OWASP Juice Shop
Penetration Test Report

Version	**1.0**
Author	**Menna Alaa Aldeen 10006535** **Miriam Ibrahim 10005930** **Mayar Mohammad 10005684**
Issue Date	**14/12/2024**

Table of Contents

1.0 OWASP Juice shop Penetration Test Report

1.1 Introduction

Subject of this document is a summary of penetration tests performed against web applications owned by OWASP Juice shop company. Test was conducted according to rules of engagement defined and approved at the beginning by both parties – customer and contractor. Black-box pentesting assignment was requested.

Black-box penetration test classification means that penetration tester has no internal knowledge about target system architecture. He/she can use information commonly available on the Internet. More data can be collected during the reconnaissance phase based on observation of target system behavior. Black-box penetration test results give overview of vulnerabilities exploitable from outside the company network. It shows how unauthenticated actors can take advantage of weaknesses existing in tested web applications.

Time frame defined:

Penetration test start: 27/11/2024

Penetration test end: 15/12/2024.

1.2 Scope

To perform a Black Box Web Application Penetration Test against the web applications of the organization named OWASP Juice shop

This is what the client organization defined as scope of the tests:

Dedicated Web Server: 127.0.0.1

Domain: localhost:3000

Subdomains: all subdomains

2.0 – EXECUTIVE SUMMARY

Conducted penetration test uncovered several security weaknesses present in web applications owned by OWASP Juice Shop

When performing the penetration test, there were several alarming vulnerabilities that were identified on OWASP Juice Shop networks. When performing the attacks, I was able to gain access to multiple sensitive systems, primarily due to outdated patches and poor security configurations. During the testing, I had unauthorized administrative access to user management portal. These systems as well as a brief description:

- Subdomain Found: admin.juiceshop.localhost
- X High, X Med, X Low issue has been identified.

2.1 – Recommendations

I recommend patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered at a later date.

3.0 – RISK ASSESSMENT

3.1 Likelihood

The likelihood is a measurement of the capacity to carry out an attack. The factor will be the difficulty or skill required to carry out the attach.

Risk	Description
Critical	An attacker is near-certain to carry out the threat event
High	An untrained user could exploit the vulnerability. The vulnerability is obvious or easily accessed
Medium	The vulnerability required some hacking knowledge to carry out the attack
Low	The vulnerability required significant time, skill , access and other resource to carry out the attack

3.2 Impact

The impact is a measurement of the adverse effect carrying out an attack would have on the organization.

Risk	Description
Critical	An attack would cause catastrophic or severe effect on operation, asset or other organization
High	An attack would severely degrade mission capability. The attack may result in damage to asset(data exposure)
Medium	An attack would degrade the mission capability. An attack would allow for primary function to application to resume, but at reduced effectiveness
Low	An attack would degrade mission capability in a limited capacity. The attack may result in marginal damage to assets

4.0 -- FINDINGS SUMMARY

Findings	Likelihood	Impact	Rating	Status
Login Admin (SQL Injection)	High	Critical	High	**Open**
Client-side XSS Protection	High	High	High	**Open**
Admin Registration (Mass assignment)	High	Critical	High	**Open**
Missing Encoding (Improper Input Validation)	High	Medium	Medium	**Open**
Password Strength (Broken Authentication)(brute force)	High	High	High	**Open**
View Basket (Broken Access Control)	High	High	High	**Open**
Attack Analysis for API-Only XSS	High	High	High	**Open**
Ephemeral Accountant (SQL-Injection)	High	High	High	**Open**
Forgotten Sales Backup (Sensitive Data Exposure)	High	High	High	**Open**
Repetitive Registration (Improper Input Validation)	High	Medium	Medium	**Open**
DOM XSS (Client-Side)	High	High	High	**Open**
Bonus Payload (Client-Side)	High	High	High	**Open**
Confidential Document (Server-Side)	High	High	High	**Open**
Upload File Size Larger Than 100 KB (Server-Side)	High	High	High	**Open**
Deprecated Interface (Server-Side)	Medium	Medium	Medium	**Open**
Weird Crypto (Server-Side)	High	High	High	**Open**
Access to Admin Section (Broken Access Control)	High	High	High	**Open**
Get Rid of All 5-Star Feedback (Broken Access Control)	High	High	High	**Open**
Reflected XSS (Client-Side)	High	High	High	**Open**

