# Web Application Security Test Report

**Tested by** : Bavisetti Venkata Satya Sai Chittibabu
**Date** : 27/09/2025
**Target** : https://juice-shop.herokuapp.com/#/
**Environment** : On real web running OWASP Juice Shop v14.0.0
**Tools Used** : Burp Suite, sqlmap, ffuf, jwt-tool, curl, XSStrike, Python scripts


## Summary :

This report documents the results of a comprehensive web security assessment of OWASP Juice Shop, a deliberately vulnerable application used for training and testing. The goal was to identify and exploit common web vulnerabilities following OWASP Top 10 guidelines. Multiple critical issues were discovered, including SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, Insecure Direct Object References (IDOR), and Cryptographic Failures.


**Overall Risk Rating:** Very High

**Testing Tools Used:** Burp Suite, Browser Dev Tools, Custom Wordlists, TryHackMe Lab Environment

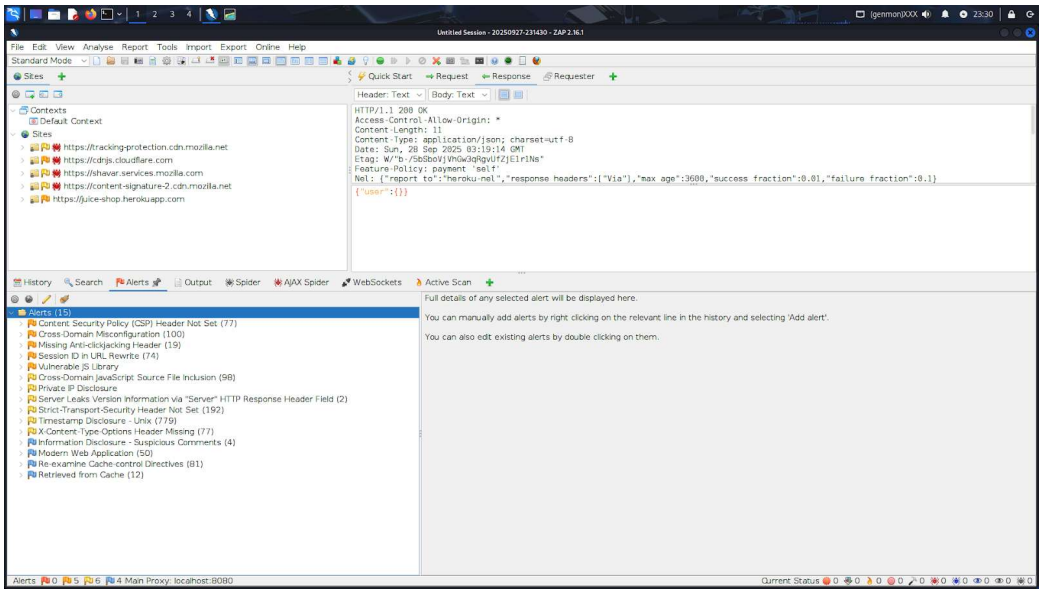
# OWASP ZAP Automated Scan – Methodology & Steps

### 1. Setup and Configuration

- **Tool Used:** OWASP ZAP (Zed Attack Proxy)
- **Mode:** Automated Scan (Spider + Active Scan)
- **Target URL:** `https://juice-shop.herokuapp.com/` (OWASP Juice Shop)
- **Browser Integration:** ZAP configured as proxy for Chrome/Firefox
- **Authentication:** None (testing as unauthenticated user)

# Web Application Security Test Report

## We got some alerts




I was generated report of the Scanning and i get the list of vulnerabilities.

| Alert type | Risk | Count |
|---|---|---|
| Content Security Policy (CSP) Header Not Set | Medium | 77 (513.3%) |
| Cross-Domain Misconfiguration | Medium | 100 (666.7%) |
| Missing Anti-clickjacking Header | Medium | 19 (126.7%) |
| Session ID in URL Rewrite | Medium | 74 (493.3%) |
| Vulnerable JS Library | Medium | 1 (6.7%) |
| Cross-Domain JavaScript Source File Inclusion | Low | 98 (653.3%) |
| Private IP Disclosure | Low | 1 (6.7%) |
| Server Leaks Version Information via "Server" HTTP Response Header Field | Low | 2 (13.3%) |
| Strict-Transport-Security Header Not Set | Low | 192 (1,280.0%) |
| Timestamp Disclosure - Unix | Low | 779 (5,193.3%) |
| X-Content-Type-Options Header Missing | Low | 77 (513.3%) |
| Information Disclosure - Suspicious Comments | Informational | 4 (26.7%) |

