Web Application Penetration Testing (OWASP Juice Shop)

Presented By:

Mohamed Mostafa Elemam

Ahmed Hesham Soliman

Yousef Mohammed abdelkareem

Mosatafa Mohamed

Osama Hesham

Presented for:

Eng. Beshoy Vector

Digital Egypt Pioneers Initiative(DEPI)

Table of Contents:

OWASP Juice Shop Vulnerability Assessment

Executive summary:

A penetration test was performed on the OWASP Juice Shop, a well-known intentionally vulnerable application used for security training. The assessment focused on identifying potential weaknesses, specifically targeting key **OWASP Top 10** vulnerabilities such as SQL injection, Cross-Site Scripting (XSS), and authentication flaws. The evaluation revealed several vulnerabilities, ranging from insecure input validation to broken authentication mechanisms. This report details these findings, categorizes them by severity, and provides actionable remediation steps to enhance security and mitigate the risk of unauthorized access or data breaches.

Scope:

IP Address	http://localhost:3000/
Name	OWASP Juice Shop
Type	Web Application

Methodology:

A black-box approach was employed, utilizing Burp Suite to intercept and analyze all web traffic, manipulating requests and responses to actively test for vulnerabilities. Browser Developer Tools were used to inspect client-side code and data handling. This involved examining HTTP interactions, employing Burp's tools for targeted testing, and client-side analysis with browser Dev Tools complemented this by revealing potential vulnerabilities in JavaScript and how the application managed data within the browser, all without prior knowledge of the application's internal workings.

1) Vulnerability: Sensitive Data Exposure – Access a confidential document

Executive Summary:

A sensitive data exposure vulnerability was identified, allowing unauthorized access to confidential documents through /ftp directory and direct file access to acquisition, as found **the /ftp directory in the robots.txt file.**

Impact:

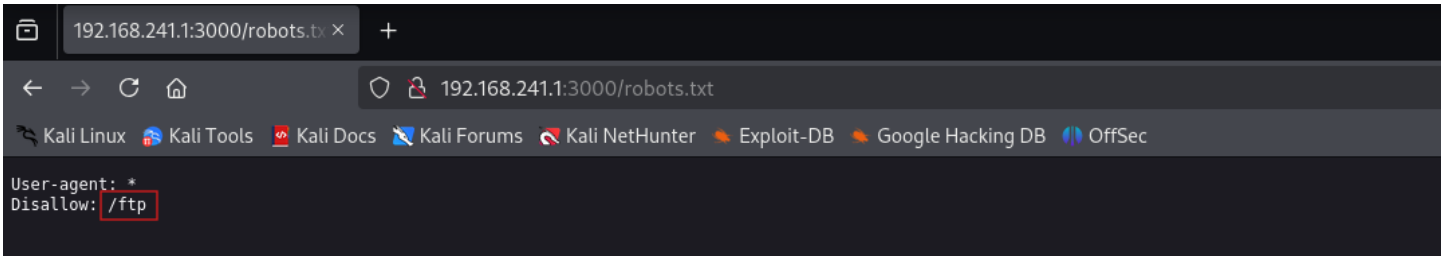
Exposure of confidential documents, such as acquisition details, could lead to significant business risks. This may include the leakage of sensitive financial information **as it has a big impact on the stock market** potentially harming the organization's competitive advantage, reputation, and potentially leading to legal issues depending on the nature of the exposed data.

Severity:

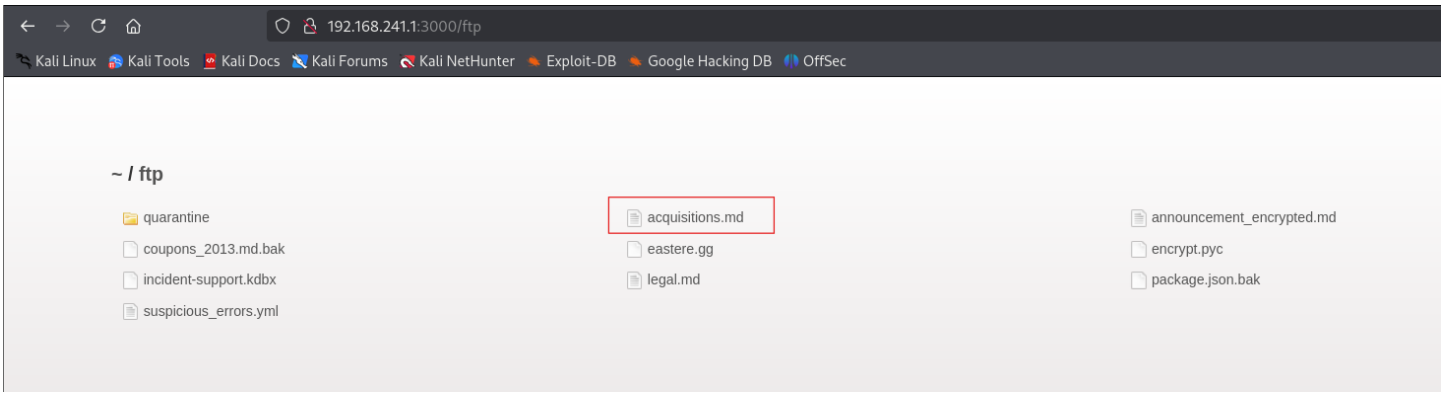
High

PoC (Steps to Reproduce):

