Juice Shop Report

Made by
Farah Mohamed Salama

Table of Content

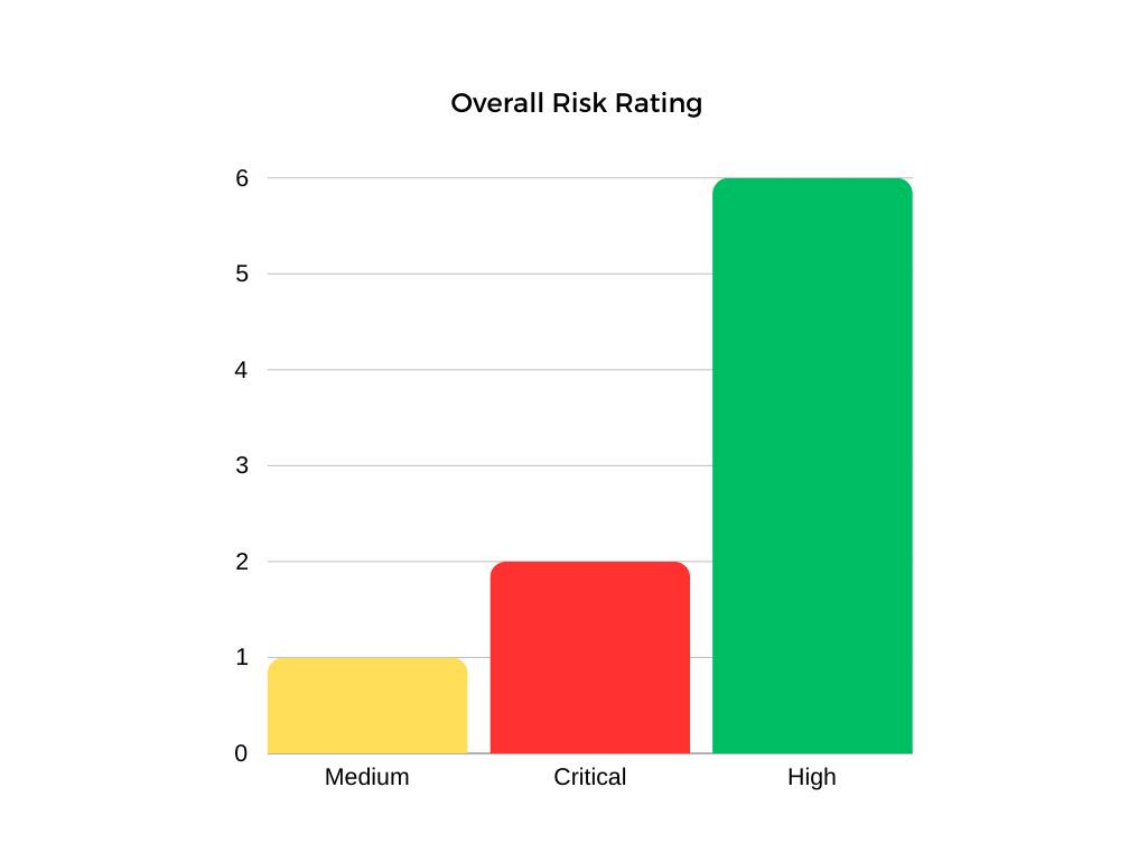
Executive Summary

I conducted a comprehensive penetration test to evaluate the security posture of the Juice Shop web application. This assessment aimed to identify and exploit vulnerabilities in the configuration and implementation of the Juice Shop service. The penetration test was designed to emulate the actions of an external threat actor attempting to compromise various external systems by exploiting multiple vulnerabilities within the service.

Test Scope and Method

Allowed Scope The allowed scope for this engagement was the following:
OWASP Juice Shop:http://localhost:3000

Overall Risk Rating



Found Vulnerabilities

Vulnerability	Severity
1. SQL Injection (login admin).	Critical
2.Unauthorized Privilege Escalation.	Critical
3.API-only Cross-Site Scripting (XSS)	High
4.Broken Access Control.	High
5.Unauthorized Modification of Data.	High
6.Unauthorized Viewing of Another User's Basket.	High
7.Unauthorized Deletion of Items in Another User's Basket.	High
8.Sensitive Information Exposure	High
9.DOM-Based Cross-Site Scripting (XSS) via Search Bar.	Medium

Technical Explanation

1-SQL Injection (login admin)

Vulnerability	SQL Injection
Weakness Type	CWE-89
CVSS Rating	CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
Endpoint	/rest/user/login

Description:

A critical SQL injection vulnerability was identified in the login functionality of the web application at http://localhost:3000. The vulnerability exists because user inputs are not properly sanitized before being used in SQL queries. An attacker can exploit this flaw to bypass authentication mechanisms and gain unauthorized access to the admin panel.

Impact:

Exploiting this vulnerability allows an attacker to bypass authentication and gain administrative access to the application. This could lead to complete compromise of the application, including the ability to view, modify, or delete sensitive data, and potentially execute arbitrary commands on the server.

Proof of Concept (PoC):

1.Navigate to the Login Page:

Open your web browser and go to http://localhost:3000.

2.Inject the SQL Payload:

In the username field of the login form, input the following payload:

```
' OR 1 -- -
```

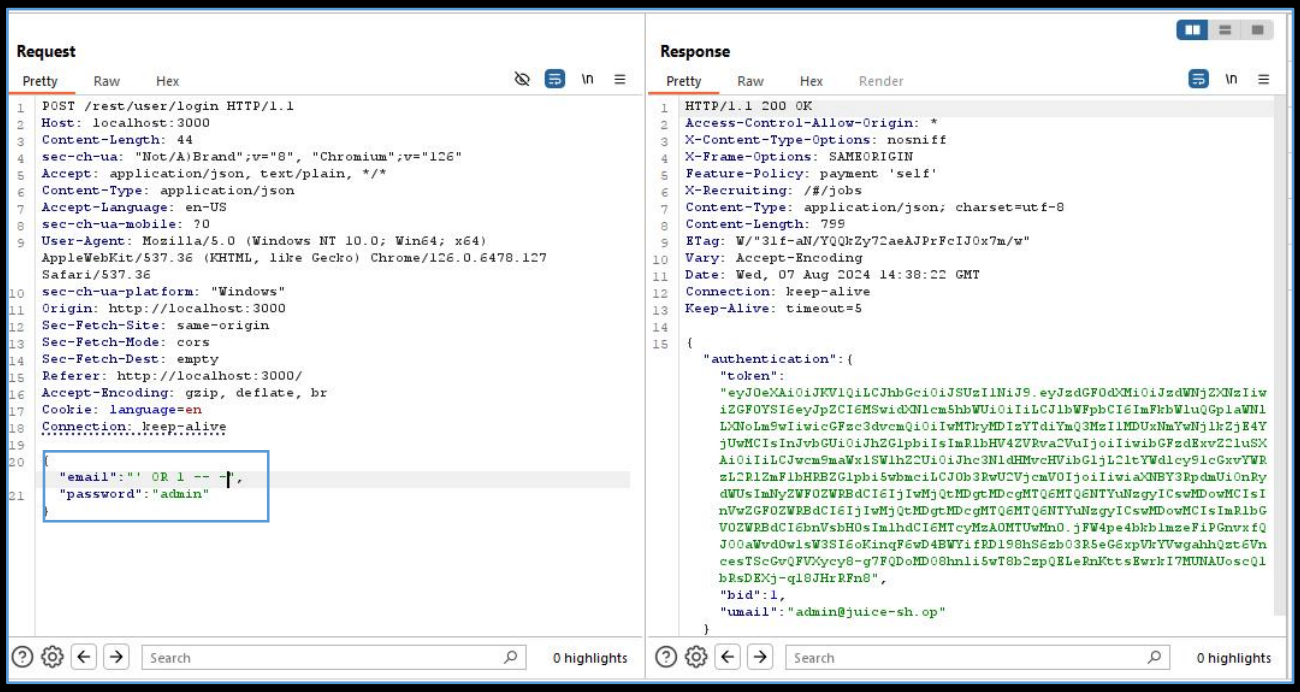
3.Submit the Form:

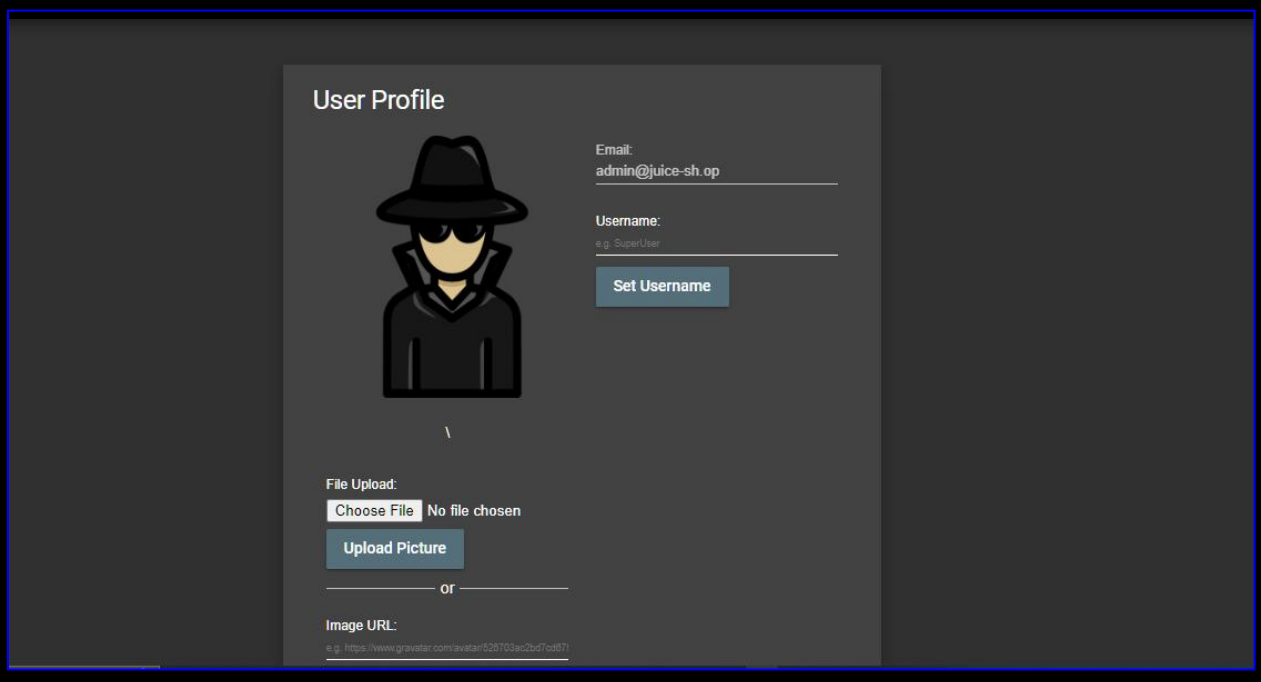
Click on the login button or press Enter to submit the form.

