Internship Report (Week 1)

DEVELOPERS.HUB CORPORATION

Security Assessment Report

Submitted By: Muhammad Imran

Intern ID: DHC-2097

Email: muhammad.imran4842@gmail.com

TABLE OF CONTENTS

1 INTRODUCTION

In this Week 1 Security Assessment Report, we detail the evaluation of the OWASP Juice Shop web application, an intentionally vulnerable platform designed to serve as an effective training ground for web application security testing. The primary objective of this assessment is to identify potential security weaknesses within the application by employing a combination of automated scanning and manual testing techniques, all executed within a controlled environment on Kali Linux.

Objectives of the Assessment:

- **Environment Setup:**
  Deploy and configure the OWASP Juice Shop application on a local machine running Kali Linux. This includes installing essential tools such as Git, Node.js, and npm, cloning the Juice Shop repository, and ensuring the application is accessible at http://localhost:3000.
- **Automated Vulnerability Scanning:**
  Utilize OWASP ZAP to perform an automated scan of the application. The scan is designed to detect common vulnerabilities—such as missing security headers and potential Cross-Site Scripting (XSS) issues—by systematically probing various endpoints.
- **Manual Vulnerability Testing:**
  Conduct targeted manual tests using browser developer tools. This involves simulating attacks by injecting payloads such as <script>alert('XSS');</script> into input fields and attempting SQL Injection via payloads like admin' OR '1'='1 in the login form, in order to observe the application's response.
- **Documentation of Findings:**
  Systematically record all identified vulnerabilities along with their potential impacts and recommendations for remediation. This documentation will serve as the foundation for the next phases, where these issues will be addressed through various security measures.

The deliberately vulnerable nature of OWASP Juice Shop allows us to simulate real-world attack scenarios and evaluate the effectiveness of our security testing methodologies. By analyzing the results from both automated and manual tests, we gain valuable insights into the application's current security posture. These findings will guide the implementation of necessary fixes in subsequent phases, such as input sanitization, password hashing, secure token-based authentication, and the utilization of secure HTTP headers.

This report not only documents the detailed steps taken during the setup and testing processes but also provides a reproducible methodology for web application security assessments. The insights derived from this assessment are essential for establishing secure coding practices and preparing to mitigate vulnerabilities in more complex systems.

---

2 ENVIRONMENT SETUP

Setting up the right environment is a critical step before conducting any security assessment. In this task, we deployed OWASP Juice Shop, a vulnerable web application, on Kali Linux to simulate real-world security testing. This setup allows us to analyze vulnerabilities using both automated scanning tools like OWASP ZAP and manual testing techniques using browser developer tools.

A. Installing Requirement Tools:

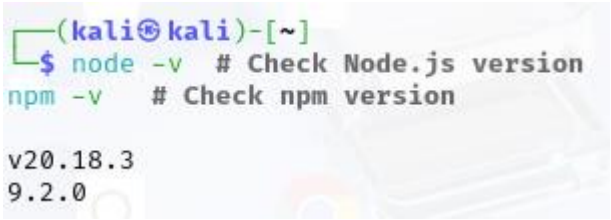
To ensure a smooth setup, we first installed the necessary dependencies on Kali Linux:

- **Git** – To download the OWASP Juice Shop source code.
- **Node.js & npm (Node Package Manager)** – Required to run Juice Shop, as it is built using Node.js.

Installation Commands:

sudo **apt** update	sudo **apt** install git nodejs npm -y

To confirm successful installation:



node -v # Check Node.js version	npm -v # Check npm version

B. Downloading and Running OWASP Juice Shop

Once the dependencies were installed, we downloaded and launched Juice Shop using the following steps:

- **Clone the repository from GitHub:**

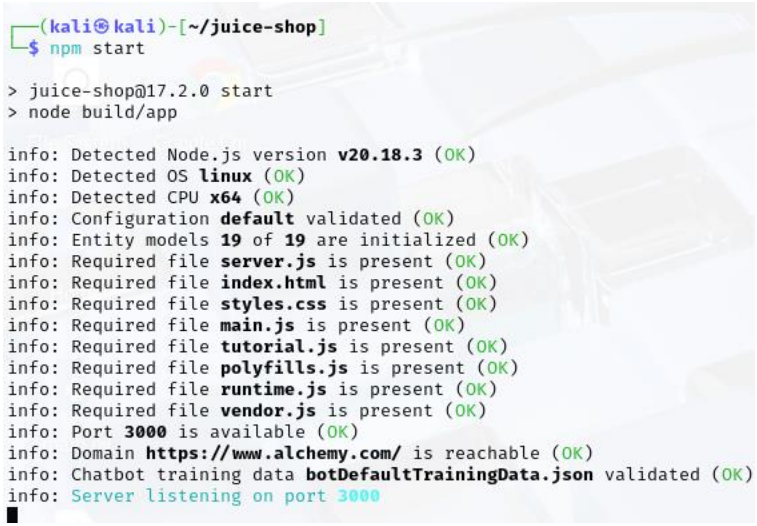
git clone https://github.com/juice-shop/juice-shop.git	**cd** juice-shop