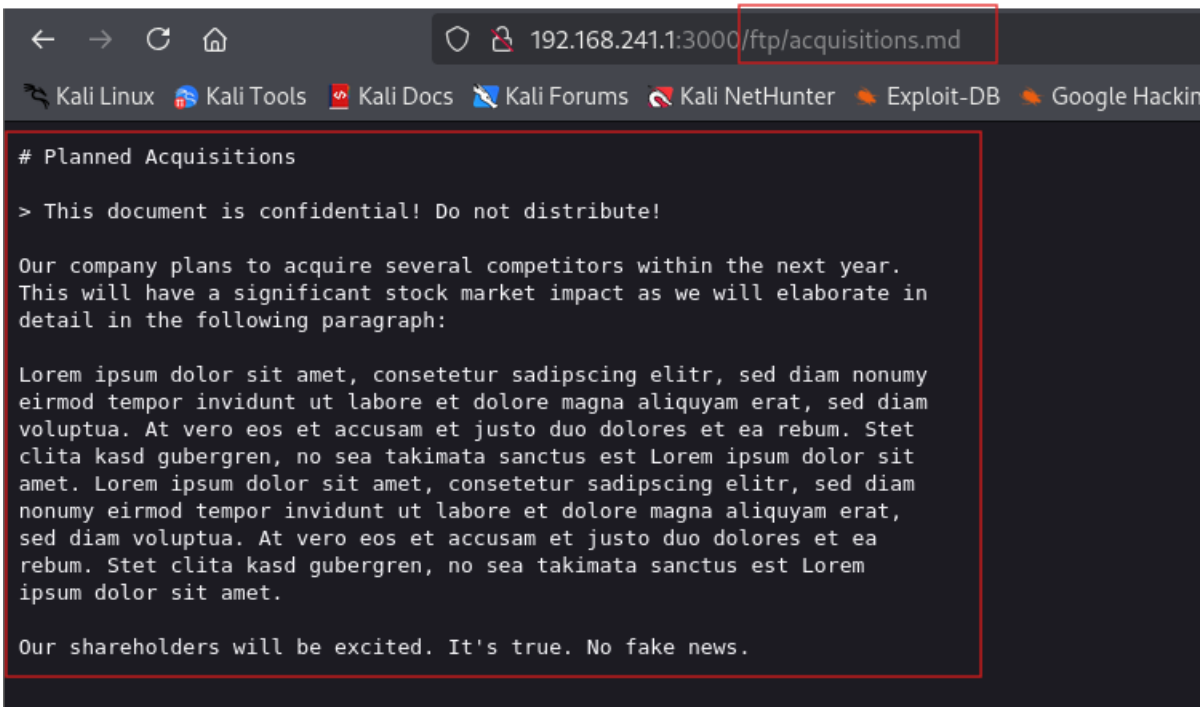
1. Go to http://192.168.241.1:3000/robots.txt , after that we will find interesting directory.



2. Successfully attempt to browse the directory by changing the URL into http://192.168.241.1:3000/ftp



3. You will find very interesting files like package.json.bak and acquisitions.md , open it.



Mitigation:

- **Disable Directory Listing:** Configure the web server to prevent directory listing for the /ftp directory. This will prevent attackers from browsing the contents of the directory. **The presence of /ftp in robots.txt only advises search engine crawlers and does not provide any actual security against direct access by malicious actors.**

2) Vulnerability: SQL Injection leading to Admin Login

Executive Summary:

A critical SQL injection vulnerability was discovered in the login functionality, allowing attackers to bypass authentication and potentially gain administrative access by manipulating the email parameter. Successful exploitation allows login as the first user in the database, which is often the administrator account.

Impact:

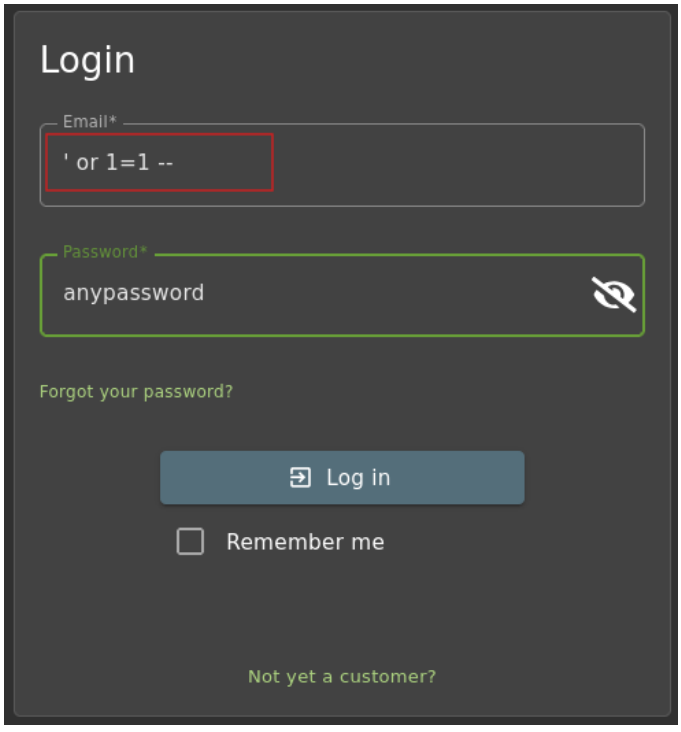
Complete account takeover, including administrative privileges. An attacker gaining admin access could perform any administrative action, leading to severe consequences such as data breaches, unauthorized modifications, and further malicious activities. This represents a catastrophic compromise of the application's security and integrity.

