Contents

[1. Scope of Work 2](#_Toc216126162)

[1.1 Timeline 2](#_Toc216126163)

[1.2 Targets 2](#_Toc216126164)

[2. Finding Overview 3](#_Toc216126165)

[3. Risk Overview 4](#_Toc216126166)

[4. Potential Business Impact [after all vulnerabilities] 5](#_Toc216126167)

[5. Technical Vulnerabilities Details 7](#_Toc216126168)

[5.1 SQL Injection in Search Functionality 7](#_Toc216126169)

[5.2 DOM XSS 9](#_Toc216126170)

[5.3 Exposed Prometheus Metrics Endpoint 11](#_Toc216126171)

[5.4 bussiness logic bypass via client-side manipulation 14](#_Toc216126172)

[5.5 Unrestricted File Upload with Extension Manipulation 16](#_Toc216126173)

[5.6 Broken Authentication 20](#_Toc216126174)

[5.7 Information Disclosure via Client-Side Debugging 24](#_Toc216126175)

[5.8 Improper Access Control in Chatbot Functionality 26](#_Toc216126176)

[5.9 Repetitve Registration 28](#_Toc216126177)

[5.10 UNVALIDATED Redirects 30](#_Toc216126178)

[5.11 Sensitive Data exposure 32](#_Toc216126179)

[5.12 Insecure Direct Object Reference 35](#_Toc216126180)

[5.13 Empty user registration (Improper Input Validation) 38](#_Toc216126181)

[5.14 Forced Zero Rating 42](#_Toc216126182)

# Scope of Work

This section outlines the specific scope of the penetration testing, referred to throughout the document as the targeted application.

## Timeline

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Start Date** | **End Date** | **Man-days** |
| OWASP Juice Shop | Nov 21st, 2025 | Nov 27th, 2025 | 6 |

## Targets

The below scope was OWASP Juice Shop

|  |  |
| --- | --- |
| **ID** | **Asset Identifier (URL/IP)** |
| 1 | <http://localhost:3000/> |

# Finding Overview

The table below presents a high-level summary of the security vulnerabilities that have been successfully found and verified by **ThreatHive**.

|  |  |  |
| --- | --- | --- |
| ID | Vulnerability Name | Severity |
| [Vuln 01](#_VULNERABLE_AND_OUTDATED) | SQL Injection in Search Functionality | **Critical** |
| [Vuln 02](#_Broken_Authentication) | DOM XSS | **High** |
| [Vuln 03](#_Exposed_Prometheus_Metrics) | Exposed Prometheus Metrics Endpoint | **High** |
| [Vuln 04](#_bussiness_logic_bypass) | business logic bypasses via client-side manipulation | **High** |
| [Vuln 05](#_Unrestricted_File_Upload) | Unrestricted File Upload with Extension Manipulation | **High** |
| [Vuln 06](#_Broken_Authentication_2) | Broken Authentication | **High** |
| [Vuln 07](#_Information_Disclosure_via_1) | Information Disclosure | **Medium** |
| [Vuln 08](#_Improper_Access_Control_1) | Improper Access Control in Chatbot Functionality | **Medium** |
| [Vuln 09](#_Repetitve_Registration) | Repetitive registration | **Medium** |
| [Vuln 10](#_UNVALIDATED_REDIrects) | Unvalidated redirects | **Medium** |
| [Vuln 11](#_iNsecure_logging) | Sensitive Data Exposure | **Medium** |
| [Vuln 12](#_iNsecure_logging_1) | Insecure Direct Object Reference | **Medium** |
| [Vuln 13](#_Empty_user_registration) | Improper Input Validation | **Medium** |
| [Vuln 14](#_exposed_web3_sandbox) | Forced Zero Rating | **Low** |

# Risk Overview

The Below chart shows the Number of vulnerabilities, and their risks, categorized as Critical, High, Medium, Low, and Informative.

# Potential Business Impact [after all vulnerabilities]

The highlights of business impact possible due to the Pentest are listed below:

* Exposure of Personally Identifiable Information (PII) such as names, email addresses, phone numbers, and physical addresses.
* Exposure of session cookies.
* Exposure to class-action lawsuits from affected customers.
* Malicious scripts can be designed to harvest sensitive information displayed on the user's screen and send it to an attacker-controlled server.
* Displaying fraudulent or malicious content can severely erode user trust and damage the company's brand reputation.
* Spread false promotions, redirect users to phishing sites, or display offensive content.
* Theft of financial assets.
* Loss of reputation and customers.
* Making unauthorized purchases, accessing order history, redeeming loyalty points, and viewing personal/financial information.
* Accessing and exfiltrating confidential data, manipulating user data, and performing privileged actions if an admin account is compromised.
* Competitors can reverse-engineer application features.
* Attackers can craft precise attacks against discovered endpoints.
* Requires code obfuscation and restructuring.
* Competitors can reverse-engineer application features.
* Hidden functionalities become targets for privilege escalation.
* Demonstrates poor security practices to technical users.
* May violate data protection requirements for secure coding.
* Unauthorized distribution of discount coupons reduces profit margins.
* Coupon codes intended for specific campaigns are compromised.
* Demonstrates weak security controls to customers.
* Violates principle of least privilege in access control design.
* Attackers gain access to all user accounts and data.
* Violation of data protection laws.
* Competitors gain advanced knowledge of acquisition plans.
* Insider information that could be used for illegal trading.
* Loss of competitive advantage in merger negotiations.
* Potential SEC violations for inadequate information protection.
* Loss of investor confidence due to poor security controls.
* Intelligence for DDoS attacks based on resource limits.
* Understanding application architecture and scale.
* Data to plan targeted resource exhaustion attacks.
* Exposure of operational patterns and user activity metrics.
* Blueprint of the application's technical environment.
* Unauthorized access to paid features without payment.
* Widespread circumvention of payment systems.
* Premium features become accessible for free.
* Paying customers subsidize unauthorized users.
* Demonstrates weak payment security controls.
* Customers lose confidence in review authenticity.
* Increased effort to identify and remove fake content.
* Potential defamation or false representation issues.
* Inaccurate feedback data for decision-making.
* Potential for full application takeover.
* Platform could be used to host and distribute malicious files.
* Potential for server compromise leading to data theft.
* Loss of trust if platform is abused for malicious activities.
* Copyright infringement or hosting of illegal content.
* Server resource exhaustion from abusive uploads.

# Technical Vulnerabilities Details

Below are the finding details with their respective exploitation scenarios in severity ordered from Critical to Informative.

## SQL Injection in Search Functionality

|  |  |
| --- | --- |
| Severity | Critical |
| CVSS | [CVSS 9.1](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N) |
| Affected Assets | http://localhost:3000/#/search |
| Reference | [A03:2021 – Injection](https://owasp.org/Top10/A03_2021-Injection/) |

Description

The application's search functionality is vulnerable to SQL injection attacks, allowing attackers to execute arbitrary SQL queries by manipulating the q parameter. The vulnerability exposes both SQLite error messages and the underlying SQL query structure, enabling successful UNION-based attacks to extract database schema information from sqlite\_master. The attack payload 1%'))+UNION+SELECT+sql,2,3,4,5,6,7,8,9+from+sqlite\_master-- successfully bypasses application controls and retrieves database metadata.

