

Name-Samriddhi Dixit

CYBER SECURITY INTERNSHIP-TASK 1

WEB APPLICATION SECURITY INTERNSHIP

1 Introduction

This report presents the web application security assessment conducted as part of the *Future Interns – Cyber Security* virtual internship, under Task 1: Web Application Security Testing.

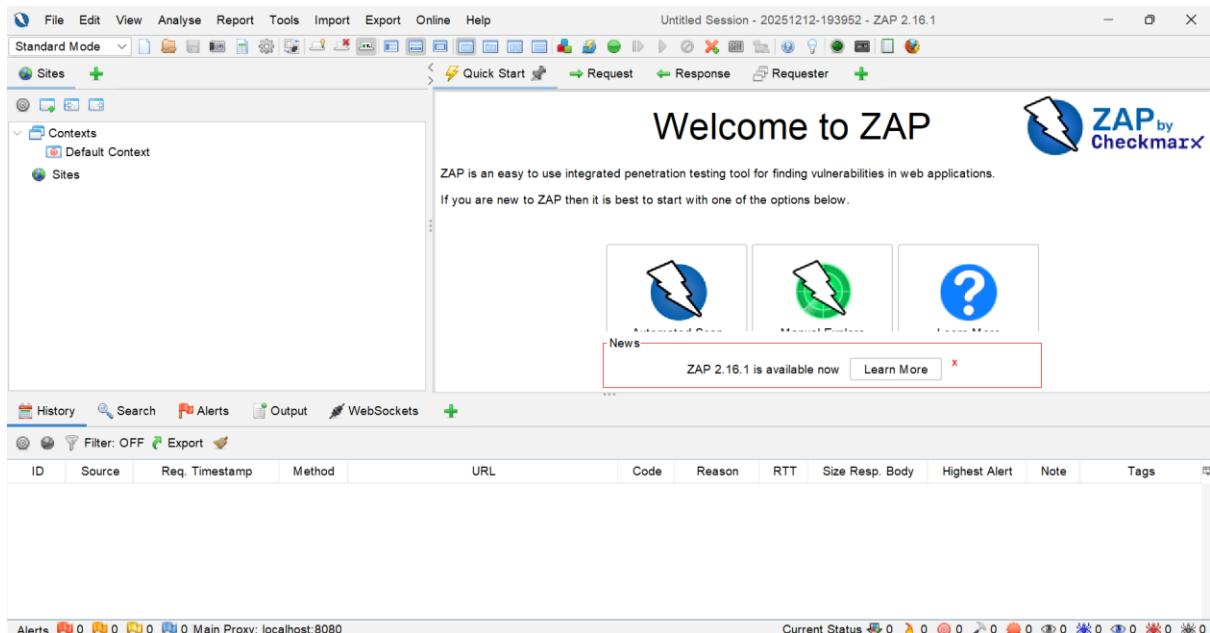
The assessment was carried out on a locally deployed instance of **OWASP Juice Shop**, an intentionally vulnerable modern web application developed to demonstrate and educate users about common web security flaws aligned with the OWASP Top 10.

The objectives of this assessment were to:

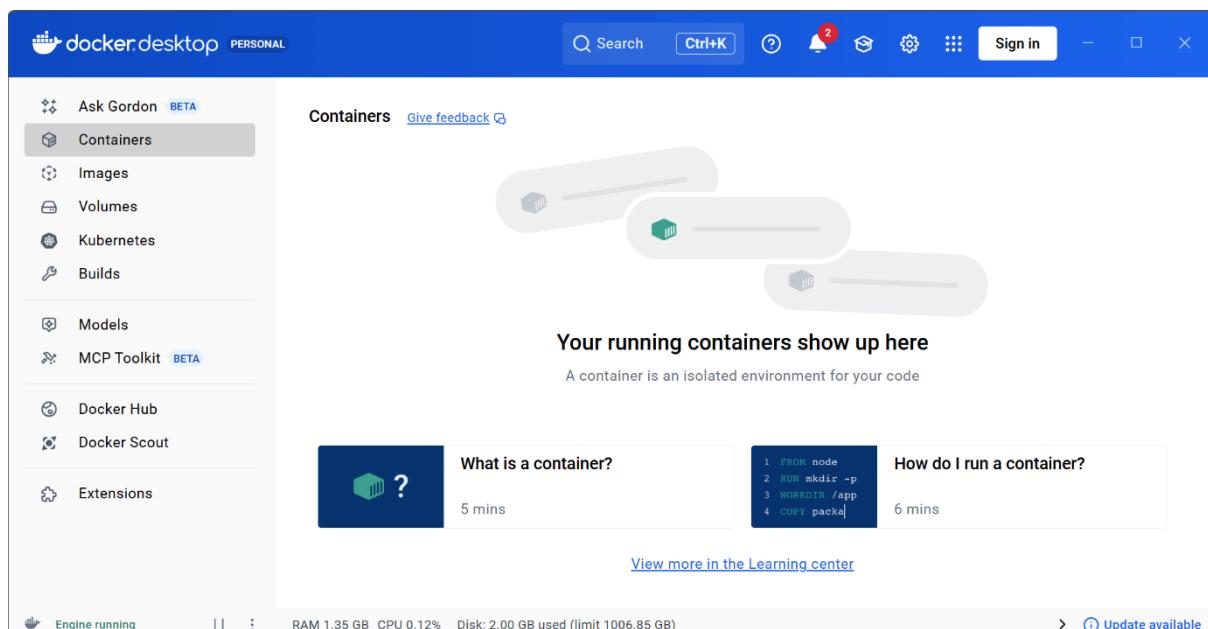
- Identify and exploit common web application vulnerabilities
- Understand their impact on confidentiality, integrity, and availability
- Map findings to OWASP Top 10 categories
- Recommend appropriate mitigation technique

