

# YENEPOYA INSTITUTE OF ARTS SCIENCE COMMERCE AND MANAGMENT



## FINAL PROJECT REPORT

ON

## VULNERABILITY ASSESSMENT ON A WEB APPLICATION

### TEAM MEMBERS:

NAME	REGISTER NUMBER	EMAIL ID
Muhammed afeef	22BCIECS086	22205@yenepoya.edu.in
Muhammed Savas K V	22BCIECS095	22361@yenepoya.edu.in
Aktar Zameel	22BCIECS022	21207@yenepoya.edu.in
Ruhaim Riswan	22BCIECS107	22283@yenepoya.edu.in
Muhammed sakariya	22BCIECS094	33313@yenepoya.edu.in

GUIDED BY: MR. SHASHANK

# Table of Contents

<b>Executive summary .....</b>	<b>1</b>
<b>1. Background.....</b>	<b>2</b>
1.1 Scope .....	2
1.2 Aim .....	2
1.3 Technologies .....	2
<b>2. Overall Summary .....</b>	<b>4</b>
2.1 Overall Risk Rating .....	4
2.2 Found Vulnerabilities .....	4
<b>3. Implementation .....</b>	<b>5</b>
3.1 Installing OWASP Juice Shop Locally Using Node.js .....	5
<b>4. Technical Explanation .....</b>	<b>7</b>
4.1 Sql injection .....	7
4.2 Information Disclosure .....	8
4.3 Insecure Direct Object Reference .....	9
4.4 Information Disclosure .....	10
4.5 Business Logic .....	11
4.6 Information Disclosure .....	13
4.7 HTML Injection through Feedback .....	14
<b>5. Snapshots of The Project .....</b>	<b>16</b>
<b>6. Conclusion .....</b>	<b>19</b>
<b>7. References .....</b>	<b>20</b>

# Executive summary

## Assessment overview

This vulnerability assessment was performed on **OWASP Juice Shop**, an open-source, deliberately insecure web application developed by OWASP for educational and testing purposes. Juice Shop simulates a real-world e-commerce platform and includes a broad spectrum of security flaws drawn from the OWASP Top 10, as well as other known vulnerabilities.

The purpose of these Web Security Assessments was to identify exploitable vulnerabilities and insufficiently configured security controls to determine the likelihood that users with considerable, little, or no prior knowledge of the applications (e.g., informed, and uninformed insiders and outsiders) could exploit weaknesses in their applications as those cataloged in the OWASP Top 10, OWASP ASVS, SANS, NIST, OWASP testing guide and Penetration Testing Execution Standard.

The assessment involved both automated and manual testing techniques to identify security flaws present in the application. The findings provide practical insight into how attackers exploit common vulnerabilities and offer recommendations on secure coding practices and mitigation strategies.

## High-Level Test Outcomes

The team uncovered multiple vulnerabilities inside of the web application. The penetration testing team identified vulnerabilities that demand immediate attention. The most severe vulnerabilities include Business Logic and Authentication Bypass, both of which pose significant risks to the system's integrity and security. Following these critical issues, the wInsecure Direct Object Referencecritical, the ability to recycle signups as other users, reflected XSS, Business Logic (repeated), Password leak, and Regular User capability to delete feedback on the admin panel. These high-risk vulnerabilities should be promptly addressed to mitigate potential exploits. Additionally, medium-level vulnerabilities such as User Name Enumeration and CSRF were identified, requiring attention to prevent security breaches. The team also observed low-risk issues, such as Information Disclosure, Bully chat bot, Front End showing routes, and HTML Injection through Feedback. While these are less critical, addressing them enhances overall system security. Lastly, two informational vulnerabilities were noted: Cookies Missing HTTP Only flags and HTML Injection through Feedback, providing insights for improved security measures

# 1 Background

## 1.1 Scope

- Web Application URL : <http://localhost:3000/>
- Assessment period : 01/03/2025 – 30/04/2025
- Testing methodology : Black Box
- Technologies Reviewed : Frontend, Backend, APIs, Authentication Mechanisms, Input Validation, etc.

## 1.2 Aim

The aim of this vulnerability assessment is to systematically identify, analyze, and document security weaknesses within the OWASP Juice Shop web application, in order to understand how such vulnerabilities can be exploited and to recommend appropriate mitigation strategies. This assessment is conducted in a controlled environment for educational purposes, with the broader goal of enhancing practical skills in web application security and promoting awareness of secure development practices.

## 1.3 Technologies

This section outlines the technologies, tools, and platforms used throughout the vulnerability assessment of the OWASP Juice Shop application. These were selected to provide both automated and manual testing capabilities, ensuring comprehensive coverage of potential vulnerabilities.

### 1. Target application stack

- Application Name : OWASP juice shop
- Frontend : Angular
- Backend : Node.js with Express.js
- Database : SQLite
- Hosting Enviroment : localhost(Node.js server)

Juice Shop mimics a modern e-commerce platform and is intentionally designed with a wide range of vulnerabilities based on OWASP Top 10, allowing for realistic security assessments.

### 2. Assessment tools

- OWASP ZAP (Zed Attack Proxy)

Used for automated scanning, spidering, and manual vulnerability testing.

- Burp Suite (Community Edition)

Used for intercepting

and modifying HTTP traffic, testing input validation, and simulating various attacks.

- Nikto

Used for basic web server vulnerability scanning.

- Nmap

Network scanning to identify open ports and services (optional, if network-level analysis was performed).

- Dirb / Gobuster

Used to enumerate hidden files and directories.