IMPACT

* **Database Compromise**: Full read access to all database tables and data.
* **Schema Extraction**: Ability to enumerate database structure via sqlite\_master.
* **Data Exfiltration**: Extraction of sensitive information including user credentials.
* **Query Manipulation**: Execution of arbitrary SELECT, UPDATE, DELETE operations.
* **Error Information Disclosure**: Detailed SQL errors reveal query structure.

RECOMMENDATIONS

* **Emergency Patching**: Implement parameterized queries immediately
* **Input Validation**: Add strict input validation for search parameters
* **Error Handling**: Configure custom error pages to prevent information leakage
* **Input Sanitization**: Whitelist acceptable characters for search terms
* **ORM Usage**: Implement Object-Relational Mapping with built-in protection
* **Output Encoding**: Properly encode database errors to prevent information leakage

Steps to Reproduce

* Enter single quote in the search bar to reveal the SQL query.

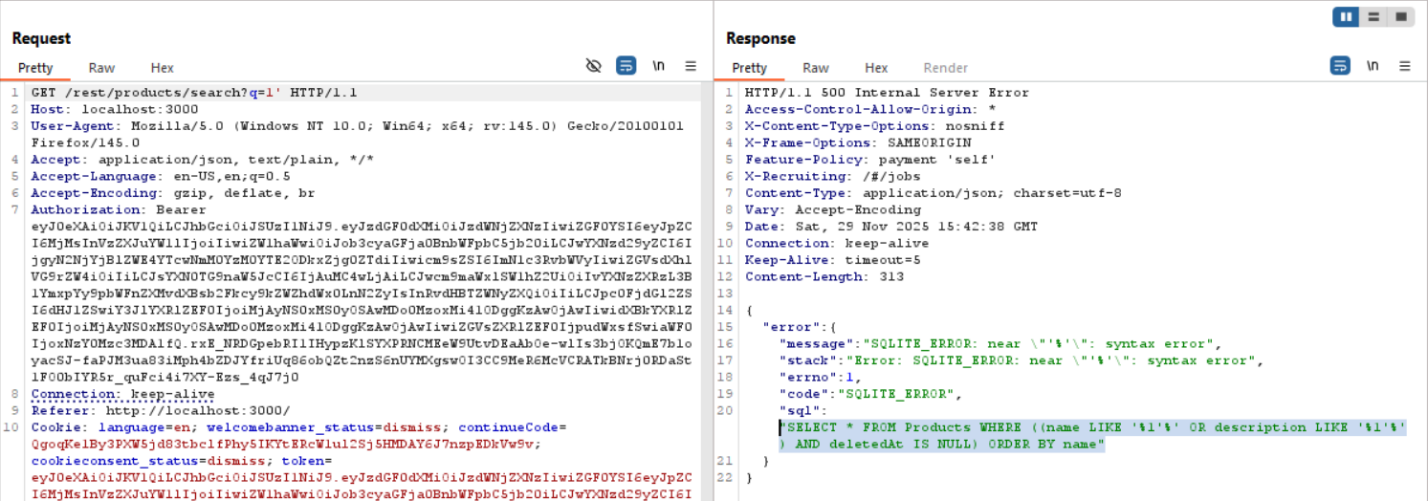


Figure 1: Enter single quote in the search bar to reveal the SQL query

* Then enter **1%'))+UNION+SELECT+sql,2,3,4,5,6,7,8,9+from+sqlite\_master--** to reveal the database metadata

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2:**enter 1%'))+UNION+SELECT+SQL,2,3,4,5,6,7,8,9+FROM+SQLITE\_MASTER--** to reveal the database metadata

## DOM XSS

|  |  |
| --- | --- |
| Severity | High |
| CVSS | [CVSS 7.1](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:L/A:N) |
| Affected Assets | http://localhost:3000/#/search |
| Reference | [OWASP cheat sheet](https://cheatsheetseries.owasp.org/cheatsheets/DOM_based_XSS_Prevention_Cheat_Sheet.html) |

Description

A DOM-based Cross-Site Scripting (XSS) vulnerability has been identified in the website's search functionality. User input from the search bar is unsafely incorporated into the webpage's Document Object Model (DOM) without proper sanitization or encoding. Specifically, the input is directly written into a span tag, allowing for the injection and execution of arbitrary JavaScript code. The application takes the user-supplied value from the search parameter and uses JavaScript to dynamically update the content of a HTML element (innerHTML()). Because the application does not validate or encode this input, an attacker can craft a malicious payload containing JavaScript code. When a victim views the manipulated page, the embedded script executes in the context of the victim's browser session.

IMPACT

* Attackers can execute any JavaScript code in the context of the victim's session.
* Modify page content, deface website, or create fake login forms.
* Steal session cookies and tokens.

RECOMMENDATIONS

* **Input Validation**: Implement strict input validation on both client and server sides
* **Output Encoding**: Properly encode user-controlled data before writing to the DOM
* **Content Security Policy**: Implement CSP headers to restrict script execution
* **HTTPOnly Cookies**: Mark session cookies as HTTPOnly to prevent client-side access
* **Sanitization**: Use trusted libraries like DOMPurify to sanitize HTML content

Steps to Reproduce

* use this payload on the search bar: <iframe src="javascript:alert(document.cookie)" allow=’autoplay’></A screenshot of a computer

  AI-generated content may be incorrect.

Figure :

Figure 3: payload used: **<iframe src="javascript:alert(document.cookie)" allow=’autoplay’></iframe>**

## Exposed Prometheus Metrics Endpoint

|  |  |
| --- | --- |
| Severity | High |
| CVSS | [CVSS 7.5](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N) |
| Affected Assets | http://localhost:3000/Metrics |
| Reference | [Prometheus Security](https://prometheus.io/docs/prometheus/latest/configuration/configuration/) |

**Description**

The application exposes an unauthenticated Prometheus metrics endpoint at /metrics that discloses extensive system and application performance data. This endpoint provides real-time monitoring information including system resource usage, application performance metrics, file upload statistics, and internal application task execution times. The exposed data reveals detailed insights into the application's infrastructure, performance characteristics, and operational patterns.

**IMPACT**

* **System Intelligence**: Attackers gain detailed knowledge of server infrastructure.
* **Performance Monitoring**: Real-time visibility into application health and resource usage.
* **Operational Patterns**: Understanding of file upload behaviors and error rates.
* **Reconnaissance Data**: Information about application tasks and startup processes.
* **Resource Analysis**: CPU, memory, and file descriptor usage patterns.

**RECOMMENDATIONS**

* **Access Restriction**: Implement authentication for the metrics endpoint
* **Network Controls**: Restrict access to internal IP ranges only
* **Endpoint Obfuscation**: Change the default /metrics path
* **Authentication**: Require valid credentials for metrics access
* **Network Segmentation**: Limit to internal monitoring networks
* **IP Whitelisting**: Restrict to known monitoring servers
* **TLS Encryption**: Ensure metrics are served over encrypted channels

**Steps to Reproduce**

* A screenshot of a computer

  AI-generated content may be incorrect.In a Linux shell, use the dirb tool with the quickhits.txt wordlist.

