OWASP JUICE SHOP

“ An intentionally insecure web application for security testing. ”

WEB APPLICATION SECURITY TESTING REPORT

PREPARED BY JIGYASHA RAJBHANDARI

05 JULY 2025

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PURPOSE OF THE REPORT

This report presents a structured security assessment of the OWASP Juice Shop web application, performed using a combination of automated scanning and manual penetration testing techniques.

The primary objective is to uncover exploitable vulnerabilities, insecure configurations, and gaps in input validation, session management, and access control. The findings are aligned with the OWASP Top 10 security risks to ensure relevance and industry best practices.

The report serves as a reflection of experiential exploration rather than a formal audit. It emphasizes the practical application of theoretical concepts from cybersecurity education by simulating attacker techniques in a safe and controlled environment.

1.2 SCOPE OF ASSESSMENT

The assessment focuses on identifying and exploiting security vulnerabilities within the OWASP Juice Shop web application using both automated tools and manual penetration testing techniques. The objective is to evaluate the application's resistance to common web threats, including injection flaws, authentication weaknesses, and client-side attacks such as XSS and SQL.

This scope covers vulnerability discovery, exploitation, and risk analysis to provide actionable insights for improving the security posture of the Juice Shop only, excluding underlying network and backend systems.

2. TOOLS AND ENVIRONMENT

2.1 TOOLS UTILIZED

- OWASP ZAP
- Burp Suite Community Edition
- Firefox with FoxyProxy
- Docker container environment

2.2 RATIONALE FOR TOOL SELECTION

OWASP ZAP handled automated scans, detecting vulnerabilities based on the OWASP Top 10. It provided a solid baseline, quickly highlighting common issues without requiring extensive manual effort.

Burp Suite Community Edition was used for deeper testing. It allowed interception and modification of requests and responses, uncovering complex injection points and business logic flaws that automated tools often miss.

Firefox paired with FoxyProxy routed browser traffic through proxies, enabling precise control and dynamic interaction with the application during testing.

The OWASP Juice Shop application and testing tools were deployed inside **Docker** containers. This setup ensured an isolated and consistent environment while avoiding the overhead of full virtual machines.

Overall, this combination of automated scanning and manual testing enabled effective identification of both common vulnerabilities and deeper, more complex flaws.

2.3 SYSTEM CONFIGURATION DETAILS

- OWASP Juice Shop version: 18.0.0
- OWASP ZAP version: 2.16.1
- Burp Suite Community Edition version: 2025.5.6
- Firefox version: 140.0.2
- FoxyProxy version: 9.2
- Docker version: 4.41.2
- Host OS: Windows 11 Pro 24H2

3. WEB APPLICATION SCANNING - AUTOMATED TESTING WITH OWASP ZAP

The goal of this phase was to perform a baseline security assessment of the OWASP Juice Shop application using automated scanning techniques. This process identifies commonly known vulnerabilities, misconfigurations, and missing security controls without the need for complex manual testing.

***Note**: Deeper context-aware injections (XSS and SQL) are handled separately in the manual testing phase, refer to* Web Application Manual Testing – Burp Suite Assessment

3.1 ALERT SUMMARY

Risk Level	Alerts	Summary Risk Type
🔴 High	0	Exploitable flaws leading to system-level compromise
🟠 Medium	3	Misconfigurations, missing headers, or CORS issues
🟡 Low	2	Minor leaks and unnecessary metadata in responses
🔵 Informational	2	Non-critical indicators helpful for recon or context

Table 1 OWASP ZAP Automated Scan Alert Summary Table

The full ZAP scan report (HTML format) has been attached as a supplementary document in Appendix 10.1

3.2 OWASP TOP 10 VULNERABILITY MAPPING AND RECOMMENDATIONS

3.2.1 Content Security Policy (CSP) Header Not Set

OWASP Top 10 Mapping: A05: Security Misconfiguration

Severity Level: 🟠 Medium

Instances Detected: 57

Description: The application does not define a `Content-Security-Policy` header, exposing it to cross-site scripting (XSS) and data injection threats.

Impact: Users may load unauthorized or malicious content, leading to data theft or session hijacking.