Severity:

Critical

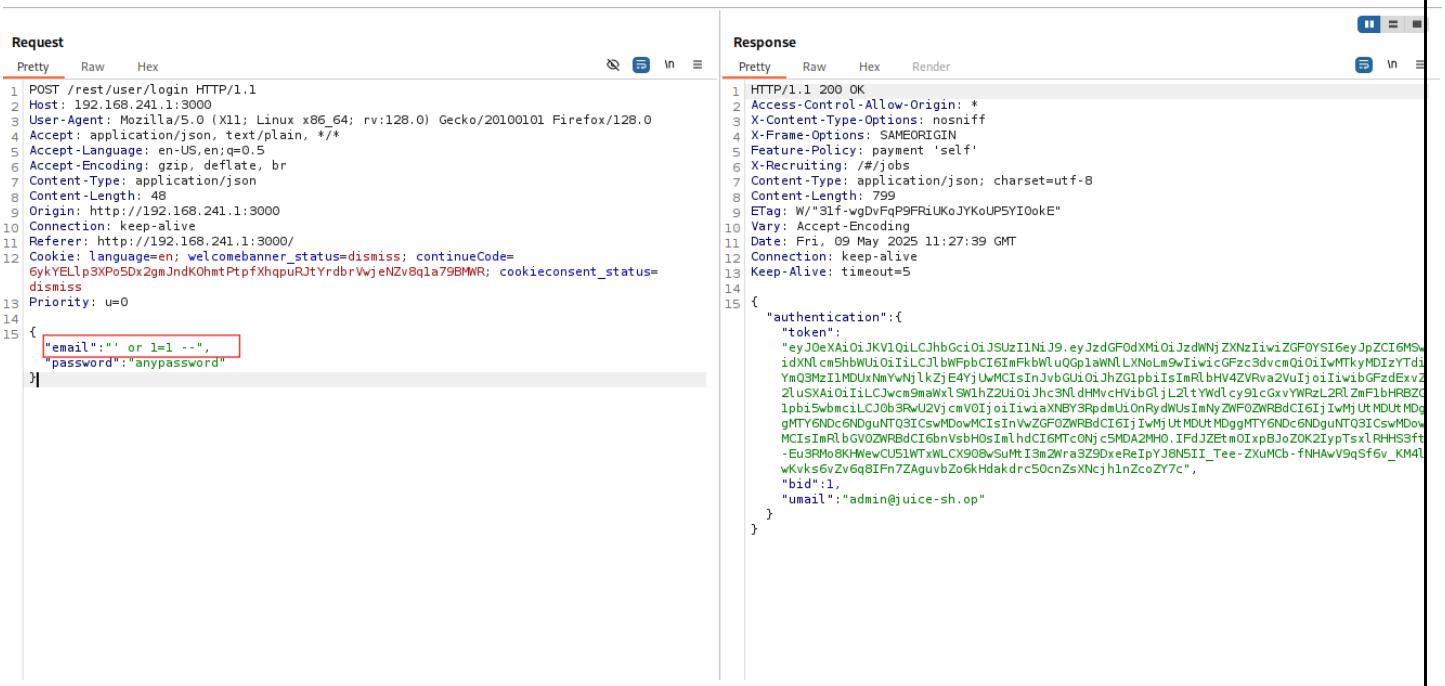
PoC (Steps to Reproduce):

1. Navigate to the login page: http://192.168.241.1:3000/#/login



2.Send a POST request to /rest/login

3.Observe the response. A successful bypass of the authentication and the reception of an authentication token ("token": "...") and user details ("bid": 1, "email": "admin@juice-sh.op") indicates successful login as the first user in the database, which is likely the administrator.



Mitigation:

- **Input Validation and Sanitization:** Implement robust input validation and sanitization on all user-provided data, especially the email and password fields, to prevent the injection of malicious SQL code.
- **Web Application Firewall (WAF):** Consider deploying a Web Application Firewall (WAF) that can help detect and block common SQL injection attempts. However, a WAF is not a substitute for secure coding practices.

3) Vulnerability: Broken Access Control (Access the administration section of the store)

Executive Summary:

A broken access control vulnerability exists that prevents unauthorized access to the administration panel initially. However, after successfully logging in with an administrator's account (either through the previously identified SQL injection), the /administration route becomes accessible, indicating a flaw in access control enforcement.

Impact:

Unauthorized access to the administration section allows malicious actors who have somehow obtained administrator credentials (or bypassed authentication) to perform administrative functions like **deleting Registered accounts and deleting reviews**. This could lead to severe consequences, including the ability to manage users, products, orders, and potentially modify critical system settings, leading to data breaches, service disruptions, and a complete compromise of the store's functionality and data integrity.