Figure : use the dirb tool with the quickhits.txt wordlist.

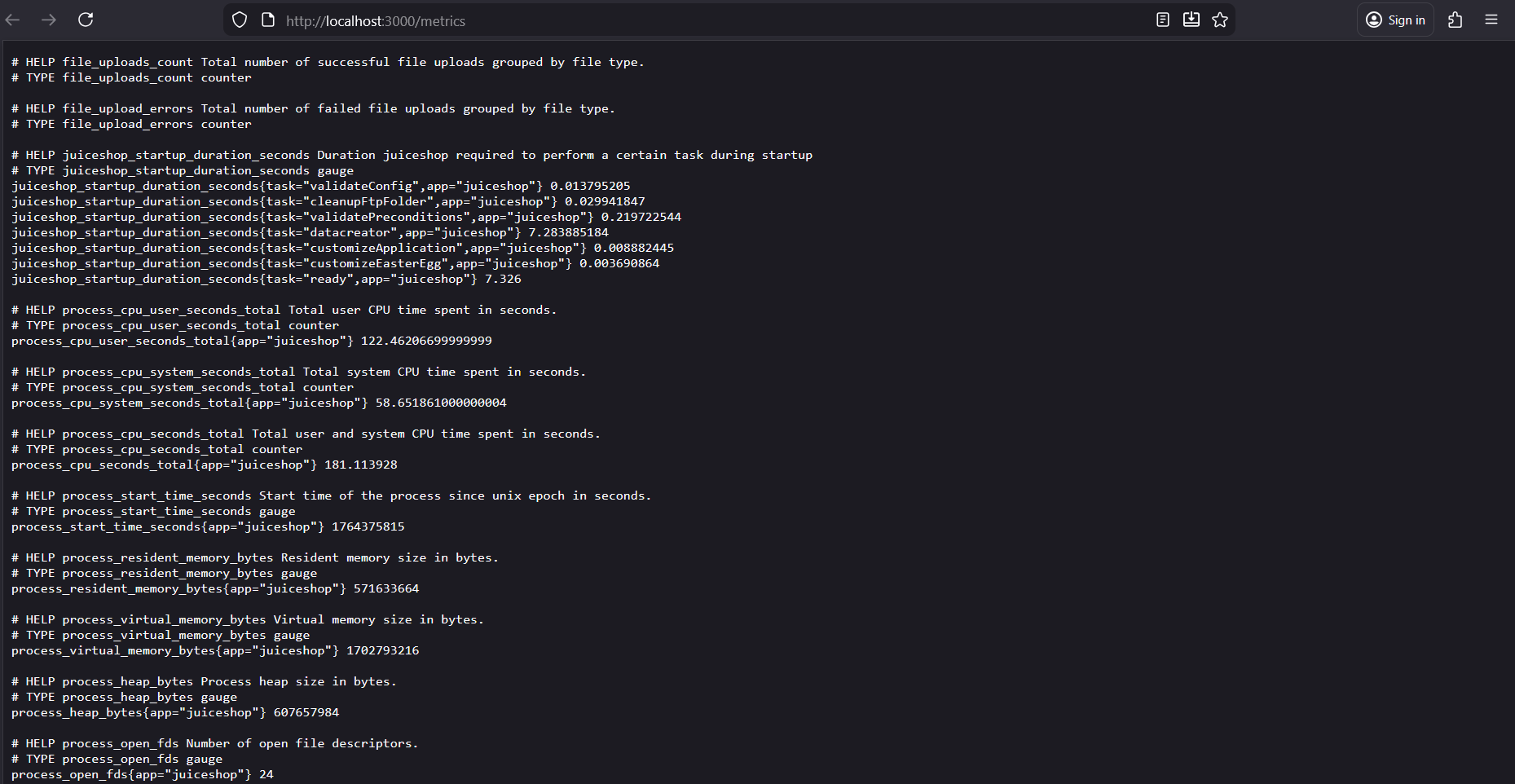
* Navigate to metrics endpoint

Figure : navigate to metrics endpoint

## bussiness logic bypass via client-side manipulation

|  |  |
| --- | --- |
| Severity | High |
| CVSS | [CVSS 8.2](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:H/A:N) |
| Affected Assets | http://localhost:3000/#/payment/deluxe |
| Reference | [CWE-602](https://cwe.mitre.org/data/definitions/602.html) |

Description

The application's payment system relies on client-side controls to enforce business logic, specifically disabling payment buttons and validating payment methods. Attackers can easily bypass these controls by modifying HTML attributes (removing disabled from buttons) and manipulating API request parameters (changing paymentMode from "wallet" to "paid"). This allows unauthorized access to premium features without proper payment verification to get deluxe membership, demonstrating a fundamental failure in server-side validation of business rules.

IMPACT

* **Authorization Bypass**: Circumvention of payment requirements for premium features.
* **Business Logic Manipulation**: Alteration of payment processing workflows.
* **Client-Side Trust**: Reliance on untrustworthy client-side controls.
* **API Parameter Tampering**: Unvalidated request parameter manipulation.

RECOMMENDATIONS

* Server-Side Validation: Implement payment verification on the server.
* Parameter Validation: Validate all payment-related parameters server-side.
* Status Verification: Confirm payment completion before granting access.

Steps to Reproduce

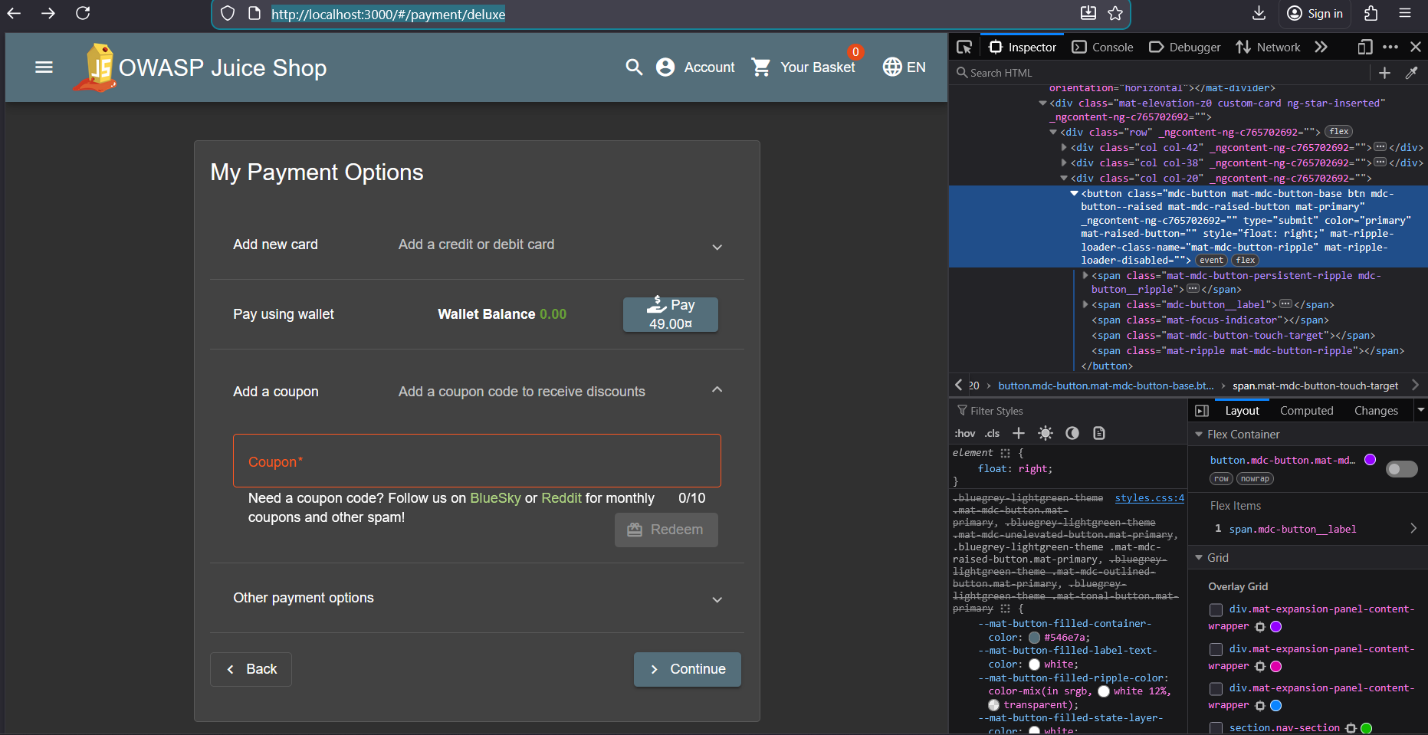
* Make an account then navigate to localhost:3000/#/payment/deluxe then active the pay button from the developer tool.