Recommendation: Implement a strict CSP header. Example:

```
Content-Security-Policy: default-src 'self';
```

Evidence:

```
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: Fri, 27 Jun 2025 04:49:57 GMT
ETag: W/"138f5-197afb88eda"
Content-Type: text/html; charset=UTF-8
Content-Length: 80117
Vary: Accept-Encoding
Date: Fri, 27 Jun 2025 06:08:16 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

Figure 1 Absence of CSP Header

3.2.2 Cross-Domain Misconfiguration (CORS)

OWASP Top 10 Mapping: A01: Broken Access Control

Severity Level: 🟠 Medium

Instances Detected: 72

Description: The server is configured to allow cross-origin requests from any domain (*), making it vulnerable to unauthorized data access.

Impact: Untrusted domains can interact with the backend, potentially leading to data leakage or abuse of session tokens.

Recommendation: Use precise domain restrictions in CORS settings. Avoid wildcard origins in production environments.

Evidence:

```
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
```

Figure 2 Wildcard Access Control Origin

3.2.3 Hidden Files Found

OWASP Top 10 Mapping: A05: Security Misconfiguration

Severity Level: 🟠 Medium

Instances Detected: 4

Description: Files such as .env, .git, or other temporary resources were accessible via the browser.

Impact: Could expose sensitive information such as credentials, keys, or internal source code.

Recommendation: Restrict access to sensitive files using server configuration (e.g., .htaccess) and remove any dev files from production.

Evidence:

```
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: Fri, 27 Jun 2025 04:49:57 GMT
ETag: W/"138f5-197afb88eda"
Content-Type: text/html; charset=UTF-8
Content-Length: 80117
Vary: Accept-Encoding
Date: Fri, 27 Jun 2025 06:08:50 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

Hidden File Found

URL:	http://localhost:3000/.hg
Risk:	Medium
Confidence:	Low
Parameter:	
Attack:	
Evidence:	HTTP/1.1 200 OK
CWE ID:	538
WASC ID:	13
Source:	Active (40035 - Hidden File Finder)

Figure 3 Hidden File Access Vulnerability

3.2.4 Cross-Domain JavaScript Inclusion

OWASP Top 10 Mapping: A08: Software and Data Integrity Failures

Severity Level: 🟡 Low

Instances Detected: 96

Description: External scripts from third-party domains are being loaded without integrity checks.

Impact: These scripts could be tampered with to deliver malicious payloads to users

Recommendation: Host critical scripts locally or use Subresource Integrity (SRI) to ensure file authenticity.

Evidence:

URL:	http://localhost:3000/
Risk:	Low
Confidence:	Medium
Parameter:	//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js
Attack:	
Evidence:	<script src="//cdnjs.cloudflare.com/ajax/libs/cookieconsent2/3.1.0/cookieconsent.min.js"></script>
CWE ID:	829
WASC ID:	15

Figure 4 Hidden File Access Vulnerability

4. WEB APPLICATION MANUAL TESTING – BURP SUITE ASSESSMENT

This section documents the manual vulnerability testing phase using Burp Suite Community Edition. It complements the automated OWASP ZAP scan by targeting business logic issues, input validation flaws, and context-specific vulnerabilities not typically uncovered by automated tools.

4.1 TOOLS USED

- Manual Web Application Testing (Browser)
- Burp Suite (Community Edition) for request inspection and testing
- Firefox + Foxy Proxy for routing requests through Burp

Vulnerability	OWASP Top 10	Risk Level
Reflected Cross-Site Scripting (XSS)	A03: Injection	🔴 High
SQL Injection (Login Bypass)	A03: Injection	🔴 High

Table 2 Burp Suite Vulnerability Findings Summary Table

4.2 VULNERABILITIES PROOF OF CONCEPT (POC)

4.2.1 Reflected Cross-Site Scripting (XSS)

Endpoint:

`/search?q=`

Payload Used:

`<script>alert("Reflected-XSS-Success!!")</script>` **BLOCKED**

`<iframe src=javascript:alert('Reflected-XSS-Success')>` **EXECUTED**

Observation:

`<script>` tags are filtered or escaped by the frontend (Angular's built-in sanitization). `<iframe src=javascript:...>` bypassed sanitization and successfully executed. Therefore, certain tag-based payloads like `<iframe>` bypass Angular's sanitization in specific contexts (e.g., inside dynamic routing with query parameters like /#/search?q=).

