WEB APPLICATION SECURITY TESTING

REPORT BY: AKSHITA SHARMA

INTRODUCTION

This report is prepared as part of the **Cyber Security Internship with Future Interns**. The goal of Task 1 was to perform **web application security testing** on a vulnerable web application (OWASP Juice Shop) to identify and check common security flaws such as **Cross-Site Scripting (XSS)**, **misconfigured security headers**, **information disclosure**, and other vulnerabilities.

Using the **OWASP ZAP tool**, an automated vulnerability scanner, a complete scan of the Juice Shop application was conducted. The tool helped identify several weaknesses in the application's security posture, which are commonly seen in modern web platforms.

Each vulnerability discovered was carefully reviewed and mapped to the relevant category in the **OWASP Top 10**, which

is a globally recognized framework that highlights the most critical security risks to web applications.

The objective of this task was not only to detect security issues but also to understand the underlying causes and recommend practical mitigation strategies. This report provides a summary of the vulnerabilities found, their potential impact, and suggested remediations to strengthen the application's security.

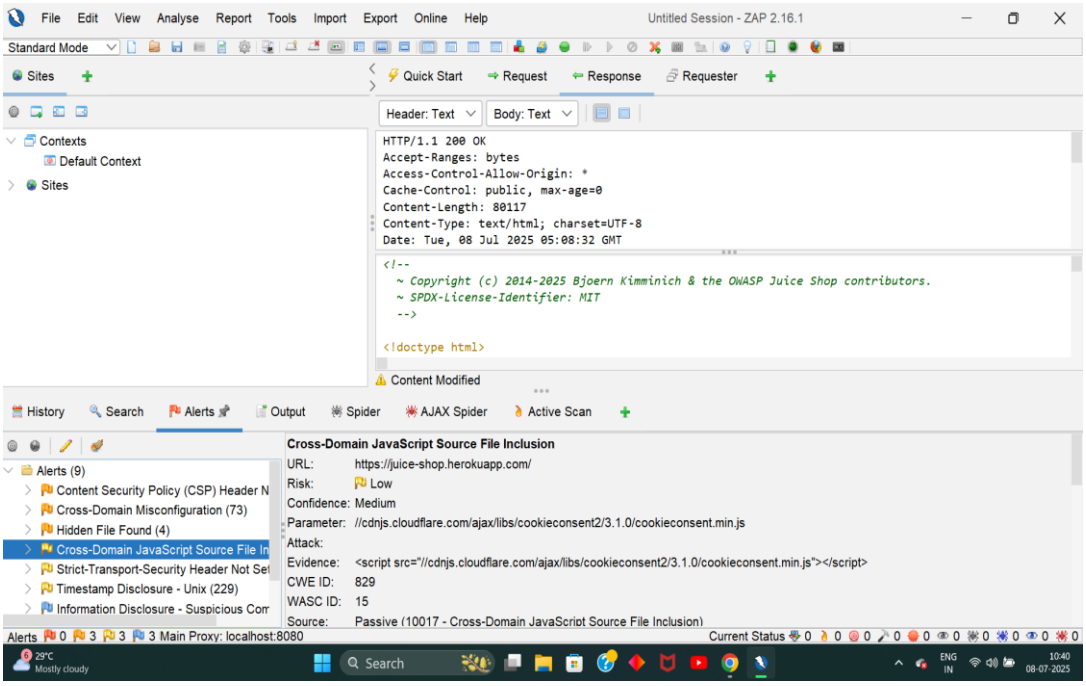
VULNERABILITIES

Vulnerability 1:

🔐 **Vulnerability 1: Cross-Domain JavaScript Source File Inclusion**

- **Tool Used**: OWASP ZAP
- **Location**: Site loads external JS files from multiple domains
- **OWASP Mapping**: A05 – Security Misconfiguration
- **Risk Level**: Medium
- **Description**: The application includes JavaScript files from external sources without strict validation. This increases the risk of loading malicious scripts from compromised domains.