### 3. Manual Testing Techniques

- SQL Injection (SQLi)
- Cross-Site Scripting (XSS)
- Broken Access Control
- Insecure Direct Object References (IDOR)
- CSRF and others from the OWASP Top 10
- Browser Developer Tools for manual inspection and DOM analysis

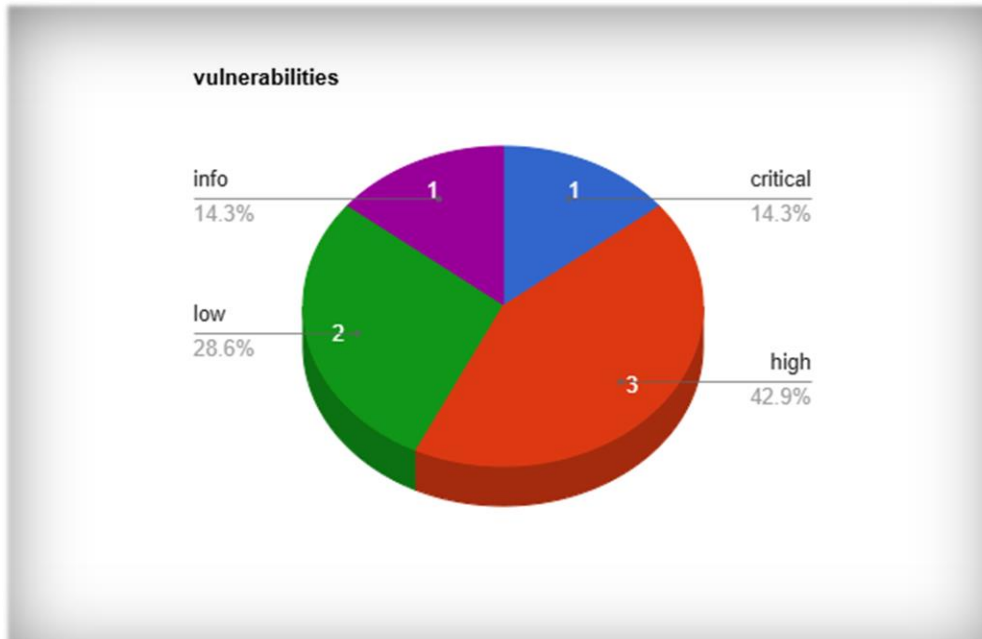
### 4. Operating environment

- **Operating System:** Kali Linux
- **Browser:** Firefox

This combination of tools and technologies allowed for a thorough exploration of the vulnerabilities intentionally embedded in OWASP Juice Shop, simulating real-world attack scenarios and aiding in the development of hands-on skills in ethical hacking and web security testing.

## 2 Overall Summary

### 2.1 Overall Risk Rating



### 2.2 Found Vulnerabilities

vulnerability	Severity
Sql injection	Critical
Information Disclosure	High
Insecure Direct Object Reference	High
Information Disclosure	High
Business Logic	Low
Information Disclosure	Low
Cross site scripting	Info

## 3 Implementation

### 3.1 Installing OWASP Juice Shop Locally Using Node.js

This section outlines the process to set up the OWASP Juice Shop web application on a local machine for vulnerability assessment using Node.js.

Make sure you have the following installed on your Linux system:

- **Node.js and npm** (Node Package Manager)
- **Git**
- **Build tools** (like `build-essential`) for compiling native modules if needed

You can install these with:

```
“sudo apt update”
```

```
“sudo apt install -y nodejs npm git build-essential”
```

To check installed versions:

```
node -v
```

```
npm -v
```

```
git --version
```

Step 1: Clone the Juice Shop Repository

Open a terminal and clone the official OWASP Juice Shop repo:

```
git clone https://github.com/juice-shop/juice-shop.git
```

Step 2: Navigate into the Project Directory

```
cd juice-shop
```

Step 3: Install Dependencies

Use npm to install all required Node.js packages:

```
npm install
```

Step 4: Start the Application

Start the Juice Shop server:

```
npm start
```

Step 5: Access the Application

Open a web browser on your Linux machine and go to:

<http://localhost:3000>

should see the OWASP Juice Shop homepage.



## 4 Technical Explanation

## 4.1 Sql injection

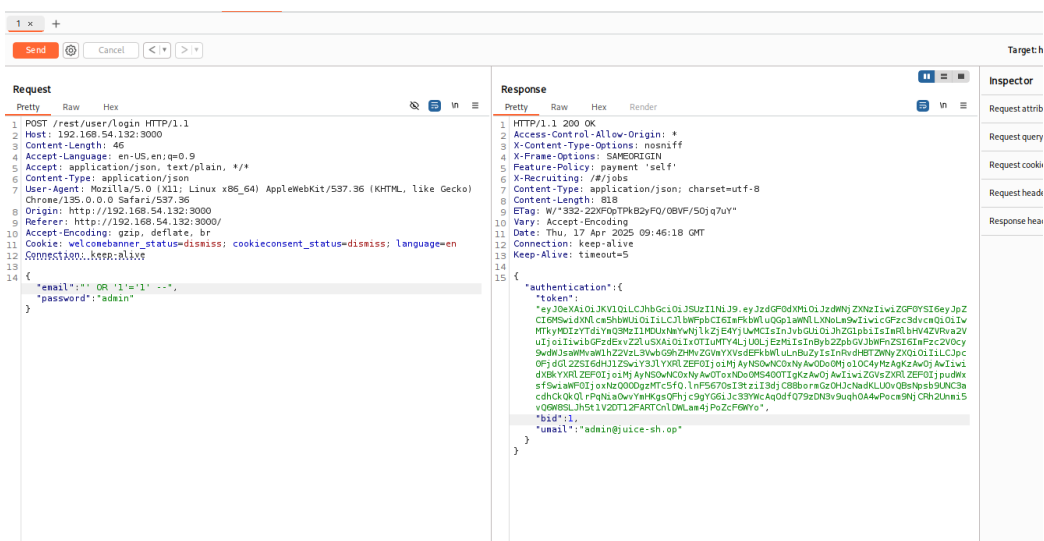
## Overview

Vulnerability	Sql injection
Description	The product constructs all or part of an SQL command using externally influenced input from an upstream component, but it does not neutralize or incorrectly neutralizes special elements that could modify the intended SQL command when it is sent to a downstream component.
CVE/CWE	CWE-89
Rating	Critical
Endpoint	/login

## How to replicate

it is possible to login as admin with the following username and password:

' OR '1'='1' --



this entirely bypasses authentication and allows us to use the application as admin:

## Remediation

To mitigate the SQL injection vulnerability, several steps can be taken. First, employ parameterized queries or prepared statements instead of directly concatenating user inputs into SQL queries. This ensures that user-supplied data is treated as data rather than executable code. Additionally, implement input validation and sanitization routines to filter

out potentially malicious characters and patterns from user inputs. This can help to block SQL injection payloads before they reach the database. Furthermore, enforce the principle of least privilege by ensuring that database users have only the necessary permissions required for their intended tasks, reducing the potential impact of successful SQL injection attacks. Regularly update database software and libraries to patch any known vulnerabilities that could be exploited by attackers. Lastly, conduct regular security audits and penetration tests to identify and remediate any SQL injection vulnerabilities that may exist within the application. By following these measures, the risk of SQL injection attacks can be significantly reduced