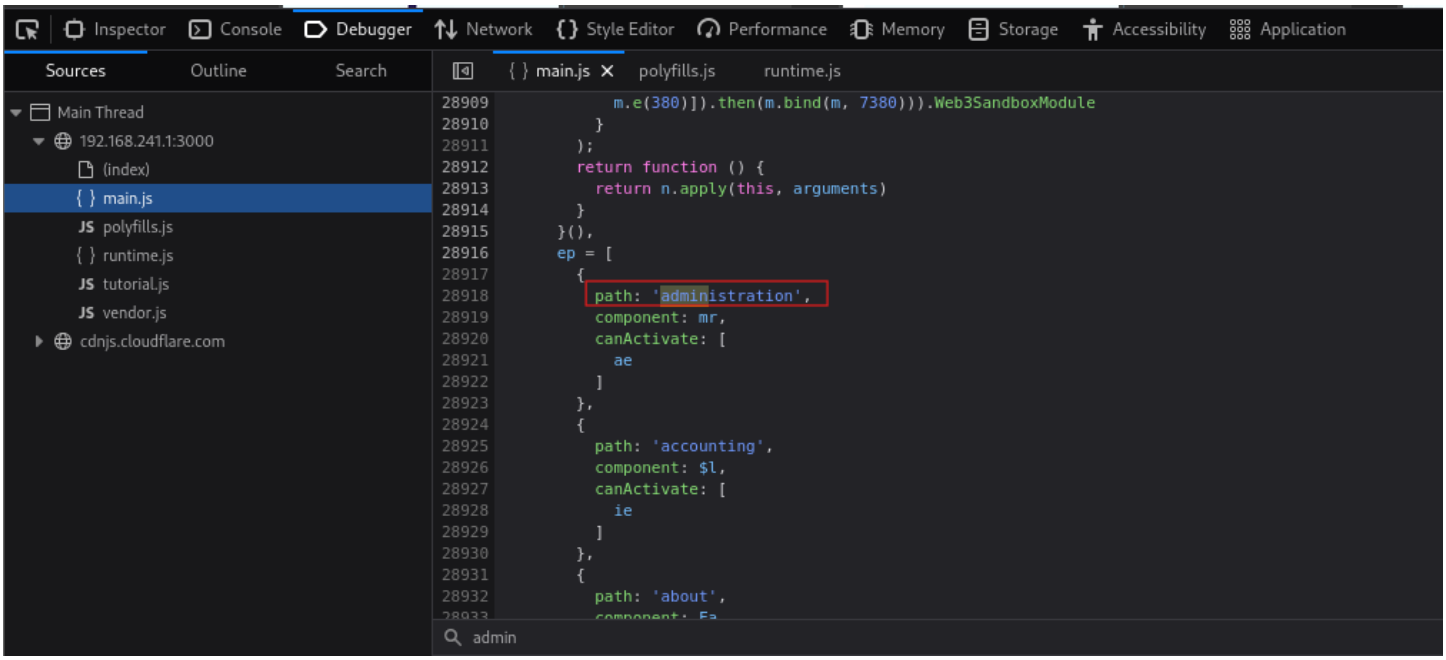
Severity:

High

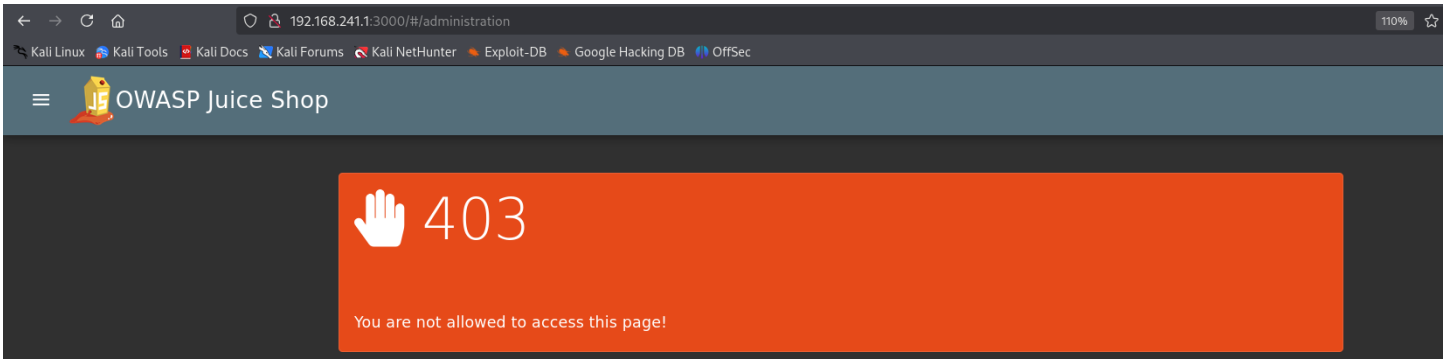
PoC (Steps to Reproduce):

1.Open the main.js file in your browser's developer tools and search for the string "admin".

2.Identify the route mapping with path: "administration".