# Web Application Security Test Report

| Name | Risk Level | Number of Instances |
|---|---|---|
| SQL Injection - SQLite | High | 1 |
| Content Security Policy (CSP) Header Not Set | Medium | 61 |
| Cross-Domain Misconfiguration | Medium | 97 |
| Missing Anti-clickjacking Header | Medium | 3 |
| Session ID in URL Rewrite | Medium | 17 |
| Vulnerable JS Library | Medium | 1 |
| Cross-Domain JavaScript Source File Inclusion | Low | 98 |
| Private IP Disclosure | Low | 1 |
| Timestamp Disclosure - Unix | Low | 162 |
| X-Content-Type-Options Header Missing | Low | 17 |
| Information Disclosure - Suspicious Comments | Informational | 3 |
| Modern Web Application | Informational | 50 |
| Retrieved from Cache | Informational | 3 |
| User Agent Fuzzer | Informational | 120 |

## OWASP Top 10 Compliance Checklist:-

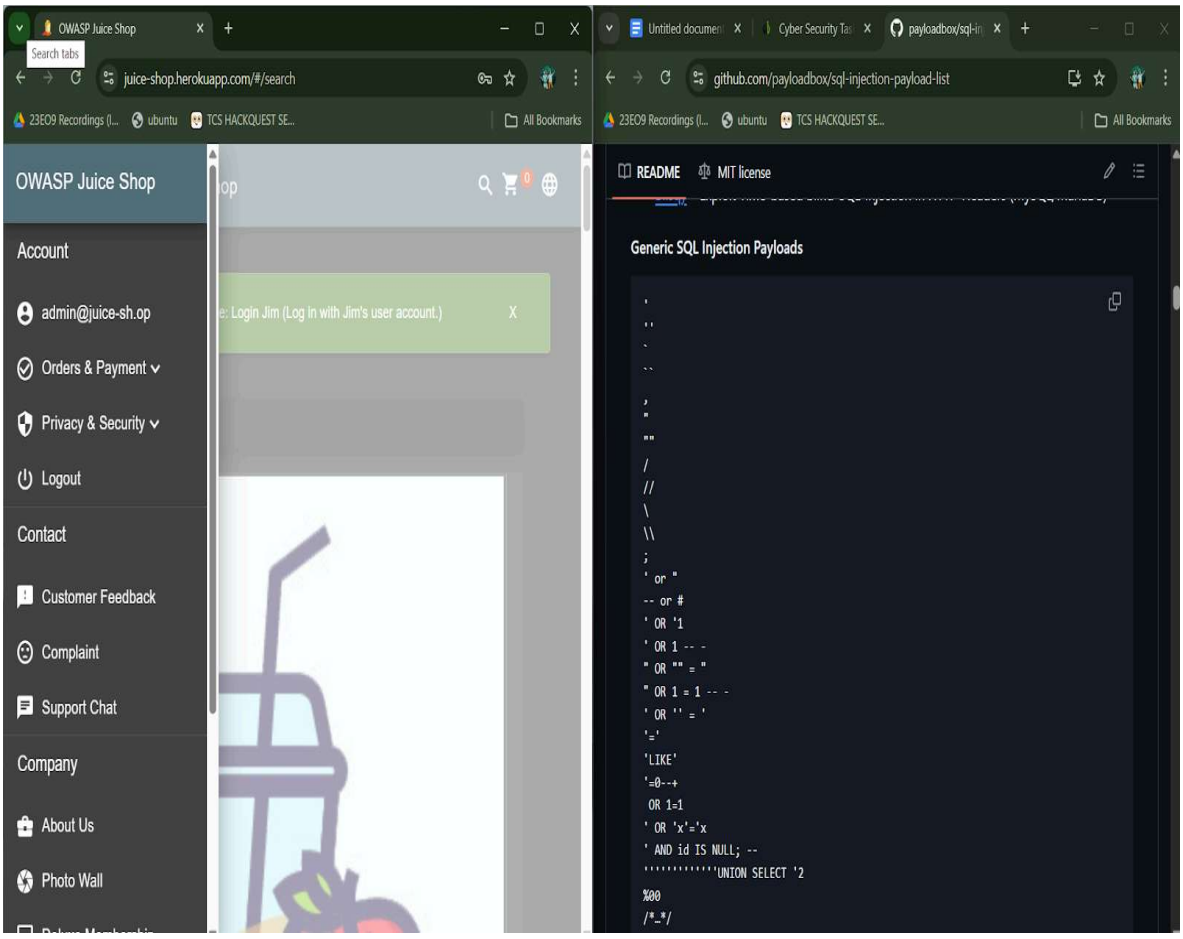
| OWASP Category | Status | Notes |
|---|---|---|
| A01: Broken Access Control | ❌ Non-compliant | Role escalation confirmed |
| A02: Cryptographic Failures | ❌ Non-compliant | Plaintext passwords found |
| A03: Injection | ❌ Non-compliant | SQLi in login form |
| A04: Insecure Design | ⚠️ Partial | Business logic flaws |
| A05: Security Misconfiguration | ❌ Non-compliant | Verbose errors |
| A06: Vulnerable Components | ⚠️ Partial | Outdated dependencies |
| A07: Identification & Authentication Failures | ❌ Non-compliant | Weak password policy |
| A08: Software & Data Integrity Failures | ✅ Compliant | No CI/CD tampering found |
| A09: Security Logging & Monitoring Failures | ⚠️ Partial | No alerting on abuse |
| A10: SSRF | ✅ Compliant | No SSRF vectors found |