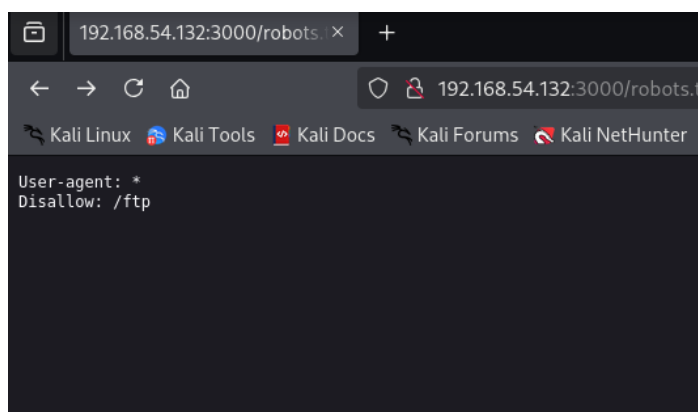
## 4.2 Information Disclosure

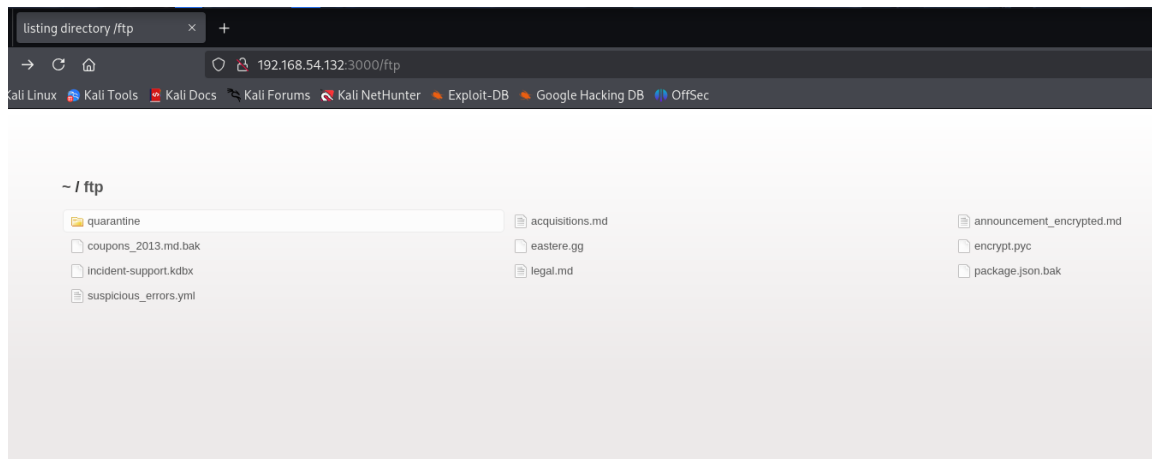
### Overview

Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information
CVE/CWE	CWE-220
Rating	High
Endpoint	/robots.txt

### How to replicate

Navigate to <http://localhost/robots.txt> you will then find a link to the ftp public access folder of the website:





## Remediation

Disallow the access to the ftp folder through .htaccess or other methods.

## 4.3 Insecure Direct Object Reference

### Overview

Vulnerability	Insecure Direct object Reference
Description	The system's authorization functionality does not prevent one user from gaining access to another user's data or record by modifying the key value identifying the data.
CVE/CWE	CWE-639
Rating	High
Endpoint	/rest/basket/{id}

### How to replicate

The view basket endpoint is vulnerable to insecure direct object reference it is possible to view other accounts baskets through manipulation of the ID that is in the url, a request of me viewing my basket:

[illegible]

With this we can also get the UserID of the account we are viewing the basket from

## Remediation

To remediate the Insecure Direct Object Reference (IDOR) vulnerability, several key measures can be implemented. First, establish robust authentication and authorization mechanisms to ensure that users can only access their own basket data. Next, replace direct object identifiers in URLs with indirect references to prevent manipulation by unauthorized users. Validate user permissions before allowing access to sensitive resources, ensuring that only authorized users can view and modify their own baskets. Enforce Role-Based Access Control (RBAC) to restrict users to actions and resources appropriate for their roles. Apply contextual access controls based on user context to add an extra layer of security. Log access attempts to sensitive resources for monitoring and detecting potential malicious activities. Regularly conduct security assessments, including penetration testing and code reviews, to identify and remediate vulnerabilities. Educate developers and users on secure coding practices and the importance of data protection. Keep software dependencies up to date to mitigate known vulnerabilities. Consider implementing a bug bounty program to encourage responsible disclosure of vulnerabilities. By implementing these measures, the IDOR vulnerability can be effectively mitigated, enhancing overall application security.

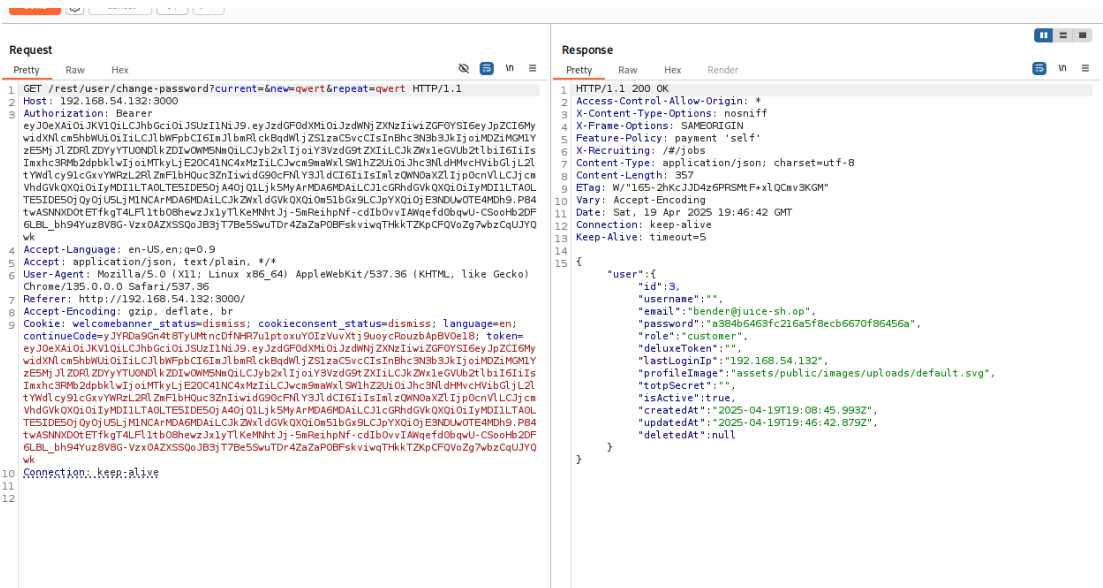
## 4.4 Information Disclosure

## Overview

Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CVE/CWE	CWE-200
Rating	High
Endpoint	/rest/memories

