Web Application Security Assessment Report

Project: Web Application Security Testing
Internship: Cyber Security Internship – Future Interns
Intern Name: Harshvardhan Duboliya
Application Tested: OWASP Juice Shop
Assessment Type: Vulnerability Assessment & Manual Exploitation.

Executive Summary

The results of a web application security assessment on OWASP Juice Shop, a purposefully vulnerable web application used for security testing and education, are presented in this report.Using automated tools and manual testing methods in line with OWASP Top 10 standards, the assessment's goal was to find common web application vulnerabilities.

Numerous security flaws, including serious ones like Cross-Site Scripting (XSS) and SQL Injection leading to authentication bypass, as well as a number of configuration and information disclosure problems, were found during the assessment. If these flaws are found in a real-world application, attackers may be able to take over user accounts, run malicious scripts, and erode overall security.

Scope of Assessment

In Scope:

- Web application: OWASP Juice Shop (online version)
- Login functionality
- Search input functionality
- HTTP headers and configuration
- Client-side input handling

Out of Scope:

- Denial of Service (DoS) attacks
- Brute force attacks
- Source code review

Tools & Techniques Used

- **OWASP ZAP** – Automated vulnerability scanning
- **Burp Suite (Community Edition)** – Manual request inspection
- **Web Browser (Chrome)** – Manual testing
- **OWASP Top 10** – Vulnerability classification standard

Vulnerability Summary Table

ID	Vulnerability	Risk Level	Testing Method
V-01	SQL Injection (Login Bypass)	High	Manual
V-02	Cross-Site Scripting (XSS)	Medium	Manual
V-03	Content Security Policy Header Not Set	Low	Automated
V-04	Strict-Transport-Security Header Not Set	Low	Automated
V-05	Information Disclosure – Suspicious Comments	Informational	Automated

Detailed Vulnerability Findings

1) V-01: Bypassing Authentication through SQL Injection

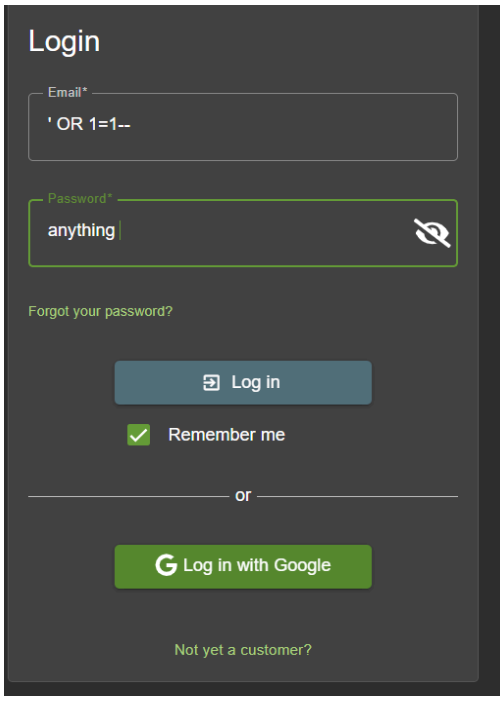
Risk Level: High A03-Injection OWASP Mapping

Description: By altering SQL queries, an attacker can get around authentication because the login feature is susceptible to SQL Injection.

Used Payload: **OR '1'='1--**

Impact: Unauthorized access and complete account takeover can result from an attacker using invalid credentials to log in as an administrator.