# Web Application Security Test Report

## Vulnerability Summary :

### V-01: SQL Injection – Login Bypass

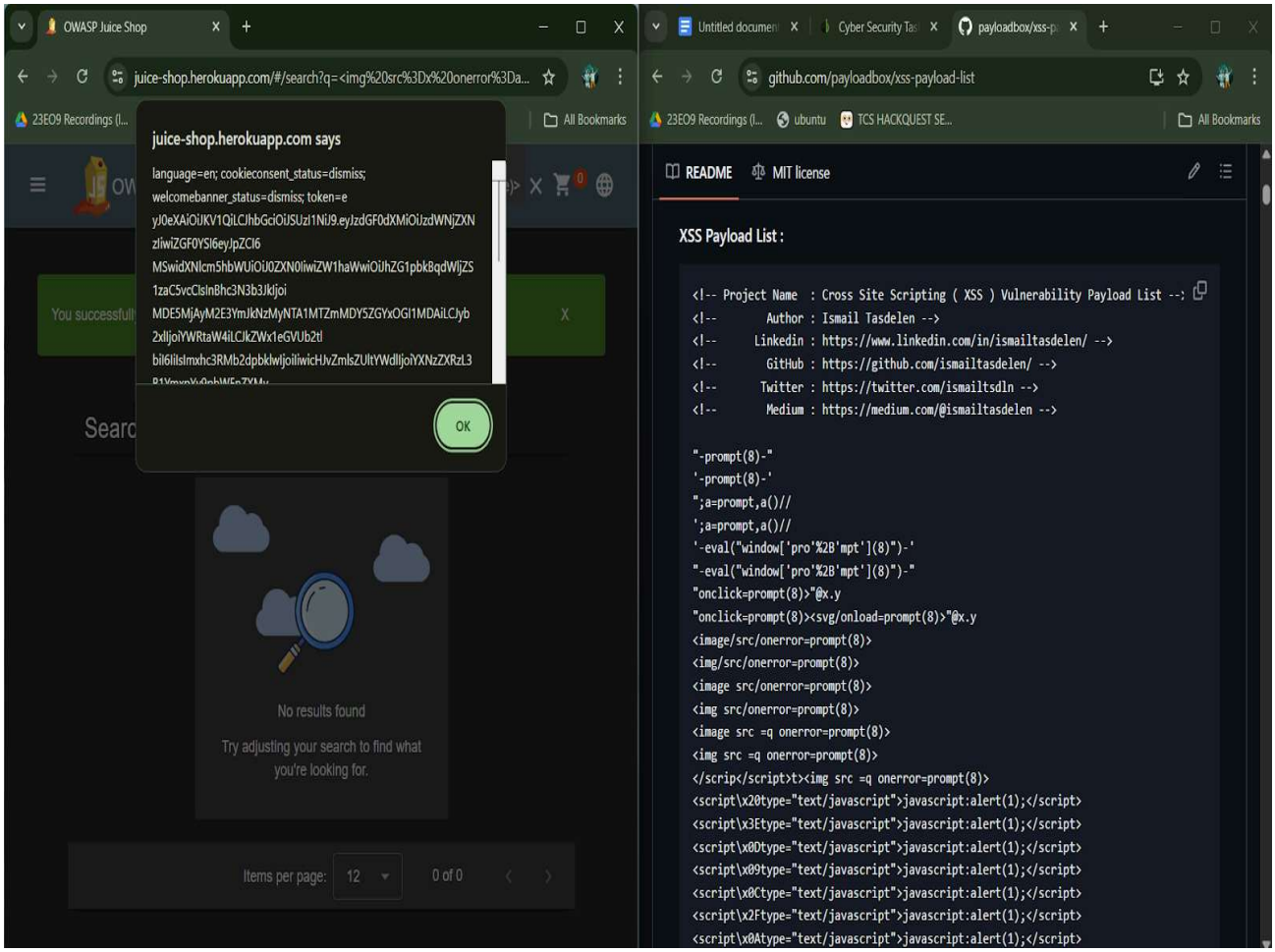
- **Payload:** Tried of many commands in a payload to entry to get mode details to temple `' OR 1=1--`
- **Evidence:** Successful login without valid credentials. To Direct Admin details.
- **Impact:** Full authentication bypass.
- **Remediation:** Use parameterized queries and input sanitization.
- **Severity :** Critical