Bully Chatbot (Client-Side)	Medium	Medium	Medium	**Open**
Security Policy Disclosure (Client-Side)	Medium	Medium	Medium	**Open**
Login Jim Injection (injection, server)	High	High	Critical	**Open**
Database Schema (Injection,server)	High	High	Critical	**Open**
User Credentials Retrieval (Injection, Server)	High	Critical	Critical	**Open**
Error Handling Vulnerability (Security Misconfiguration, Server)	Medium	medium	High	**Open**
Outdated Whitelist (unvalidated Redirects , Server)	High	Medium	High	**Open**
Forged Feedback (Broken Access Control, Client)	Medium	Medium	Medium	**Open**
Forged Review (Broken Access Control, Client)	Medium	High	High	**Open**
Basket Manipulation (Broken Access Control , client)	High	High	Critical	**Open**
Empty User Registration (Improper Input validation , client)	Medium	Low	Medium	**Open**
Zero Star Feedback (Improper Input validation , Client)	Medium	Low	Low	**Open**

5.0 – VULNERABILITY & REMEDIATION REPORT

Penetration test finding classification, description and recommendations mentioned in the report are taken mostly from OWASP TOP 10 project documentation available on site: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

The OWASP TOP 10 is a list of definitions of web application vulnerabilities that pose the most significant security risks to organizations when exploited.

6.0 – INFORMATION GATHERING

The information gathering portion of a penetration test focuses on identifying the scope of the penetration test. I started the pentest with finding all subdomains.

I have used multiple tools to make sure that I haven't missed any domain.

Tools Used:

1) WFUZZ

2) Virustotal

3) Sublist3r

Above tools discovered multiple subdomains, the working one's are below:

5.0 High Findings

5.1. Title: SQL Injection in Login Page (server-side)

Rating: High

URL:/# /login

Description: A SQL Injection vulnerability was identified in the login page, allowing an attacker to bypass authentication and gain unauthorized access to the application.

Proof of Concept:

Step 1: Go to the /login page.

Step 2: Enter ' OR '1'='1 in both the username and password fields.

Step 3: Successfully log in as an admin without valid credentials.

Remediation Steps:

1.Use prepared statements and parameterized queries for database interactions.

2.Validate and sanitize all user inputs on the server side.

3.Implement robust error handling to avoid revealing database information.

5.2. Title: XSS Vulnerability in Registration Page (client-side)

Rating: High

URL: /# /register

Description: A Cross-Site Scripting (XSS) vulnerability in the registration form allows attackers to inject malicious scripts, which are executed in the victim's browser.

Proof of Concept:

Step 1: Intercept a registration request using a proxy (e.g., Burp Suite).

Step 2: Replace the email field with an iframe containing malicious JavaScript: <iframe src="javascript:alert('XSS')"></iframe>.

Step 3: Submit the request and observe the popup message in the browser.

Remediation Steps:

1. Validate and sanitize all user inputs, especially in form fields.
2. Use Content Security Policy (CSP) headers to restrict script execution.
3. Implement proper encoding for all user inputs when rendering them on the page.

5.3. Title: Admin Registration - Mass Assignment Attack (server-side)

Rating: High

URL: /#/register

Description: The application allows unauthorized attribute manipulation via the registration API, enabling attackers to modify the role attribute to "admin," resulting in the creation of an admin account. This issue occurs due to improper input validation and lack of restrictions on assignable attributes.

Proof of Concept:

Step 1: Intercept the registration request using a proxy tool (e.g., Burp Suite).

Step 2: Inspect the response, which includes an attribute called "role" with the value "customer."

Step 3: Modify the request and manually add the "role" attribute with the value "admin" in the request payload.

Step 4: Submit the modified request, successfully creating an admin account.

Remediation Steps:

1.Implement proper input validation and ensure that only allowed attributes are accepted in the request body.

2. Use a whitelist approach to explicitly define which attributes are allowed to be assigned.

3.Apply server-side checks to ensure that sensitive attributes (e.g., "role") cannot be modified by end-users through the API.