- **Install application dependencies:**

npm install

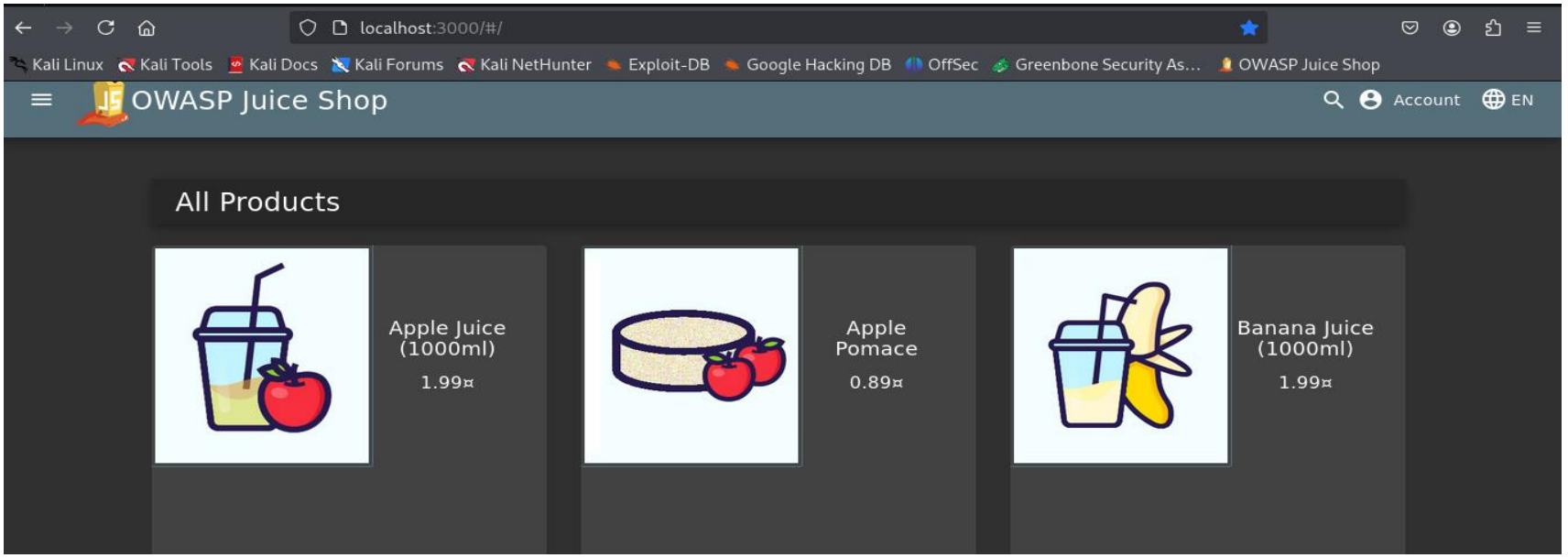
- **Start the application:**

npm start



➢ **Access Juice Shop in a browser:**

Open **http://localhost:3000** to verify successful deployment.



C. Verifying the Setup

After launching the application, we performed the following checks:

☑ Terminal Verification: Ensured that Juice Shop was running successfully without errors.

☑ Browser Check: Opened http://localhost:3000 to confirm that the Juice Shop homepage loaded correctly.

3 TOOLS USED

1. OWASP ZAP
OWASP ZAP (version XYZ) is used for automated scanning of the web application. This tool helps in detecting common vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection, and missing security headers. By automatically probing various endpoints, OWASP ZAP provides a comprehensive overview of the application's security posture.

2. Browser Developer Tools
Browser Developer Tools (accessed via the F12 key) are employed for manual testing. These tools allow us to inspect and manipulate input fields, enabling the testing of various payloads directly within the browser. This manual approach helps in identifying vulnerabilities like XSS and SQL Injection by observing how the application handles user inputs.