Impact:

The Attacker can execute arbitrary scripts in a user's browser session.

Remediation:

Properly encode user inputs in HTML contexts.

PoC **Screenshot**:

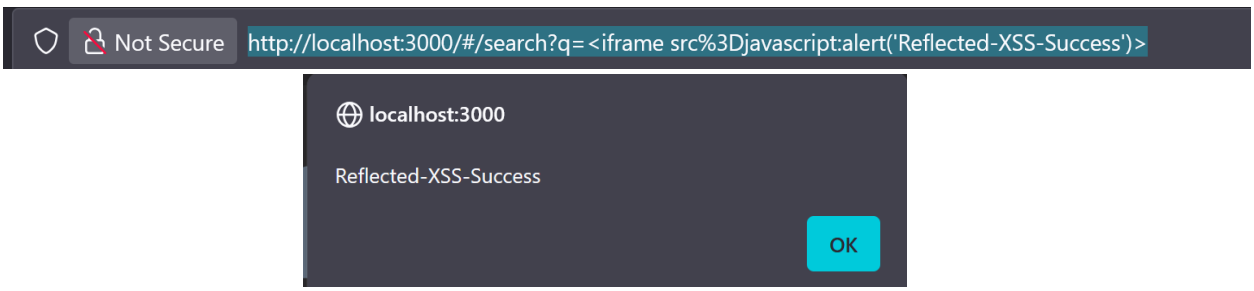


Figure 2 Reflected XSS Executed in OWASP Juice Shop Search Input Field

4.2.2 SQL Injection (Login Bypass)

Endpoint:

`POST /rest/user/login`

Payload Used:

Email: `' OR 1=1--` Password: `demo123`

Observation:

The payload ' OR 1=1-- in the email field bypassed the authentication logic. Upon sending the request, the server responded with a valid authentication token, allowing access without a valid user credential.

This confirms that the backend does not properly sanitize user-supplied input in SQL queries.

Impact:

An attacker can log in as an arbitrary user without knowing the correct password. This can lead to unauthorized access to :

- Unauthorized access to protected user accounts
- Elevation of privileges if admin accounts are targeted
- Potential data leakage or manipulation

Remediation:

- Use parameterized queries or prepared statements to handle user input securely.
- Avoid directly concatenating user input into SQL queries.
- Implement strong input validation on the server side.

PoC Screenshot:

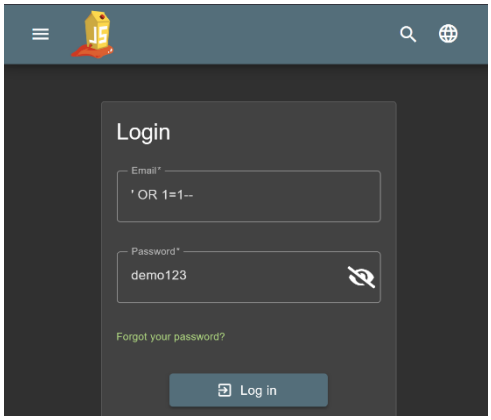


Figure 3 SQL Injection Payload

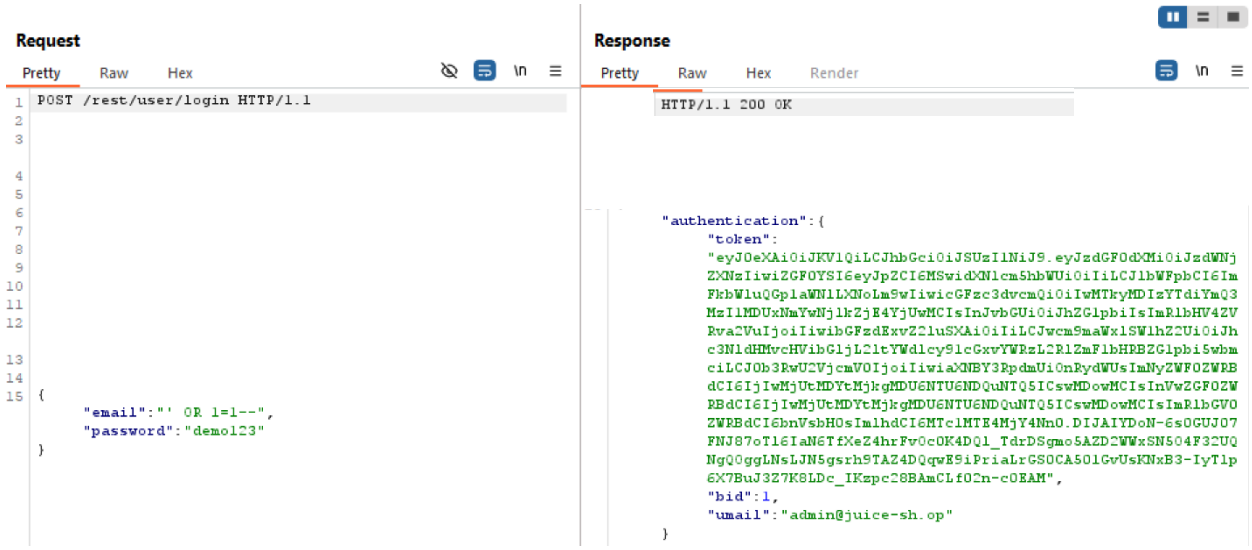


Figure 4 SQL Injection Burp Suite Request and Response

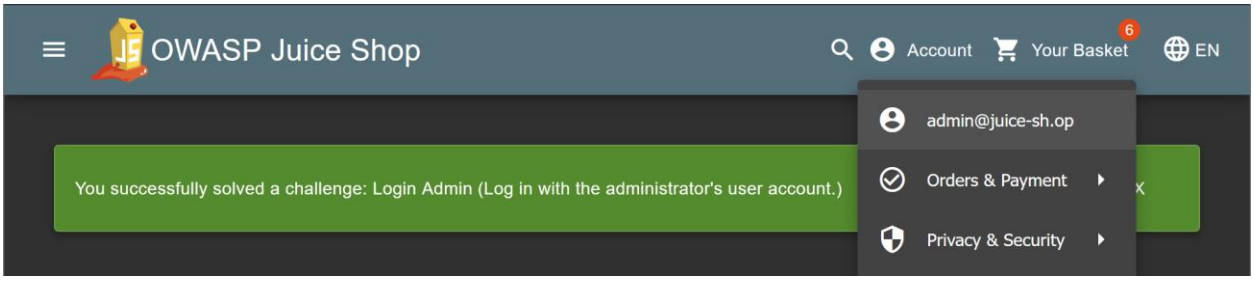


Figure 5 SQL Injection Successful and Admin Account Access

5. KEY INSIGHTS AND TECHNICAL LEARNINGS

5.1 OBSERVATIONS FROM THE ASSESSMENT

The assessment revealed critical vulnerabilities in both the frontend and backend components of the Juice Shop. Angular's client-side sanitization was bypassed using alternate tag formats, proving client-only defenses are not enough.

Backend issues like SQL injection showed poor input validation and unsafe query handling, allowing full authentication bypass in some cases.

Hidden files, open CORS policies, and missing security headers indicated weak deployment and insecure CI/CD practices.

These findings expose a gap between assumed frontend security and actual backend enforcement.