4.Result:

You will be logged in as an administrator.

Screenshots:





Recommended Fix:

1. **Input Validation**:

   Implement strict input validation to ensure that user inputs do not contain any SQL-specific characters such as ', ", --, etc.

2. **Parameterized Queries**:

   Use parameterized queries or prepared statements to separate user input from the SQL code. This prevents user input from being executed as SQL commands.

3. **ORM Usage**:

   Consider using an Object-Relational Mapping (ORM) framework that abstracts SQL queries and provides built-in protection against SQL injection.

4. **Regular Security Audits**:

   Conduct regular security audits and code reviews to identify and remediate vulnerabilities in the application code.

2.Unauthorized Privilege Escalation

Vulnerability	Improper Input Validation
Weakness Type	CWE-20
CVSS Rating	CVSS:3.1AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Endpoint	/api/Users

Description:

A critical vulnerability in the admin registration process of the web application at http://localhost:3000 allows unauthorized users to escalate their privileges by exploiting improper input validation. The vulnerability exists because the application does not adequately validate user inputs when assigning roles during the registration process, allowing attackers to assign themselves an admin role.

Impact:

Exploiting this vulnerability enables an attacker to gain administrative privileges without proper authorization. This could lead to a complete compromise of the application, including unauthorized access to sensitive data, modification of critical settings, and potential disruption of services.

Proof of Concept (PoC):

1.Intercept the Registration Request:

· Register a new user on the application by filling out the registration form.

· Use a web proxy tool like Burp Suite to intercept the request sent to the server during registration.

2.Modify the Role Parameter:

· In the intercepted request, find the section where the user's role is defined.

· Modify the role parameter to "role":"admin".

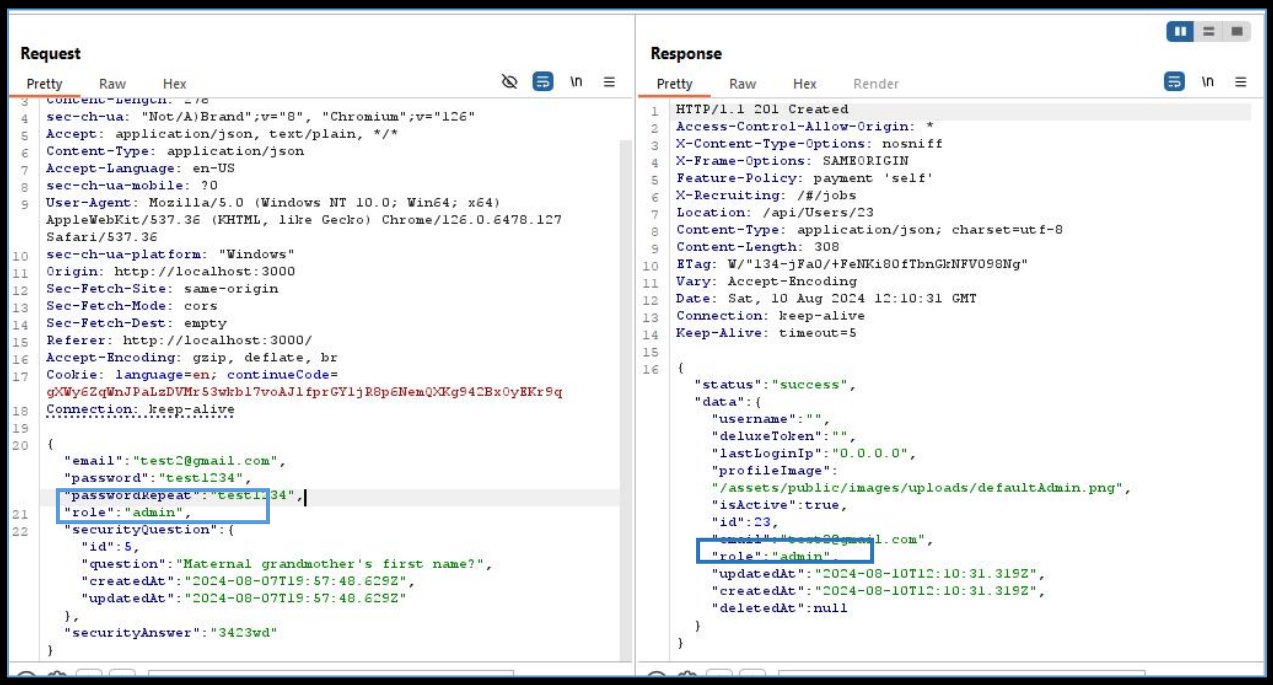
3.Forward the Request:

Forward the modified request to the server.

4.Result:

The server processes the request and assigns the user the "admin" role without any validation, granting the user full administrative privileges.

Screenshots:



Request

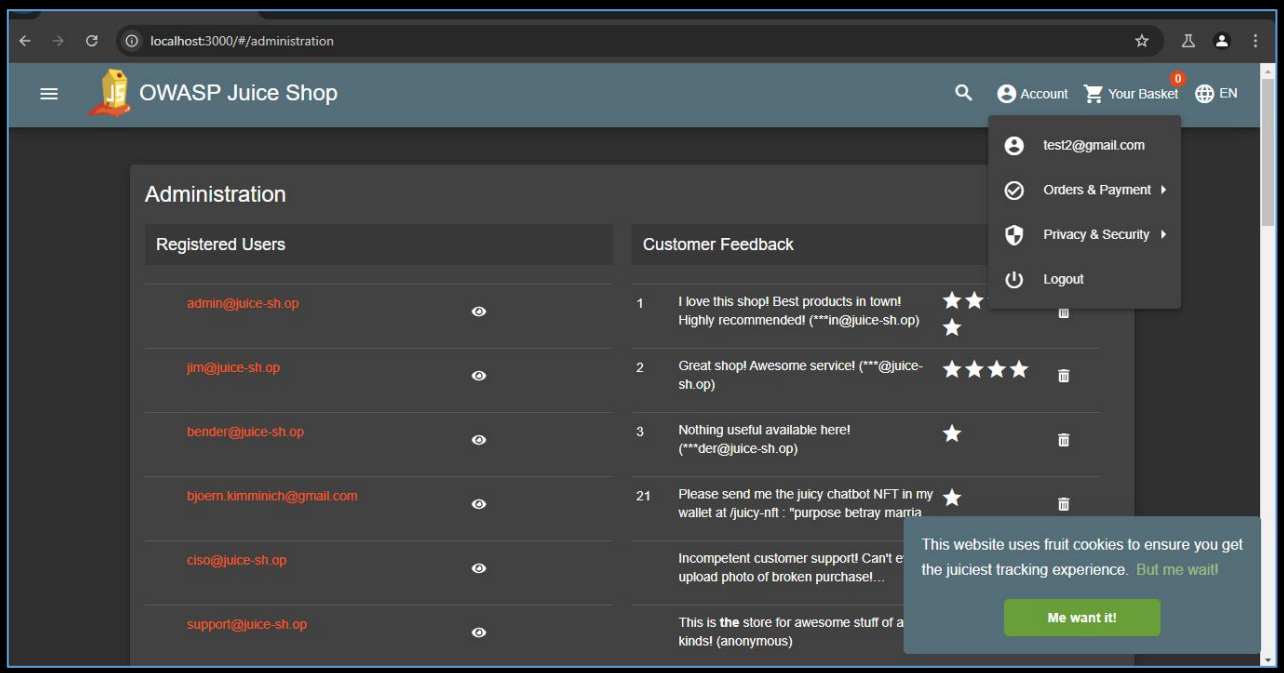
Pretty Raw Hex

```
sec-ch-ua: "Not/A)Brand";v="8", "Chromium";v="126"
Accept: application/json, text/plain, */*
Content-Type: application/json
Accept-Language: en-US
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/126.0.6478.127
Safari/537.36
sec-ch-ua-platform: "Windows"
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; continueCode=
gXWy6ZqWnJPaLzDVMr53wkbl7voAJ1fprGY1jR8p6NemQXKg942Bx0yEKr9q
Connection: keep-alive

{
  "email":"test2@gmail.com",
  "password":"test1234",
  "passwordRepeat":"test1234",
  "role":"admin",
  "securityQuestion":{
    "id":5,
    "question":"Maternal grandmother's first name?",
    "createdAt":"2024-08-07T19:57:48.629Z",
    "updatedAt":"2024-08-07T19:57:48.629Z"
  },
  "securityAnswer":"3423wd"
}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Location: /api/Users/23
Content-Type: application/json; charset=utf-8
Content-Length: 308
ETag: W/"134-jFaO/+FeNKi80fTbnGkNFVO98Ng"
Vary: Accept-Encoding
Date: Sat, 10 Aug 2024 12:10:31 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
  "status":"success",
  "data":{
    "username":"",
    "deluxeToken":"",
    "lastLoginIp":"0.0.0.0",
    "profileImage":
    "/assets/public/images/uploads/defaultAdmin.png",
    "isActive":true,
    "id":23,
    "email":"test2@gmail.com",
    "role":"admin",
    "updatedAt":"2024-08-10T12:10:31.319Z",
    "createdAt":"2024-08-10T12:10:31.319Z",
    "deletedAt":null
  }
}
```



Recommended Fix:

1.Strict Input Validation:

Implement strict input validation on all user-supplied data, particularly when assigning roles. Ensure that users cannot modify their roles through client-side input.