Figure 6: navigate to **localhost:3000**/#/payment/deluxe then active the pay button from the developer tool

* Intercept the request after clicking on pay and change the paymentMode value in the request from wallet into paid.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 7: change the **paymentMode** value from wallet into paid

## Unrestricted File Upload with Extension Manipulation

|  |  |
| --- | --- |
| Severity | High |
| CVSS | [CVSS 7.5](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N) |
| Affected Assets | http://localhost:3000/#/complaint |
| Reference | [OWASP](https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload) |

Description

The application's file upload functionality in the complaint section relies on client-supplied filename parameters without proper server-side validation. Attackers can manipulate the filename parameter to remove or change file extensions, bypassing intended file type restrictions. The system accepts the modified filename without the .pdf extension and processes the upload successfully, demonstrating inadequate server-side file type verification.

IMPACT

* **File Type Bypass**: Circumvention of intended file type restrictions.
* **Malicious File Upload**: Potential to upload executable files disguised as documents.
* **Server Compromise**: Risk of web shell deployment and remote code execution.
* **Content Spoofing**: Upload of malicious content with misleading filenames.
* **Storage Abuse**: Consumption of server resources with unauthorized files.

RECOMMENDATIONS

* **Server-Side File Validation**: Implement MIME type and content verification.
* **Extension Enforcement**: Server-side file extension validation.
* **File Type Restrictions**: Whitelist acceptable file types.

Steps to Reproduce

* Intercept the request before sending the uploaded file
* POST /file-upload HTTP/1.1
* Host: localhost:3000
* User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:145.0) Gecko/20100101 Firefox/145.0
* Accept: \*/\*
* Accept-Language: en-US,en;q=0.5
* Accept-Encoding: gzip, deflate, br
* Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..no\_J3LRGilGOTgZuoerAWT2yI3Uh0bjUoi2C2gfmKTjF-jNm0Pwwv0712PVXaKXxspUwgGWIHQNuiAYeQeCYUCclTpTxVtjNe1H\_THq74c61olP21l\_Fcgnaz1sPaYP83dvbZzCtNy7lL-oLNPvssLcPhau7nqeqaSTVfJEyjug
* Content-Type: multipart/form-data; boundary=----geckoformboundary575bce179241c8899f2e44973891ffec
* Content-Length: 98405
* Origin: http://localhost:3000
* Connection: keep-alive
* Referer: http://localhost:3000/
* Cookie: language=en; welcomebanner\_status=dismiss; continueCode=2alPzXRBnWb5dv1tMcQIyflhxXIzBtOaUrzFBZT4pTVEuaZAoqDMOLmvJ194; cookieconsent\_status=dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..no\_J3LRGilGOTgZuoerAWT2yI3Uh0bjUoi2C2gfmKTjF-jNm0Pwwv0712PVXaKXxspUwgGWIHQNuiAYeQeCYUCclTpTxVtjNe1H\_THq74c61olP21l\_Fcgnaz1sPaYP83dvbZzCtNy7lL-oLNPvssLcPhau7nqeqaSTVfJEyjug
* Sec-Fetch-Dest: empty
* Sec-Fetch-Mode: cors
* Sec-Fetch-Site: same-origin
* Priority: u=0
* ------geckoformboundary575bce179241c8899f2e44973891ffec
* Content-Disposition: form-data; name="file"; filename="documentation.pdf"
* Content-Type: application/pdf
* Manipulate the value of the filename parameter by deleting the .pdf extension
* POST /file-upload HTTP/1.1
* Host: localhost:3000
* User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:145.0) Gecko/20100101 Firefox/145.0
* Accept: \*/\*
* Accept-Language: en-US,en;q=0.5
* Accept-Encoding: gzip, deflate, br
* Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..no\_J3LRGilGOTgZuoerAWT2yI3Uh0bjUoi2C2gfmKTjF-jNm0Pwwv0712PVXaKXxspUwgGWIHQNuiAYeQeCYUCclTpTxVtjNe1H\_THq74c61olP21l\_Fcgnaz1sPaYP83dvbZzCtNy7lL-oLNPvssLcPhau7nqeqaSTVfJEyjug
* Content-Type: multipart/form-data; boundary=----geckoformboundary575bce179241c8899f2e44973891ffec
* Content-Length: 98401
* Origin: http://localhost:3000
* Connection: keep-alive
* Referer: http://localhost:3000/
* Cookie: language=en; welcomebanner\_status=dismiss; continueCode=2alPzXRBnWb5dv1tMcQIyflhxXIzBtOaUrzFBZT4pTVEuaZAoqDMOLmvJ194; cookieconsent\_status=dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9..no\_J3LRGilGOTgZuoerAWT2yI3Uh0bjUoi2C2gfmKTjF-jNm0Pwwv0712PVXaKXxspUwgGWIHQNuiAYeQeCYUCclTpTxVtjNe1H\_THq74c61olP21l\_Fcgnaz1sPaYP83dvbZzCtNy7lL-oLNPvssLcPhau7nqeqaSTVfJEyjug
* Sec-Fetch-Dest: empty
* Sec-Fetch-Mode: cors
* Sec-Fetch-Site: same-origin
* Priority: u=0
* ------geckoformboundary575bce179241c8899f2e44973891ffec
* Content-Disposition: form-data; name="file"; filename="documentation"
* Content-Type: application/pdf

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8: successfully uploading the file without its extension

## Broken Authentication

|  |  |
| --- | --- |
| Severity | High |
| CVSS | [CVSS 8.2](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N) |
| Affected Assets | <http://localhost:3000/#/login> |
| Reference | [Broken authentication](https://owasp.org/www-project-top-ten/2017/A2_2017-Broken_Authentication) |

Description

The Juice Shop application does not implement sufficient protections against password brute‑force or credential stuffing attacks. After obtaining a valid user email address, such as the administrator’s address, an attacker can automate repeated login attempts using common or guessable passwords. The login endpoint imposes no rate limiting, account lockout, CAPTCHA, or delays between attempts. Because authentication requests can be made without restriction, an attacker can systematically try multiple passwords until the correct one is identified, leading to unauthorized access.