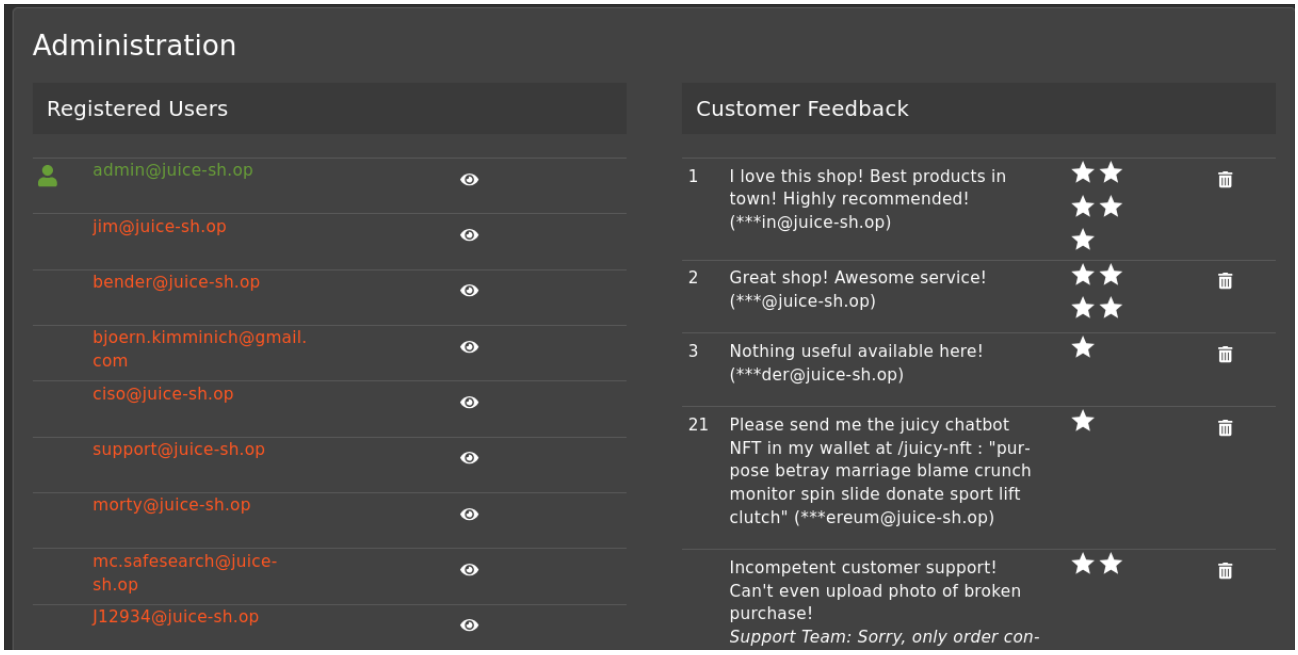
3.Attempt to navigate directly to the administration panel: http://192.168.241.1:3000/#/administration. Observe the 403 Forbidden error, (only if you are an average user)



4.Log in to the application using a valid administrator's account. This can be achieved by:

- As demonstrated in the previous vulnerability, exploiting the SQL injection vulnerability to log in as the administrator.

5.After successfully logging in as an administrator, navigate to http://192.168.241.1:3000/#/administration. Observe that the administration panel is now accessible.



Mitigation:

- **Regular Security Audits:** Conduct regular security audits to identify and address any weaknesses in access control mechanisms.

4) Vulnerability: Improper Input Validation (Admin User Registration)

Executive Summary:

An insecure user registration process allows a regular user to register with administrative privileges by manipulating the role parameter during account creation. This circumvents intended access controls and leads to unauthorized privilege escalation.

Impact:

Successful exploitation of this vulnerability allows malicious users to create administrator accounts, granting them full control over the application. This can lead to severe consequences, including data breaches, unauthorized modifications, and further malicious activities, effectively compromising the entire application's security and integrity.

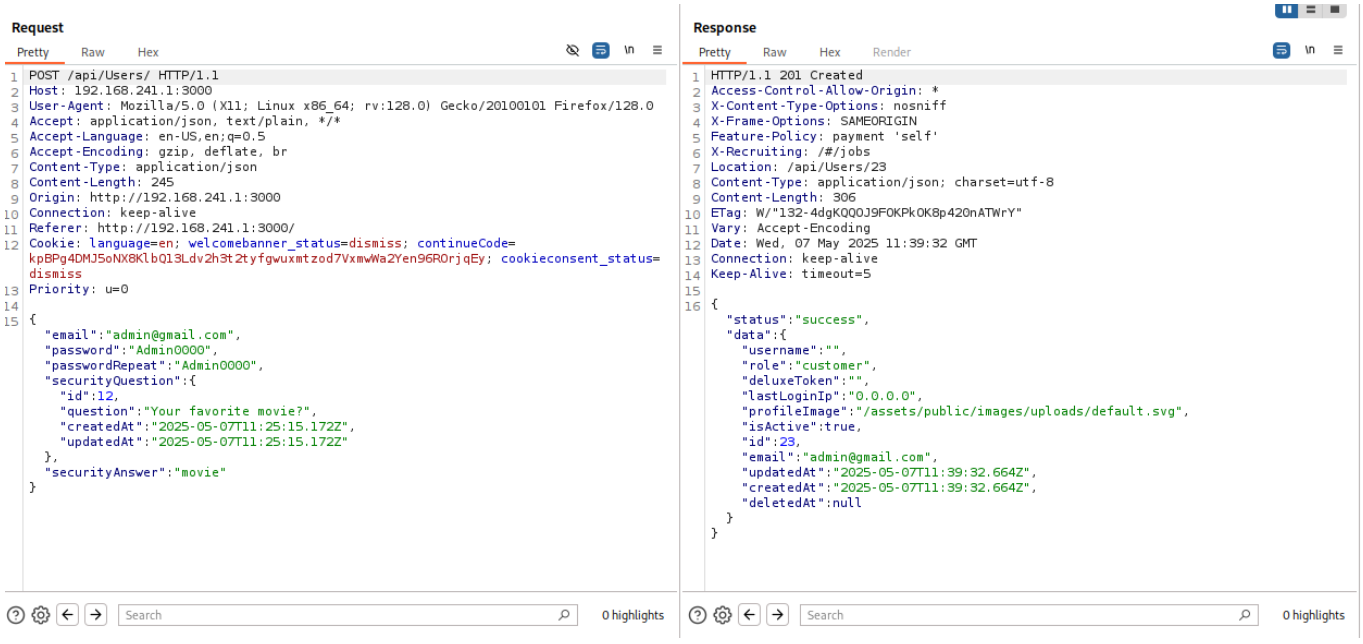
Severity:

Critical

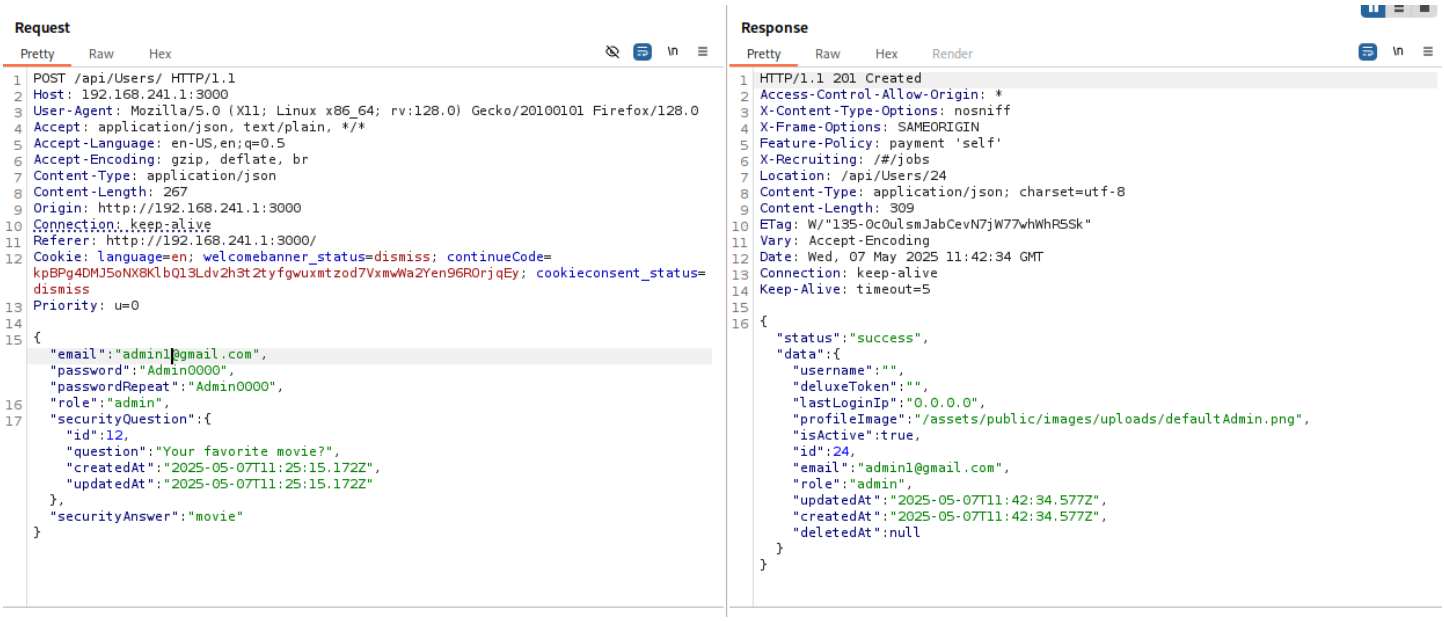
PoC (Steps to Reproduce):

1.Navigate to the registration page: http://192.168.241.1:3000/#/register

2.Submit a POST request to http://192.168.241.1:3000/api/Users



3. Then submit a POST request but send role set to admin and the response will be successful too.



Request

```
POST /api/Users/ HTTP/1.1
Host: 192.168.241.1:3000
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
Content-Type: application/json
Content-Length: 267
Origin: http://192.168.241.1:3000
Connection: keep-alive
Referer: http://192.168.241.1:3000/
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
kpBPg4DMJ5oNX8KlbQ13Ldv2h3t2tyfgwuxmtzod7VxmwWa2Yen96ROrjqEy; cookieconsent_status=
dismiss
Priority: u=0

{
  "email":"admin1@gmail.com",
  "password":"Admin0000",
  "passwordRepeat":"Admin0000",
  "role":"admin",
  "securityQuestion":{
    "id":12,
    "question":"Your favorite movie?",
    "createdAt":"2025-05-07T11:25:15.172Z",
    "updatedAt":"2025-05-07T11:25:15.172Z"
  },
  "securityAnswer":"movie"
}
```

Response

```
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/24
Content-Type: application/json; charset=utf-8
Content-Length: 309
ETag: W/"135-0cOulsmJabCevN7jW77whWhR5Sk"
Vary: Accept-Encoding
Date: Wed, 07 May 2025 11:42:34 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":24,
    "email":"admin1@gmail.com",
    "role":"admin",
    "updatedAt":"2025-05-07T11:42:34.577Z",
    "createdAt":"2025-05-07T11:42:34.577Z",
    "deletedAt":null
  }
}
```

Mitigation:

- **Input Validation on the Server-Side:** While the server should ignore the role parameter, it's still good practice to validate all user input on the server-side to prevent unexpected data from being processed.