## How to replicate

on the request to the view memories password hashes are disclosed:



## Remediation

The development team should implement access controls to restrict unauthorized access to sensitive user information, such as password hashes. Additionally, consider using secure hashing algorithms (e.g., bcrypt, Argon2) with proper salting and iteration counts to hash passwords securely. It's crucial to avoid storing or exposing password hashes directly and instead provide functionalities for password reset or authentication using secure mechanisms. Conduct thorough security testing, including vulnerability scanning and code reviews, to identify and remediate any similar vulnerabilities within the application. Lastly, prioritize user education on password security best practices, emphasizing the importance of using strong, unique passwords and enabling multi-factor authentication. By diligently implementing these measures, the vulnerability can be effectively mitigated, safeguarding user passwords and enhancing overall application security.

## 4.5 Business Logic

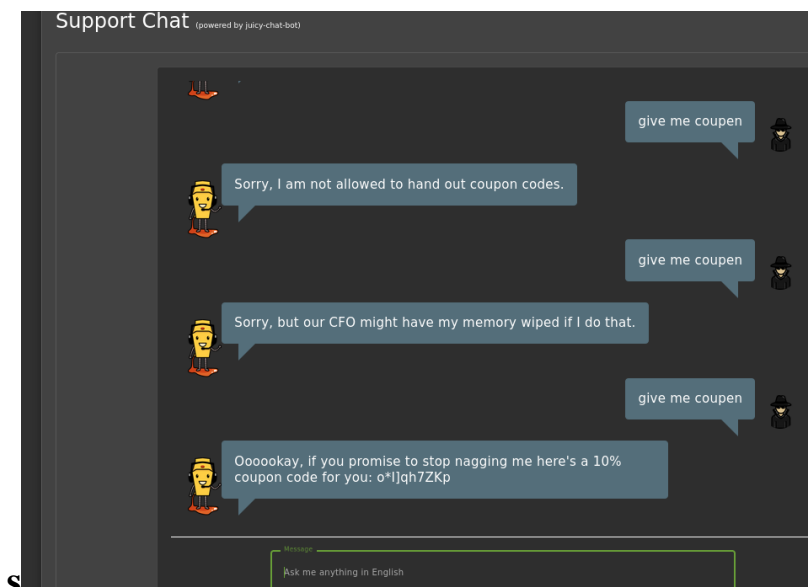
## Overview

Vulnerability	Business logic
Description	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application. They can be difficult to find automatically, since they typically involve legitimate use of the application's functionality. However, many business logic errors can exhibit patterns that are similar to well-understood implementation and design weaknesses
CVE/CWE	CWE-840
Rating	Low
Endpoint	/chatbot

## How to replicate

it is possible to spam the chat bot up until it gives you a code:

mNYT0g+yBo



## Remediation

First, implement rate limiting or cooldown mechanisms within the chatbot to prevent users from excessively querying discount codes within a short period of time. This ensures that legitimate users can still access the chatbot without disruption while mitigating abuse. Additionally, introduce authentication and authorization checks to ensure that only authenticated users are eligible to receive discount codes, and limit the number of codes a user can request within a specified time frame. Furthermore, consider implementing CAPTCHA or other bot detection mechanisms to distinguish between human users and

automated scripts attempting to exploit the system. Regularly monitor chatbot interactions and analyze usage patterns to detect and mitigate suspicious activity. Lastly, review and update the business logic governing discount code generation and distribution to ensure that it aligns with the intended functionality and security requirements of the application. By implementing these measures, the vulnerability can be effectively mitigated, reducing the risk of abuse and unauthorized access to discount codes

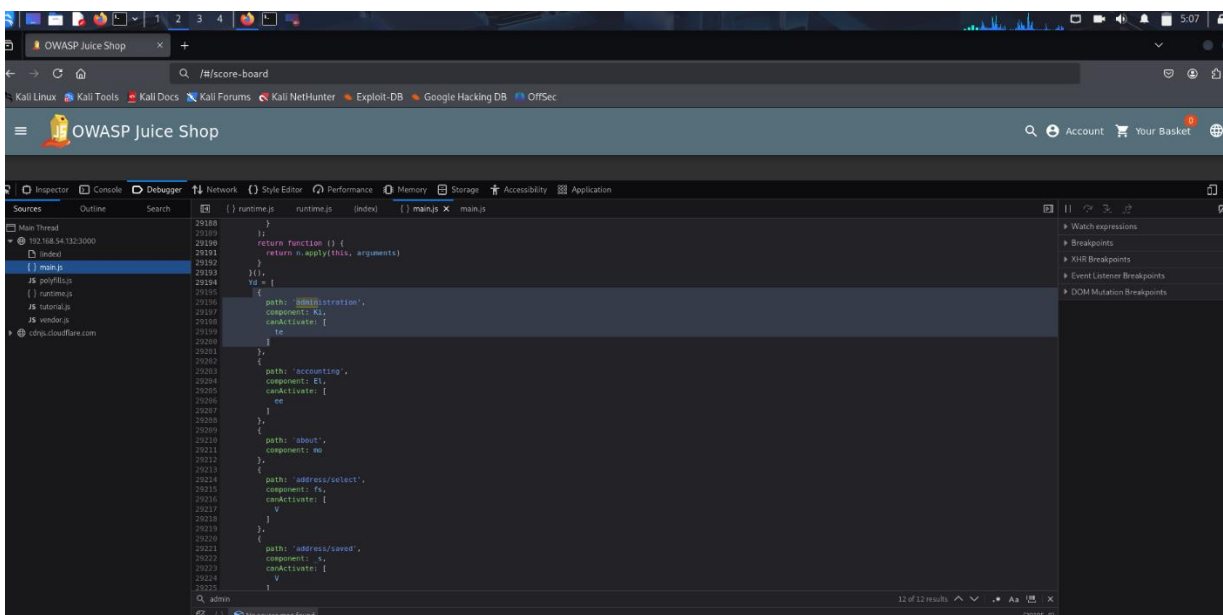
## 4.6 information Disclosure

### Overview

Vulnerability	Information Disclosure
Description	The product exposes sensitive information to an actor that is not explicitly authorized to have access to that information.
CVE/CWE	CWE-220
Rating	Low
Endpoint	/main.js

### How to replicate

when opening the file main.js you can view the different routes since this is a front end built route system:



### Remediation

Do not manage on the front-end the routes of sensitive pages. Front-end code can easily be modified and accessed. It is recommend to leave access control management handling to the backend since that code is not as easily bypassed as front-end code.