5.4. Title: Password Strength (Broken Authentication)(brute force)(server-side)

Rating: High

URL: /#/login

Description: The web application allows weak passwords such as admin123 for highly privileged accounts like admin@juice-sh.op. Additionally, there are no brute-force prevention mechanisms, such as rate-limiting, account lockouts, or CAPTCHAs. This enables attackers to perform credential-stuffing or brute-force attacks, compromising sensitive user accounts and the overall application security.

Proof of Concept:

Step 1: Identify the admin email: admin@juice-sh.op.

Step 2: Intercept the login request and send it to a tool like Burp Suite Intruder. Load a list of commonly used passwords.

Step 3: Send the payload and analyze the responses. Observe the response status code 200 OK for the correct password (admin123) and 401 Unauthorized for incorrect attempts.

Remediation Steps:

1.Enforce strong password policies, requiring a mix of uppercase, lowercase, special characters, and minimum length.

2. Implement rate-limiting for login attempts to slow down brute-force attacks

3. Introduce account lockouts or CAPTCHA after a certain number of failed login attempts.

5.5. Title: View Basket (Broken Access Control)(IDOR)(Client-side)

Rating: High

URL: /#/basket

Description: The application does not validate whether the currently logged-in user is authorized to access the basket associated with a specific bid (basket ID). By modifying the bid value in session storage, an attacker can view the contents of other users' baskets, leading to a breach of sensitive user data and a violation of privacy.

Proof of Concept:

Step 1: Log in to the application and navigate to the basket page at http://localhost:3000/#/basket.

Step 2: Open the browser's developer tools, go to the Storage tab, and locate the bid (basket ID) in session storage

Step 3: Modify the bid value to a different (likely sequential) ID and reload the page. Observe that the contents of another user's basket are displayed.

Remediation Steps:

1. Implement Server-Side Authorization Checks:

Ensure that the server validates the ownership of the bid before returning basket data. Only allow access if the authenticated user is authorized to view the specific basket ID.

2. Use Session Tokens Instead of Client-Side Identifiers:

Avoid exposing sensitive identifiers like bid directly in session storage. Instead, use session tokens tied to the user's account to fetch basket data securely on the server side.

5.6. Title: API-Only XSS (Cross-Site Scripting)(client-side)

Rating: High

URL: http://localhost:3000/api/products

Description: The API of the web application was found to be vulnerable to a Cross-Site Scripting (XSS) attack. This vulnerability allows an attacker to inject malicious JavaScript into a product-related API endpoint, which is then executed by the client when the response is rendered. The vulnerability arises due to improper validation and lack of input sanitization in API requests. By modifying the request, an attacker can inject harmful scripts that would execute on the user's browser, potentially leading to session hijacking, data theft, and other client-side attacks.

Proof of Concept:

Step 1: Intercept the API request using a tool like Burp Suite or a browser's developer tools to identify the API endpoint. In this case, the endpoint used is /api/products.

Step 2: Modify the request method from GET to PUT to test if the API allows for data modification. Notice that the server returns a 200 OK response, indicating that the method change is accepted.

Step 3: Add a malicious script payload into the JSON request. Example:

{ "name": "Product Name", "description": "<script>alert('XSS Attack!');</script>" }

Step 4: Send the request with a valid authentication token and specify the Content-Type as application/json. Upon receiving the response and rendering the data on the client side, the script gets executed in the user's browser, triggering the XSS payload.

Remediation Steps:

1.Input Validation:

Ensure that all user inputs, especially in API requests, are properly validated and sanitized. Any user-provided data (like product names or descriptions) should be treated as untrusted input.

5.7. Title: Ephemeral Accountant (SQL Injection)(server-side)

Rating: High

URL: /#/login

Description: A SQL injection vulnerability was discovered in the login functionality of the OWASP Juice Shop. The attack involves injecting a single quote (') into the email field during the login request, causing the backend SQL query to break and return unauthorized data. By exploiting this, an attacker can bypass authentication and log in with arbitrary credentials.

Proof of Concept:

Step 1: Intercept the login request using a tool like Burp Suite or OWASP ZAP.