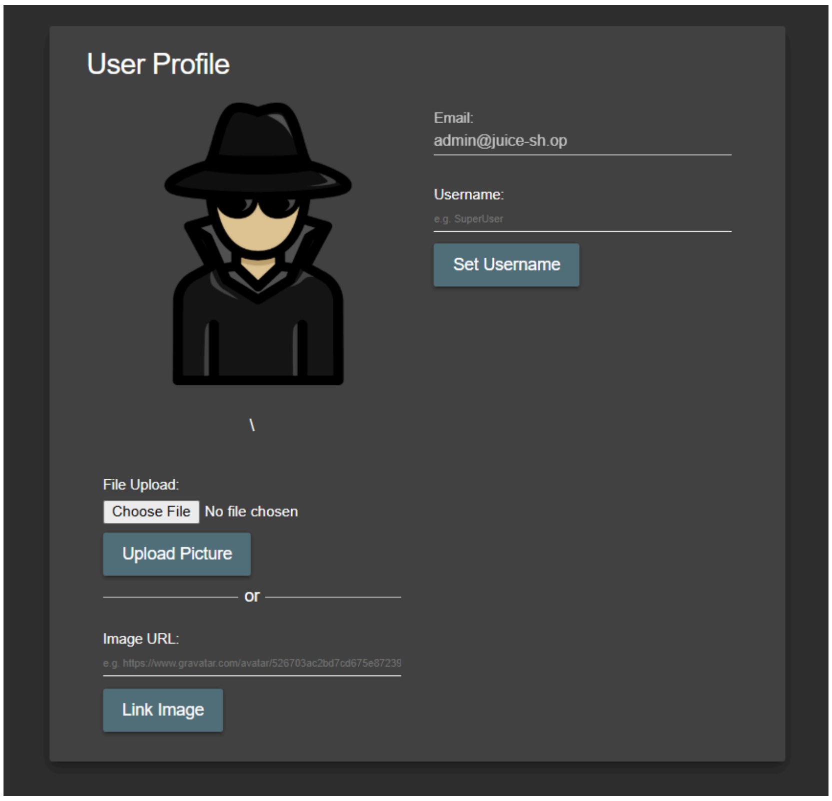
Evidence:



Login

Email*: ' OR 1=1--

Password*: anything



User Profile

Email: admin@juice-sh.op

Username: e.g. SuperUser

\

File Upload: Choose File No file chosen

Image URL: e.g. https://www.gravatar.com/avatar/526703ac2bd7cd675e87239

Corrective action:

- Make use of parameterized queries, or prepared statements.
- Put appropriate input validation into practice.
- Don't reveal database error behavior

2) V-02: XSS (Cross-Site Scripting)

Level of Risk: Medium

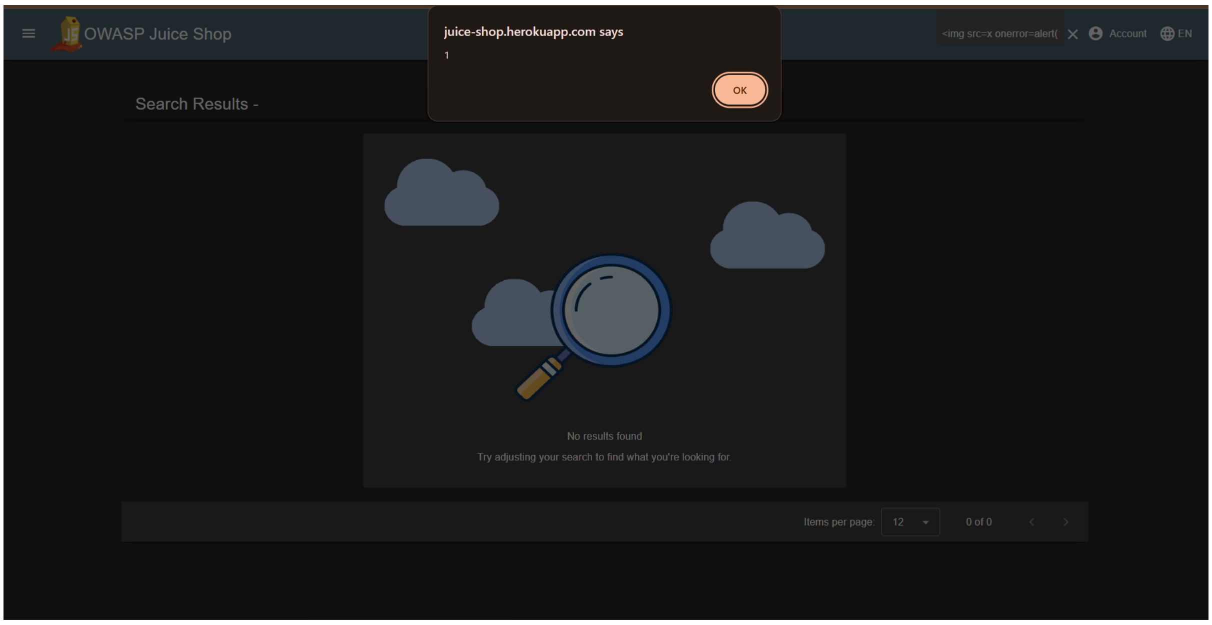
OWASP Mapping: Cross-Site Scripting, A07