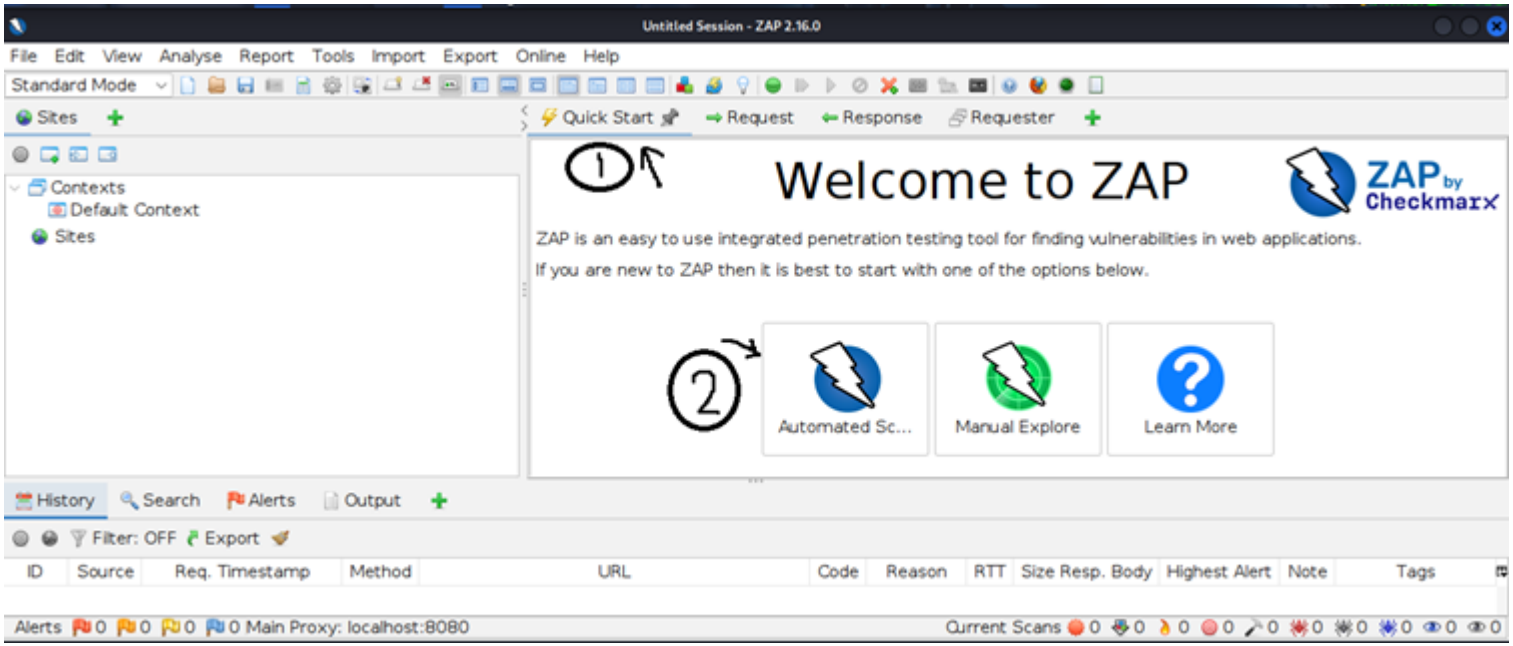
4 VULNERABILITY ASSESSMENT PROCESS

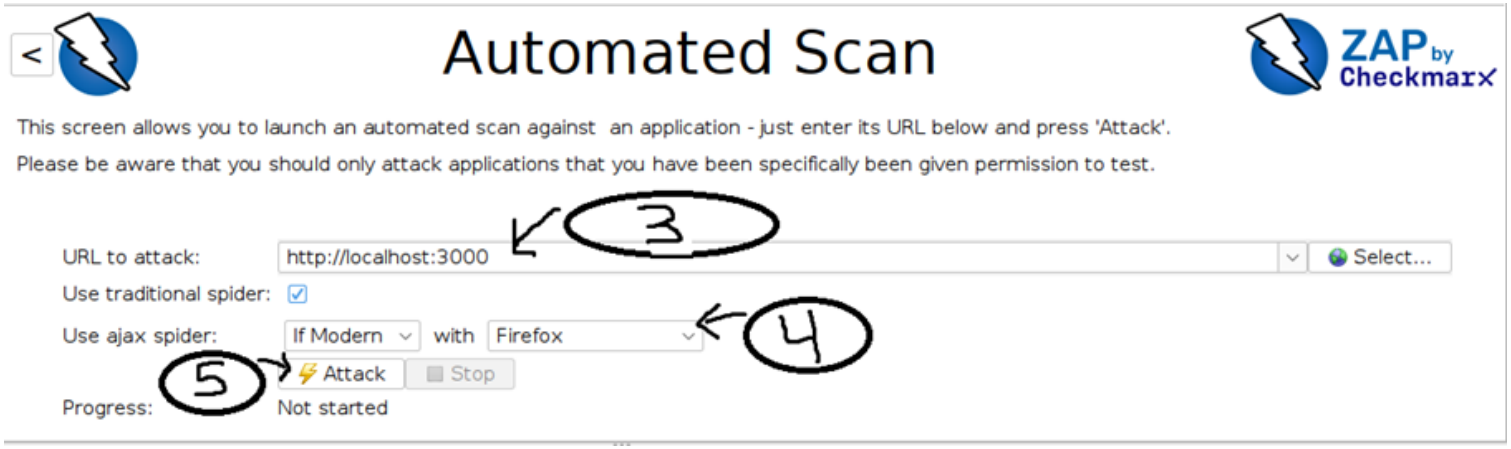
A. Automated Scan (OWASP ZAP)

ZAP Launch and Session Setup:
OWASP ZAP was launched, and a prompt appeared asking whether to save the session. We opted to persist the session for future reference, ensuring that all scan data and findings are retained for later analysis and reporting.

Setting the Target URL:
The target URL was set to http://localhost:3000, which is where the OWASP Juice Shop application is running. Once the target was configured, we initiated the scan by using either the Quick Start or Active Scan options within ZAP.

Scan Process:
After the scan was initiated, OWASP ZAP automatically probed multiple endpoints of the application. During this process, the tool identified and listed various vulnerabilities, such as Cross-Site Scripting (XSS), SQL Injection, and missing security headers. The automated scanning process provided a comprehensive overview of the application's security posture by systematically examining its endpoints.


Untitled Session - ZAP 2.16.0
File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Quick Start
Request
Response
Requester
Contexts
Default Context
Sites
1
Welcome to ZAP
ZAP by Checkmarx
ZAP is an easy to use integrated penetration testing tool for finding vulnerabilities in web applications.
If you are new to ZAP then it is best to start with one of the options below.
2
Automated Sc...
Manual Explore
Learn More
History
Search
Alerts
Output
Filter: OFF
Export
ID
Source
Req. Timestamp
Method
URL
Code
Reason
RTT
Size Resp. Body
Highest Alert
Note
Tags
Alerts 0 0 0 0 Main Proxy: localhost:8080
Current Scans 0 0 0 0 0 0 0 0 0 0


Automated Scan
ZAP by Checkmarx
This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.
3
URL to attack:
http://localhost:3000
Select...
Use traditional spider:
Use ajax spider:
If Modern
with
Firefox
4
5
Attack
Stop
Progress:
Not started

Report Results: Here is general view of the report but I attached detailed report with this file folder.

		Confidence				
		User Confirmed	High	Medium	Low	Total
Risk	High	0 (0.0%)	0 (0.0%)	1 (5.0%)	0 (0.0%)	1 (5.0%)
	Medium	0 (0.0%)	2 (10.0%)	3 (15.0%)	0 (0.0%)	5 (25.0%)
	Low	0 (0.0%)	0 (0.0%)	3 (15.0%)	1 (5.0%)	4 (20.0%)
	Informational	0 (0.0%)	0 (0.0%)	9 (45.0%)	1 (5.0%)	10 (50.0%)
	Total	0 (0.0%)	2 (10.0%)	16 (80.0%)	2 (10.0%)	20 (100%)

Alert counts by site and risk

This table shows, for each site for which one or more alerts were raised, the number of alerts raised at each risk level.

Alerts with a confidence level of "False Positive" have been excluded from these counts.

(The numbers in brackets are the number of alerts raised for the site at or above that risk level.)

		Risk			
		High (= High)	Medium (>= Medium)	Low (>= Low)	Informational (>= Informational)
Site	http://cdnjs.cloudflare.com	0 (0)	1 (1)	0 (1)	3 (4)
	http://localhost:3000	1 (1)	4 (5)	4 (9)	7 (16)

Alert counts by alert type

This table shows the number of alerts of each alert type, together with the alert type's risk level.