Step 2: Modify the "email" field by inserting ' UNION SELECT * FROM (SELECT 10 AS 'id' , 'username' AS 'username', 'accOunt4nt@juice-sh.op' AS 'email', 'asdfasdf' AS 'password', 'accounting' AS 'role', '' AS 'deluxeToken', '127.0.0.1' AS 'lastLoginIp', 'default.svg' AS 'profileImage','' AS 'totpSecret' , 1 AS 'isActive', '2020-08-30 11:12:13.456 +00:00' AS 'createdAt', '2020-08-30 11:12:13.456 +00:00' AS 'updatedAt', NULL AS 'deletedAt')-- .

Step 3: Send the modified request, and the system logs you in with the injected user credentials (accOunt4nt@juice-sh.op).

Remediation Steps:

1. Implement parameterized queries (prepared statements) to prevent SQL injection attacks by separating the data from the SQL query.
2. Validate and sanitize all user inputs on both the client and server sides.

5.8. Title: Forgotten Sales Backup (Sensitive Data Exposure)(server-side)

Rating: High

URL: http://localhost:3000/ftp/coupons_2013.md.bak.

Description:

Proof of Concept:

Step 1: Find the backup file (coupons.bak) on the FTP server but notice that downloading it directly results in a "Bad Request" error.

Step 2: modify url to http://localhost:3000/ftp/coupons_2013.md.bak.%2500.md

Step 3: backup file is downloaded

Remediation Steps:

1.Enforce access controls to prevent unauthorized access to sensitive files like backups.

2.Avoid storing sensitive data in publicly accessible locations, such as FTP servers, and ensure that backups are securely stored.

3.Validate input to ensure that file extensions and paths are correctly sanitized before processing user requests.

5.9. Title: DOM XSS (Client-Side)

Rating: High

URL: http://localhost:3000 /#/search

Description: A DOM-based Cross-Site Scripting (XSS) vulnerability was found in the search bar, allowing malicious scripts to be executed in the user's browser. This can lead to session hijacking or theft of sensitive information.

Proof of Concept:

Step 1: Navigate to the search bar on the home page.

Step 2: Insert the payload `<iframe src="javascript:alert('xss')">`.

Step 3: Observe the alert pop-up indicating script execution.

Remediation Steps:

1. Ensure proper input sanitization and encoding of all user inputs.
2. Implement a Content Security Policy (CSP) to block unauthorized script execution.

5.91. Title: Bonus Payload (Client-Side)

Rating: High

URL: http://localhost:3000 /#/search

Description: An attacker can embed a malicious iframe payload in the search input, causing unwanted audio playback or disrupting the user experience when the page is rendered.

Proof of Concept:

Step 1: Insert the payload:

<iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?url=https%3A//api.soundcloud.com/tracks/771984076&color=%23ff5500&auto_play=true&hide_related=false&show_comments=true&show_user=true&show_reposts=false&show_teaser=true"></iframe>

Step 2: Observe the iframe rendering and audio playback.

Remediation Steps:

1. Restrict input types to prevent HTML or iframe embedding.
2. Use a CSP to control iframe usage.

5.92. Title: Confidential Document (Server-Side)

Rating: High

URL: http://localhost:3000/ftp/legal.md

Description: A lack of proper access control allows unauthorized access to sensitive documents stored on the server. An attacker can enumerate files and retrieve confidential documents by manipulating the URL.

Proof of Concept:

Step 1: Use Burp Suite to intercept requests to the /ftp/legal.md endpoint.

Step 2: Delete legal.md from the URL and resend the request.

Step 3: In the response, identify a directory containing multiple files. Replace `legal.md` with acquisitions.md to retrieve another document.

Remediation Steps:

1. Implement strict access controls on sensitive directories.
2. Restrict directory listing and enforce authentication for file access.

5.93. Title: Upload File Size Larger Than 100 KB (Server-Side)

Rating: High

URL: http://localhost:3000/#/file-upload

Description: The file upload feature lacks proper validation, allowing an attacker to bypass file size restrictions and upload larger-than-allowed files.

Proof of Concept:

Step 1: Upload a PDF file under 100 KB and intercept the request in Burp Suite.