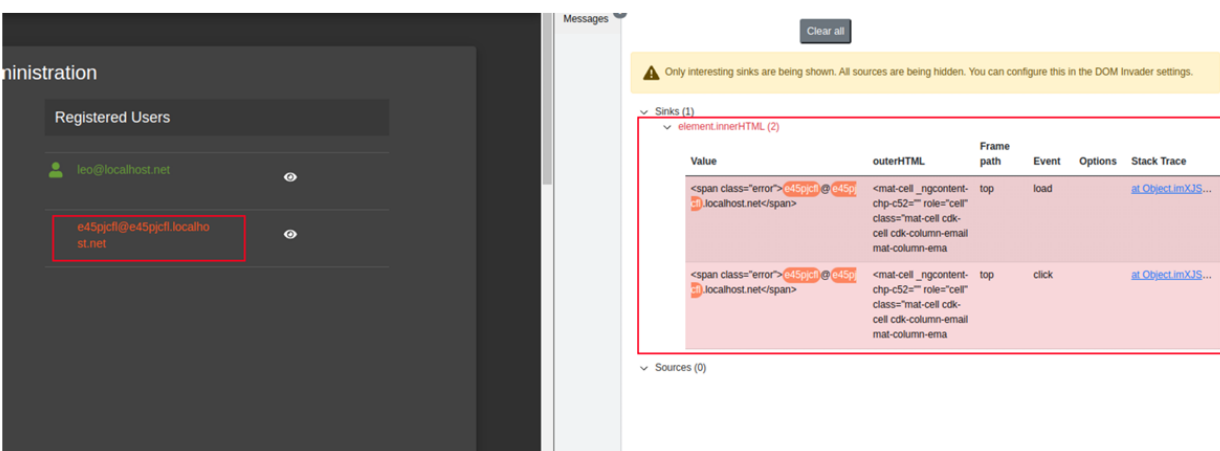
## 4.7 HTML Injection through Feedback

### Overview

Vulnerability	Cross Site Scripting
Description	Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it
CVE/CWE	CWE-79
Rating	Informantional
Endpoint	/#/administration

### How to replicate

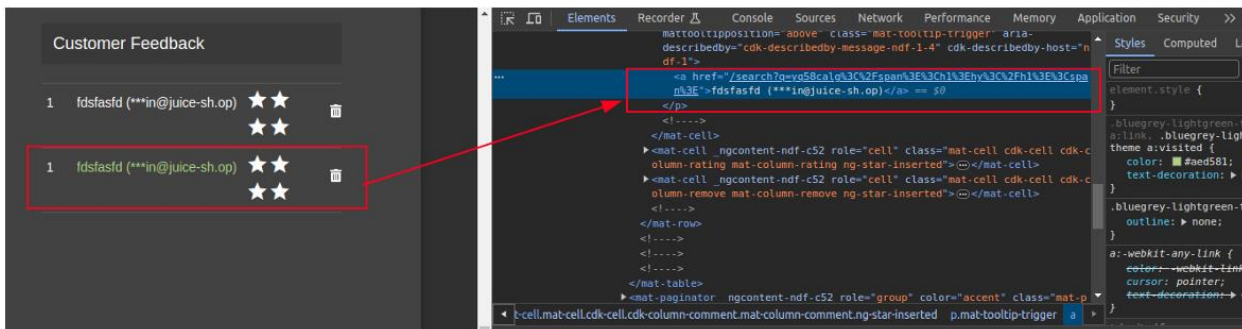
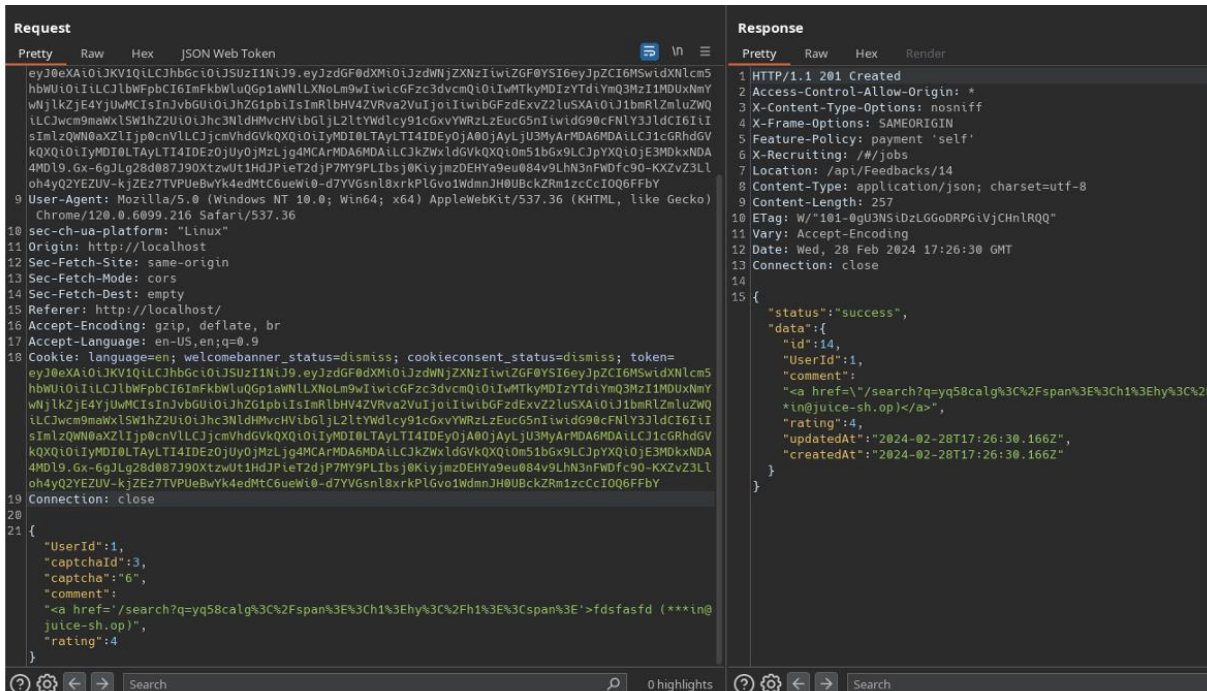
Inside of the administration page there is a stored HTML injection present since the email address is improperly sanitized and injected in the page with document.innerHTML :