- **Impact**: Could allow attackers to inject harmful code if external sources are hijacked.
- **Mitigation**:
  - Use **Subresource Integrity (SRI)** to verify file integrity
  - Load scripts only from **trusted domains**
  - Apply **Content Security Policy (CSP)** to limit external script execution



🔐 Vulnerability 2: Strict-Transport-Security (HSTS) Header Not Set

- **Tool Used**: OWASP ZAP
- **Location**: All pages of the application
- **OWASP Mapping**: A05 – Security Misconfiguration

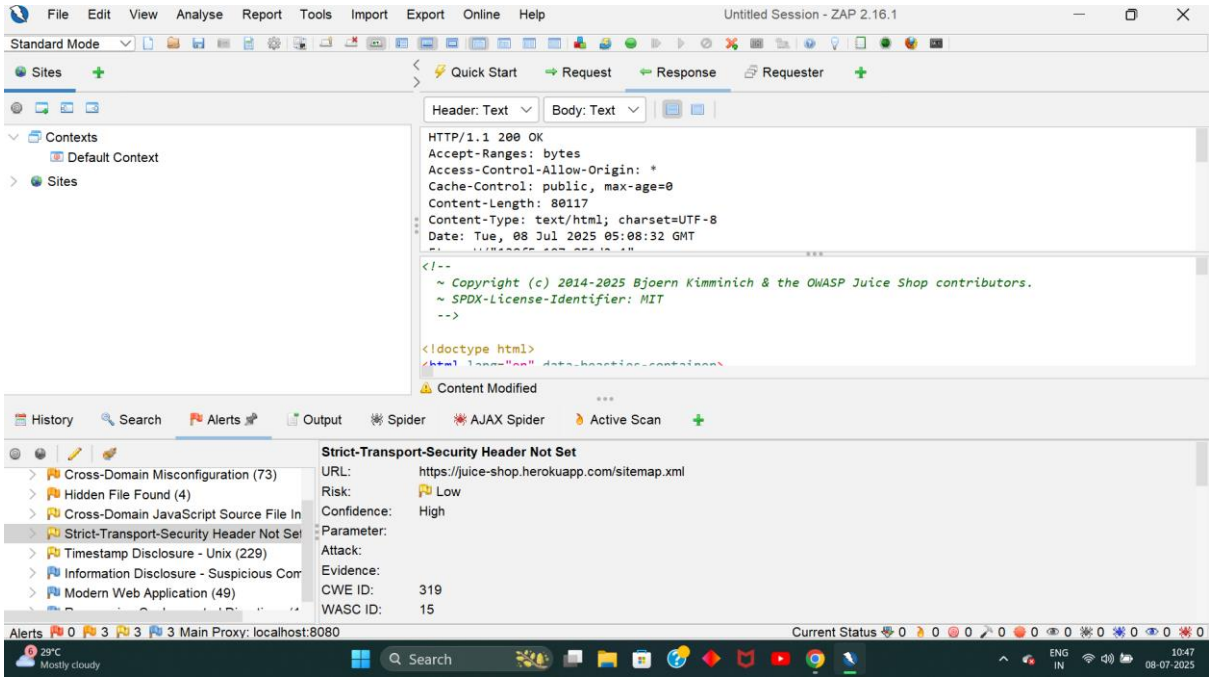
- **Risk Level**: Low
- **Description**: The application does not enforce HTTPS connections using the HSTS header.
- **Impact**: Users may connect over insecure HTTP, making them vulnerable to man-in-the-middle attacks.

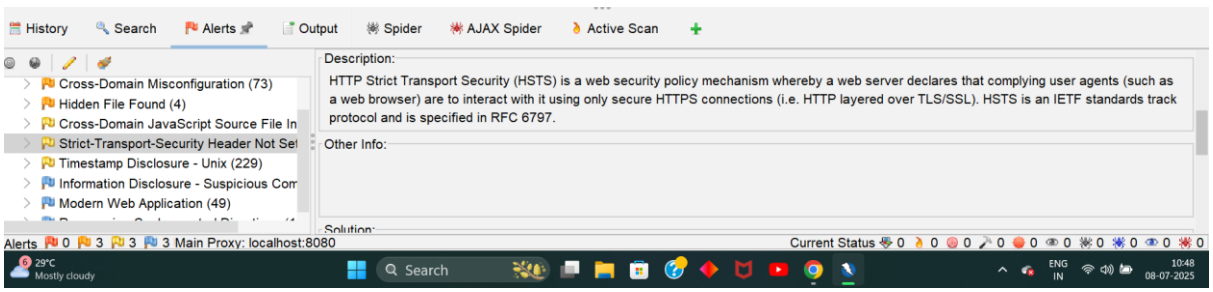
Mitigation:

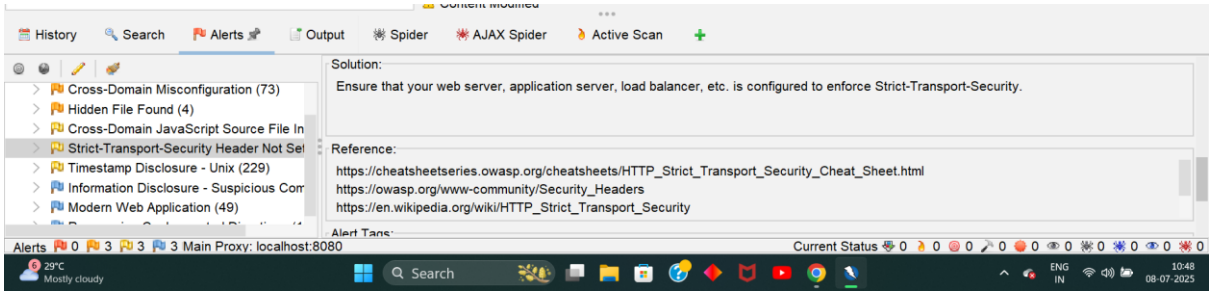
- Add the following HTTP header:

Strict-Transport-Security: max-age=63072000; includeSubDomains; preload

Redirect all HTTP traffic to HTTPS







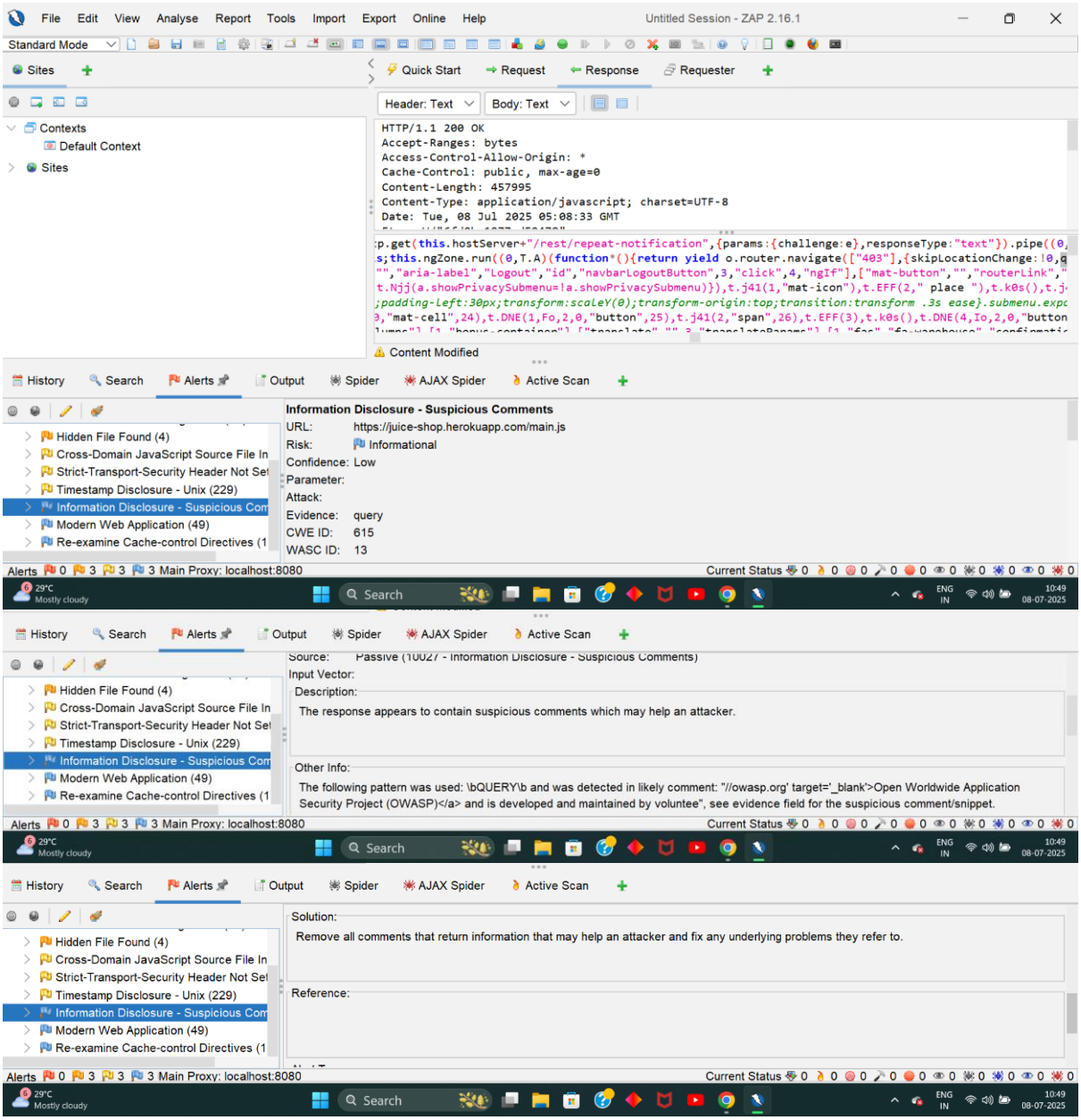
🔓 Vulnerability 3: Information Disclosure – Suspicious Comments in Source Code