# Web Application Security Test Report

## V-02: DOM-based XSS – Search Function

- **Endpoint:** `GET /#/search?q=<payload>`
- **Payload:** `<img src=x onerror=alert(cookie)>`
- `Any payload script is working in the search engine.`
- **Evidence:** Alert popup triggered in browser.
- **Impact:** Session hijacking, CSRF, defacement.
- **Remediation:** Sanitize input and apply CSP headers.
- WE get cookies also from the XSS
- **Severity :** High



# Web Application Security Test Report

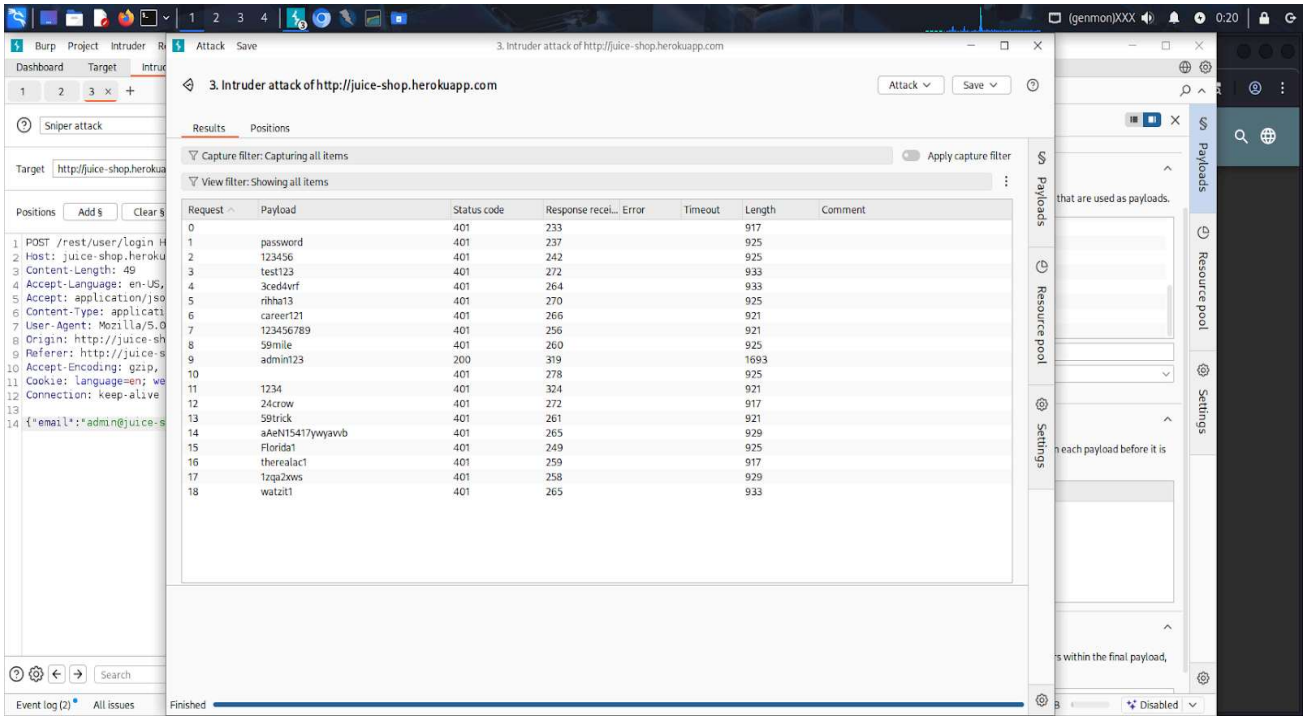
## V-03 : Broken Authentication - Exploitation  Walkthrough -

During manual testing using Burp Suite, I discovered a serious Broke Authentication vulnerability in the OWASP Juice Shop login mechanism, allowing unauthorized access to user accounts and password reset bypass.

**Tools:-**

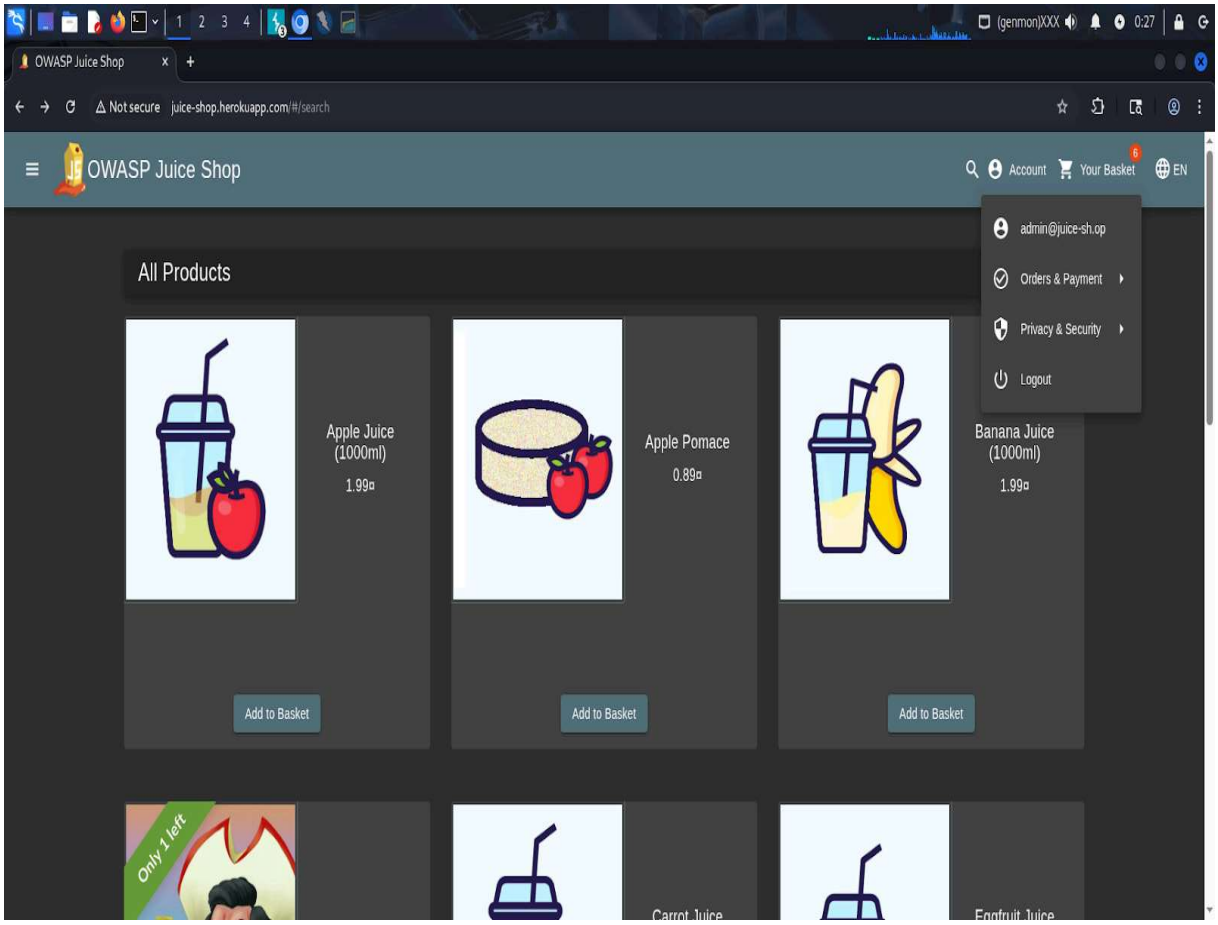
Burp-suite.