Step 2: Replace the data in the request with a larger PDF file (e.g., over 100 KB).

Step 3: Resend the modified request to successfully upload the larger file.

Remediation Steps:

1. Implement server-side file validation to enforce size limits.
2. Ensure proper error handling for invalid file uploads.

5.94. Title: Weird Crypto (Server-Side)

Rating: High

URL: http://localhost:3000/#/security

Description: The application's GitHub repository reveals that MD5 hashing is used for password storage, which is a weak algorithm susceptible to collision attacks. This compromises the integrity of the stored data.

Proof of Concept:

Step 1: Navigate to the security folder in the Juice Shop's GitHub repository.

Step 2: Identify the use of MD5 in the codebase.

Remediation Steps:

1. Replace MD5 with a more secure hashing algorithm such as bcrypt or Argon2.
2. Implement password salting to further enhance security.

5.95. Access to Admin Section (Broken Access Control)

Rating: High

URL: http://localhost:3000/#/administration

Description: The admin section is accessible without proper authentication or authorization, allowing attackers to access sensitive administrative functions.

Proof of Concept:

Step 1: Inspect the home page and find the admin path: /administration.

Step 2: Manually navigate to http://localhost:3000/#/administration in the browser.

Step 3: Gain unauthorized access to the admin panel

Remediation Steps:

1. Implement role-based access control (RBAC) to restrict access to administrative sections.

3. Validate session tokens and user roles on the server side before granting access.

5.96. Get Rid of All 5-Star Feedback (Broken Access Control)

Rating: High

URL: http://localhost:3000/#/feedback

Description: The feedback deletion functionality allows unauthorized manipulation of reviews. Attackers can delete 5-star reviews by exploiting an SQL injection vulnerability.

Proof of Concept:

Step 1: Log in using the payload ' OR 1=1-- in the email field.

Step 2: Navigate to the feedback page and delete 5-star reviews as an admin.

Remediation Steps:

1. Validate and sanitize all input fields to prevent SQL injection.
2. Implement proper authorization checks for review deletion.

5.97. Reflected XSS (Client-Side)

Rating: High

URL: http://localhost:3000/#/track-order

Description: The order tracking page is vulnerable to reflected XSS, allowing attackers to inject malicious scripts through manipulated URL parameters. This can result in arbitrary script execution in the victim's browser.

Proof of Concept:

Step 1: Place an order and navigate to the tracking page.

Step 2: Replace the id parameter in the URL with <iframe src="javascript:alert('xss')">.

Step 3. Observe the alert pop-up in the browser, indicating successful script execution.

Remediation Steps:

1. Sanitize and validate all URL parameters to ensure they conform to expected formats.
2. Use output encoding to prevent the injection and execution of malicious scripts.

5.98 login Jim injection

Rating: High

URL: http://localhost3000/#/login

Description:

This vulnerability allows unauthorized access to a privileged account through SQL injection in the login form. An attacker can bypass authentication by crafting a malicious SQL query.

Proof of Concept:

1. Go to the login page.
2. Enter the email: `jim@juice-sh.op` .
3. In the password field, enter: `'--` .
4. Submit the form.
5. Observe successful login to Jim's account without knowing the password.

Remediation Steps:

- Use prepared statements (parameterized queries) for database interactions.
- Validate and sanitize all user inputs.
- Implement multi-factor authentication

5.99 User Credentials Retrieval

Rating: High

URL: http://localhost3000/#/search?q=apple

Description:

- SQL injection can also retrieve sensitive user credentials, including email addresses and passwords stored in the database.

Proof of Concept:

Step 1: Change in search the Apple to this :

```
Step 2: apple')) UNION SELECT email, password, 3, 4, 5, 6, 7, 8, 9 FROM users-
```

Step 3:Send the modified request.

Step 4: Observe the list of user credentials returned in the response.

Remediation Steps:

- Encrypt sensitive data such as passwords using modern hashing algorithms (e.g., bcrypt).
- Use prepared statements for database queries.
- Implement input validation to block malicious queries.

5.991 Basket manipulation

Rating: High

URL: http://localhost3000/#/basket

Description:

Manipulation of basket item IDs allows attackers to add unauthorized products or duplicate items without validation.