Description: Malicious JavaScript code can be injected into the search function and run in the user's browser.

Used Payload: <img src=x onerror=alert(1)>

Impact: Scripts that steal cookies, carry out phishing scams, or alter page content can be executed by an attacker.

Proof:



juice-shop.herokuapp.com says

1

OK
OWASP Juice Shop
<img src=x onerror=alert(
Account
EN
Search Results -
No results found
Try adjusting your search to find what you're looking for.
Items per page: 12
0 of 0

Corrective action:

- Encrypt user input correctly before rendering
- Put the Content Security Policy (CSP) into practice.
- Verify and clean up every user input.

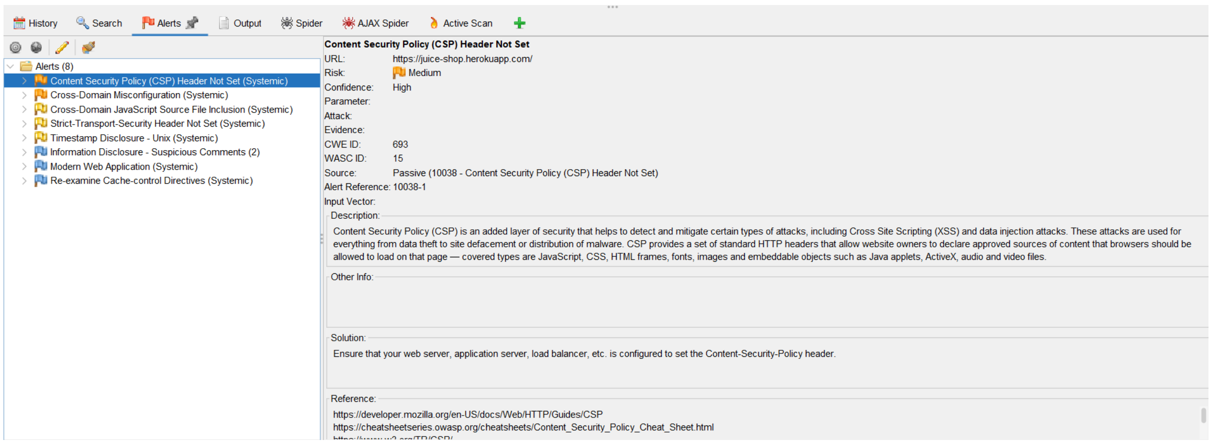
3) V-03: Content Security Policy (CSP) Header Not Set

Risk Level: Low
OWASP Mapping: A05 – Security Misconfiguration

Description:
The application does not define a Content Security Policy, increasing the risk of XSS attacks.

Impact:
Without CSP, browsers cannot restrict execution of unauthorized scripts.