5.2 PRACTICAL RELEVANCE OF FINDINGS

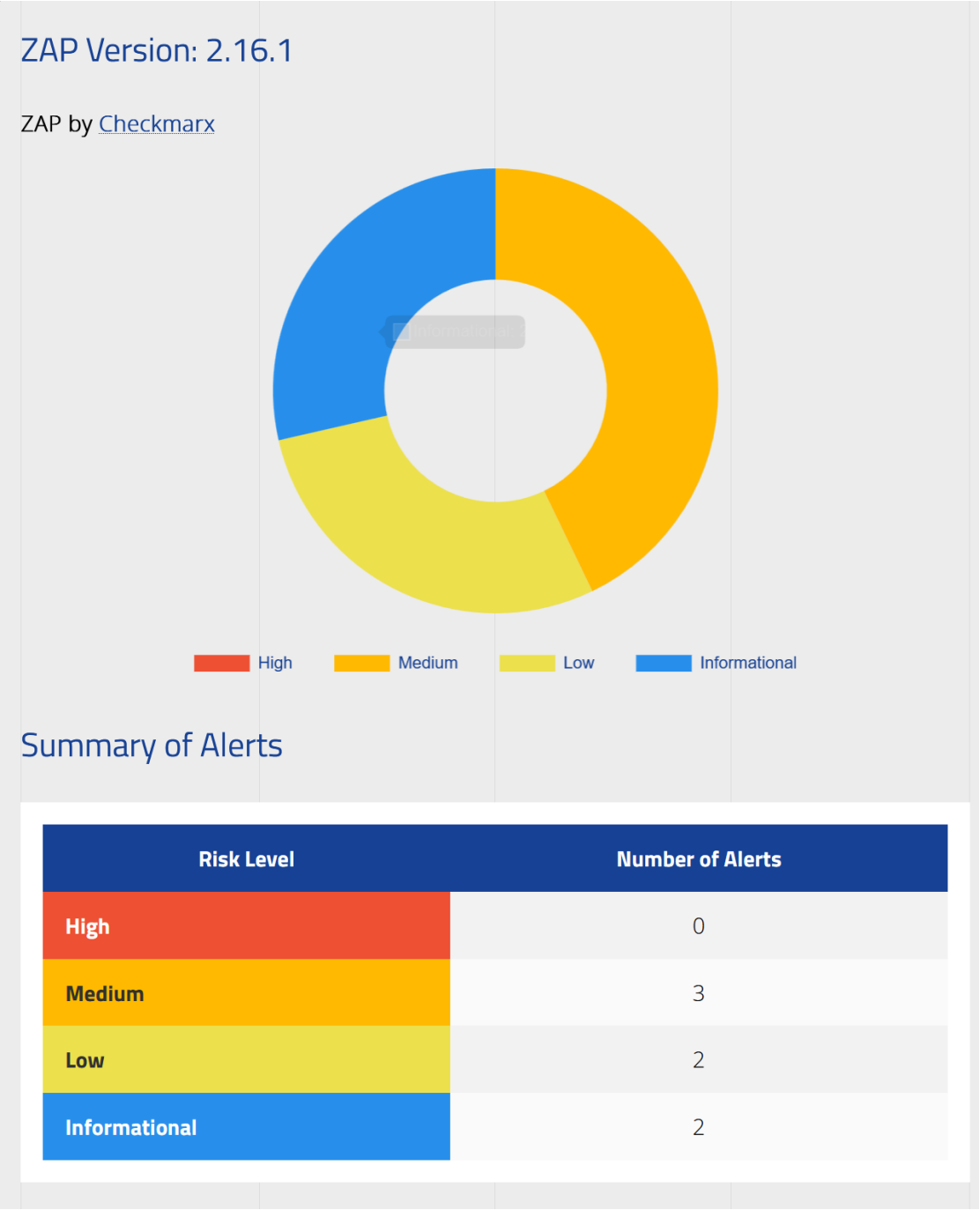
SQL injection remains a high-impact threat because directly inserting user input into database queries is still common. This reinforces the need for prepared statements and strict input validation. Reflected XSS attacks exposed the limits of Angular's sanitization mechanisms, making it clear that context-aware encoding and strong content security policies are essential.

Misconfigured CORS and exposed hidden files may not look dangerous individually, but they significantly increase the attack surface and can be exploited as attack vectors in chained attacks, especially when token-based authentication is involved.

Overall, the findings stress the importance of secure development practices and continuous dynamic testing, even for modern frameworks.

6. APPENDIX

6.1 OWASP ZAP REPORT



Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	3
Low	2
Informational	2

Figure 6 OWASP ZAP Alert Summary

Alerts

Name	Risk Level	Number of Instances
Content Security Policy (CSP) Header Not Set	Medium	57
Cross-Domain Misconfiguration	Medium	72
Hidden File Found	Medium	4
Cross-Domain JavaScript Source File Inclusion	Low	96
Timestamp Disclosure - Unix	Low	155
Information Disclosure - Suspicious Comments	Informational	2
Modern Web Application	Informational	49

Figure 7 Alert Categories, Severity and its Instances in Juice Shop

Sites

http://localhost:3000

HTTP Response Code	Number of Responses
403 Forbidden	16
404 Not Found	4
200 OK	586

Figure 8 OWASP Juice Shop Site Response Details