IMPACT

* Unauthorized access to user or administrative accounts can be gained.
* Sensitive information and administrative functionality become exposed.
* Compromise of an administrator account may lead to some control over the application, such as viewing users’ emails, details and deleting comments.

RECOMMENDATIONS

* Implement account lockout policies after several failed login attempts.
* Use rate limiting and throttling to prevent automated login attempts.
* Add mandatory multi-factor authentication (MFA) to protect high-privilege accounts.
* Monitor login attempts for unusual patterns and alerts on suspicious activity.
* Educate users to avoid common or weak passwords.

Steps to Reproduce

* Navigate to <http://localhost:3000/#/login>
* Enter the admin’s email address **admin@juice-sh.op** and any random password such as **0000**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 9: Entering admin's username and a random password

* Turn the interception on in the burp suite and click on **login** to **capture the HTTP request**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 10: capturing HTTP request

* Send the captured post request to **the intruder**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 11 The request in the intruder

* Add **position** to the password then use a list of commonly used password such as from <https://nordpass.com/most-common-passwords-list/> and copy them into the **payload**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 12 Adding position and payload

* Start the attack and look for the password with the **status code 200**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 13 Finding the admin’s password

* Navigate back to <http://localhost:3000/#/login> and enter the admin’s email **admin@juice-sh.op** with the password **admin123** then login successfully

A screenshot of a computer

AI-generated content may be incorrect.

Figure 14 Logged in the admin's account

## Information Disclosure via Client-Side Debugging

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 5.3](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N) |
| Affected Assets | http://localhost:3000/#/ |
| Reference | [OWASP](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure) |

**Description**

The application exposes sensitive client-side JavaScript files (main.js) containing application logic and endpoint references through browser developer tools. This exposure allowed the discovery of hidden pages such as score-board, administration and all paths of this application that should not be publicly accessible. The debug information and unobfuscated source code remain accessible in production environments, enabling attackers to map application structure, identify hidden functionalities, and discover potential attack vectors.

**IMPACT**

* **Application Mapping:** Attackers can discover hidden endpoints and administrative interfaces.
* **Business Logic Discovery**: Critical application logic and API endpoints are visible.
* **Attack Surface Expansion**: dentification of additional targets for exploitation (score-board page).
* **Reconnaissance Aid**: Foundation for further, more severe attacks.
* **Information Gathering**: Fuels reconnaissance phase for more sophisticated attacks.

**RECOMMENDATIONS**

* **Build Process Security**: Separate development and production buildings.
* **Security Headers**: Implement Content Security Policy (CSP).
* **Regular Audits**: Conduct source code exposure assessments.
* **Developer Training**: Secure coding practices for production deployments.

**Steps to Reproduce**

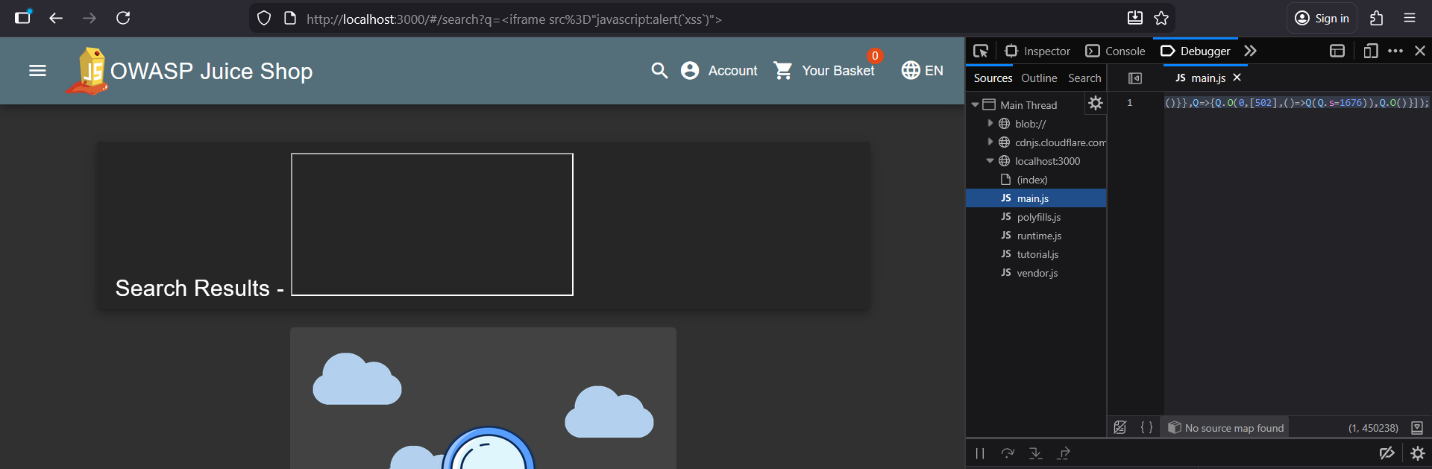
* open developer tool and navigate to debugger

Figure 15: as shown in the screenshot, the main.js file is shown in debugger

* A screenshot of a computer

  AI-generated content may be incorrect.use online JavaScript beautifier to understand the code

Figure 16: use online JavaScript beautifier to understand the code

## Improper Access Control in Chatbot Functionality

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 5.3](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N) |
| Affected Assets | http://localhost:3000/#/chatbot |
| Reference | [A01:2021 – Broken Access Control](https://owasp.org/Top10/A01_2021-Broken_Access_Control/) |

**Description**

The application's chatbot functionality contains improper access control mechanisms that allow unauthorized users to escalate privileges through social engineering techniques. By claiming to be "the manager" or "the CFO," attackers can bypass intended authorization checks and extract sensitive business information (coupon codes) that should be restricted to authorized personnel only. The chatbot fails to properly verify user identity and implements weak role-based access control.

**IMPACT**

* **Authorization Bypass**: Circumvention of intended access controls through simple text-based claims
* **Information Disclosure**: Unauthorized access to discount codes and promotional materials
* **Business Logic Flaw**: Failure to implement proper identity verification for privileged actions
* **Privilege Escalation**: Ability to claim administrative roles without authentication

**RECOMMENDATIONS**

* **Identity Verification**: Integrate chatbot with existing authentication system
* **Role-Based Controls**: Implement proper RBAC for sensitive operations
* **Session Management**: Maintain user context and permissions throughout chat
* **Input Validation**: Sanitize and validate role claims against authorized users
* **AI/ML Security**: Train chatbot to detect social engineering patterns
* **Multi-factor Verification**: Require additional verification for sensitive actions
* **Regular Testing**: Conduct social engineering tests on chatbot functionality
* **Incident Response**: Develop procedures for coupon code compromise

**Steps to Reproduce**

* **A screenshot of a chat

  AI-generated content may be incorrect.**claim that you are the CFO of the company and insist of that

Figure 17: insist that you are the CFO

## Repetitve Registration

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 5.3](https://www.first.org/cvss/calculator/3-1#CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N) |
| Affected Assets | register |
| Reference | [Input Validation - OWASP Cheat Sheet Series](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html) |

**Description**

The registration workflow does not correctly validate the “password” and “repeatPassword” fields on the server side. In particular, submitting a registration request with mismatched or manipulated password fields may still succeed under certain conditions. This allows the creation of a new user account with an already existing email address. i.e., the same email can be registered again.