2.Server-Side Role Assignment:

Enforce role assignment logic server-side, where only authorized users or predefined processes can assign admin roles.

3.Authorization Checks:

Add robust authorization checks to verify that only users with proper privileges can modify roles or access sensitive endpoints.

4.Logging and Monitoring:

Implement logging and monitoring to detect any unauthorized role assignments and respond promptly to suspicious activity.

3.API-only Cross-Site Scripting (XSS)

Vulnerability	Cross-site Scripting(xss)
Weakness Type	CWE-79
CVSS Rating	CVSS:3.1AV:/AC:/PR:/UI:/S:/C:/I:/A:
Endpoint	/api/products/1

Description:

In this specific case, the vulnerability is exploitable by an authenticated admin user and can be triggered by modifying certain content via the API. Although the XSS is confined to API responses, it poses a significant risk as it can be leveraged to compromise admin accounts or perform unauthorized actions on behalf of the admin user.

Impact:

Perform unauthorized actions on behalf of the admin, such as modifying or deleting critical application data.

Proof of Concept (PoC):

1.Authenticate as Admin:

Log in to the application as an admin user. Make sure you have access to the API used for modifying content.

2.Send Malicious Payload via API:

Send a PUT request to the API endpoint that allows content modification.

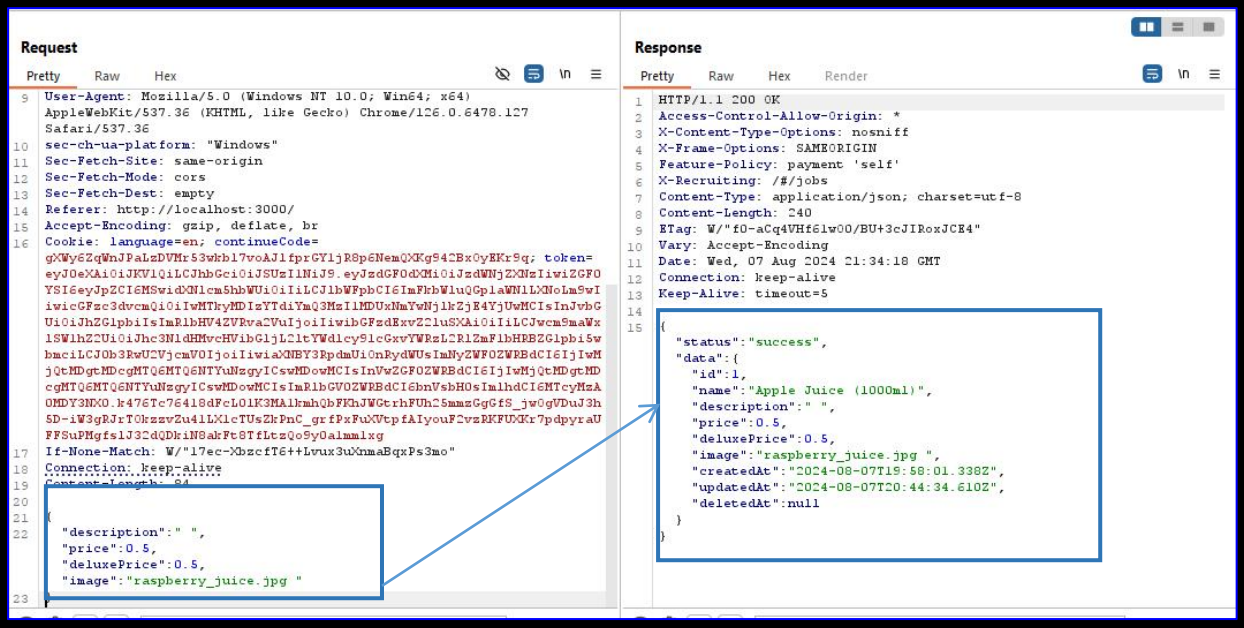
3.Trigger the XSS:

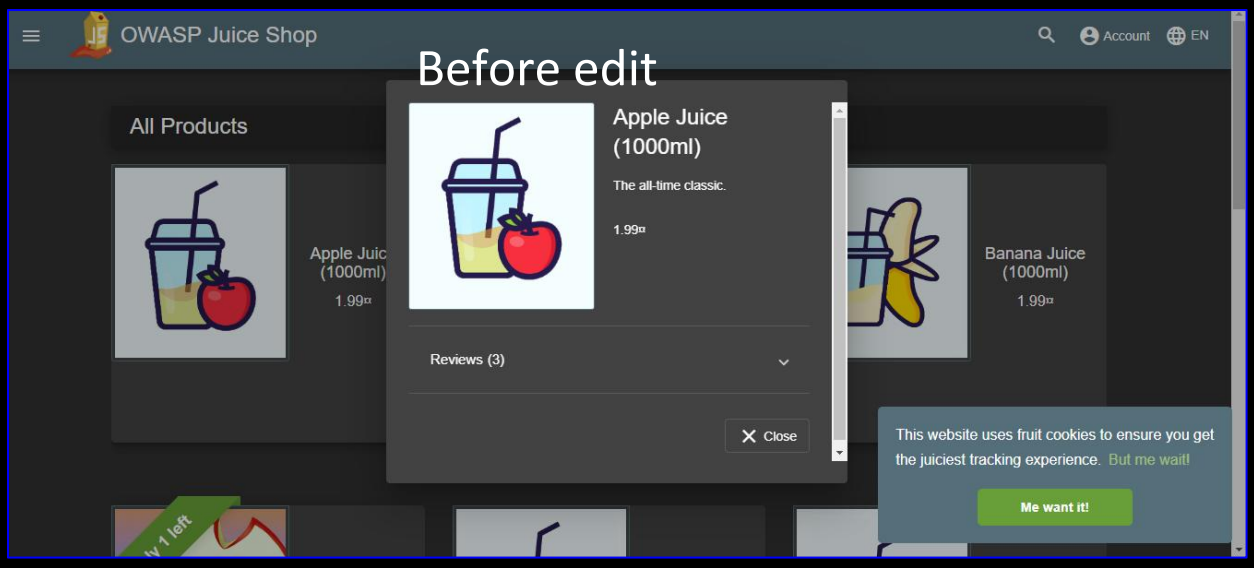
Access the admin dashboard or any other admin interface where the modified content is displayed.

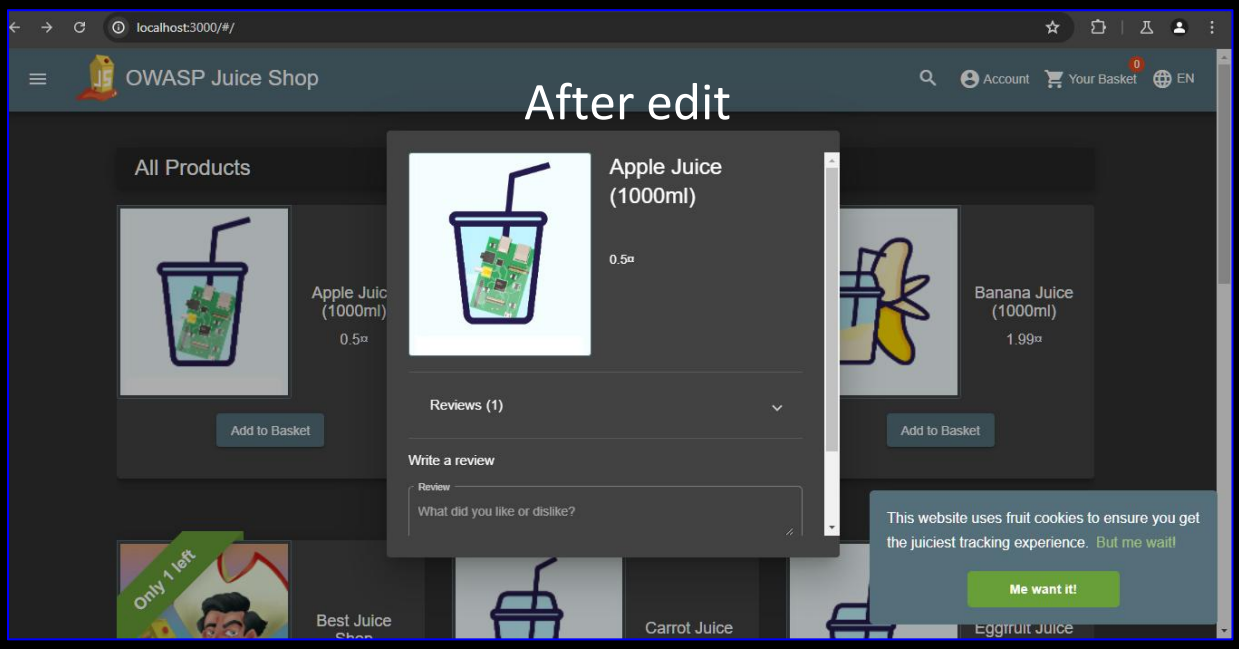
4.Result:

Observe that the modified content is executed or displayed as-is, indicating that proper sanitization is missing.

Screenshots:






localhost:3000/#/
OWASP Juice Shop
Account
Your Basket
EN
After edit
All Products
Apple Juic
(1000ml)
0.5¤
Add to Basket
Apple Juice
(1000ml)
0.5¤
Reviews (1)
Write a review
Review
What did you like or dislike?
Banana Juice
(1000ml)
1.99¤
Add to Basket
Only 1 left
Best Juice
Carrot Juice
Eggfruit Juice
This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!
Me want it!

Recommended Fix:

1.Input Sanitization:

Ensure that all input passed to the API is sanitized and encoded before being rendered on the page.

2.Output Encoding:

Apply output encoding to any user-generated content before rendering it in the admin interface.

3.Validation of Content Types:

Restrict the types of content that can be input via the API to ensure only safe and expected types are accepted.