- **Tool Used: OWASP ZAP**
- **Location: Public JavaScript files and HTML source**
- **OWASP Mapping: A05 – Security Misconfiguration**
- **Risk Level: Low**
- **Description: The application includes developer comments in publicly accessible files. These may expose hints about internal logic, unfinished features, or system structure.**
- **Impact: Attackers can use this information for further reconnaissance and exploit development.**
- **Evidence:**

Comments like // TODO: Disable login bypass or // Admin route for testing were found.

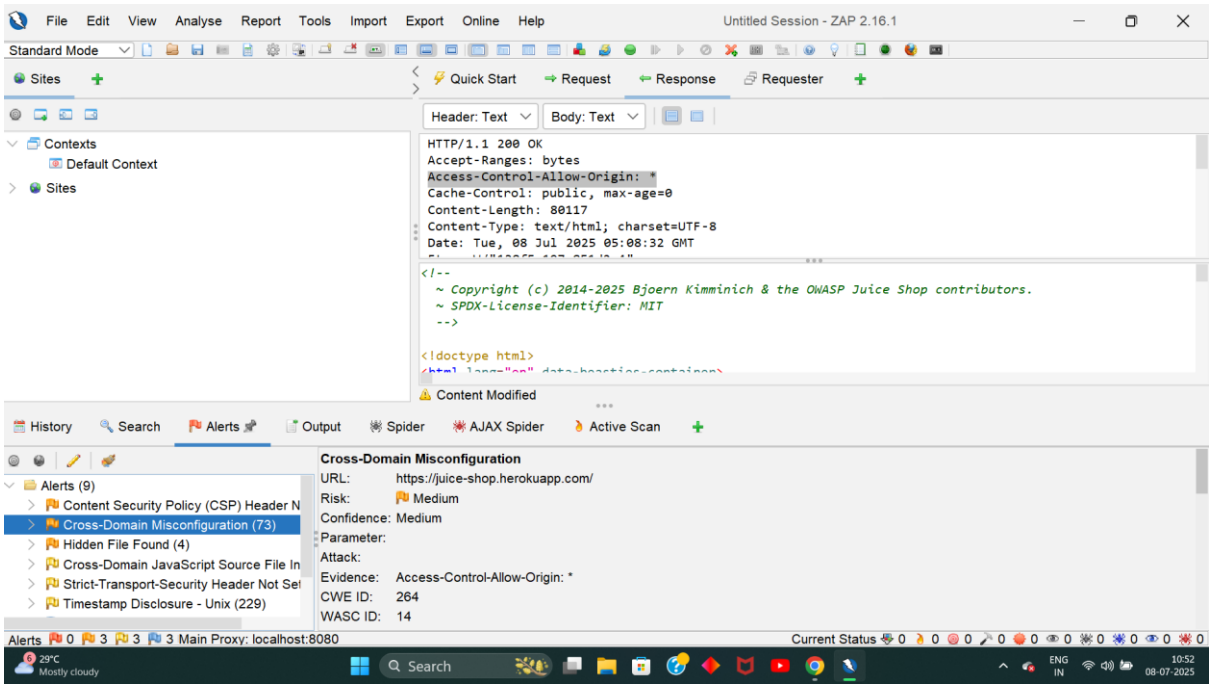
- **Mitigation:**
    - **Remove all comments from production code**