Proof of Concept:

1. Add a product (e.g., "Apple") to the basket.
2. Intercept the request in Burp Suite and modify the product ID to a different product (e.g., 7).
3. Submit the modified request.
4. Observe that the unauthorized product is added to the basket.

Remediation Steps:

- Implement server-side validation of basket operations to ensure users can only modify their own baskets.
- Validate product IDs against the inventory database.
- Apply rate-limiting to prevent abuse.

5.992 Outdated Whitelist (Unvalidated Redirects)

Rating: High

URL: http://localhost:3000/#/score-board

Description:

The application contains an outdated whitelist for redirect validation. An attacker can exploit this by modifying the URL to redirect users to an external, potentially malicious site.

Proof of Concept:

Step 1: Inspect the web application using developer tools and locate the included JavaScript files.

Step 2: Search for the term `redirect` in the JavaScript files.

Step 3: Identify a vulnerable redirection endpoint that allows users to be redirected to external sites, such as `https://blockchain.info`.

Step 4: Copy the identified URL and modify it in the website's address bar. Replace `/#/scoreboard it will be like this` https://<website-url>/?redirect=https://blockchain.info

Step 5: Observe that the user is successfully redirected to the external site.

Remediation Steps:

- Validate all redirect URLs on the server side, ensuring they conform to a strict whitelist of allowed domains.
- Avoid using client-side redirects for critical workflows.
- Log and monitor all redirection activities to detect potential abuse.

5.993 Database Schema Disclosure

Rating: High

URL: http://localhost3000/#/search?q=apple

Description :

SQL injection in the search functionality allows attackers to enumerate the database schema, exposing critical tables

Proof of Concept:

Step 1 : Perform a search for any product, such as "apple".

Step 2: Intercept the request using Burp Suite.

Step 3:Replace Search term “apple” with apple')) UNION SELECT sql,2,3,4,5,6,7,8,9 FROM sqlite_master--

Step 4: send the modified request.

Step 5: Observe a list of all database tables returned in the response.

Remediation Steps:

- Use parameterized queries for search inputs.
- Restrict database error messages to avoid leaking sensitive information.
- Validate and sanitize all user inputs.

6.0 Medium Findings

6.1. Title: Missing Encoding - Improper Input Validation (Client-side)

Rating: Medium

URL: /photo-wall

Description: The application does not properly encode user inputs when handling image URLs. This allows attackers to manipulate URL parameters by replacing special characters like # with its encoded form %23. This improper handling can lead to unintended behavior, such as revealing resources or images that should not be accessible or altering how URLs are processed by the application.

Proof of Concept:

Step 1: Log in to the application and navigate to the photo wall.

Step 2: Observe that an image is missing, and inspect the URL of the image in the developer tools.

Step 3: In the URL, replace the # character with %23 (URL encoding of #). The image will now be accessible or processed differently by the application.

Remediation Steps:

1. Implement proper encoding of special characters in URL parameters before processing them.
2. Validate and sanitize inputs to ensure that only valid, safe characters are passed in the URL and any parameters.
3. Ensure that special characters like #, %, and others are handled appropriately and cannot be exploited to alter the behavior of the application.

6.2. Title: Repetitive Registration Improper Input Validation(client-side)

Rating: Medium

URL: /#/register

Description: this is vulnerability which allows users to create account without confirming their passwords

Proof of Concept:

Step 1: Go to the registration page.

Step 2: Enter a password in the "Password" field. Ex(astf1)

Step 3: In the "Confirm Password" field, enter a different password. Just the password you enterd but repeated ex.(astf1astf1)

Step 4: Submit the registration form. And it isregistered successfully

Remediation Steps:

1.Client-side: Ensure that JavaScript is properly validating that the "Password" and "Confirm Password" fields match before allowing the form to be submitted.

2. Provide user feedback immediately when passwords do not match, preventing the form submission altogether.

6.3. Title: Deprecated Interface

Rating: Medium

URL: http://localhost:3000/#/complaint

Description: The application's file upload interface only supports ZIP and PDF formats; however, inspecting the page reveals additional support for XML files, which poses a potential risk if improperly handled.