4.Security Audits:

Conduct regular security audits focusing on API inputs and admin interface functionality to identify and fix vulnerabilities.

4.Broken Access Control

Vulnerability	Access Control
Weakness Type	CWE-284
CVSS Rating	CVSS:3.1AV:/AC:/PR:/UI:/S:/C:/I:/A:
Endpoint	/adminstration

Description:

A critical broken access control vulnerability was identified in the web application at http://localhost:3000. The issue lies in the /adminstration endpoint, which allows unauthorized users to access the administration panel without proper authentication or authorization checks. This vulnerability occurs due to the application failing to enforce access control measures on sensitive resources.

Impact:

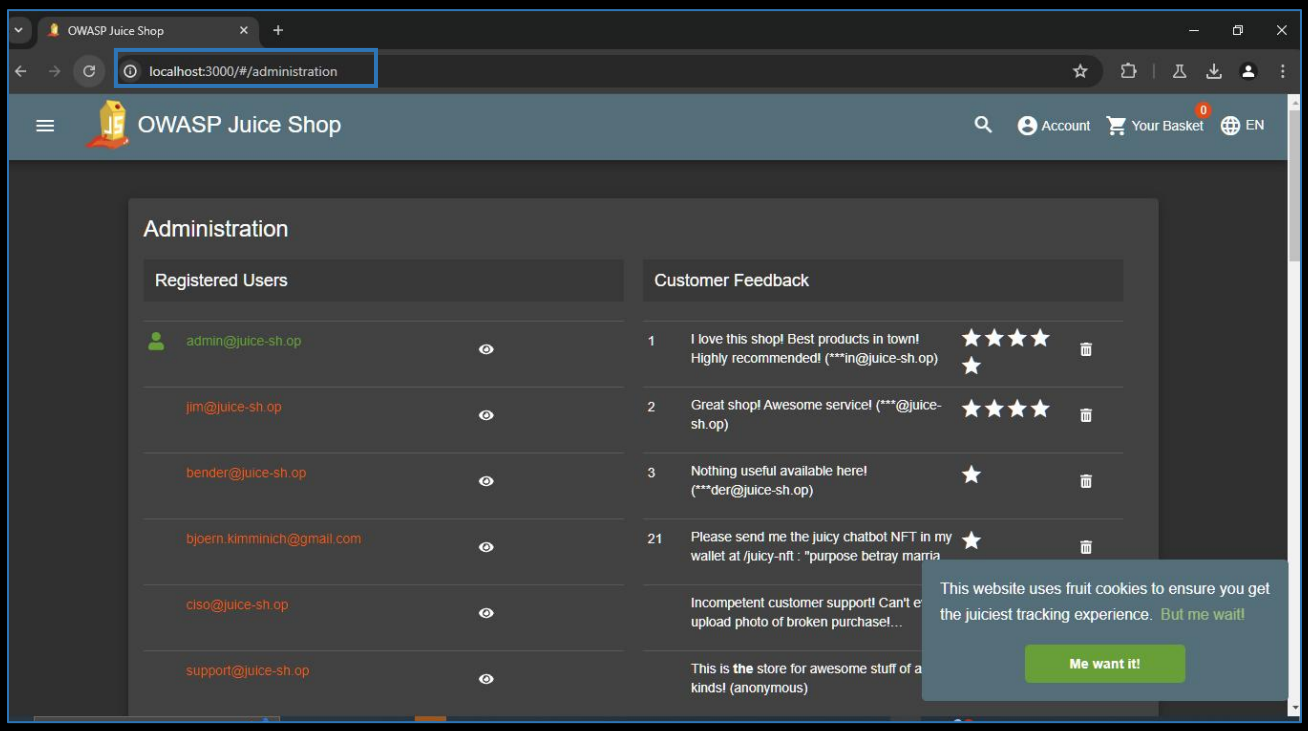
Exploiting this vulnerability allows an attacker to access the administration panel without valid credentials. This can lead to unauthorized access to sensitive data, the ability to modify application settings, and potential control over critical functions of the application.

Proof of Concept (PoC):

1. **Navigate to the Vulnerable Endpoint**:
   Open your web browser and go to the following URL:
   http://localhost:3000/adminstration

2. **Observe the Result:**
   The application grants access to the administration panel without requiring any authentication or validation of user privileges.

Screenshots:



Recommended Fix:

1.Enforce Proper Access Control:

Implement strict access control checks on the /adminstration endpoint. Ensure that only authorized users with the appropriate roles (e.g., admins) can access this resource.

2.Authentication and Authorization Middleware:

Utilize authentication and authorization middleware to verify user credentials and permissions before granting access to sensitive endpoints.

3.Security Testing:

Regularly test the application for access control vulnerabilities using both automated tools and manual testing to ensure that sensitive resources are protected.

4.Logging and Monitoring:

Implement logging and monitoring mechanisms to detect and respond to unauthorized access attempts.

5.Unauthorized Modification of Data

Vulnerability	Improper Access Control
Weakness Type	CWE-284
CVSS Rating	CVSS:3.1AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Endpoint	/api/BasketItems/

Description:

A critical access control vulnerability was identified in the web application at http://localhost:3000. The issue allows an authenticated user to add items to another user's basket by directly interacting with the /api/BasketItems/ endpoint. This vulnerability exists because the application does not properly validate or enforce ownership of basket items, enabling unauthorized modification of another user's data.

Impact:

Exploiting this vulnerability allows an attacker to add or modify items in another user's basket, leading to unauthorized access and potential manipulation of user orders. This can result in financial losses, trust issues, and potential legal consequences for the affected users and the application.

Proof of Concept (PoC):

1.Log in as a User:

Authenticate to the application using valid credentials.

2.Intercept the Request:

Add an item to your basket and intercept the request to /api/BasketItems/ using a web proxy tool like Burp Suite.

3.Modify the Request:

In the intercepted request, change the userId or any parameter that identifies the basket to another user's ID.

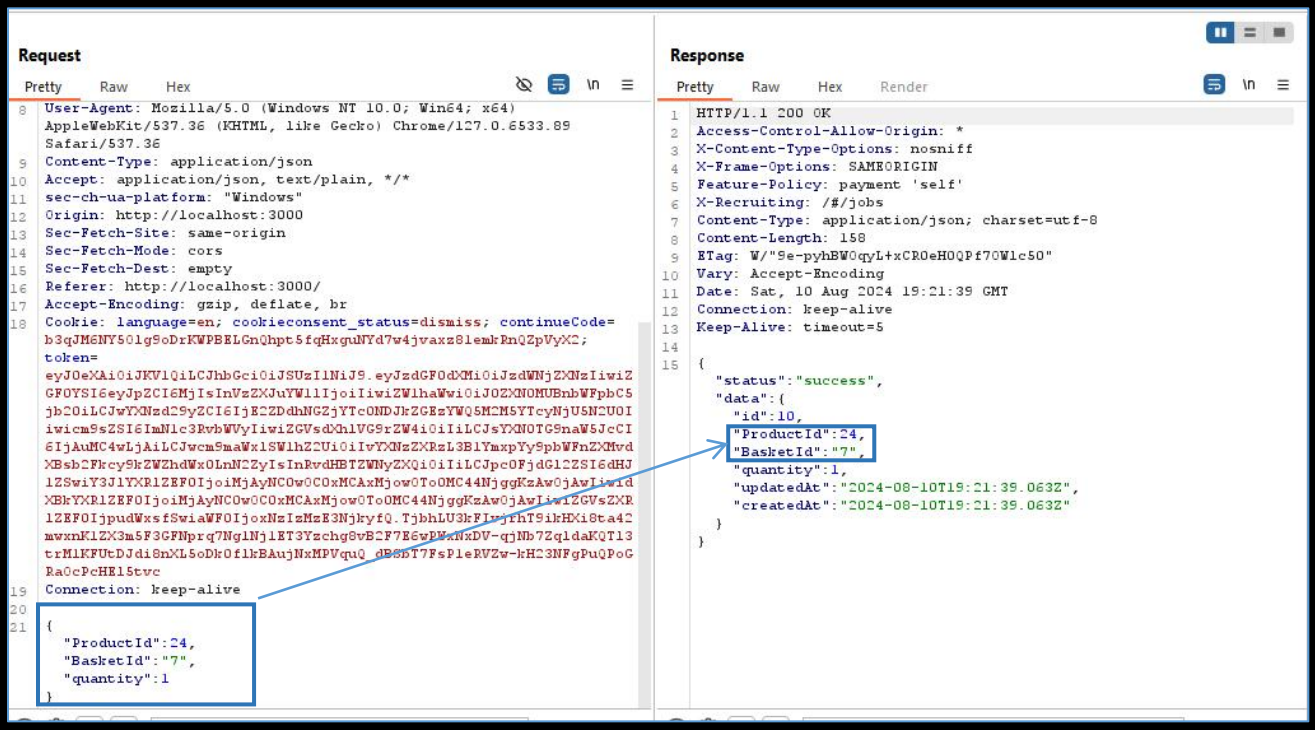
4.Forward the Request:

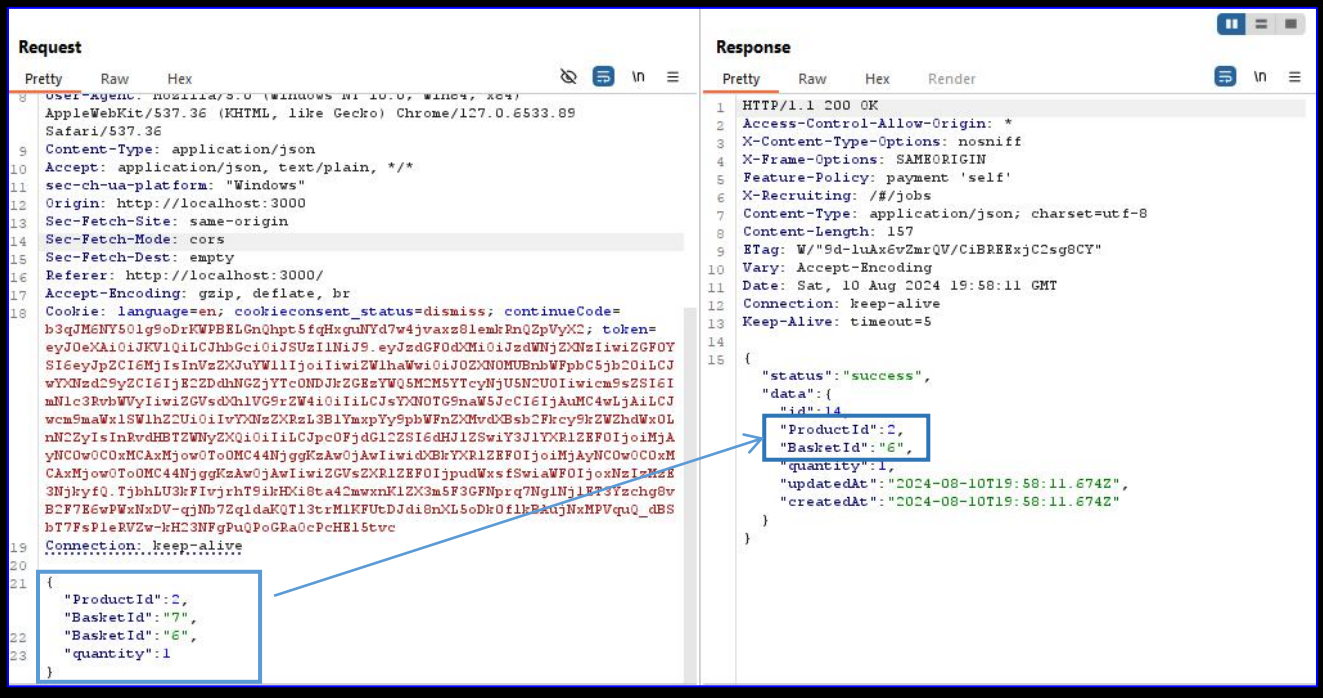
Forward the modified request to the server.

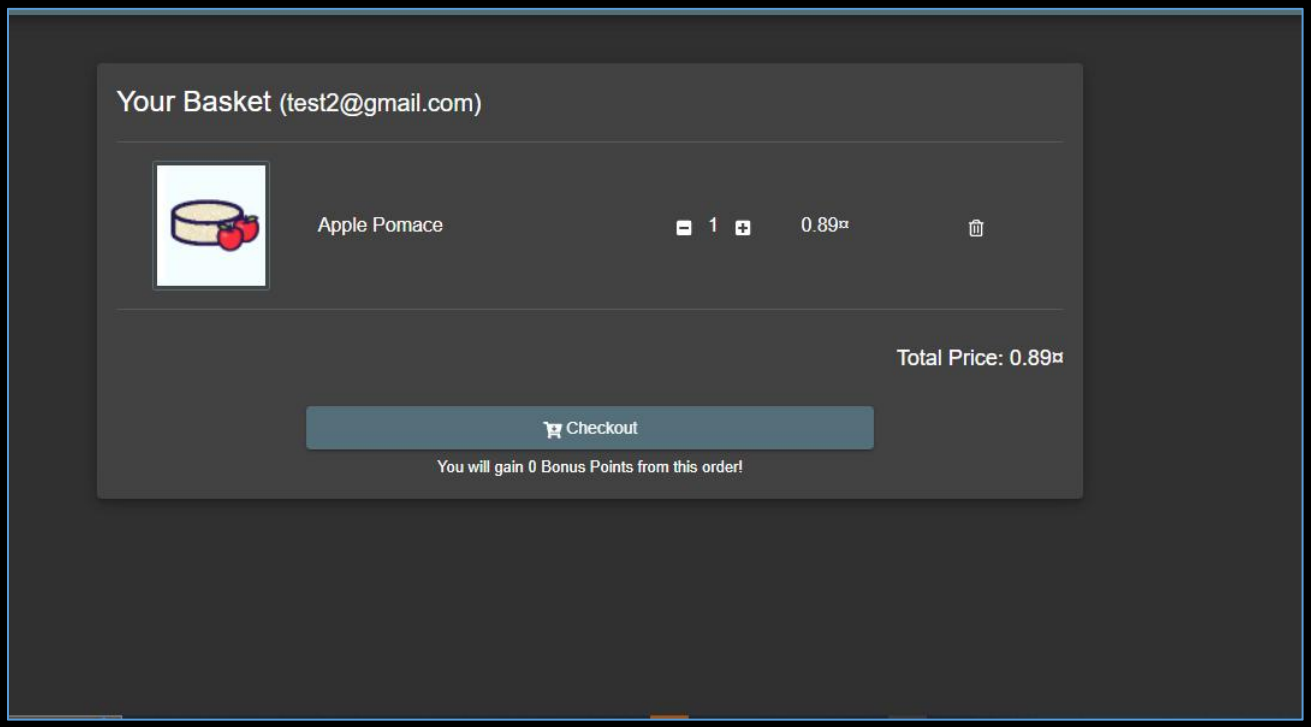
5.Result:

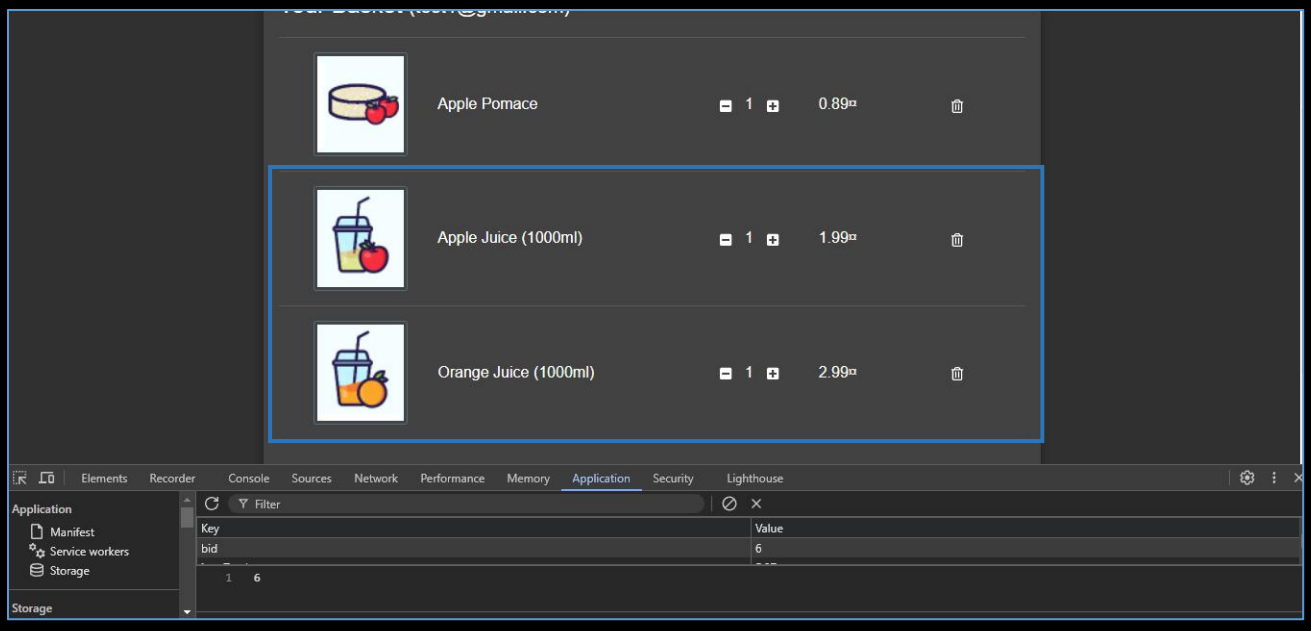
The item will be added to the basket of the specified user (anotherUserId), without any validation checks.

Screenshots:






Your Basket (test2@gmail.com)
Apple Pomace
1
0.89¤
Total Price: 0.89¤
Checkout
You will gain 0 Bonus Points from this order!


Apple Pomace
1
0.89¤
Apple Juice (1000ml)
1
1.99¤
Orange Juice (1000ml)
1
2.99¤
Elements
Recorder
Console
Sources
Network
Performance
Memory
Application
Security
Lighthouse
Application
Manifest
Service workers
Storage
Storage
Filter
Key
Value
bid
6
1 6

Recommended Fix:

1.Enforce Ownership Validation:

Ensure that the application strictly enforces ownership validation on the /api/BasketItems/ endpoint. Only allow modifications to the basket by the user who owns it.

2.Authorization Middleware:

Implement authorization middleware to check user permissions before allowing any modifications to the basket.

3.Avoid User-Controlled Identifiers:

Avoid using user-controlled identifiers (e.g., userId) in sensitive API requests. Instead, use session-based identifiers or server-side mechanisms to associate requests with the correct user.

4.Regular Security Audits:

Conduct regular security audits and penetration testing to identify and address access control vulnerabilities.

6.Unauthorized Viewing of Another User's Basket

Vulnerability	Unauthorized Data Access
Weakness Type	CWE-639
CVSS Rating	CVSS:3.1AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Endpoint	/rest/basket/

Description:

A critical Insecure Direct Object Reference (IDOR) vulnerability was identified in the web application at http://localhost:3000. The issue allows authenticated users to view the contents of another user's basket by modifying the basketId parameter in the request to the /rest/basket/ endpoint. The vulnerability exists because the application fails to validate that the user making the request owns the specified basket, leading to unauthorized access to other users' data.

Impact:

Exploiting this vulnerability allows an attacker to view the contents of another user's basket, potentially exposing sensitive information such as product selections, quantities, and prices. This can lead to privacy violations, unauthorized data access, and a breach of user trust.

Proof of Concept (PoC):

1.Log in as a User:

Authenticate to the application using valid credentials.

2.Intercept the Request:

Add an item to your basket and intercept the request to /rest/basket/{basketId} using a web proxy tool like Burp Suite.

3.Modify the Request:

Change the basketId in the request URL from your current basket ID (e.g., 6) to another ID (e.g., 2).