The screenshot shows a web application interface with a 'Registered Users' table. One entry is highlighted with a red box: 'e45pjcfl@e45pjcfl.localho st.net'. To the right, a DOM Inspector tool is open, showing a table with columns: Value, outerHTML, Frame path, Event, Options, and Stack Trace. The table contains two rows of data, both representing the injected HTML code. The first row shows the HTML code for a table cell, and the second row shows the HTML code for a table cell. The code is: `<span class="error">e45pjcfl@e45pjcfl.localho st.net</span>`. The DOM Inspector also shows the 'innerHTML' property of the table cell, which contains the injected HTML code.

With this it is then possible to inject a pointing to any link that we would like. Which could then be changed to another vulnerability to cause more damage.

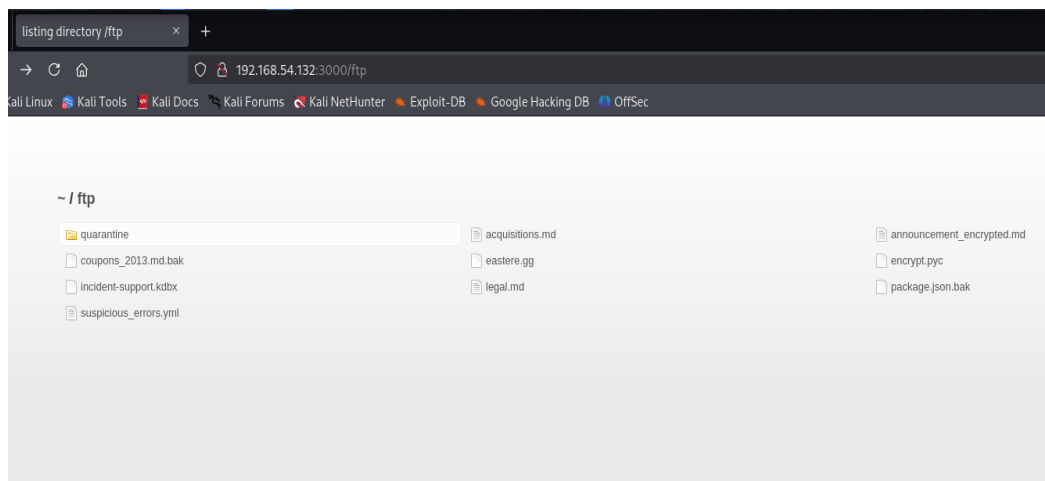
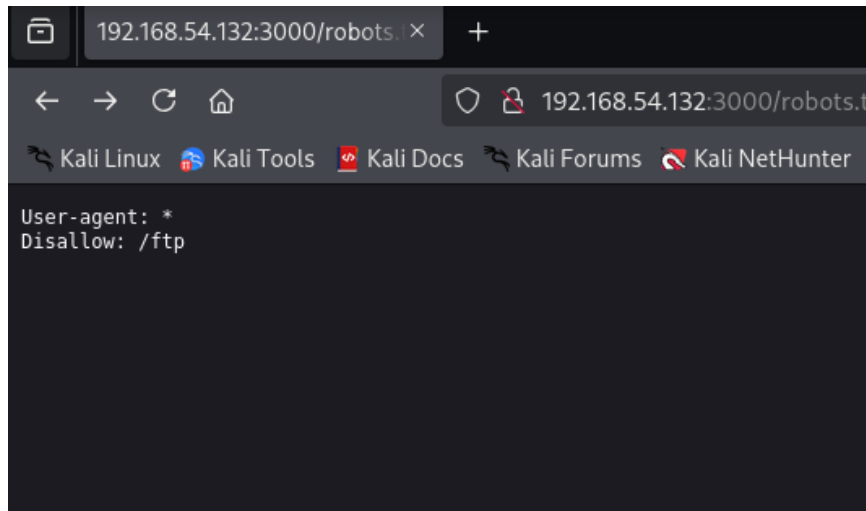