Evidence:



Alerts (8)
- Content Security Policy (CSP) Header Not Set (Systemic)
- Cross-Domain Misconfiguration (Systemic)
- Cross-Domain JavaScript Source File Inclusion (Systemic)
- Strict-Transport-Security Header Not Set (Systemic)
- Timestamp Disclosure - Unix (Systemic)
- Information Disclosure - Suspicious Comments (2)
- Modern Web Application (Systemic)
- Re-examine Cache-control Directives (Systemic)

```
Content Security Policy (CSP) Header Not Set
URL:             https://juice-shop.herokuapp.com/
Risk:            Medium
Confidence:      High
Parameter:
Attack:
Evidence:
CWE ID:          693
WASC ID:         15
Source:          Passive (10038 - Content Security Policy (CSP) Header Not Set)
Alert Reference: 10038-1
Input Vector:
```

Description:
Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are JavaScript, CSS, HTML frames, fonts, images and embeddable objects such as Java applets, ActiveX, audio and video files.

Other Info:

Solution:
Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.

Reference:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

Remediation:

- Implement a strict Content Security Policy header

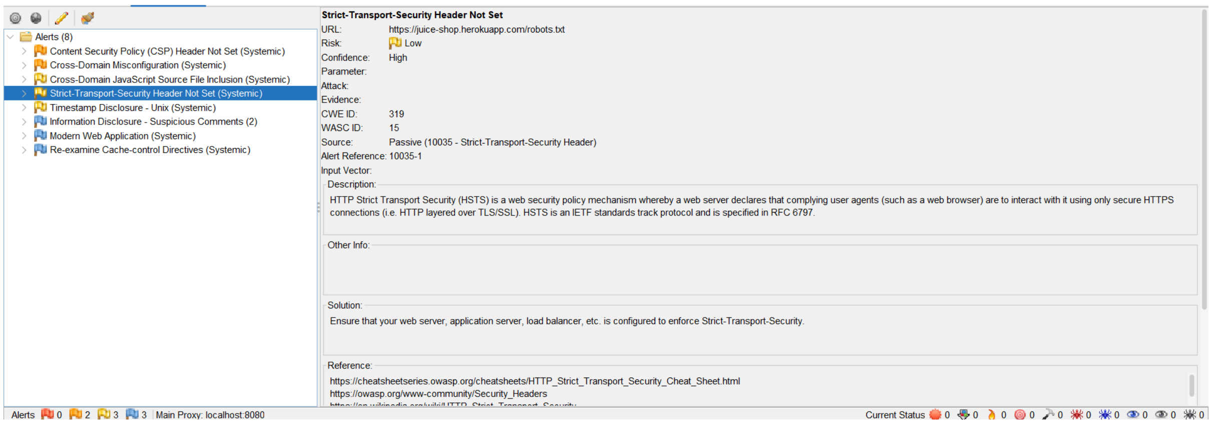
4) V-04: Strict-Transport-Security (HSTS) Header Not Set

Risk Level: Low
OWASP Mapping: A05 – Security Misconfiguration

Description:
The application does not enforce HTTPS through the HSTS header.

Impact:
Users may be vulnerable to downgrade and man-in-the-middle attacks.

Evidence:



Alerts (8)
- Content Security Policy (CSP) Header Not Set (Systemic)
- Cross-Domain Misconfiguration (Systemic)
- Cross-Domain JavaScript Source File Inclusion (Systemic)
- Strict-Transport-Security Header Not Set (Systemic)
- Timestamp Disclosure - Unix (Systemic)
- Information Disclosure - Suspicious Comments (2)
- Modern Web Application (Systemic)
- Re-examine Cache-control Directives (Systemic)

Strict-Transport-Security Header Not Set

URL:	https://juice-shop.herokuapp.com/robots.txt
Risk:	Low
Confidence:	High
Parameter:	
Attack:	
Evidence:	
CWE ID:	319
WASC ID:	15
Source:	Passive (10035 - Strict-Transport-Security Header)
Alert Reference:	10035-1
Input Vector:	

Description:
HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL). HSTS is an IETF standards track protocol and is specified in RFC 6797.

Other Info:

Solution:
Ensure that your web server, application server, load balancer, etc. is configured to enforce Strict-Transport-Security.