2. Lab Environment

- **Operating System:** Windows (Local Machine)
- **Target Application:** OWASP Juice Shop
- **Deployment Method:** Docker (Localhost)
- **Application URL:** <http://localhost:3000>
- **Tools Used:**
 - Docker Desktop
 - OWASP ZAP
 - Web Browser (Chrome / Firefox)



OWASP ZAP HOMEPAGE



DOCKER DESKTOP-ENGINE RUNNING

3. Methodology

The security assessment followed a **black-box testing approach**, simulating the perspective of an external attacker with no prior knowledge of the application's internal implementation.

The methodology consisted of the following steps:

1. Deploying the vulnerable application locally using Docker
2. Configuring OWASP ZAP as an intercepting proxy
3. Browsing the application to capture HTTP requests and responses
4. Performing passive scanning during application exploration
5. Conducting active scans using OWASP ZAP
6. Executing manual input-based testing for common vulnerabilities
7. Analyzing request and response behavior

4. Vulnerability Findings

4.1 SQL Injection (Authentication Bypass)

Location: Login Functionality

Description:

The login feature was tested using a classic SQL Injection payload. Improper validation of user input allowed manipulation of the authentication query, resulting in an authentication bypass and unauthorized access.

Steps:

1. Navigate to the login page.
2. Enter a SQL Injection payload in the email field.
3. Enter any value in the password field.
4. Submit the login form and observe successful authentication.

Impact:

- Unauthorized access to application accounts
- Potential exposure of sensitive data

A screenshot of the OWASP Juice Shop application. At the top, there is a navigation bar with icons for back, forward, search, account, and basket. The URL in the address bar is http://localhost:3000/#/search. A green success message box says "You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)". Below the message, there is a heading "All Products" followed by two product cards. The first card shows an illustration of a juice cup with a straw and a red apple next to it. The second card shows an illustration of a container of pomace with three red apples next to it.

SQL-Entered as Admin Successfully

The screenshot shows the NetworkMiner interface with the following details:

Header: Text

```
POST http://localhost:3000/rest/user/login HTTP/1.1
host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:146.0) Gecko/20100101 Firefox/146.0
Accept: application/json, text/plain, */*
Accept-Language: en-US;q=0.5
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.
```

Body: Text

```
{"email": "OR 1=1", "password": "anything"}
```

Network Tab Data:

ID	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size	Resp. Body	Highest Alert	Note	Tags
38.554	Proxy	12/12/25, 10:25:04 PM	GET	http://localhost:3000/api/Challenges/?name=S...	200	OK	3.54 s	449 bytes		Medium		JSON
38.555	Proxy	12/12/25, 10:25:58 PM	GET	http://localhost:3000/rest/user/whoami	200	OK	46 ms	140 bytes		Medium		JSON
38.556	Proxy	12/12/25, 10:25:56 PM	GET	http://localhost:3000/rest/user/whoami	304	Not Modified	68 ms	0 bytes		Medium		
38.557	Proxy	12/12/25, 10:25:58 PM	POST	http://localhost:3000/rest/user/login	200	OK	125 ms	799 bytes		Medium		JSON
38.558	Proxy	12/12/25, 10:25:58 PM	GET	http://localhost:3000/rest/user/whoami	200	OK	98 ms	125 bytes		Medium		JSON
38.559	Proxy	12/12/25, 10:25:56 PM	GET	http://localhost:3000/rest/user/whoami	200	OK	165 ms	125 bytes		Medium		JSON
38.560	Proxy	12/12/25, 10:25:58 PM	GET	http://localhost:3000/rest/continue-code	200	OK	92 ms	79 bytes		Medium		JSON

4.2 Reflected Cross-Site Scripting (XSS)

Location: Search Functionality

Description:

User input entered in the search field is reflected back in the server response without proper output encoding, allowing injected JavaScript to be rendered in the browser

Steps:

1. Navigate to the search bar in OWASP Juice Shop.
2. Enter the following payload: <script>alert('XSS')</script>
3. Submit the search request.
4. Inspect the HTTP response using OWASP ZAP.