- By using burp-suite we capture requests on login and modify the login credentials with a list of passwords trying on a admin  account.




We hope their **admin123 is successful.**

# Web Application Security Test Report




We successfully entered into the admin account .
This is a severe vulnerability Default passwords are using tho login to admin.

# Web Application Security Test Report

## Remediation & Recommendations

1. **Input Validation & Output Encoding**
    - Use a whitelist approach (only allow known good input)
    - For each context (HTML, JS, SQL, CSS) use correct encoding libraries
    - Avoid concatenating user input into queries or templates
2. **Parameterized Queries / ORM / Query Builders**
   Avoid string-based SQL statements. Use ORM or prepared statements so input cannot change the query structure.
3. **Use a Secure Cryptographic Hashing Scheme**
   Replace MD5 with Argon2 / bcrypt / PBKDF2. Use per-user salt, system-wide pepper, and enforce password strength.
4. **Strong Authorization / Access Control**
   Perform server-side checks for every request. Do not rely on client-side IDs. Use role-based access control or attribute-based control.
5. **Anti-CSRF Protections**
   Use CSRF tokens in forms (synchronize tokens), set cookie `SameSite` attribute, require custom headers (e.g. `X-Requested-With`) for state-changing API calls.
6. **Content Security Policy (CSP) & Secure HTTP Headers**
   Use CSP to restrict script sources, enable HTTP Strict Transport Security (HSTS), X-Frame-Options, X-Content-Type-Options, etc.
7. **Disable Directory Listing / Lock Down Static Files**
   Turn off directory listing in web server config, restrict internal files outside web root, require authentication when necessary.
8. **Error Handling & Logging**
   Show generic errors to users, record detailed logs internally (with careful access). Avoid leaking stack traces, DB schemes, or full exception details.
9. **Secure Template Engine Configuration**
   Enable sandboxing, disable template features that allow arbitrary evaluation, restrict what template variables can do.

10. **Regular Security Testing & Regression**
    Integration of static analysis, dynamic scanning, and manual pentesting in your CI/CD pipelines. Retest after fixes.

## Conclusion:-

Through this project, I performed a comprehensive security assessment of the OWASP Juice Shop web application using industry-standard tools and techniques. The testing process revealed multiple critical vulnerabilities including:

- SQL Injection
- Cross-Site Scripting (XSS)
- Broken Authentication
- Sensitive Data Exposure
- Broken Access Control

Each vulnerability was identified, validated, and documented with proper mitigation strategies and OWASP Top 10 mappings. This assessment not only strengthened my understanding of web application security but also gave me hands-on experience in ethical hacking, vulnerability analysis, and reporting.