(The percentages in brackets represent each count as a percentage, rounded to one decimal place, of the total number of alerts included in this report.)

Alert type	Risk	Count
SQL Injection - SQLite	High	1 (5.0%)
Content Security Policy (CSP) Header Not Set	Medium	65 (325.0%)
Cross-Domain Misconfiguration	Medium	98 (490.0%)
Missing Anti-clickjacking Header	Medium	8 (40.0%)
Session ID in URL Rewrite	Medium	37 (185.0%)
Vulnerable JS Library	Medium	1 (5.0%)

B. Manual Testing

I. XSS Test

➢ Test Methodology

We performed multiple injection strategies:

1. **Email Field with XSS Payload + Dummy Password**

Email Field:

Tried both a direct **<script>alert('XSS');</script>** and a partial email + script combo like **test@example.com<script>alert('XSS');</script>.**

Password Field:

Entered a dummy password such as **test123** or **12345**.

Expected Result: If the script was executed, an alert box with “XSS” would appear.

2. Valid Email + XSS Payload in Password Field

Email Field:

Used a valid email format, e.g., **test@example.com**.

Password Field:

Pasted **<script>alert('XSS');</script>**.

Expected Result: If the password input were reflected unsafely, the script might execute, producing an alert box.

3. Submission & Observation

After entering the payload, we clicked the Login button to trigger form submission.

We closely monitored the browser for any alert pop-ups and inspected how the application displayed or handled our input.

II. SQL Injection Test

Process:
On the login page, we entered the injection string:

admin' OR '1'='1

into both the username and password fields. This test is designed to determine whether the application is vulnerable to SQL Injection by bypassing authentication mechanisms.

Result:
The outcome is noted based on whether the injection attempt is successful. If the login is successful or the application exhibits any unusual behavior (such as error messages hinting at SQL query issues), it indicates the presence of an SQL Injection vulnerability. If the login fails as expected, it suggests that the vulnerability might be mitigated. These results are documented for further analysis.

III. NoSQL Injection Test

Process:
Using the browser's Developer Tools (accessed via the F12 key), we navigated to the login page of the OWASP Juice Shop application. In this test, we attempted to bypass the authentication mechanism by injecting a typical NoSQL payload into the username field. For example, we entered:

{"$gt": ""}
{"$ne": ""}

into the username field and left the password field empty (or applied a similar payload in both fields). This payload is designed to manipulate the underlying NoSQL query logic, potentially allowing unauthorized access if the input is not properly validated.

Result:
The outcome of the injection attempt was carefully observed. If the login was successful or if the application exhibited any unusual behavior—such as unexpected error messages or unauthorized access—it would indicate that a NoSQL injection vulnerability exists. If the login fails as expected, it suggests that the vulnerability may be mitigated. These results are documented for further analysis and to guide subsequent remediation efforts.

Final Results for SQL & NoSQL injection:

Test: SQL Injection on Login Form

Payload: **admin' OR '1'='1**

Result: "Invalid email or password" message appeared.

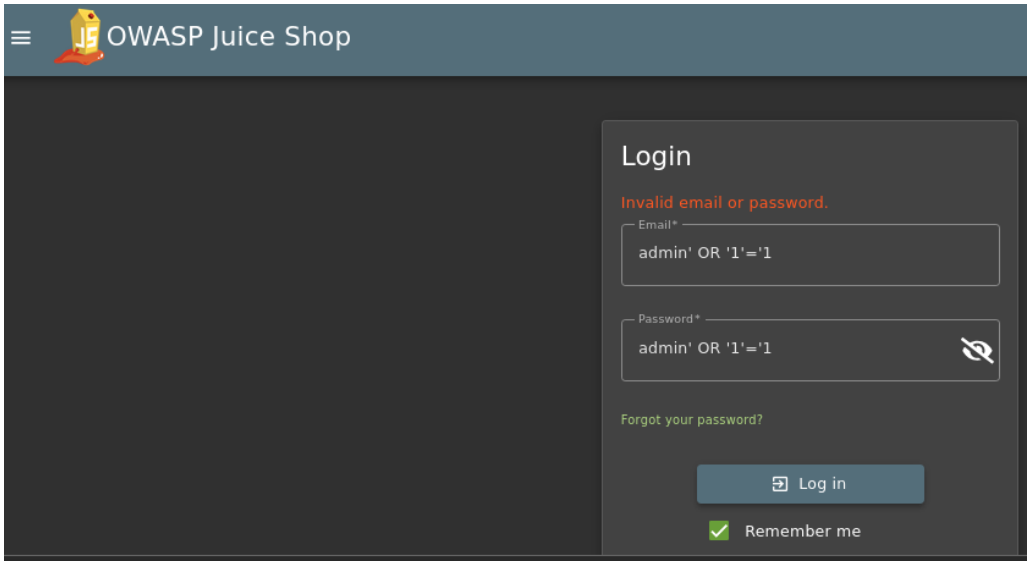
Conclusion: The login form appears secure against SQL injection.