5) Vulnerability: **Insecure Direct Object Reference (IDOR)** leads to viewing other user's shopping basket

Executive Summary:

An Insecure Direct Object Reference (IDOR) vulnerability was discovered in the shopping basket functionality. By manipulating the basketId parameter in the API request, a logged-in user can view the contents of another user's shopping basket, leading to potential information disclosure.

Impact:

Exposure of another user's shopping basket contents can lead to the disclosure of sensitive information, such as the items they intend to purchase, quantities, and potentially any discounts applied. This can reveal shopping habits, personal preferences, and could be a precursor to further malicious activities. In some cases, it might even be possible to modify or delete items in another user's basket.

Severity:

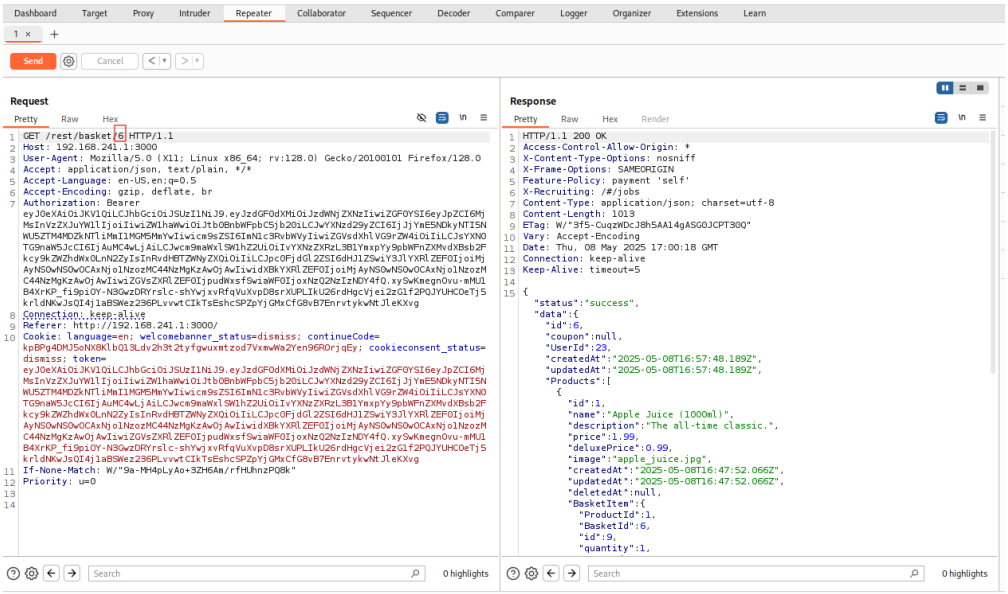
Medium (Information disclosure of shopping basket contents)

PoC (Steps to Reproduce):

1.Log in to the application as any user.

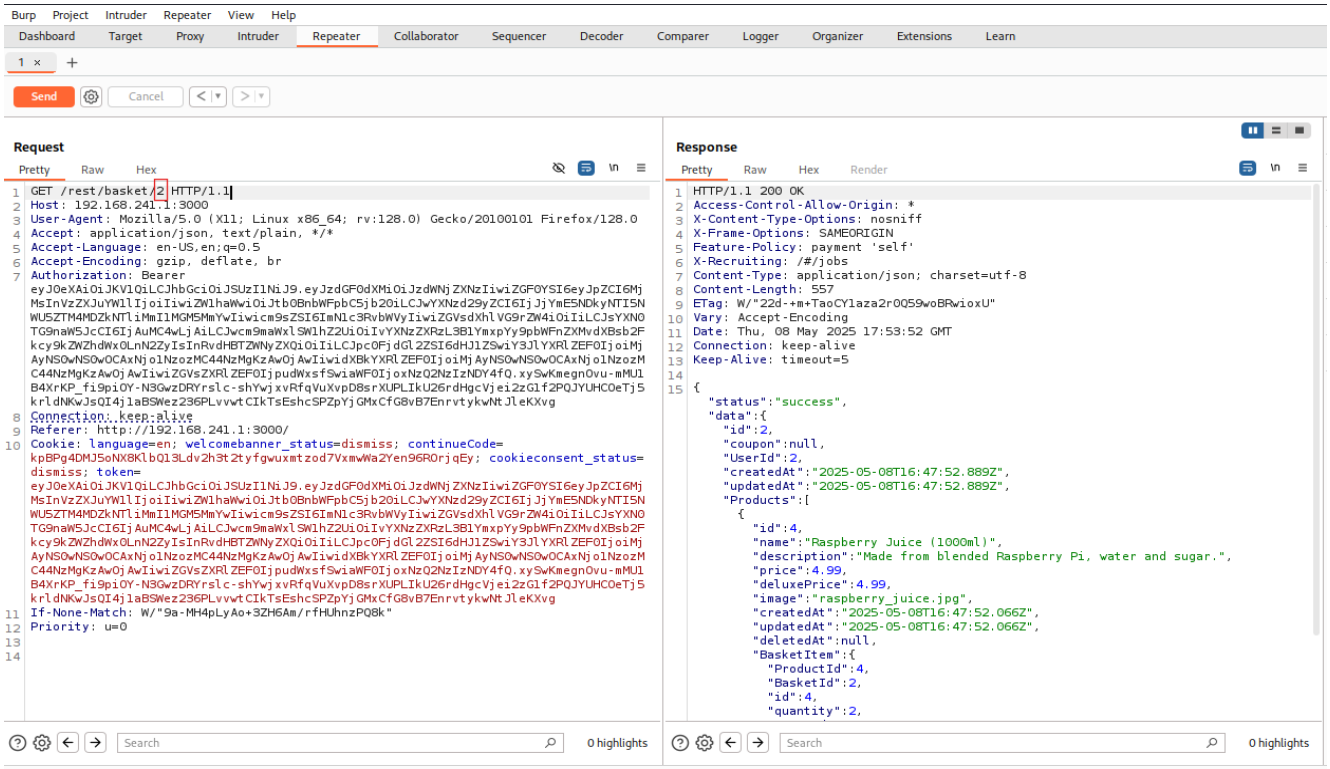
2.Add one or more products to your shopping basket.