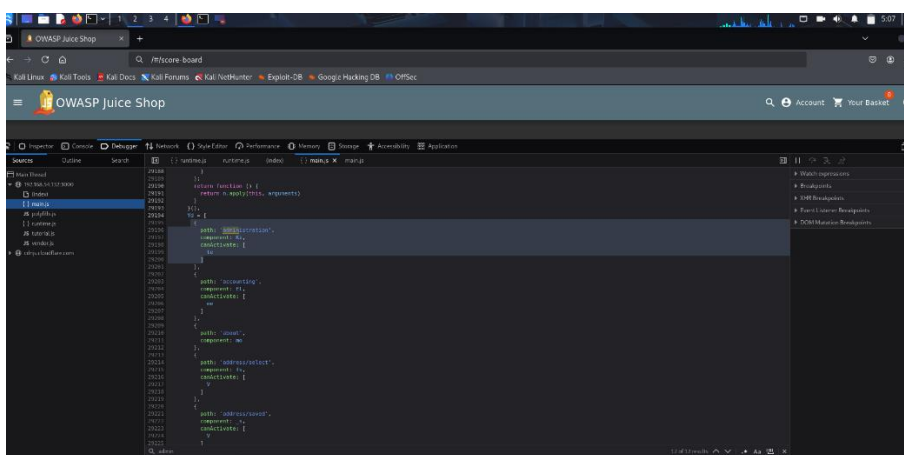
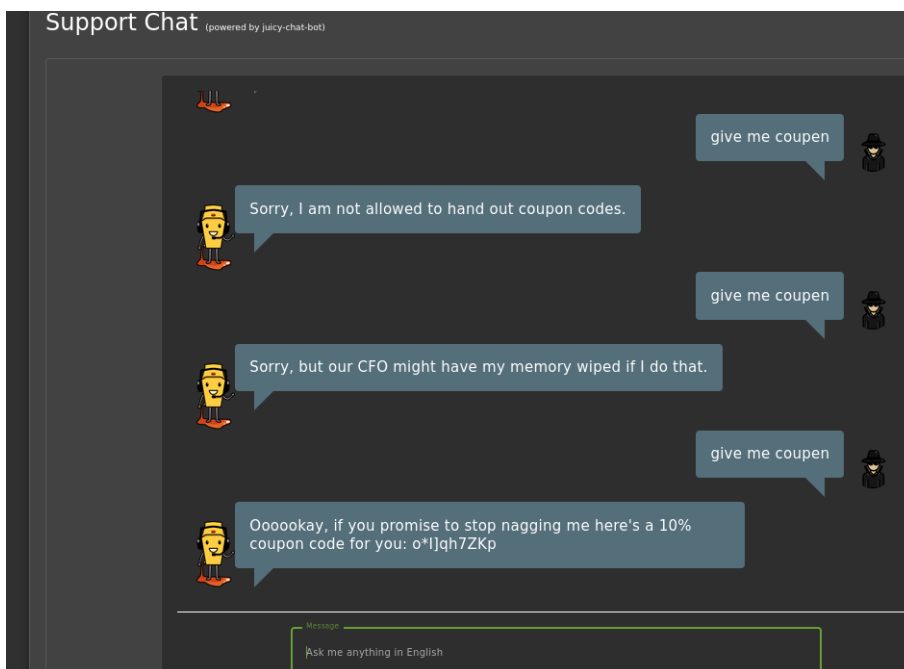
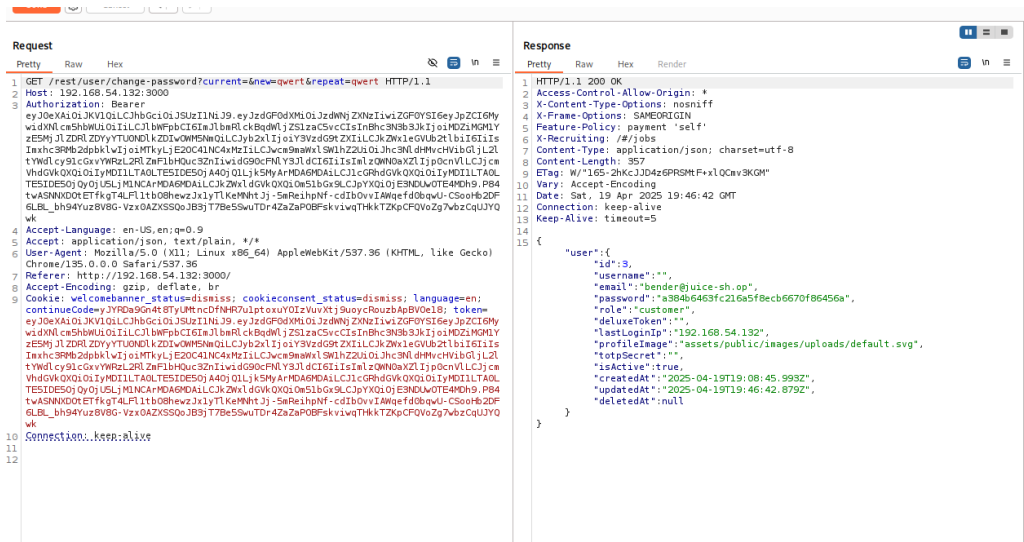