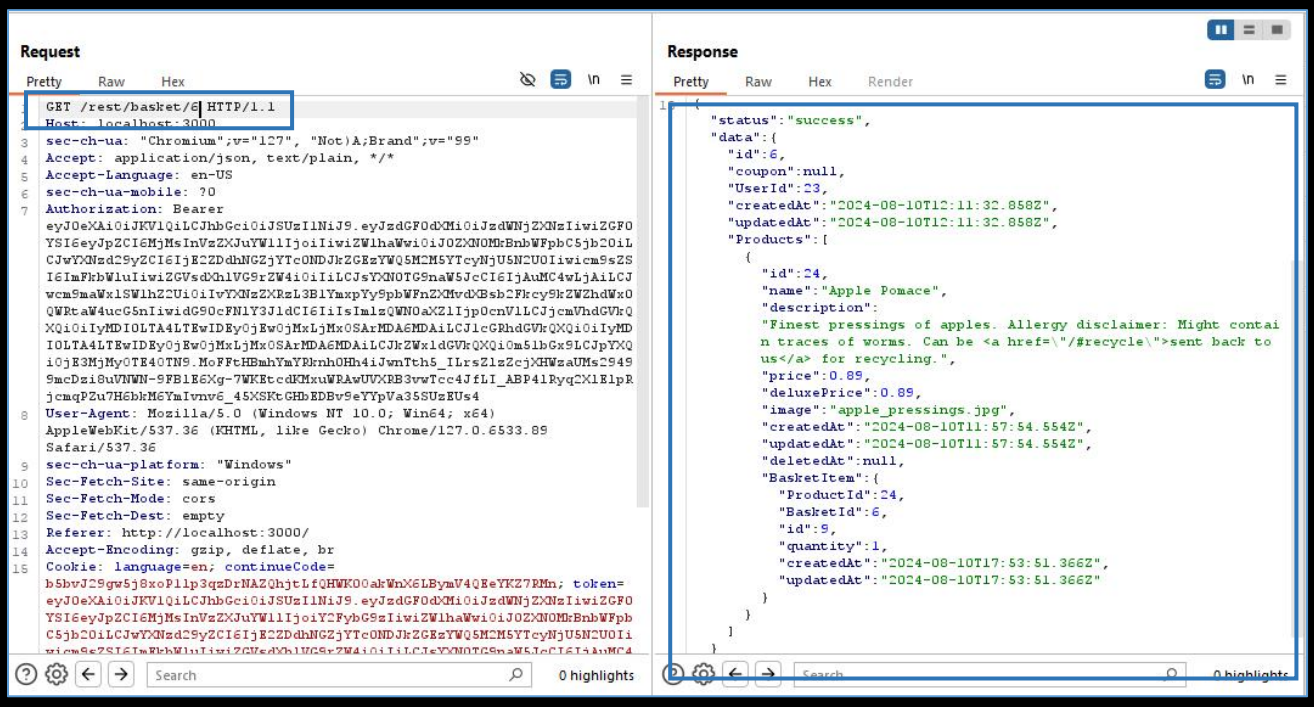
4.Forward the Request:

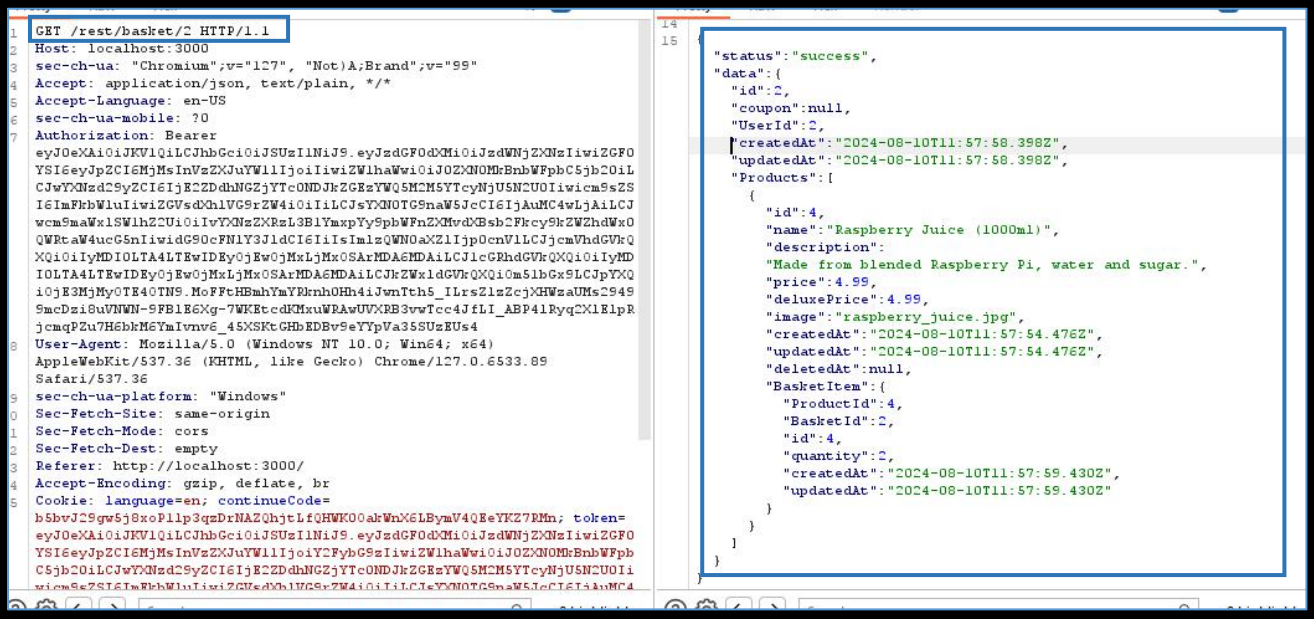
Forward the modified request to the server.

5.Result:

The server responds with the contents of the basket with ID 2, which belongs to another user.

Screenshots:





Recommended Fix:

1.Enforce Ownership Validation:

Implement strict server-side validation to ensure that users can only access resources (e.g., baskets) they own. Check that the basketId in the request belongs to the authenticated user.

2.Use Session-Based Identifiers:

Instead of allowing users to specify object identifiers (e.g., basketId), use session-based or server-generated identifiers that are not user-controllable.

3.Authorization Middleware:

Implement authorization middleware to ensure that all requests are subject to proper authorization checks before processing.

4.Penetration Testing:

Conduct regular penetration testing to identify and mitigate IDOR vulnerabilities across the application.

7.Unauthorized Deletion of Items in Another User's Basket

Vulnerability	Insecure Direct Object Reference (IDOR)
Weakness Type	CWE-639
CVSS Rating	CVSS:3.1AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Endpoint	/api/BasketItems/

Description:

A critical Insecure Direct Object Reference (IDOR) vulnerability was identified in the web application at http://localhost:3000. The issue allows an authenticated user to delete all items from another user's basket by sending a specially crafted DELETE request to the /api/BasketItems/ endpoint. The application fails to properly validate that the user making the request owns the items in the specified basket, allowing unauthorized deletion of items from other users' profiles.

Impact:

Exploiting this vulnerability enables an attacker to delete all items from another user's basket, leading to unauthorized data manipulation, potential financial loss, and a negative impact on user experience. This could also result in legal consequences for the application if users' data is compromised.

Proof of Concept (PoC):

1.Log in as a User:

Authenticate to the application using valid credentials.

2.Intercept a DELETE Request:

Attempt to delete an item from your own basket and intercept the DELETE request sent to /api/BasketItems/ using a web proxy tool like Burp Suite.

3.Modify the Request:

Change the identifier for the basket or item to target another user's basket.

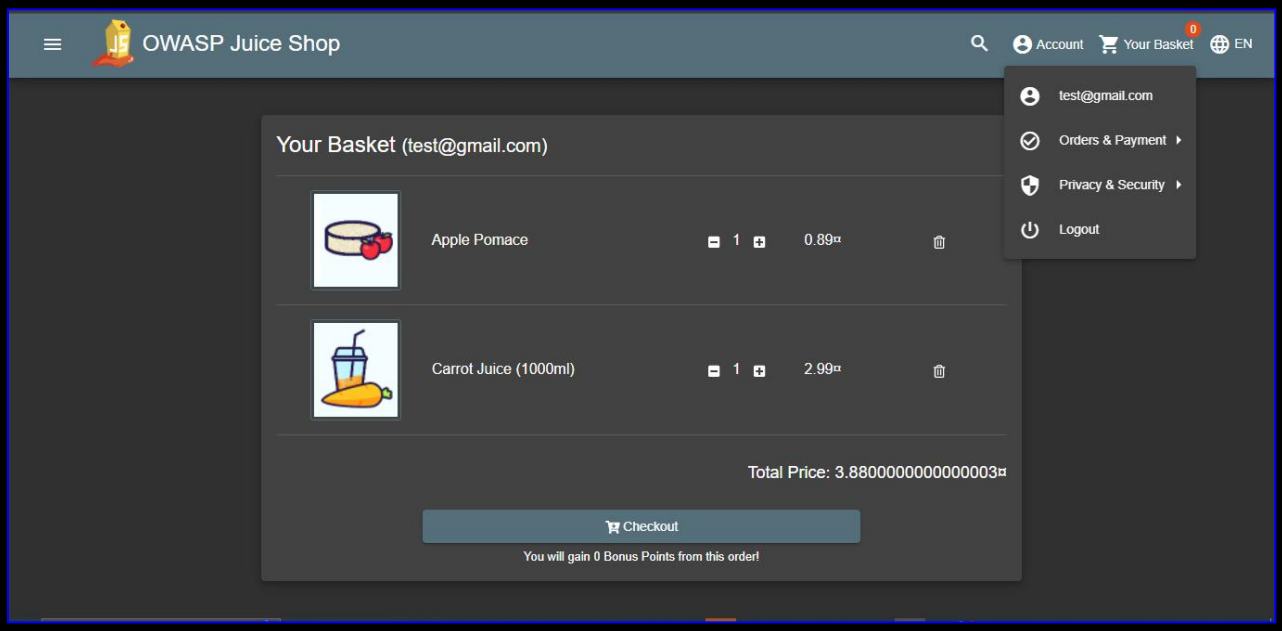
4.Forward the Request:

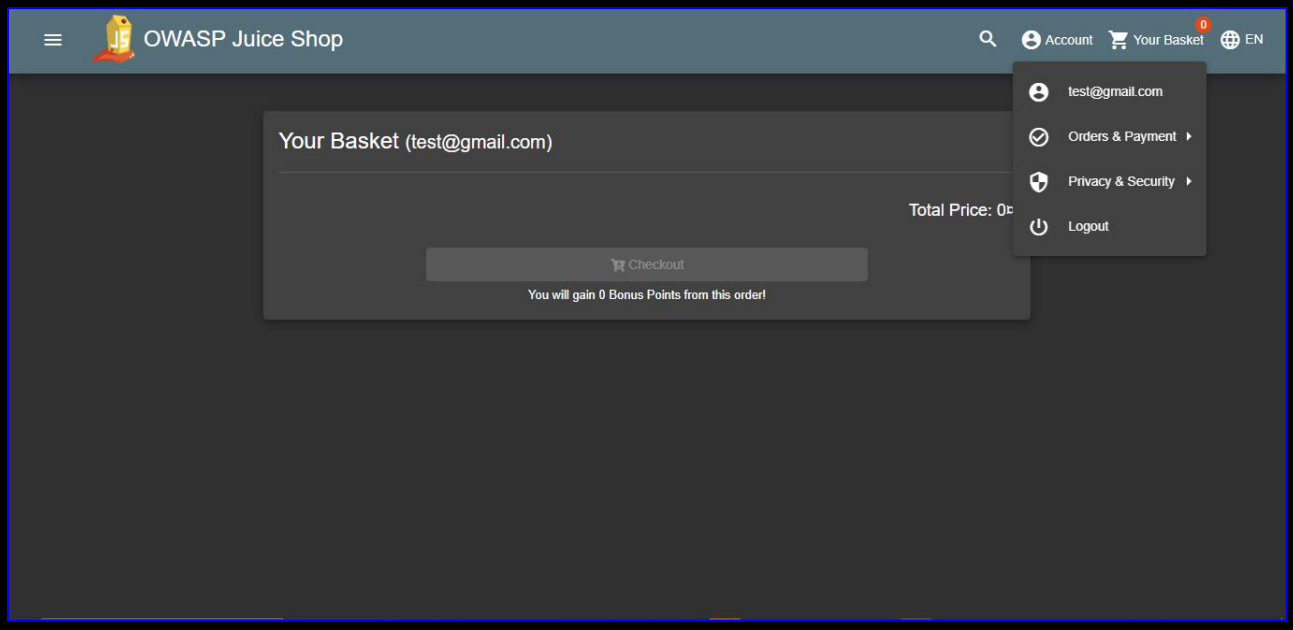
Forward the modified request to the server.

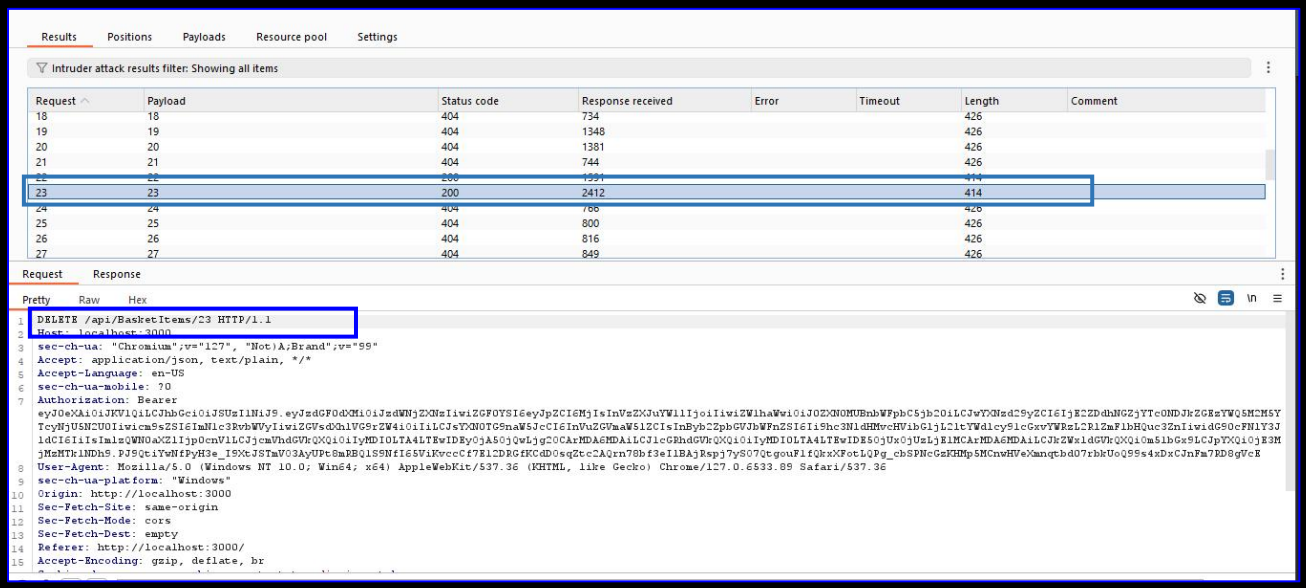
5.Result:

The server processes the request and deletes the items from the targeted user's basket without verifying that the request was made by the owner.

Screenshots:






Results
Positions
Payloads
Resource pool
Settings
Intruder attack results filter: Showing all items
Request
Payload
Status code
Response received
Error
Timeout
Length
Comment
18 18 404 734 426
19 19 404 1348 426
20 20 404 1381 426
21 21 404 744 426
23 23 200 2412 414
24 24 404 766 426
25 25 404 800 426
26 26 404 816 426
27 27 404 849 426
Request
Response
Pretty
Raw
Hex
DELETE /api/BasketItems/23 HTTP/1.1
Host: localhost:3000
sec-ch-ua: "Chromium";v="127", "Not)A;Brand";v="99"
Accept: application/json, text/plain, */*
Accept-Language: en-US
sec-ch-ua-mobile: ?0
Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjIsInVzZXJuYW1lIjoiIiwiZWlhaWwiOiJOZXNOMUBnbWFpbC5jb20iLCJwYXNzd29yZCI6IjE2ZDdhNGZjYTc0NDJkZGRzYWQ5M2M5Y
TcyNjU5NzU0IiwicmSzSI6ImNlc3RvbWVyIiwiZGVsdXhlVG9rZW4iOiIiLCJsYXN0TG9naW5JcCI6InVuZGVmaW5lZCIsInByb2ZpbGVJbWFnZSI6Ii9hc3NldHMvcHVibGljL2ltYWdlcy9uZGxYWRzL2RlZmF1bHQuc3ZnIiwidG90cFNlY3J
ldCI6IiIsImlzQWN0aXZlIjp0cnVlLCJjcmVhdGVkQXQiOiIyMDI0LTA4LTEwIDEy0jA50jQwLjg20CArMDA6MDAiLCJ1cGRhdGVkQXQiOiIyMDI0LTA4LTEwIDE50jUx0jUzLjE1MCArMDA6MDAiLCJkZWxldGVkQXQiOm51bGx9LCJpYXQiOjE3M
jMzMTk1NDh9.PJ9QtiYwNfPyH3e_ISXtJSTmV03AyUPt8mRBQlS9NfI65ViKvccCf7El2DRGfKCdD0sqZtc2AQrn78bf3eIlBAjRspj7yS07QtgouFlfQkxXFotLQPg_cbSPNcGzKHMp5MCnwHVeXamqtbd07rbkUoQ99s4xDxCJnFm7RD8gVcE
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.89 Safari/537.36
sec-ch-ua-platform: "Windows"
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br

Recommended Fix:

1.Enforce Ownership Validation:

Implement strict server-side checks to verify that the user making the request owns the basket or items being deleted. Ensure that users can only delete items they own.

2.Use Secure Identifiers:

Avoid using user-controlled identifiers (e.g., item IDs) in sensitive operations like deletion. Instead, use session-based identifiers or server-side logic to determine the correct resources to modify.

3.Authorization Middleware:

Implement authorization middleware to enforce access control before allowing modifications to user data.

4.Regular Security Audits:

Perform regular security audits and penetration testing to identify and address IDOR vulnerabilities across the application.

8.Sensitive Information Exposure

Vulnerability	Information Disclosure
Weakness Type	CWE-200
CVSS Rating	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N
Endpoint	/ftp

Description:

The /robots.txt file on the target application was found to be publicly accessible and contains a disallowed entry for the /ftp directory. This directory can potentially house sensitive files or data that were intended to be restricted. The exposure of this information allows an attacker to directly access the /ftp directory, which may contain sensitive or exploitable content.

Impact:

The disclosed /ftp directory could contain sensitive information such as configuration files, data backups, or other critical resources. Access to this directory can lead to unauthorized information gathering, which might pave the way for further attacks such as data theft, privilege escalation, or even full system compromise if exploitable files are discovered within.

