Week 1 Report: Security Assessment of Web Application

Intern Name: Fatima Muhammad

Week: 1
Internship: Developers Hub Corporation – Cybersecurity Intern
Application Tested: OWASP Juice Shop (http://localhost:3000)
Environment: Kali Linux VM (Dockerized Application)

1. Objective

The objective of Week 1 was to perform a basic security assessment on a vulnerable web application. The goal was to identify common vulnerabilities such as Cross-Site Scripting (XSS), SQL Injection (SQLi), and security misconfigurations using manual testing and browser tools.

2. Application Setup

- Deployed **OWASP Juice Shop** using Docker:

```
sudo docker run --rm -d -p 3000:3000 bkimminich/juice-shop
```

- Explored features: login, registration, search, and product pages.

3. Vulnerability Assessment

A. Cross-Site Scripting (XSS)

- **Tested Component**: Search Bar
- **Payload Used**:

```
"><img src=x onerror=alert('XSS')>
```

- **Result**: Alert popup appeared showing “XSS”.
- **Risk**: Demonstrates reflected XSS. Could allow session theft or phishing attacks.
- **Severity**: High

B. SQL Injection

- **Tested Component**: Login Page

- **Payload Used**:
  - Email: `' OR 1=1--`
  - Password: `anything`
- **Result**: Logged in without valid credentials.
- **Risk**: Demonstrates bypass of authentication.
- **Severity**: Critical

C. Security Misconfigurations

- **Tool Used**: Browser DevTools → Network tab → Response Headers
- **Findings**:
  - Present:
    - `X-Content-Type-Options: nosniff`
    - `X-Frame-Options: SAMEORIGIN`
  - Missing:
    - `Content-Security-Policy`
    - `Strict-Transport-Security`
- **Risk**: Missing headers weaken browser-based protections and open room for exploits like XSS.

4. Summary of Findings

Vulnerability Type	Location	Status	Risk Level
XSS	Search Bar	Confirmed	High
SQL Injection	Login Page	Confirmed	Critical
Misconfiguration	HTTP Headers	Partial	Medium

5. Recommendations

- Sanitize and validate all user inputs on both client and server sides.
- Use HTTP security headers:
  - Add `Content-Security-Policy` to restrict scripts.
  - Enforce HTTPS with `Strict-Transport-Security`.
  - Use `Referrer-Policy` to reduce sensitive info leakage.
- Use parameterized queries or ORM to prevent SQLi.
- Implement logging and monitoring for login attempts and unexpected input.