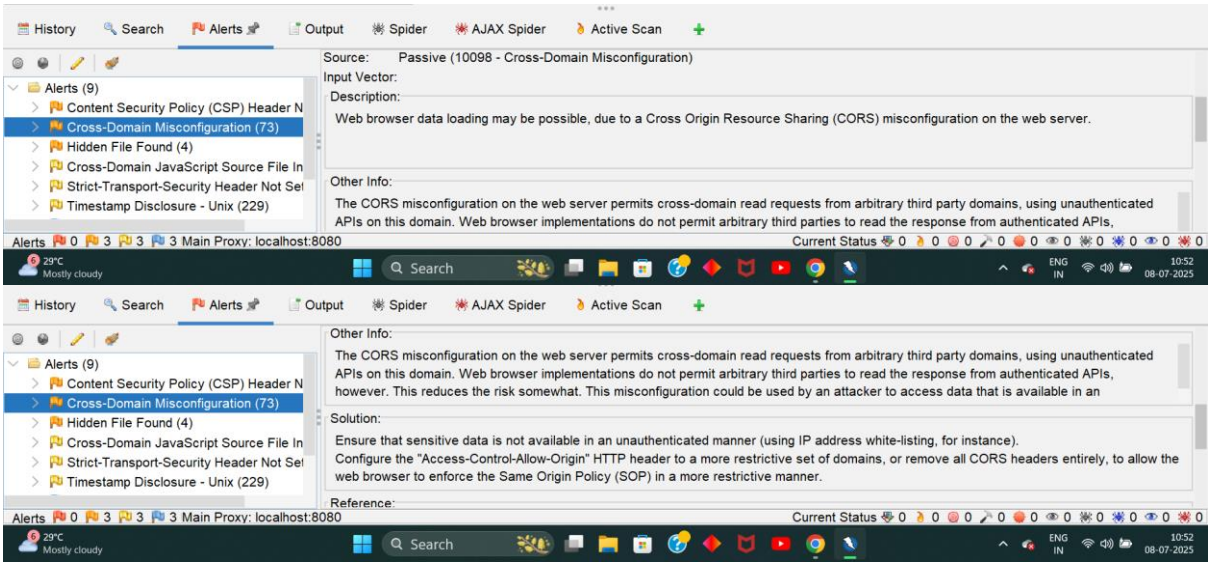
- **Minify JS/CSS/HTML files to hide internal logic**
- **Use build tools to strip comments automatically before deployment**



🔓 Vulnerability 4: Cross-Domain Misconfiguration (CORS)