Proof of Concept (PoC):

1.Navigate to the Target URL:

Open a web browser and go to the following URL: http://localhost:3000

2.Inject the Payload:

In the search bar, enter the following payload and press enter:/robots.txt

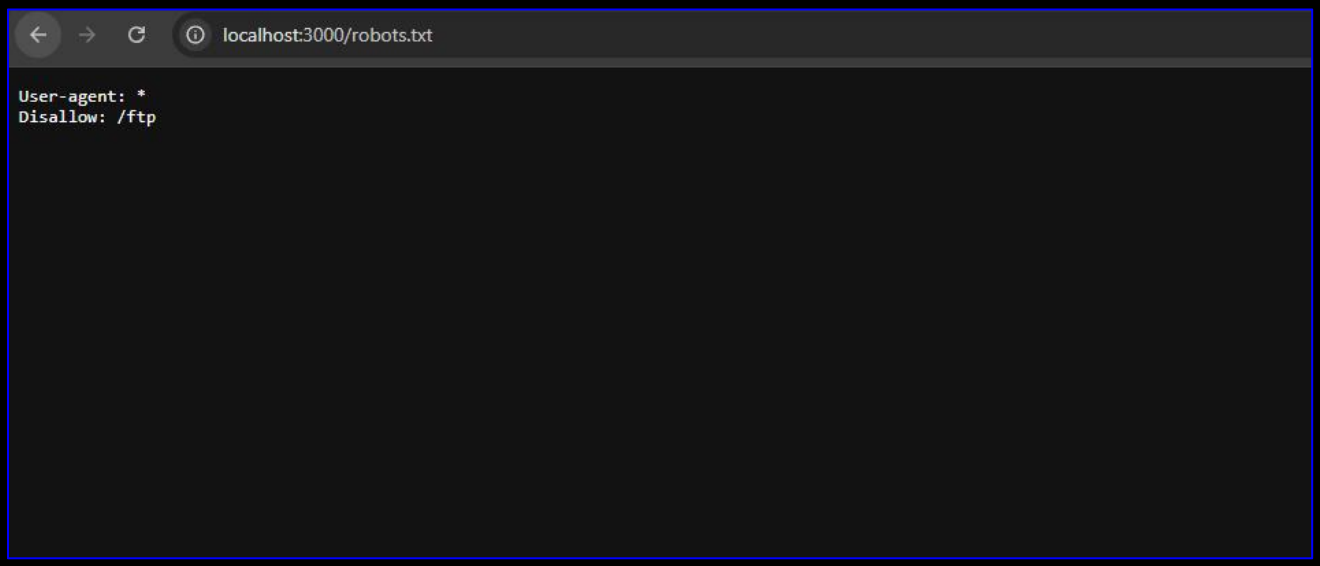
3.Identify the Sensitive Directory:

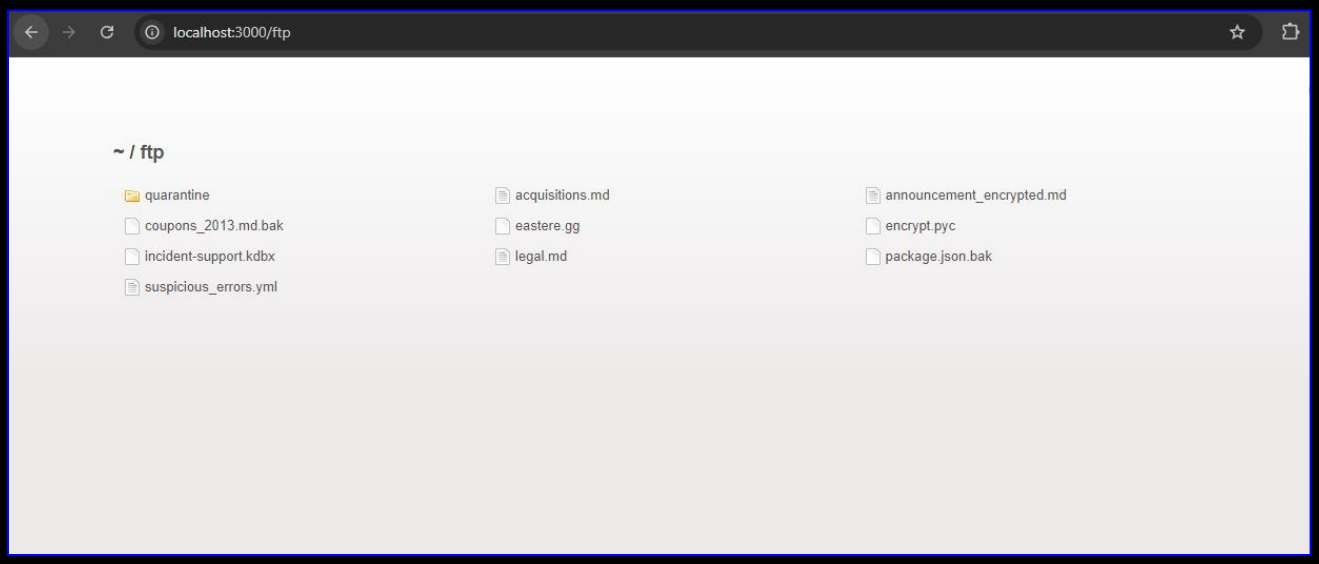
The browser will display the contents of the /robots.txt file, which includes a disallow directive pointing to the /ftp directory.

4.Access the /ftp Directory:

Manually navigate to the following URL to access the potentially sensitive /ftp directory: http://localhost:3000/ftp

Screenshots:





Recommended Fix:

1.Restrict Access to the /ftp Directory:

Ensure that the /ftp directory is not publicly accessible by implementing proper access controls. Use server configurations to restrict access to authorized users only.

2.Remove Sensitive Directories from /robots.txt:

Avoid listing sensitive directories in the /robots.txt file. Instead, use authentication and authorization mechanisms to control access to these areas.

3.Regular Security Audits:

Conduct regular security audits to identify and secure any sensitive directories or files that should not be publicly accessible.

9.DOM-Based Cross-Site Scripting (XSS) via Search Bar

Vulnerability	DOM-Based XSS
Weakness Type	CWE-79
CVSS Rating	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N
Endpoint	/search

Description:

The application at http://localhost:3000/#/ is vulnerable to DOM-Based Cross-Site Scripting (XSS). The vulnerability occurs when user-controlled input is directly manipulated in the DOM without proper sanitization, allowing attackers to inject and execute malicious scripts within the context of the user's browser. This specific vulnerability was identified in the search bar of the application, where an attacker can inject a malicious payload that executes JavaScript code.

Impact:

Successful exploitation of this vulnerability allows an attacker to execute arbitrary JavaScript code in the context of the user's session. This could lead to unauthorized actions such as stealing session cookies, performing actions on behalf of the user, or displaying fraudulent content. The impact is elevated if sensitive user data or high-privileged user accounts are affected.

Proof of Concept (PoC):

1.Navigate to the Target URL:

Open a web browser and go to the following URL: http://localhost:3000

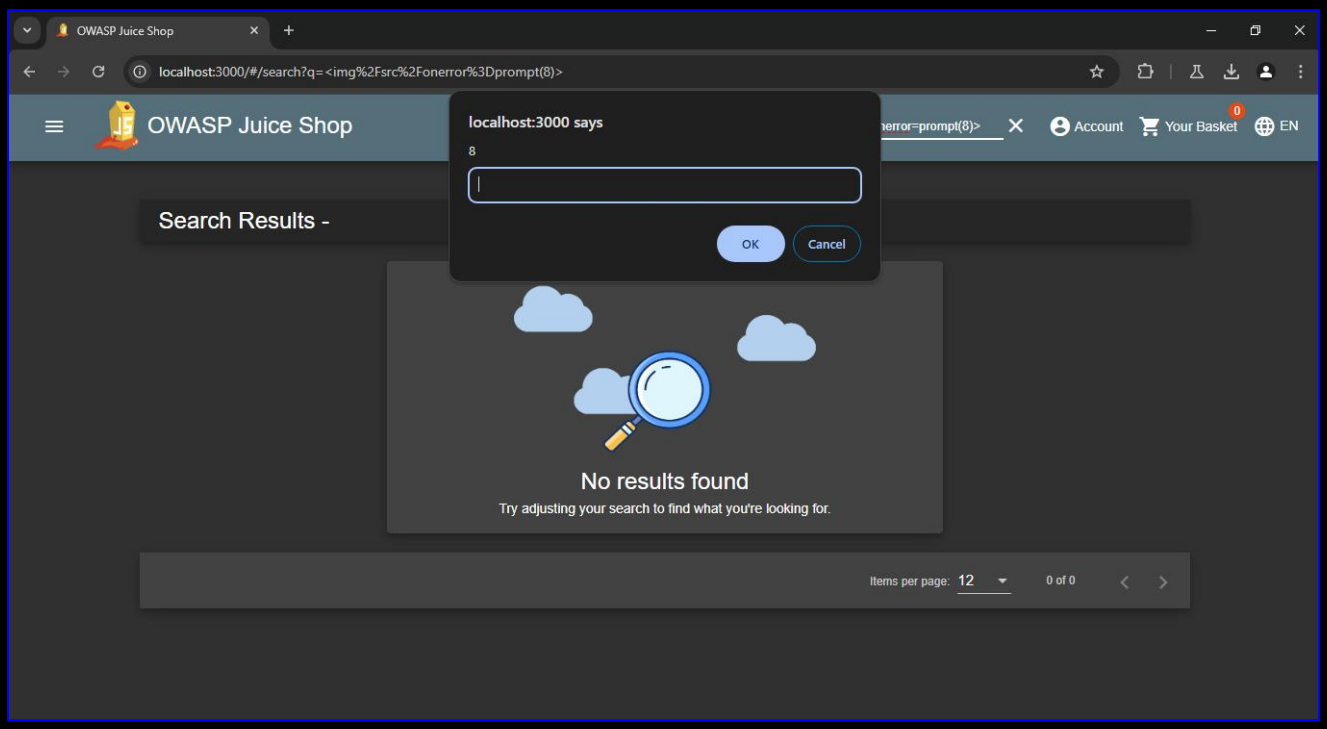
2.Inject the Payload:

In the search bar of the application, enter the following payload and press enter:
<img/src/onerror=prompt(8)>

3.Observe the Behavior:

The payload will trigger a JavaScript prompt dialog box with the number 8 displayed, confirming that the injected script was executed.

Screenshots:



Recommended Fix:

1.Input Validation and Sanitization:

Implement strict input validation and sanitization on any user-controlled data that is inserted into the DOM. Ensure that any input is properly escaped or encoded before being processed or displayed.

2.Use of Secure JavaScript Libraries:

Consider using secure JavaScript libraries or frameworks that automatically handle input sanitization and prevent direct manipulation of the DOM without proper controls.

3.Content Security Policy (CSP):

Implement a Content Security Policy (CSP) that restricts the execution of inline scripts. This can mitigate the impact of XSS vulnerabilities by preventing the execution of unauthorized scripts.