Proof of Concept:

Step 1: Use the browser's developer tools to inspect the file upload interface.

Step 2: Search for "PDF" in the debugger and observe the additional XML support.

Step 3: Upload an XML file through the complaint submission form.

Remediation Steps:

1. Update the application to enforce file type restrictions strictly.
2. Remove deprecated or unintended file type support from the backend.

6.4. Title: Bully Chatbot (Client-Side)

Rating: Medium

URL: http://localhost:3000/#/chat-support

Description: The chatbot interface allows repeated abusive requests for coupon codes, leading to resource abuse or unintentional disclosures.

Proof of Concept:

Step 1: Continuously request a coupon code from the chatbot.

Step 2: Observe the chatbot's responses and identify potential unintended behaviors or data leakage.

Remediation Steps:

1. Implement rate-limiting for user interactions with the chatbot.
2. Add abuse detection mechanisms to block repetitive and abusive requests.

6.5. Title: Security Policy Disclosure (Client-Side)

Rating: Medium

URL: http://localhost:3000/.well-known/security.txt

Description: The publicly accessible security.txt file provides details about the organization's security policy, which could aid attackers in reconnaissance.

Proof of Concept:

Step 1: Navigate to http://localhost:3000/.well-known/security.txt.

Step 2: Review the disclosed information, such as contact details for reporting vulnerabilities.

Remediation Steps:

1. Restrict access to the security.txt file, making it available only to authenticated or authorized users.
2. Periodically review the file content to ensure no sensitive information is exposed.

6.6. Error Handling Vulnerability

Rating: Medium

URL: http://localhost3000/#/search?q=apple

Description:

The application fails to handle SQL injection attempts gracefully, exposing sensitive error messages.

Proof of Concept:

Step 1: Perform a search for "apple'))--".

Step 2: Observe the SQL error message returned in the response.

Remediation Steps:

- Implement generic error messages for users.
- Log detailed errors internally for debugging purposes.
- Validate and sanitize all user inputs

6.7. Forged Review

Rating: Medium

URL: http://localhost:3000/#/

Description:

The application allows users to manipulate product reviews, compromising the integrity of the review system.

Proof of Concept:

Step 1: Write a review for a product as a logged-in user.

Step 2: Capture the request in Burp Suite.

Step 3: Modify the review content and send the modified request.

Step 4: Observe that the updated review is successfully posted.

Remediation Steps:

- Verify the authenticity of review submissions on the server side.
- Restrict review modification to the original author only.
- Implement logging to detect suspicious review activity.

6.8. Forged Feedback

Rating: Medium

URL: http://localhost:3000/#/contact

Description:

The feedback functionality can be manipulated, allowing attackers to submit forged feedback. system.

Proof of Concept:

Step 1: Submit feedback through the feedback form.

Step 2: Intercept the request in Burp Suite

Step 3: Modify the feedback content and send the modified request.

Step 4: Observe that the forged feedback is successfully posted.

Remediation Steps:

- Validate feedback submissions on the server side.
- Implement rate-limiting to prevent abuse of feedback forms.

7.0 Low Findings

7.1. . Empty User Registration

Rating: Low

URL: http://localhost:3000/#/register

Description :

The application accepts incomplete user registrations, resulting in malformed accounts in the system.

Proof of Concept:

Step 1 : Register a new account.

Step 2: Intercept the registration request in Burp Suite.

Step 3 : Remove the email and password fields and send the modified request.

Step 4: Observe that the registration succeeds without valid credentials.

Remediation Steps:

- Enforce server-side validation for all required fields during registration.
- Implement strong input validation mechanisms.

7.2. Zero-Star Feedback

Rating: Low

URL: http://localhost:3000/#/contact

Description :

The application allows invalid ratings (e.g., 0 stars), compromising the feedback system's credibility.

Proof of Concept:

Step 1 : Submit feedback with a valid rating (e.g., 5 stars) t.

Step 2: Intercept the request in Burp Suite.

Step 3 : Change the rating value to 0 and send the modified request.

Step 4: Observe that the 0-star rating is accepted.

Remediation Steps:

- Validate rating values on the server side to ensure they fall within acceptable bounds (e.g., 1-5).
- Sanitize and validate all user inputs.