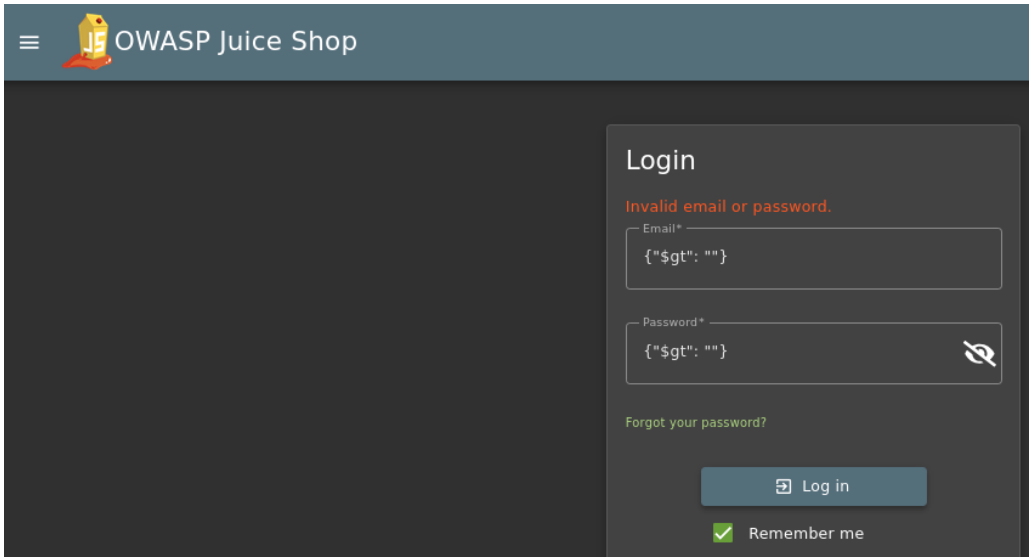
Test: NoSQL Injection on Login Form

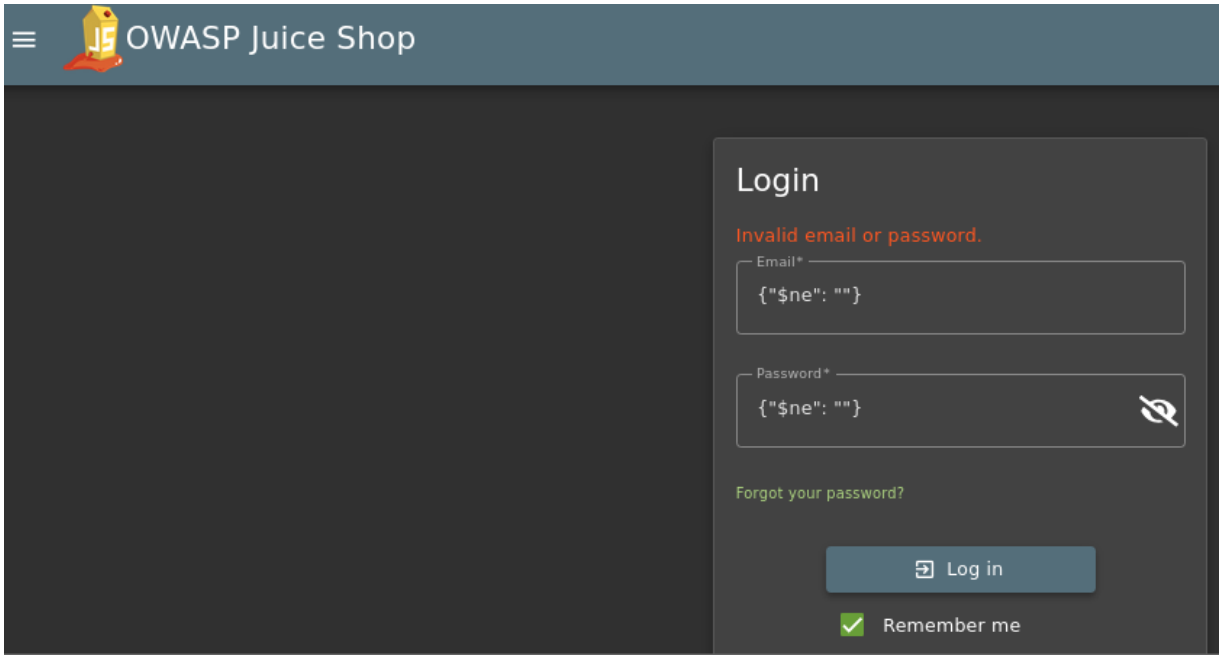
Payload: **{"$gt": ""}** & **{"$ne": ""}** for both email and password fields.

Result: "Invalid email or password" message appeared.

Conclusion: The login form appears secure against NoSQL injection.


OWASP Juice Shop
Login
Invalid email or password.
Email*
admin' OR '1'='1
Password*
admin' OR '1'='1
Forgot your password?
Log in
Remember me


OWASP Juice Shop
Login
Invalid email or password.
Email*
{"$gt": ""}
Password*
{"$gt": ""}
Forgot your password?
Log in
Remember me



Final Results for XSS injection:

Test Environment:

- **Application:** OWASP Juice Shop (accessed at **http://localhost:3000/#/login**)
- **Tool:** Browser Developer Tools (F12)
- **Payloads Tested:**
  - **Option 1:** In the Email field, we used the payload: *test@example.com<script>alert('XSS');</script>*
  - **Option 2:** In the Password field (with a valid email in the Email field), we used the payload: *<script>alert('XSS');</script>*

Observations:

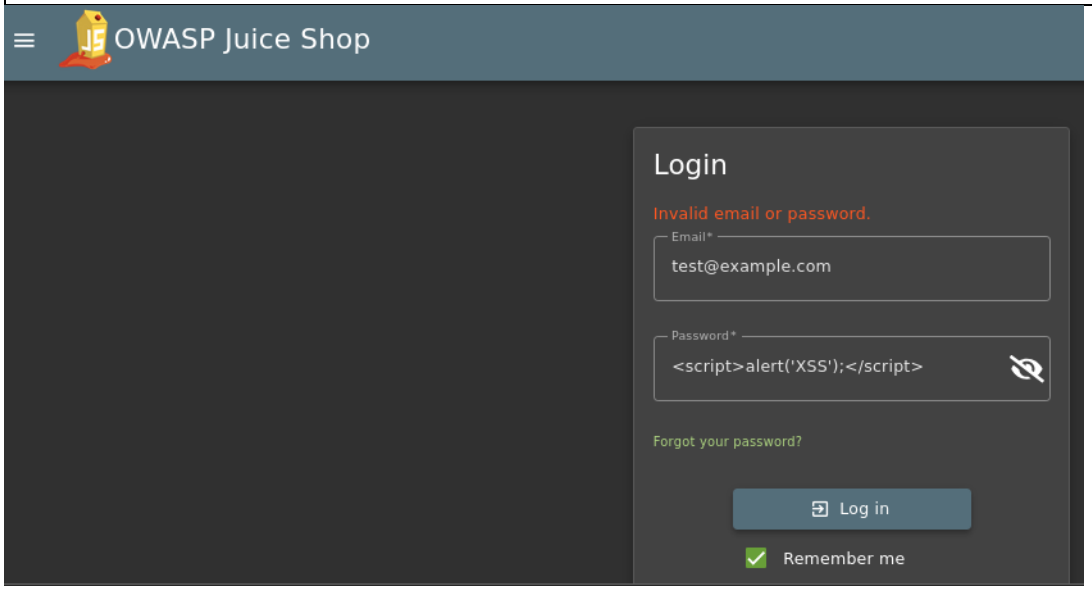
- In every test attempt, instead of triggering an alert popup, the application displayed:

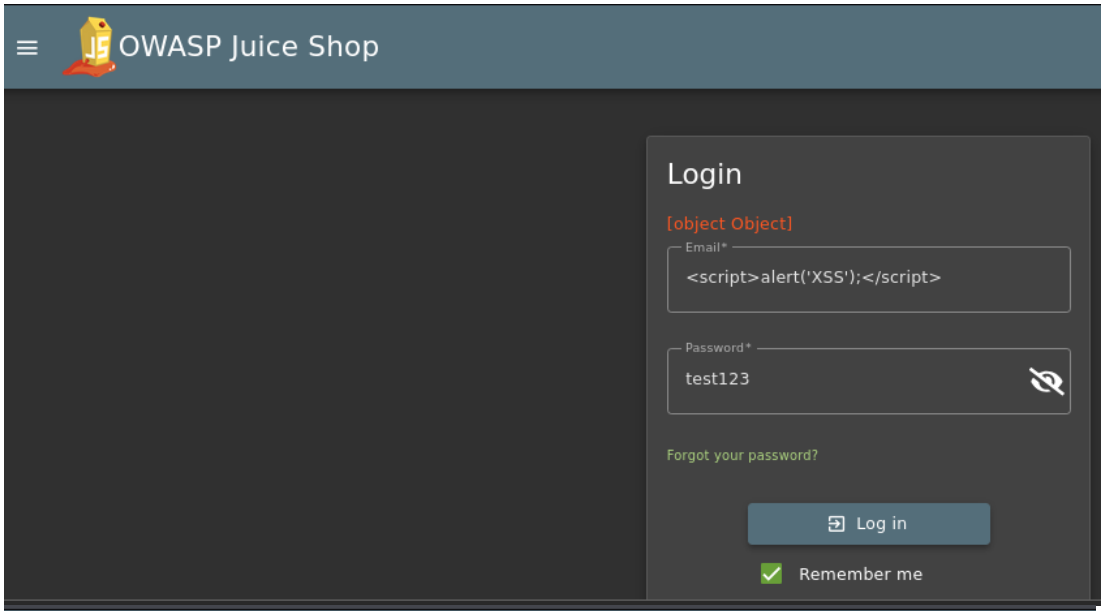
  [object Object]

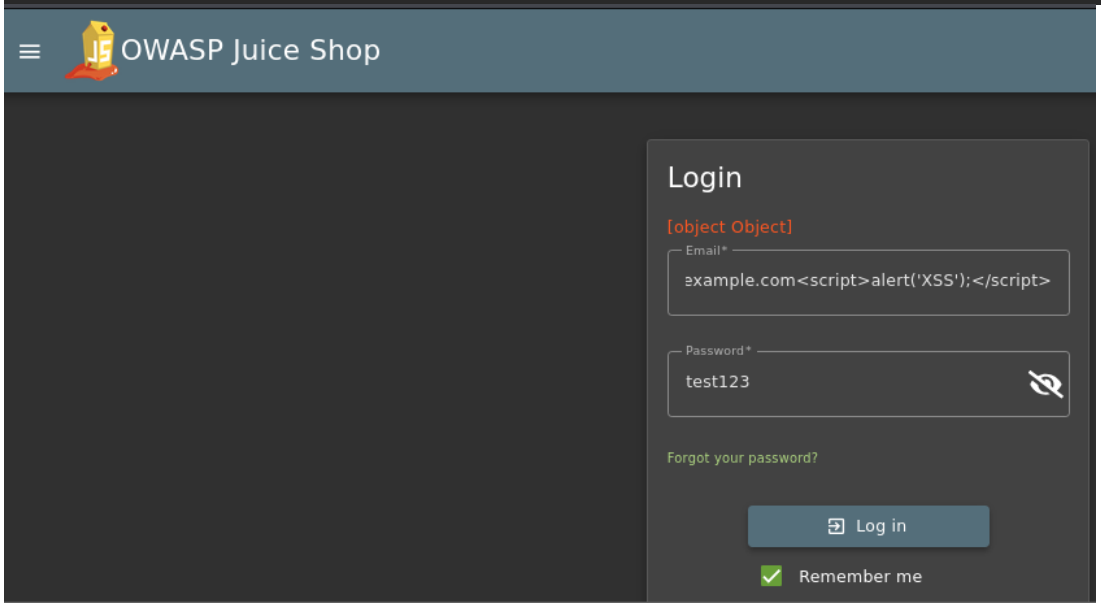
- No alert box was triggered, which indicates that the script did not execute as intended.

