Security Assessment Report

Task: FUTURE_CS_01 – Web Application Security Testing
Intern: Raviteja Vadla
Internship: Cyber Security Internship — Future Interns
Date: [11 Sep 2025]

Target Application: OWASP Juice Shop (Local Docker)
Target URL (local): http://192.168.0.102:3000

Tools Used: OWASP ZAP (v2.15.0), Burp Suite Community, Docker Desktop, Kali Linux (VMware)

Deliverables:

- report.pdf
- zap_scan.html
- screenshots/
- owasp_top10_checklist.md
  (Repo: FUTURE_CS_01)

Contact / Repo:
GitHub: https://github.com/raviteja-vadla/FUTURE_CS_01
LinkedIn: https://www.linkedin.com/in/raviteja-cybersecurity

Executive Summary

This security assessment was conducted as part of my Cyber Security Internship (Future Interns) with the objective of performing a vulnerability assessment and penetration test of the OWASP Juice Shop application.

The engagement simulated a real-world web application test where the goal was to identify security flaws, document them, and provide recommendations for mitigation.

Tools such as OWASP ZAP and Burp Suite were used to identify common vulnerabilities including SQL Injection and Cross-Site Scripting (XSS). The findings confirm that Juice Shop, being intentionally vulnerable, allowed multiple successful exploitations that demonstrate the risk of insecure coding practices.

The results of this assessment, along with screenshots and technical evidence, have been documented in this report and mapped to the OWASP Top 10 security categories.

Scope & Objectives

In Scope:

- Target Application: OWASP Juice Shop (local instance running in Docker on Windows host)
- Target URL: http://192.168.0.102:3000
- Assessment conducted from Kali Linux VM using penetration testing tools.

Objectives:

1. Identify and document vulnerabilities within the Juice Shop web application.
2. Validate findings through automated scanning (OWASP ZAP) and manual testing (Burp Suite).
3. Map vulnerabilities to the OWASP Top 10 security risks.
4. Provide remediation steps for each finding.
5. Deliver a professional assessment report suitable for client presentation.

Methodology

The security assessment followed a structured approach combining automated scanning with manual testing techniques. The methodology was aligned with industry best practices for web application security assessments.

1. Environment Setup

- OWASP Juice Shop deployed locally in Docker on Windows 11 host.
- Kali Linux VM used as the attacker machine.
- Networking configured in Bridged mode to allow cross-host communication.

2. Reconnaissance & Scanning

- OWASP ZAP used to spider the application and perform active vulnerability scanning.
- Automated tests focused on injection flaws, cross-site scripting, and common misconfigurations.

3. Manual Testing

- Burp Suite Community Edition used to intercept and manipulate requests.
- Manual SQL Injection payloads tested against login functionality.
- Manual Cross-Site Scripting (XSS) payloads tested against input forms (e.g., Contact Us).

4. Analysis & Verification

- Vulnerability alerts from ZAP were verified and validated manually.
- Screenshots and request/response samples were collected as proof-of-concept.