- **Tool Used**: OWASP ZAP

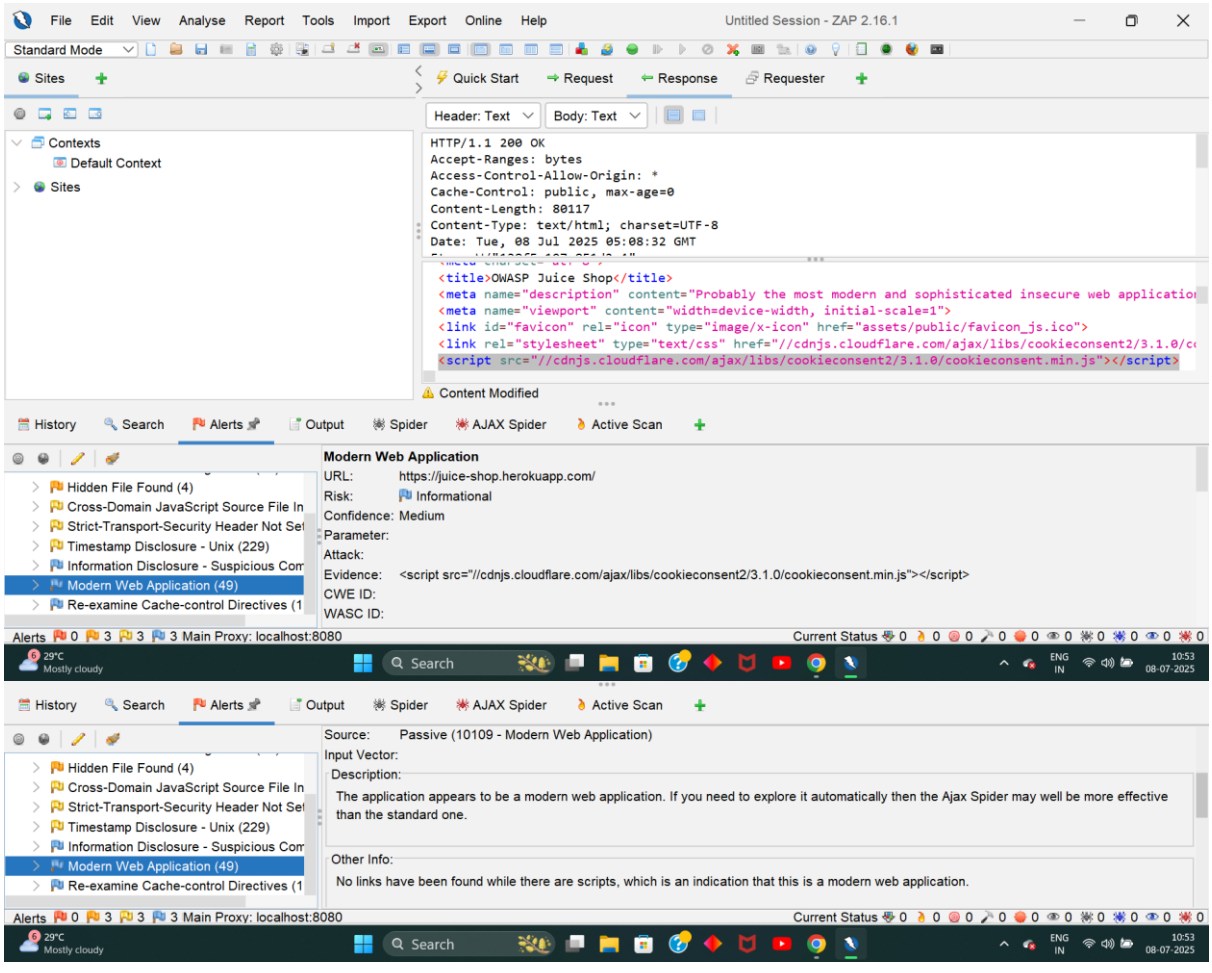
- **Location**: HTTP response headers
- **OWASP Mapping**: A05 – Security Misconfiguration
- **Risk Level**: Medium
- **Description**: The application allows cross-domain resource sharing via a permissive CORS policy (Access-Control-Allow-Origin: *).
- **Impact**: Could allow malicious sites to interact with the application's APIs or data.
- **Mitigation**:
    - Set CORS to only allow trusted domains (e.g., Access-Control-Allow-Origin: https://yourdomain.com)
    - Use proper authentication and origin checks on sensitive endpoints
- 

- 

🔓 Vulnerability 5: Modern Web Application Vulnerability

- **Tool Used**: OWASP ZAP
- **Location**: Found during dynamic analysis of JavaScript-heavy (SPA) frontend
- **OWASP Mapping**: **A04 – Insecure Design**
- **Risk Level**: Medium
- **Description**: The application exposes excessive functionality and logic in the frontend. Sensitive processes (such as ID manipulation, auth flows, or hidden fields) can be manipulated directly in the browser.
- **Impact**: Attackers may bypass controls or exploit hidden frontend functionality that was meant to be protected by backend logic.
- **Mitigation**:

- Shift all sensitive logic (validation, access control) to the backend
- Avoid relying solely on frontend frameworks (React, Angular) for security
- Implement secure APIs with strict server-side validation
- Use feature flags to hide unfinished or dev-only features

- 

Summary of Vulnerabilities and OWASP Mapping

Vulnerability	OWASP Mapping	OWASP Code
Cross-Domain JS File Inclusion	Security Misconfiguration	A05

Strict-Transport-Security Header Not Set	Security Misconfiguration	A05
Suspicious Comments in Source Code	Security Misconfiguration / Logging Failures	A05 / A09
Cross-Domain Misconfiguration (CORS)	Security Misconfiguration	A05
Modern Web Application Vulnerability	Insecure Design	A04

Conclusion

This task provided valuable hands-on experience in identifying and analyzing common security vulnerabilities in a modern web application using industry-standard tools like OWASP ZAP. Through scanning and manual analysis of the OWASP Juice Shop platform, several weaknesses were discovered, including security misconfigurations, information disclosure, and insecure design patterns.

Each vulnerability was mapped to the OWASP Top 10 framework, emphasizing its real-world relevance and potential impact. Mitigation strategies were proposed for every issue to demonstrate an understanding of how such flaws can be remediated in a production environment.

Overall, this task has strengthened my foundational knowledge of web application security, ethical hacking practices, and secure development standards — laying the groundwork for deeper learning and real-world application in the field of cybersecurity.