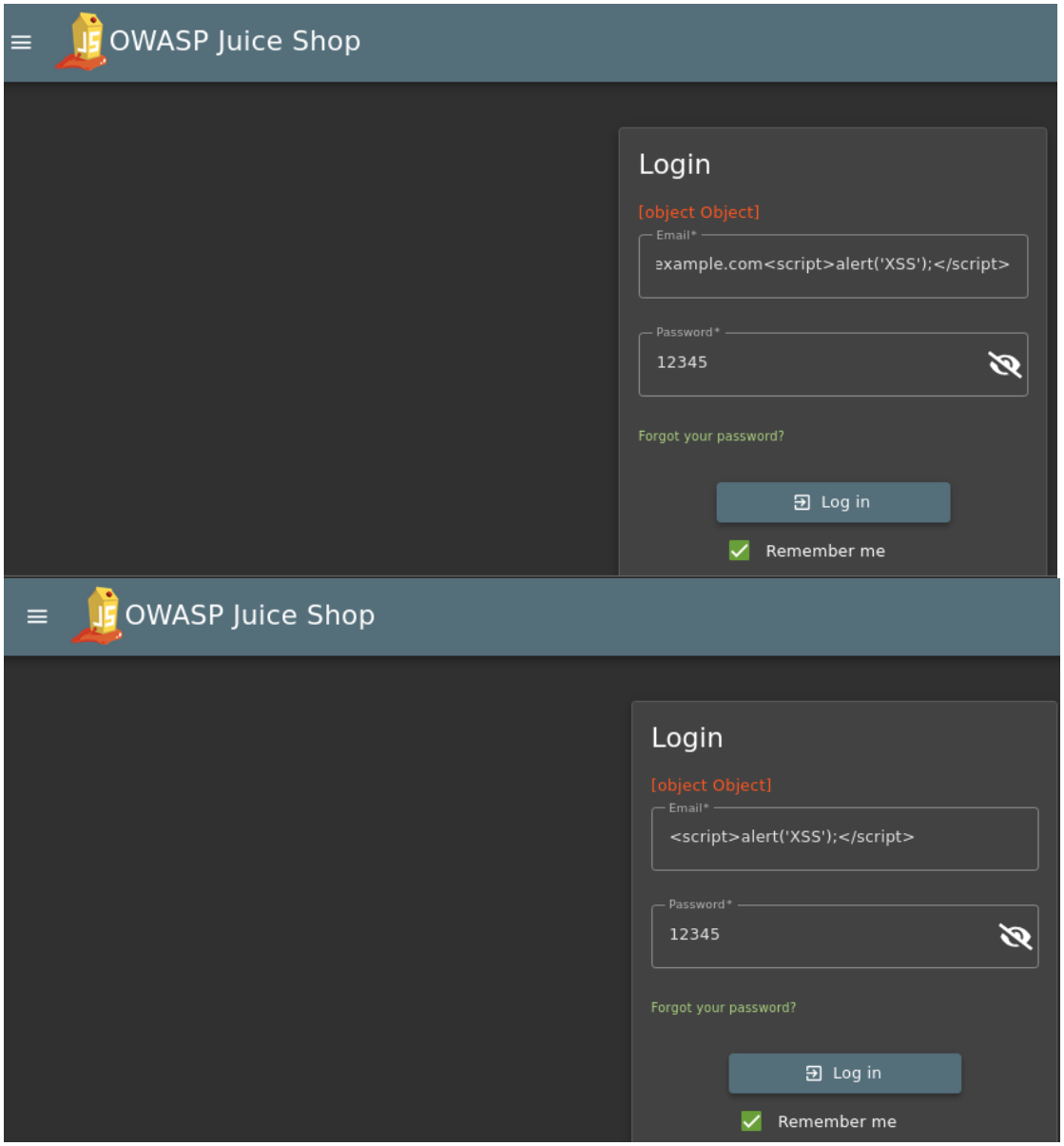
Conclusion:

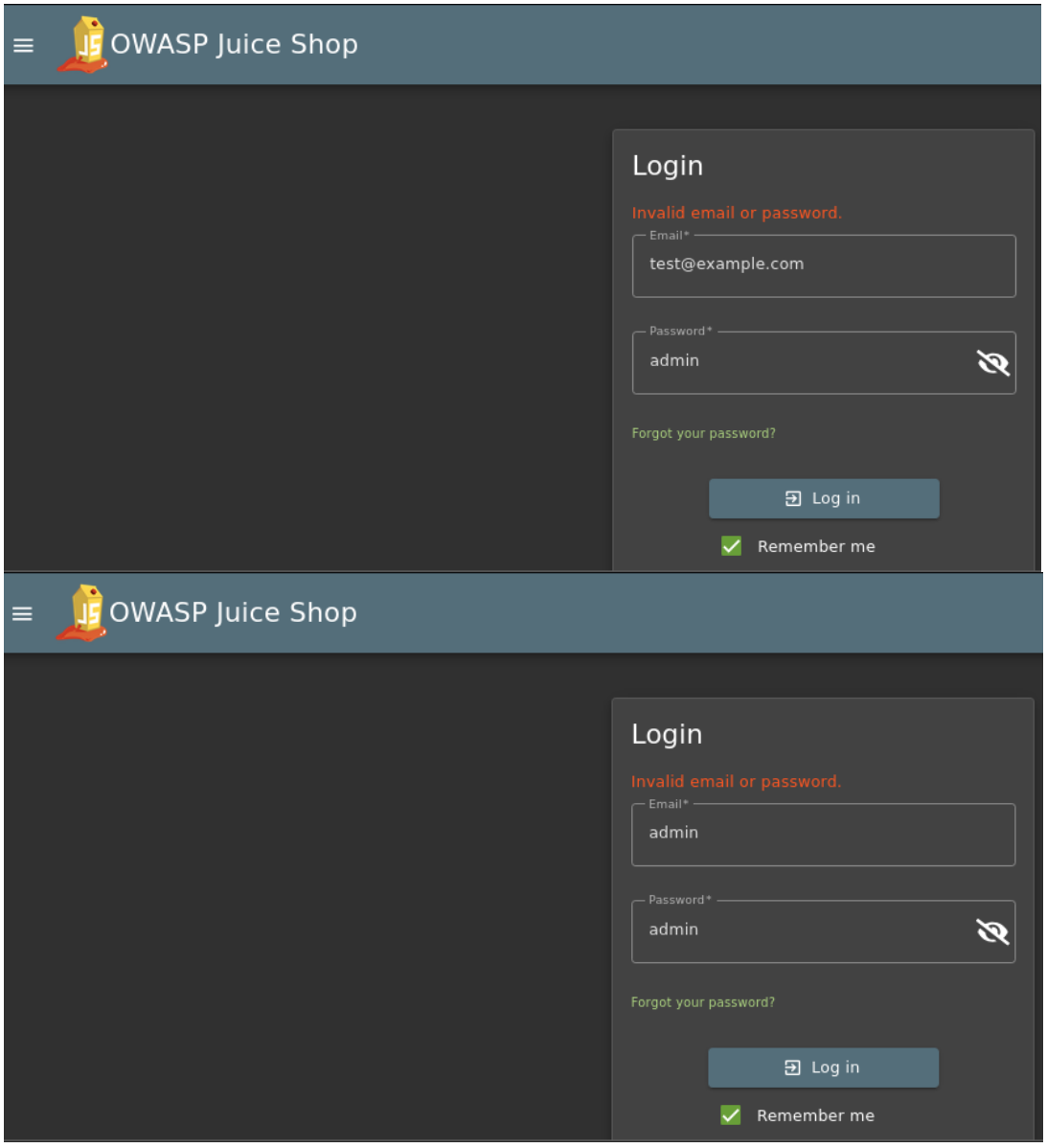
- The results suggest that the login form is not vulnerable to this basic XSS attack.
- The [object Object] output indicates that the application is likely sanitizing or escaping the input, treating it as plain text rather than executable HTML/JavaScript.