**IMPACT**

* Multiple accounts can be registered with the same email identity. This undermines uniqueness and identity integrity, potentially enabling account duplication, confusion over user identity, misuse of application features under duplicate accounts, or skewing of user‑based metrics, especially relevant in real-world systems relying on unique user accounts for authorization, logging, or rate-limiting.

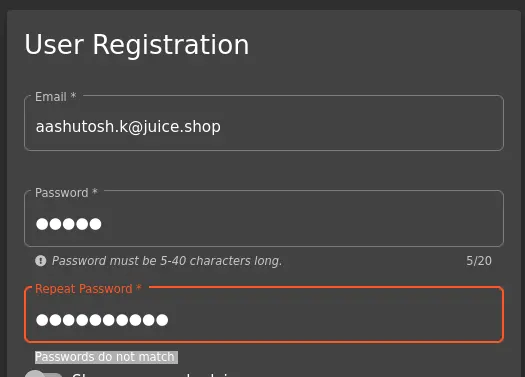
**RECOMMENDATIONS**

* Enforce **server‑side validation** for both password and confirm‑password fields. Ensure that the two values must match and meet minimum password strength requirements (e.g. length, complexity).
* Prevent registration when the provided email already exists in the database (unique constraint on email).
* Avoid relying solely on client-side validation.
* Log and monitor registration attempts with failed or suspicious inputs.

**Steps to Reproduce**

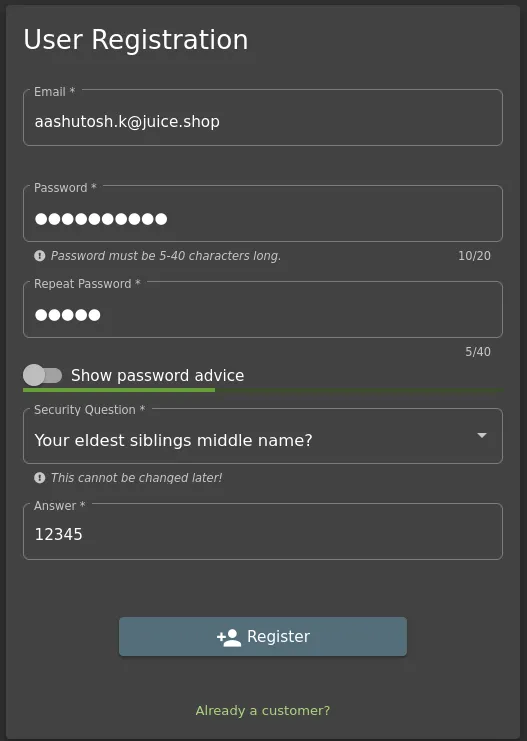
* Entered password as “12345” and repeat password as “123456” it says “Passwords do not match”

Figure 18: Entered password as “12345” and repeat password as “123456” it says “Passwords do not match”



* Then entered “1234512345” in password field and it didn’t show any kind of error.

Figure 19 Then entered “1234512345” in password field and it didn’t show any kind of error.



## UNVALIDATED REdirects

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 6.5](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:L/RL:O/CR:L/IR:L/AR:L/MAC:H) |
| Affected Assets | /redirect?to=https://blockchain.info/address/1AbKfgvw9psQ41NbLi8kufDQTezwG8DRZm |
| Reference | [Unvalidated Redirects and Forwards - OWASP Cheat Sheet Series](https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html) |

**Description**

The application accepts a user‑controlled URL parameter and performs a redirect without validating or restricting the destination. Because the redirect target is not checked against an allowlist, any external URL can be supplied, enabling forced navigation to arbitrary sites.

**IMPACT**

* Users may be redirected to malicious or misleading external pages, facilitating phishing, credential harvesting, or malware delivery. In real deployments, unvalidated redirects commonly serve as a steppingstone in social‑engineering attacks.

**RECOMMENDATIONS**

* Implement strict validation of redirect targets using an allowlist of permitted internal paths. Reject all external URLs and sanitize user‑supplied input before performing navigation.

**Steps to Reproduce**

* In the Developer Tools > Memory > main.js   
  search for the term “redirect”