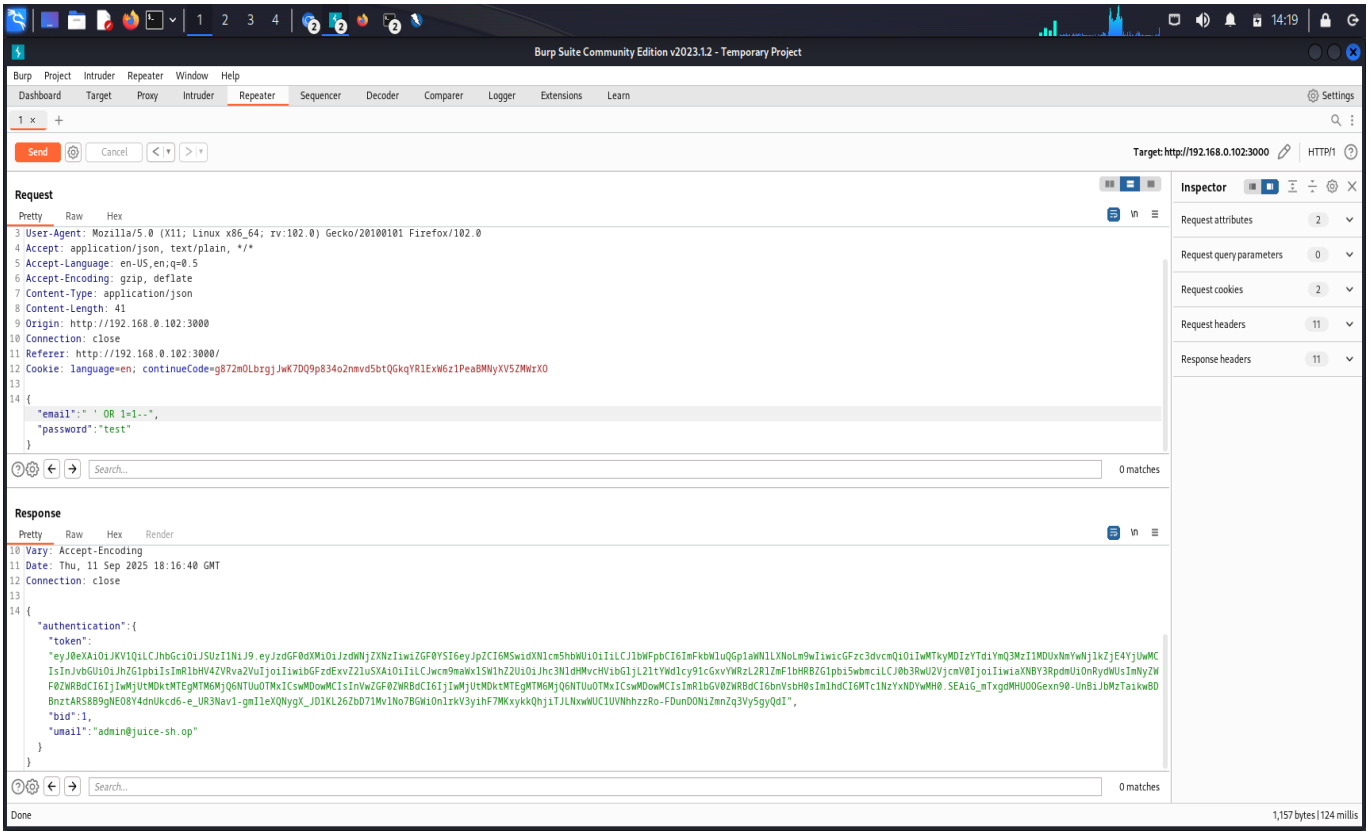
5. Reporting

- Findings documented with severity rating, impact analysis, and remediation steps.
- Mapped to OWASP Top 10 categories for standardized classification.

Findings & Vulnerabilities

1. SQL Injection – Login Bypass

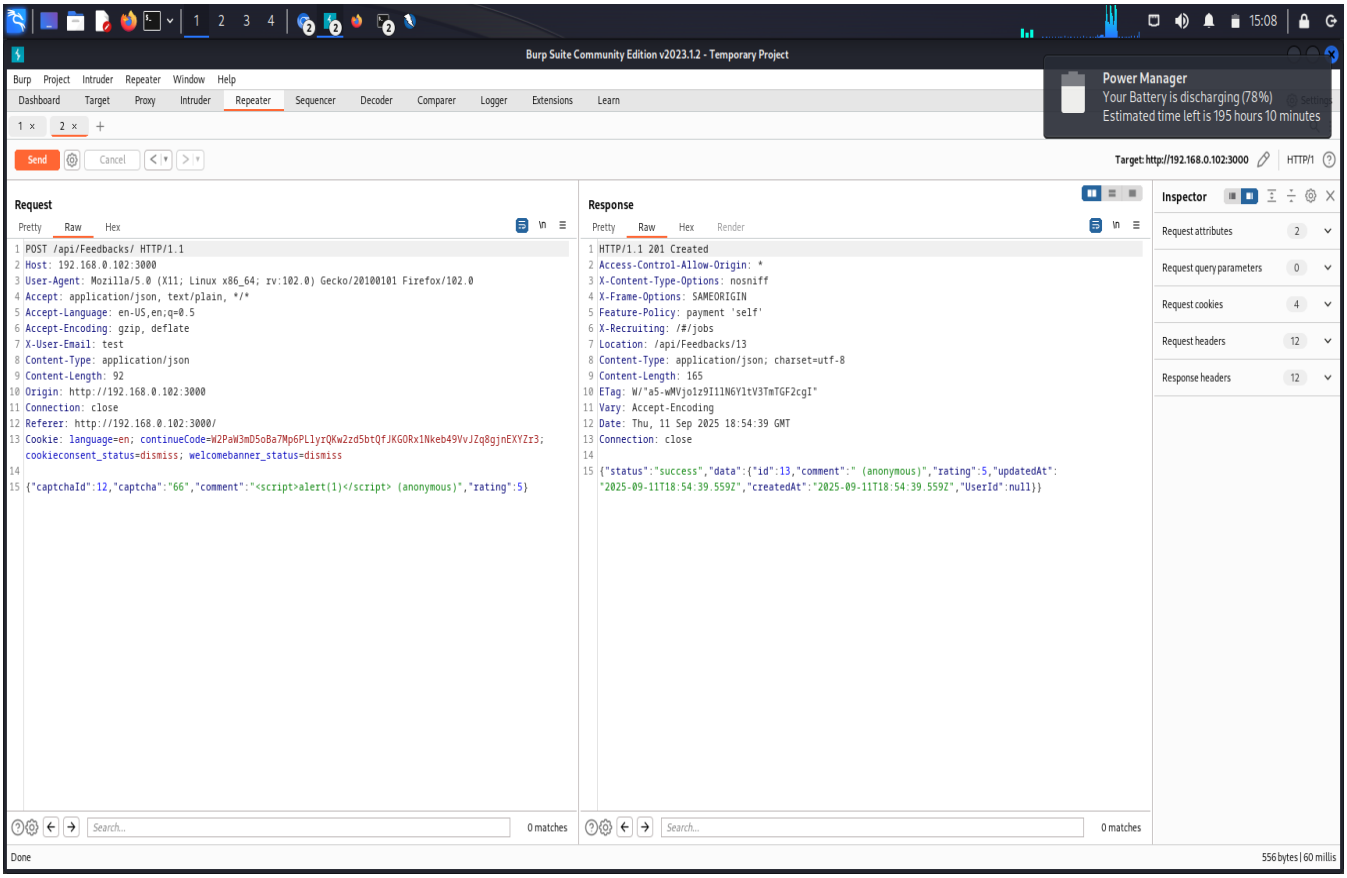
- **Severity:** High 🔴
- **Affected Endpoint:** /rest/user/login
- **Description:** The login functionality was found to be vulnerable to SQL Injection. By injecting ' OR 1=1-- into the email parameter, authentication could be bypassed.
- **Impact:** Unauthorized access to user accounts, full compromise of application authentication.
- **Proof of Concept:**
    - Payload: ' OR 1=1--
    - Response: Authentication success with token.