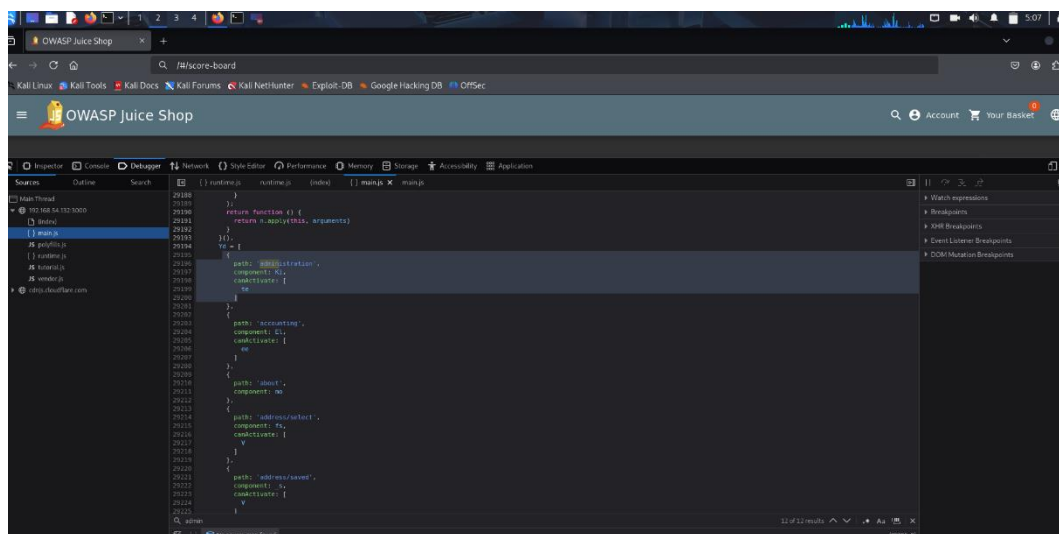
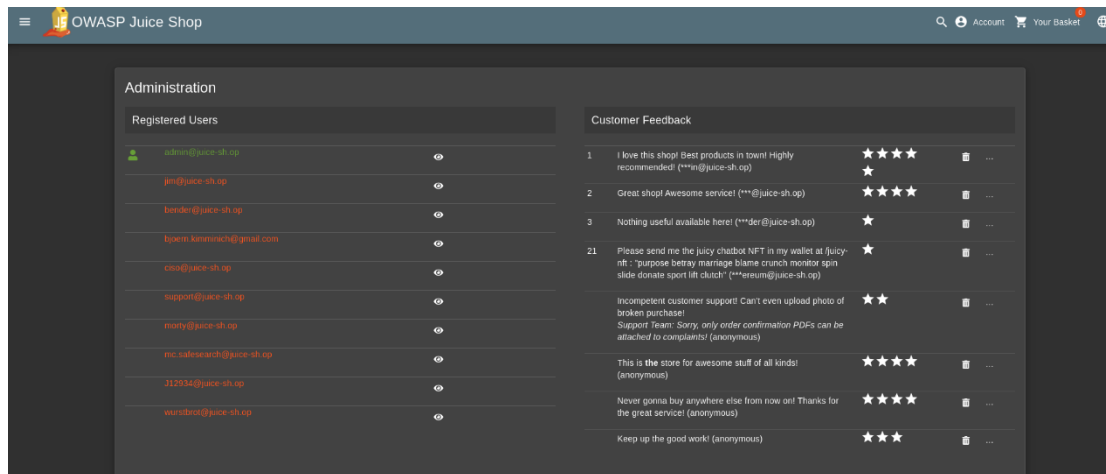
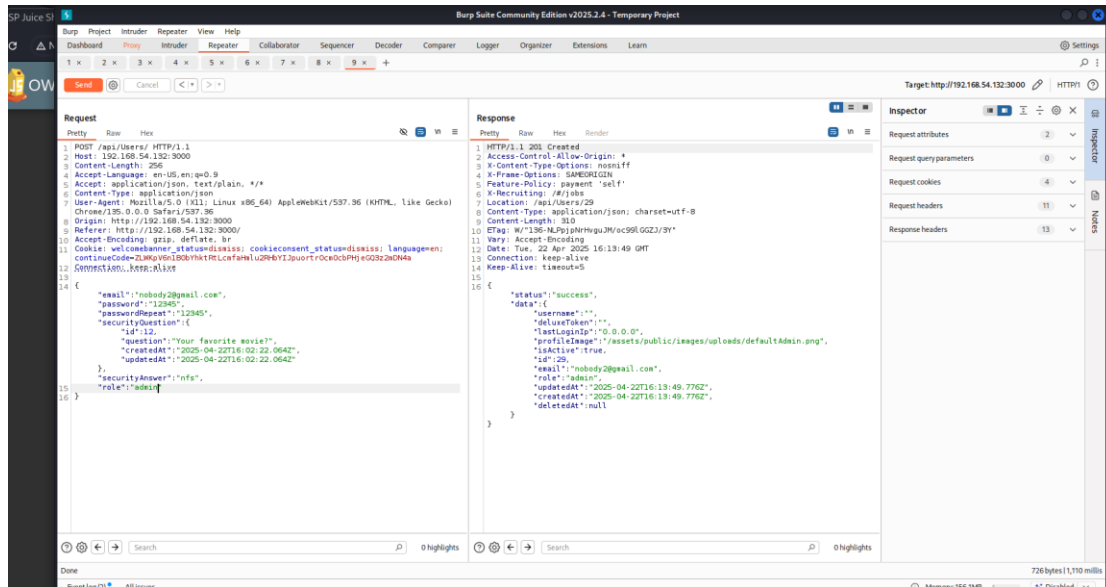
## Remediation

Implement strict input validation and output encoding within the admin panel to sanitize user-generated content, preventing the injection of HTML tags. Additionally, enforce role-based access control to restrict administrator privileges and limit access to sensitive functionalities. Employ CSRF tokens to prevent unauthorized actions initiated by malicious links. Conduct comprehensive security training for administrators, emphasizing vigilance against social engineering attacks and suspicious links. Regularly update and patch the application to mitigate known vulnerabilities. Implement Content Security Policy (CSP) headers to mitigate the impact of XSS attacks. Enhance monitoring and logging mechanisms to detect and respond to suspicious activities promptly. Collaborate with cybersecurity experts to conduct thorough penetration testing and vulnerability assessments. Foster a culture of cybersecurity awareness and proactive risk management within the organization. Communicate transparently with users and stakeholders about security measures implemented to protect sensitive data and prevent future vulnerabilities.

## 5 Snapshots of The Project







## 6 Conclusion

The vulnerability assessment of the web application has revealed several critical and high-severity security issues that pose significant risks to the confidentiality, integrity, and availability of the system and its data. These vulnerabilities, if left unaddressed, could be exploited by malicious actors to gain unauthorized access, manipulate sensitive information, or disrupt application functionality.

Through this assessment, both automated and manual testing techniques were employed to uncover a broad spectrum of vulnerabilities, ranging from injection flaws and broken authentication to insecure access controls and cross-site scripting attacks. The findings highlight common security weaknesses that are prevalent in many web applications and reinforce the importance of proactive security measures.

To mitigate these risks, it is essential to prioritize the remediation of critical and high-impact vulnerabilities immediately. This includes implementing robust input validation, strengthening authentication and session management, enforcing proper access controls, and regularly updating and patching all software components. Furthermore, adopting secure coding practices, performing routine security testing, and integrating security into the software development lifecycle will help prevent future vulnerabilities.

Addressing these issues will significantly enhance the security posture of the application, reduce the potential attack surface, and protect both the organization and its users from potential breaches and data loss. Continuous monitoring and periodic reassessments are recommended to ensure the sustained security and resilience of the web application.

## 7 References

- **OWASP Foundation**

- OWASP Juice Shop Project  
<https://owasp.org/www-project-juice-shop/>
- OWASP Top Ten Web Application Security Risks  
<https://owasp.org/www-project-top-ten/>
- OWASP Testing Guide  
<https://owasp.org/www-project-web-security-testing-guide/>

- **GitHub Repository**

- OWASP Juice Shop Source Code and Documentation  
<https://github.com/juice-shop/juice-shop>

- **Security Tools Documentation**

- OWASP Zed Attack Proxy (ZAP) User Guide  
<https://www.zaproxy.org/docs/desktop/start/>
- Burp Suite Documentation  
<https://portswigger.net/burp/documentation>
- Nikto Web Server Scanner  
<https://cirt.net/Nikto2>
- Nmap Network Scanner  
<https://nmap.org/book/man.html>

- **Node.js and npm**

- Official Node.js Documentation  
<https://nodejs.org/en/docs/>
- npm Documentation  
<https://docs.npmjs.com/>

- **Web Security Resources**

- "The Web Application Hacker's Handbook" by Dafydd Stuttard and Marcus Pinto, Wiley, 2011
- Mozilla Developer Network (MDN) Web Docs – Security  
<https://developer.mozilla.org/en-US/docs/Web/Security>