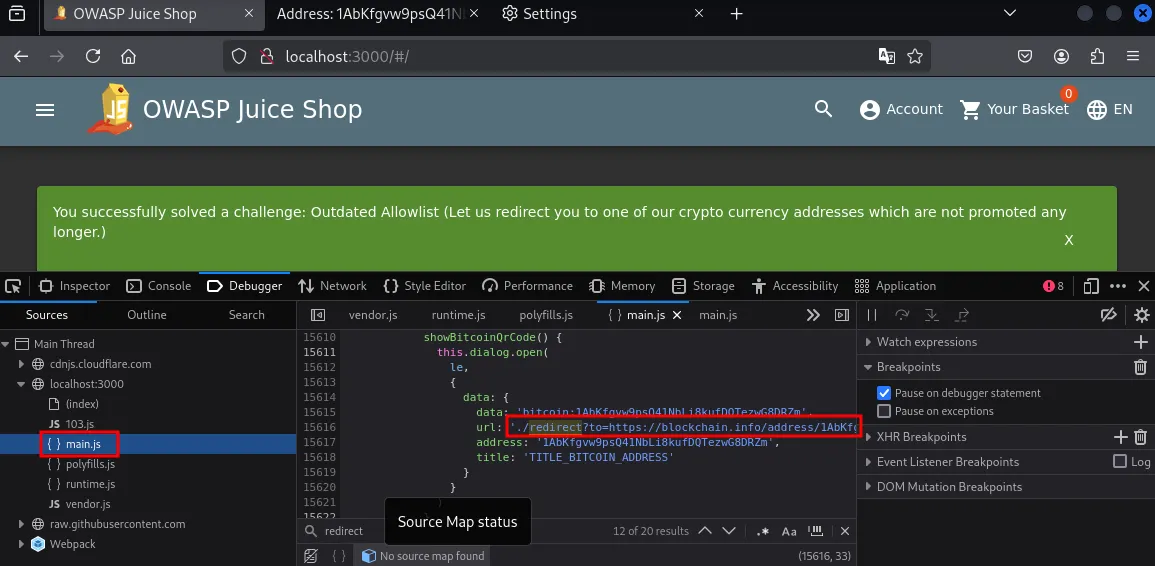


Figure 20: Developer Tools > Memory > main.js

* Copy the URL containing the redirect and paste in the URL path

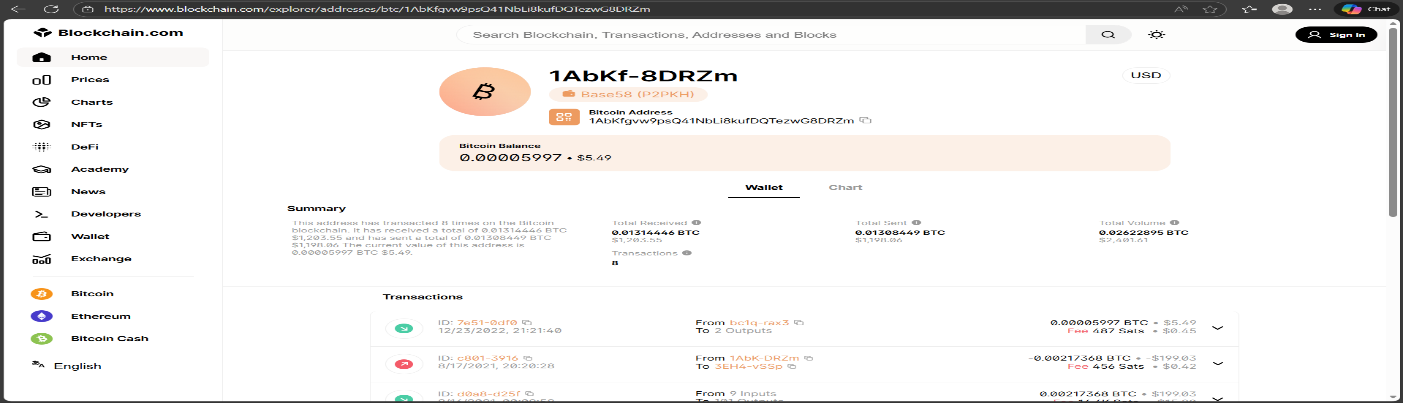


Figure 21: Copy the URL containing the redirect and paste in the URL path

## Sensitive Data exposure

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 6.5](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N) |
| Affected Assets | main.js |
| Reference | [Sensitive Data Exposure](https://owasp.org/www-project-top-ten/2017/A3_2017-Sensitive_Data_Exposure) |

Description

The Juice Shop application contains hardcoded credentials for a testing account within its client-side JavaScript **(main.js)**. These credentials are embedded directly in the source code and remain valid for authentication. Because client-side JavaScript is fully accessible to end users through browser developer tools, an attacker can easily extract these credentials and log in without authorization. This issue exposes a functional user account and represents a significant security risk caused by improper handling of sensitive information.

IMPACT

* Unauthorized users can log in using the exposed testing account credentials.
* Attackers may view or modify data associated with the compromised account.
* Exposure of internal testing data increases the risk of further exploitation and misuse.

RECOMMENDATIONS

* Remove all hardcoded credentials from client-side code.
* Disable or delete unused testing accounts in production environments.
* Store sensitive information securely on the server side and never within JavaScript assets.
* Implement regular code reviews to ensure credentials and secrets are not exposed in source files.

Steps to Reproduce

* Navigate to <http://localhost:3000/#/login>
* Open **developer tools** then click on **main.js**

A screenshot of a computer

AI-generated content may be incorrect.

Figure 22: main.js in the developer tools

* Search for keyword **“password”** and look for the testing credentials

A screenshot of a computer

AI-generated content may be incorrect.

Figure 23: Finding the email and password

* Enter the found credentials of email **“testing@juice-sh.op”** and password **“IamUsedForTesting”** in the login

A screenshot of a computer

AI-generated content may be incorrect.

Figure 24: putting the found credentials in the login form

* Click on the **login** button to login with the testing account

A screenshot of a computer

AI-generated content may be incorrect.

Figure 25: Logged in with the testing account

## Insecure Direct Object Reference

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 5.4](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N) |
| Affected Assets | <http://localhost:3000/#/basket> |
| Reference | [Insecure Direct Object Reference](https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html) |

Description

The Juice Shop application does not properly enforce authorization checks on user basket identifiers. The **basket ID (bid)** is stored in the browser’s session storage and is fully client-controlled. When a user modifies this value, the application does not verify whether the basket belongs to them. As a result, an attacker can change their own **bid** to another **valid** **basket ID**, allowing them to view and modify the contents of that basket. While the email displayed remains the attacker’s own, this issue allows unauthorized access and modification of other users’ basket contents, demonstrating a failure to enforce proper access control on server-side resources.

IMPACT

* Attackers can view and modify the **basket contents** of other users.
* Attackers can modify or delete items in another user’s basket, affecting **the integrity** of that resource.

RECOMMENDATIONS

* Do not trust client-side identifiers such as basket ids.
* Implement server-side authorization checks to ensure each request belongs to the authenticated user.
* Avoid storing sensitive or critical data like basket ids in session storage.
* Use securely generated, non-enumerable identifiers tied to user sessions.
* Perform regular access control tests on all endpoints handling user data.

Steps to Reproduce

* Login using your account then navigate to the basket page <http://localhost:3000/#/basket>

A screenshot of a computer

AI-generated content may be incorrect.

Figure 26: navigate to the basket page <http://localhost:3000/#/basket>

* Open the developer tools and go to **session storage** located in application section

A screenshot of a computer

AI-generated content may be incorrect.

Figure 27: My basket id in storage session

* Change the **bid** value to 1

A screenshot of a computer

AI-generated content may be incorrect.

Figure 28: Changing the basket id to another value

* Refresh the page

A screenshot of a computer

AI-generated content may be incorrect.

Figure 29: content of a basket of another user is displayed

## Empty user registration (Improper Input Validation)

|  |  |
| --- | --- |
| Severity | Medium |
| CVSS | [CVSS 6.5](https://www.first.org/cvss/calculator/3-0#CVSS:3.0/AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N) |
| Affected Assets | /register |
| Reference | [CWE-20: Improper Input Validation](https://cwe.mitre.org/data/definitions/20.html) |

**Description**

The application fails to properly validate input during the user registration process. It is possible to register a new user account with an empty email address and an empty password by manipulating the API request. This allows the creation of "ghost" accounts that violate database integrity constraints and standard authentication requirements.

**IMPACT**

* Attackers can create unlimited anonymous “ghost” accounts.
* Creation of malformed user records pollutes the database.
* Potentially allows login with empty credentials if the login logic does not handle null/empty strings correctly.
* Attackers could flood the database with invalid accounts.

**RECOMMENDATIONS**

* Enforce strict server-side validation for all fields.
* Require minimum password complexity.
* Reject null, empty, or malformed data.
* Implement rate limiting and CAPTCHA on registration

**steps to Reproduce**

* Navigate to the User Registration page (/#/register).and Fill in the form with dummy data.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 30: Navigate to the User Registration page (/#/register).and Fill in the form with dummy data.

* Intercept the registration request using a proxy tool (e.g., Burp Suite), Modify the JSON body to set "email" and "password" to empty strings.