- **Recommendation:** Use parameterized queries (prepared statements), sanitize inputs, and implement server-side validation.

2. Stored Cross-Site Scripting (XSS) – Contact Us Form

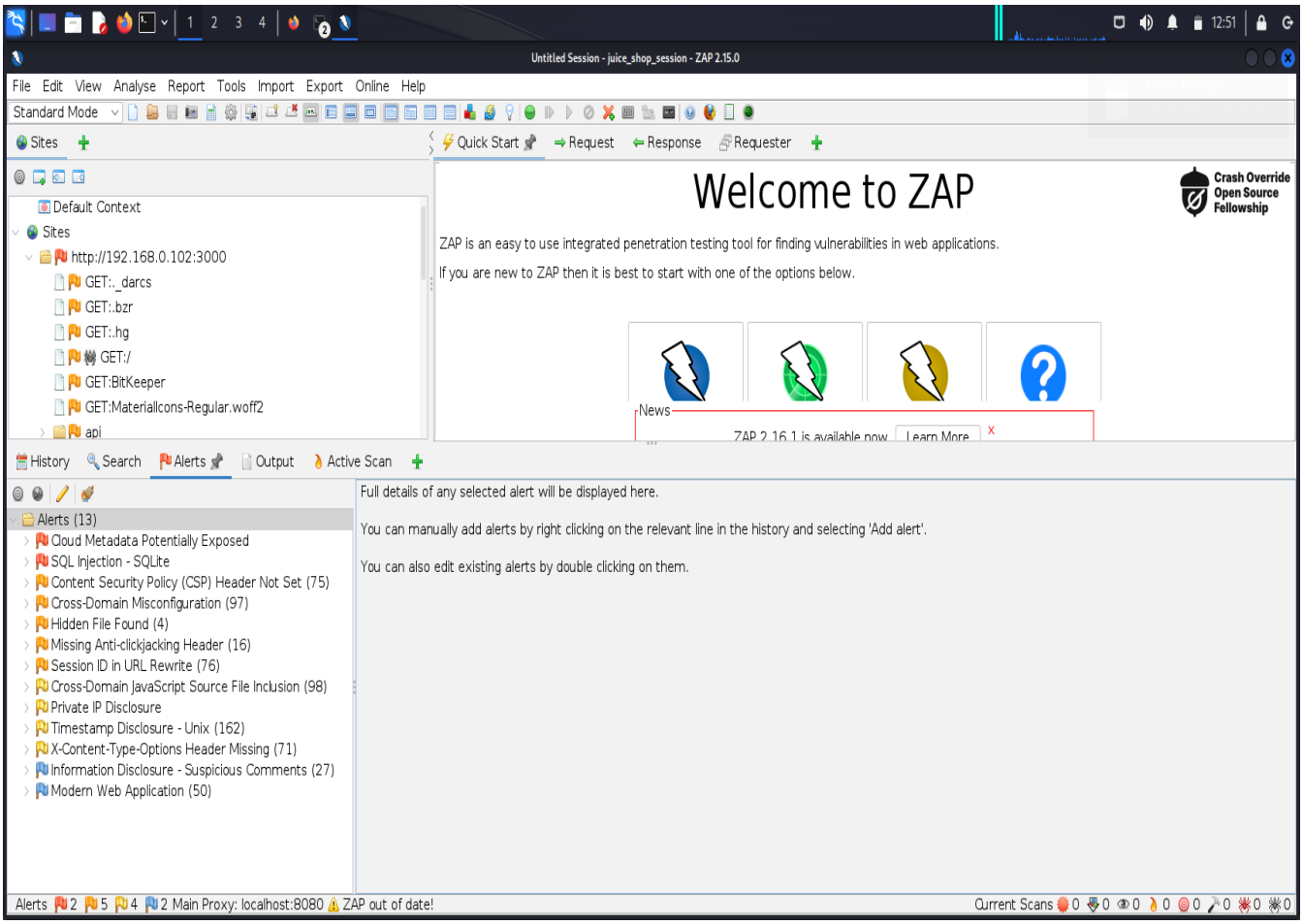
- **Severity:** High 🔴
- **Affected Endpoint:** /api/Feedbacks (Contact Us form)
- **Description:** Application accepted JavaScript payloads in feedback comments without proper sanitization. Payloads such as <script>alert(1)</script> or <img src=x onerror=alert(1)> were stored successfully.
- **Impact:** If executed in the browser, this vulnerability could allow session hijacking, data theft, or UI defacement.
- **Proof of Concept:**
    - Payload submitted: <script>alert(1)</script>
    - Response: 201 Created indicating payload stored.