Impact:

- Execution of malicious scripts in the victim's browser
- Session hijacking and phishing attacks



4.3 Stored Cross-Site Scripting (XSS)

Location: Product Review / Feedback Functionality

Description:

User input submitted through product reviews and feedback fields is stored in

the backend without proper sanitization. The stored payload is later rendered in the application user interface, resulting in a persistent XSS vulnerability.

Steps:

1. Navigate to a product review or feedback section.
2. Enter the following payload: <script>alert('Stored XSS')</script>
3. Submit the review or feedback.
4. Reload or revisit the affected page.

Impact:

- Persistent attacks affecting multiple users
- Session hijacking and credential theft

The screenshot shows the ZAP 2.16.1 interface. The 'Sites' tab is selected, displaying a tree view of assets, rest endpoints, socket.io, and vendor files. The 'Request' tab is active, showing a complex URL with a stored XSS payload: "bGFzdExvZ2luSXAiOiiIwljaUwC4wiwcH3vZmIsZUltYwddl1joiL2Fzc2V0cy9wdWJsaWMvalhZz2VzL3wbG9hZHMvZGVmXVsdc5zdmclC...". Below the URL is the raw payload: ("UserId":23,"captchaId":73,"captcha":"36","comment": "\n (**samriddhi.dixit@gmail.com)\" ,\"rating\":1]". The 'Output' tab at the bottom shows a table of requests, with row 17.040 highlighted. This row details a POST request to http://localhost:3000/api/Feedbacks/ with a status of 201 Created, 72 ms RTT, and 184 bytes size.

ID	Source	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
17.036	Proxy	12/12/25, 9:47:09 PM	GET	http://localhost:3000/rest/captcha/	200	OK	154 ms	49 bytes	Medium	JSON	
17.037	Proxy	12/12/25, 9:50:17 PM	POST	http://localhost:3000/api/Feedbacks/	201	Created	68 ms	184 bytes	Medium	JSON	
17.038	Proxy	12/12/25, 9:50:17 PM	GET	http://localhost:3000/rest/user/whoami	304	Not Modified	14 ms	0 bytes	Medium		
17.039	Proxy	12/12/25, 9:50:17 PM	GET	http://localhost:3000/rest/captcha/	200	OK	179 ms	48 bytes	Medium	JSON	
17.040	Proxy	12/12/25, 9:52:09 PM	POST	http://localhost:3000/api/Feedbacks/	201	Created	72 ms	184 bytes	Medium	JSON	
17.041	Proxy	12/12/25, 9:52:09 PM	GET	http://localhost:3000/rest/user/whoami	304	Not Modified	12 ms	0 bytes			
17.042	Proxy	12/12/25, 9:52:09 PM	GET	http://localhost:3000/rest/captcha/	200	OK	133 ms	47 bytes	Medium	JSON	

XSS Payload request

![Screenshot of OWASP ZAP tool showing captured requests and responses. The left pane shows a tree view of assets, rest, socket.io, and vendor.js. The right pane shows a detailed view of a captured request for '/rest/products/6/reviews' with a status of 200 OK. The response body contains an XSS payload: '[](x)\n", "author": "21.samridhi.dixit@gmail.com", "likesCount":0, "likedBy":[]}, {"id": "AxmsXDTEv4nw13WCN"}]'. Below this, a table lists captured requests from ID 32,063 to 32,219, showing details like URL, Method, Status Code, RTT, and Reason."/>

XSS Payload Response-Vulnerability Exists

4.4 Cross-Site Request Forgery (CSRF) Analysis

Location: Password Update / Account Settings

Description:

State-changing requests were analysed by inspecting captured HTTP traffic. The absence of visible CSRF tokens in sensitive requests indicated potential exposure to CSRF attacks.

Steps:

1. Navigate to account settings.
2. Perform a password update action.
3. Inspect the request using OWASP ZAP.

Impact:

- Unauthorized actions performed on behalf of authenticated users

Password Change Done

No CSRF Token even after password change-potentially vulnerable application

5. OWASP Top 10 Mapping

Vulnerability	OWASP ID	Category
SQL Injection	A03	Injection
Reflected XSS	A07	Cross-Site Scripting

Stored XSS	A07	Cross-Site Scripting
CSRF Risk	A01	Broken Access Control

6.Conclusion

This assessment demonstrated how common web application vulnerabilities, including SQL Injection, Cross-Site Scripting, and CSRF risks, can be identified through systematic testing using OWASP ZAP. The task highlighted the importance of analysing HTTP requests and responses to understand how user input is processed by the application, rather than relying only on visible alerts.

The hands-on testing carried out during this exercise strengthened practical knowledge of ethical hacking techniques and reinforced the significance of implementing secure development practices in web applications.