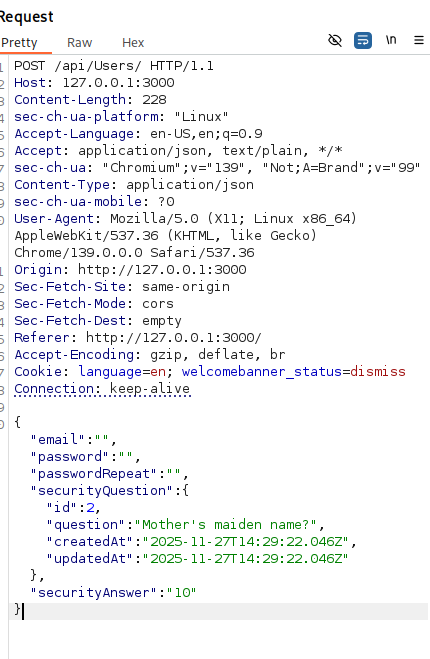


Figure 31: Intercept the registration request using a proxy tool (e.g., Burp Suite), Modify the JSON body to set "email" and "password" to empty strings.

* Forward the request, and observe the server responds with 201 Created (Success).

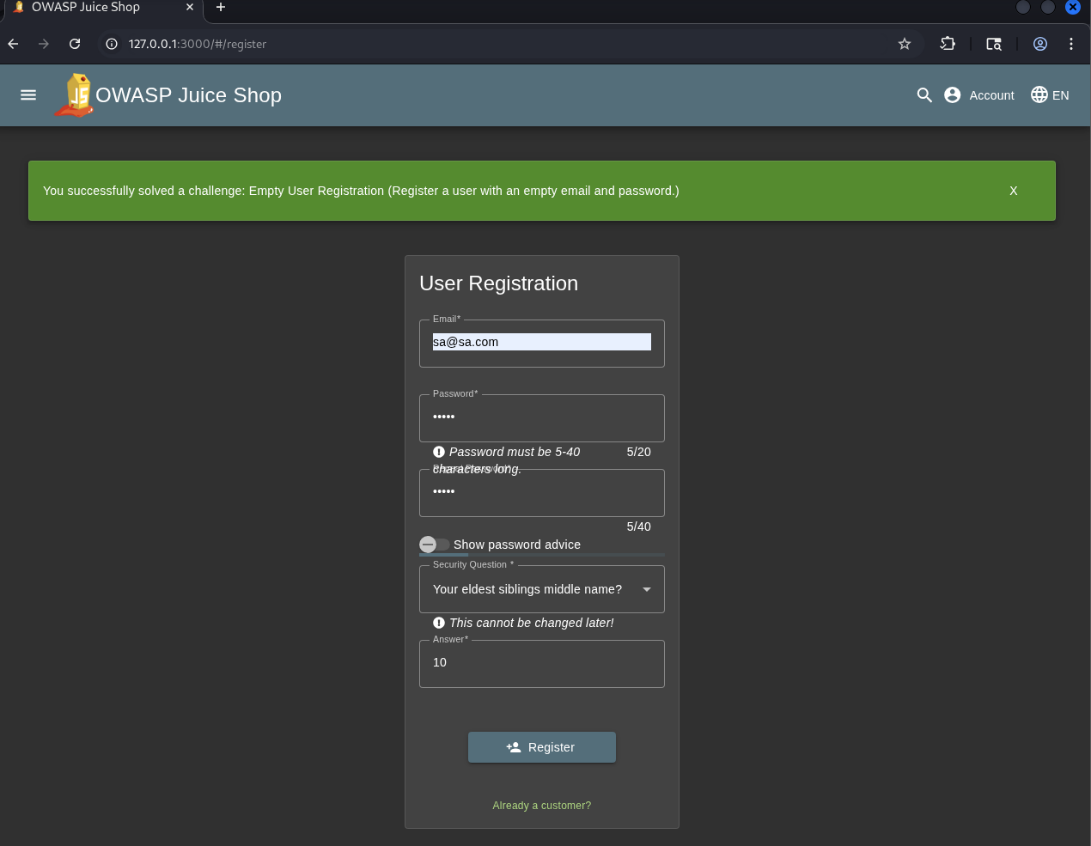


Figure 32: Forward the request, and observe the server responds with 201 Created (Success).

## Forced Zero Rating

|  |  |
| --- | --- |
| Severity | Low |
| CVSS | [CVSS 3.7](https://www.first.org/cvss/calculator/3-1#CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N) |
| Affected Assets | / |
| Reference | [Input Validation - OWASP Cheat Sheet Series](https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html) |

**Description**

The product review feature does not properly validate rating input values. By manipulating the request client‑side, it is possible to submit a review with a rating of **zero stars**, even though the UI restricts scores to 1–5. The server accepts the out‑of‑range value due to missing server‑side validation.

**IMPACT**

* Submitting invalid or unexpected rating values can undermine data integrity in product reviews. In real systems, improper validation of rating or scoring fields can distort analytics, damage product reputation, or enable automated manipulation of scoring systems.

**RECOMMENDATIONS**

* Enforce server‑side validation for review ratings, ensuring that only values within the expected range are accepted. Reject or normalize any out‑of‑range inputs.

**Steps to Reproduce**

* Go to the menu and open customer feedback, in the feedback, the least you can do is 1 star.

****

Figure 33: at least 1 star for rating in customer feedback page

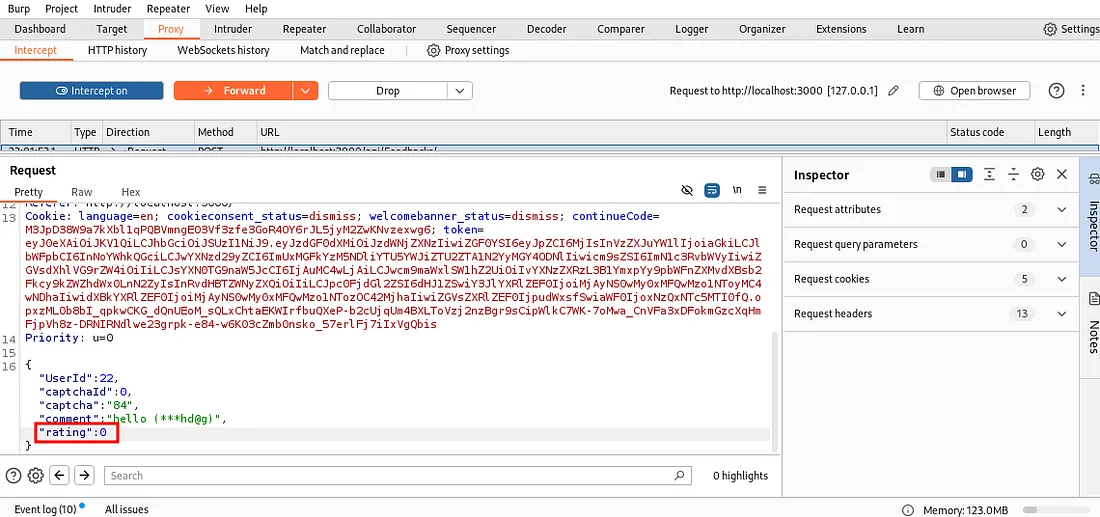
* Intercept the request with burp and change the rating number as you wish.

Figure 34: change the rating number as you wish.