Reference:
https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html
https://owasp.org/www-community/Security_Headers

Alerts 🚩0 🚩2 🚩3 🚩3 Main Proxy: localhost:8080 Current Status 0 0 0 0 0 0 0 0 0 0

Remediation:

- Enable HSTS with an appropriate max-age value

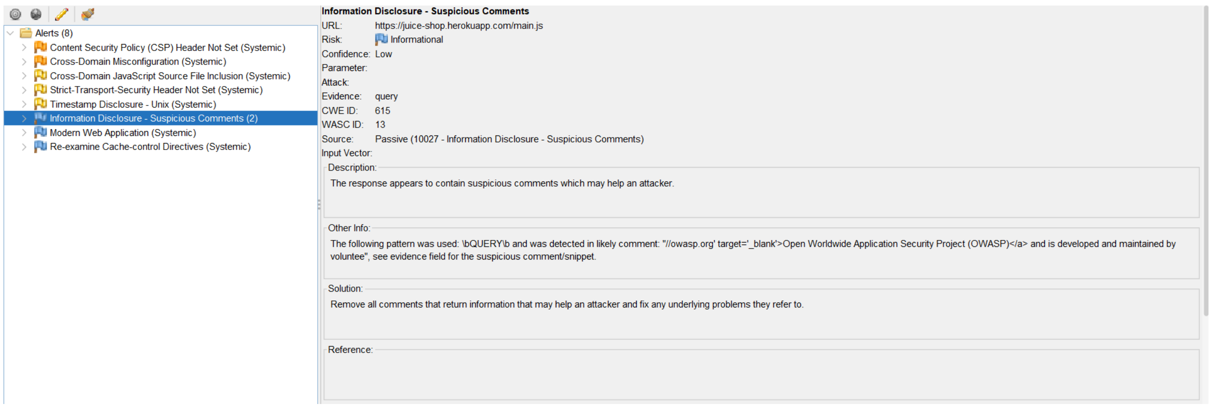
5) V-05: Information Disclosure – Suspicious Comments

Risk Level: Informational
OWASP Mapping: A05 – Security Misconfiguration

Description:
Developer comments were found within the application's client-side code.

Impact:
Such information may help attackers understand application logic.

Evidence:



Alerts (8)
- Content Security Policy (CSP) Header Not Set (Systemic)
- Cross-Domain Misconfiguration (Systemic)
- Cross-Domain JavaScript Source File Inclusion (Systemic)
- Strict-Transport-Security Header Not Set (Systemic)
- Timestamp Disclosure - Unix (Systemic)
- Information Disclosure - Suspicious Comments (2)
- Modern Web Application (Systemic)
- Re-examine Cache-control Directives (Systemic)

Information Disclosure - Suspicious Comments

URL:	https://juice-shop.herokuapp.com/main.js
Risk:	Informational
Confidence:	Low
Parameter:	
Attack:	
Evidence:	query
CWE ID:	615
WASC ID:	13
Source:	Passive (10027 - Information Disclosure - Suspicious Comments)
Input Vector:	

Description:
The response appears to contain suspicious comments which may help an attacker.

Other Info:
The following pattern was used: \bQUERY\b and was detected in likely comment: "//owasp.org' target='_blank'>Open Worldwide Application Security Project (OWASP)</a> and is developed and maintained by voluntee", see evidence field for the suspicious comment/snippet.

Solution:
Remove all comments that return information that may help an attacker and fix any underlying problems they refer to.