OWASP Juice Shop
Login
[object Object]
Email*
<script>alert('XSS');</script>
Password*
test123
Forgot your password?
Log in
Remember me


OWASP Juice Shop
Login
[object Object]
Email*
example.com<script>alert('XSS');</script>
Password*
test123
Forgot your password?
Log in
Remember me


OWASP Juice Shop
Login
[object Object]
Email*
example.com<script>alert('XSS');</script>
Password*
12345
Forgot your password?
Log in
Remember me
OWASP Juice Shop
Login
[object Object]
Email*
<script>alert('XSS');</script>
Password*
12345
Forgot your password?
Log in
Remember me


OWASP Juice Shop
Login
Invalid email or password.
Email*
test@example.com
Password*
admin
Forgot your password?
Log in
Remember me
OWASP Juice Shop
Login
Invalid email or password.
Email*
admin
Password*
admin
Forgot your password?
Log in
Remember me

5 FINDINGS AND OBSERVATIONS

SQL Injection Test:

- **Process:** The injection string admin' OR '1'='1 was entered into both the email and password fields.
- **Result:** The login form responded with an "Invalid email or password" message, indicating that the injection did not bypass authentication.
- **Conclusion:** The login form is not vulnerable to this classic SQL injection attempt.

NoSQL Injection Test:

- **Process:** The NoSQL payload {"$gt": ""} was inserted into both the email and password fields.
- **Result:** The same "Invalid email or password" message was returned, showing that the injection attempt failed.
- **Conclusion:** The login form appears secure against NoSQL injection attacks, likely due to proper input validation and sanitization.

XSS Injection Test:

- **Process:** Two approaches were attempted. In Option 1, a valid email combined with an XSS payload (test@example.com<script>alert('XSS');</script>) was used in the email field. In Option 2, a valid email was provided while the XSS payload (<script>alert('XSS');</script>) was inserted into the password field.
- **Result:** Instead of triggering an alert box, the application displayed [object Object], which indicates that the input was not executed as script but was instead processed as a plain object/string.
- **Conclusion:** The login form does not seem vulnerable to basic XSS attacks, as it sanitizes or escapes user input effectively.

Test Results Summary

Test Type	Payload Used	Observed Result	Conclusion
SQL Injection	admin' OR '1'='1 in both Email & Password fields	"Invalid email or password" message appears	Login form rejects SQL injection; appears secure against SQLi.
NoSQL Injection	{"$gt": ""} in both Email & Password fields	"Invalid email or password" message appears	Login form rejects NoSQL injection; appears secure against NoSQLi.
XSS Injection	Option 1: test@example.com<script>alert('XSS');</script> in EmailOption 2: <script>alert('XSS');</script> in Password field	[object Object] is displayed; no alert popup	XSS payload did not execute; application sanitizes/escapes input.

6 CONCLUSION

Based on the tests performed on the OWASP Juice Shop login form, our assessment indicates that the form effectively mitigates common injection attacks. Specifically:

- **SQL Injection:** The payload admin' OR '1'='1 consistently resulted in an "Invalid email or password" message, demonstrating that the login form rejects SQL injection attempts.
- **NoSQL Injection:** The use of {"$gt": ""} in both email and password fields also led to the same rejection message, confirming that the login form is resilient against NoSQL injection.

- **XSS Injection:** Despite testing with various XSS payloads (e.g., appending <script>alert('XSS');</script> to valid input), the output displayed was [object Object] rather than executing the script. This suggests that the application sanitizes or escapes user input, preventing the execution of injected scripts.

In summary, these results indicate that the login form is not vulnerable to the basic injection attacks that were tested. While these findings are promising, it is recommended to conduct further tests using advanced payloads across different parts of the application to ensure comprehensive security.

7 REFERENCES

List References:

- OWASP ZAP Documentation: https://www.zaproxy.org/
- OWASP Juice Shop GitHub: https://github.com/bkimminich/juice-shop