- **Recommendation:** Implement server-side input validation and proper output encoding.

3. Automated Scan Results – OWASP ZAP

- **Severity:** Mixed (High, Medium, Low)
- **Tool:** OWASP ZAP 2.15.0
- **Description:** Automated scan revealed multiple vulnerabilities across OWASP Top 10 categories including injection flaws, cross-site scripting, and security misconfigurations.
- **Proof of Concept:**



  - ZAP HTML Report exported: tool_logs/zap_scan.html
- **Recommendation:** Review full scan report and address identified issues in priority order.

OWASP Top 10 Mapping

The identified vulnerabilities were mapped against the OWASP Top 10 2021 security risks:

OWASP Top 10 Category	Vulnerability Found?	Notes
A01: Broken Access Control	❌	Not tested in this task
A02: Cryptographic Failures	❌	Not tested
A03: Injection (SQL Injection)	✅	Login bypass confirmed
A04: Insecure Design	⚠️	Poor validation of inputs
A05: Security Misconfiguration	✅	Findings in ZAP scan
A06: Vulnerable & Outdated Components	❌	Not tested
A07: Identification & Authentication Failures	✅	Weak login handling (via SQLi)
A08: Software & Data Integrity Failures	❌	Not tested
A09: Security Logging & Monitoring Failures	⚠️	No logs visible during testing
A10: Server-Side Request Forgery (SSRF)	❌	Not tested

Conclusion

The assessment of the OWASP Juice Shop application demonstrated multiple critical vulnerabilities, including SQL Injection and Stored Cross-Site Scripting (XSS). These findings highlight the risks of insecure coding practices and emphasize the importance of secure development, input validation, and regular penetration testing.

Although Juice Shop is intentionally vulnerable for educational purposes, the techniques used here are directly applicable to real-world applications. If these flaws existed in production, they could lead to unauthorized access, data breaches, and compromise of user trust.

General Recommendations

1. **Input Validation & Sanitization**
    - Implement strict server-side validation of all user input.
    - Encode output before rendering in the browser.
2. **Use of Secure Coding Practices**
    - Parameterized queries or ORM frameworks to prevent SQL Injection.
    - Escape dangerous characters (<, >, ", ', etc.) in all user inputs.
3. **Implement Security Controls**
    - Multi-factor authentication (MFA).
    - Strong password policies.
    - Web Application Firewall (WAF) to detect and block malicious payloads.
4. **Regular Security Testing**
    - Conduct automated and manual penetration tests at regular intervals.
    - Keep frameworks and libraries updated to patch known vulnerabilities.
5. **Logging & Monitoring**
    - Maintain detailed server-side logs for authentication attempts and input validation errors.
    - Enable real-time monitoring for suspicious activities.