Reference:

Remediation:

- Remove unnecessary comments from production code

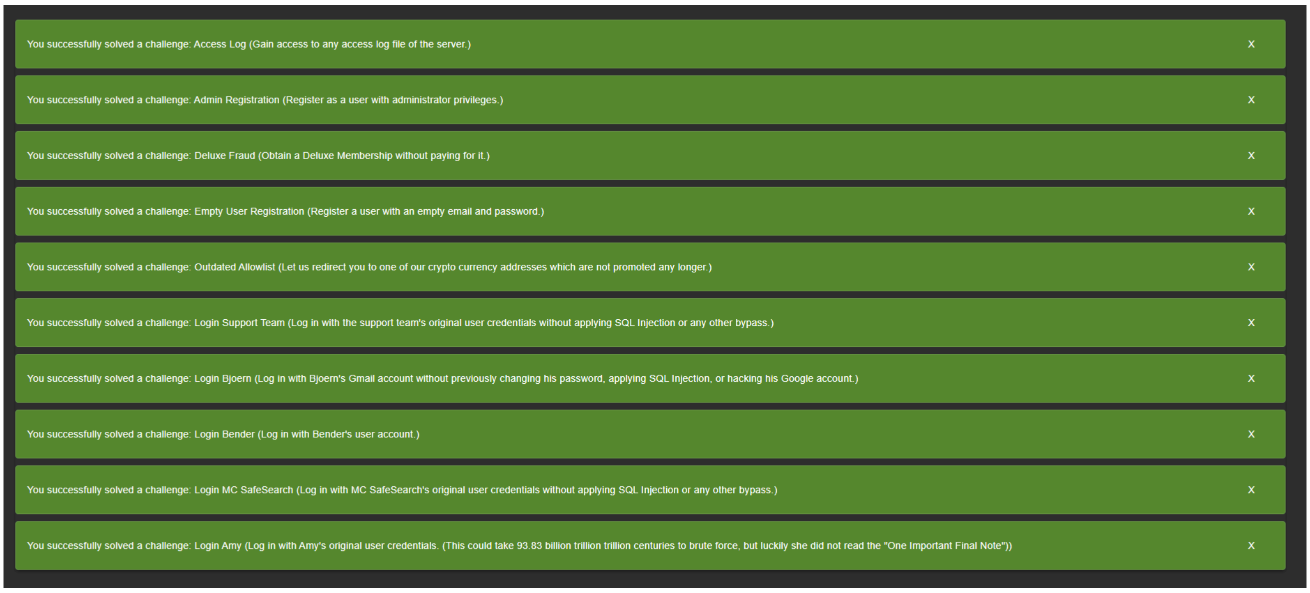
OWASP Top 10 Mapping Checklist

OWASP Category	Status
A02 – Cryptographic Failures	Fail
A03 – Injection	Done
A05 – Security Misconfiguration	Done
A07 – XSS	Done

Conclusion

The security assessment found several critical and moderate vulnerabilities that could seriously affect the confidentiality and integrity of a real-world application. The presence of SQL Injection and XSS shows how important secure coding practices, proper input validation, and strong security headers are. Fixing the identified issues will greatly improve the overall security of the application.

Result



You successfully solved a challenge: Access Log (Gain access to any access log file of the server.)

You successfully solved a challenge: Admin Registration (Register as a user with administrator privileges.)

You successfully solved a challenge: Deluxe Fraud (Obtain a Deluxe Membership without paying for it.)

You successfully solved a challenge: Empty User Registration (Register a user with an empty email and password.)

You successfully solved a challenge: Outdated Allowlist (Let us redirect you to one of our crypto currency addresses which are not promoted any longer.)

You successfully solved a challenge: Login Support Team (Log in with the support team's original user credentials without applying SQL Injection or any other bypass.)

You successfully solved a challenge: Login Bjoern (Log in with Bjoern's Gmail account without previously changing his password, applying SQL Injection, or hacking his Google account.)

You successfully solved a challenge: Login Bender (Log in with Bender's user account.)

You successfully solved a challenge: Login MC SafeSearch (Log in with MC SafeSearch's original user credentials without applying SQL Injection or any other bypass.)

You successfully solved a challenge: Login Amy (Log in with Amy's original user credentials. (This could take 93.83 billion trillion trillion centuries to brute force, but luckily she did not read the "One Important Final Note"))

Disclaimer

This assessment was conducted **strictly for educational purposes** as part of a cybersecurity internship using an intentionally vulnerable application.