3.Navigate to your shopping basket to trigger a request to fetch your basket details. Intercept this request (likely a GET request to /rest/basket/{your_basket_id} where {your_basket_id} is the unique identifier for your basket) using Burp Suite.



4.In the intercepted request, modify the basketId in the URL to the ID of another user's basket.

5.Observe the response. If the response contains the contents of a shopping basket that does not belong to your logged-in user, the IDOR vulnerability is confirmed.



Mitigation:

- **Implement Proper Authorization Checks:** Before returning or manipulating any resource based on a user-provided identifier, the application must always verify that the requested resource belongs to the currently authenticated user. This check should be performed on the server-side.

6) Vulnerability: **Broken Access Control** leads to Forged Feedback (Post some feedback in another user's name)

Executive Summary:

A broken access control vulnerability allows users to forge feedback and post it under another user's name. This can be achieved by manipulating the userId parameter either through a visible field on the "Contact Us" form (after inspecting and modifying the DOM) or directly via the /api/Feedbacks endpoint. This undermines the integrity and trustworthiness of the feedback system.

Impact:

The ability to forge feedback can be exploited to manipulate ratings or reviews, or spread misinformation under someone else's identity. This can damage the reputation of legitimate users, distort the perception of products or services, and erode trust in the feedback system.

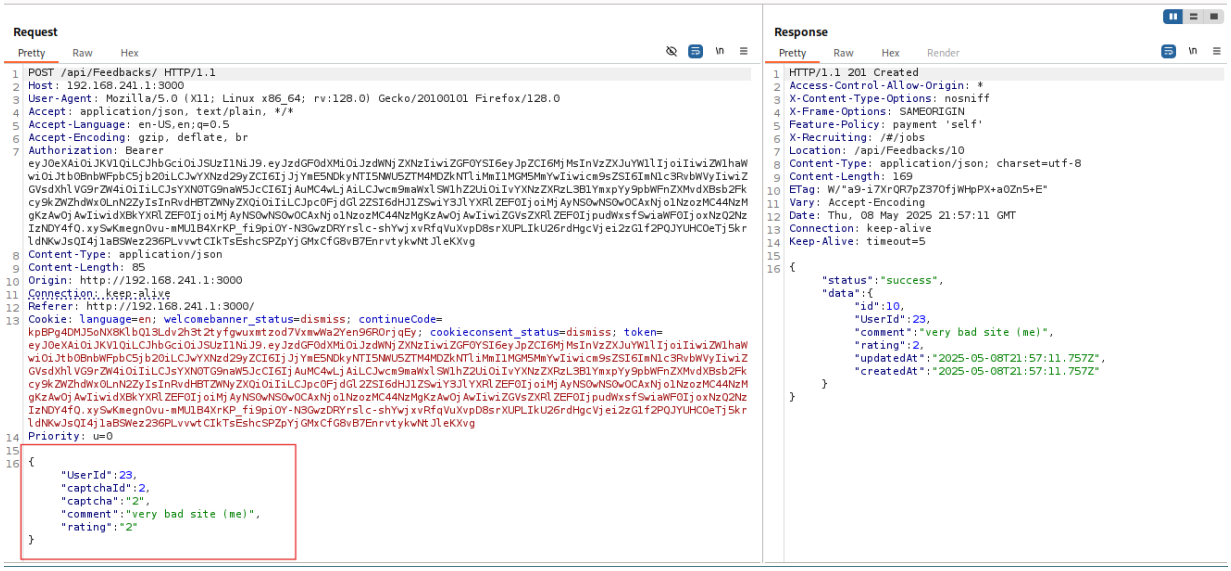
Severity:

Medium (Potential for reputational damage and manipulation of feedback)

PoC (Steps to Reproduce):

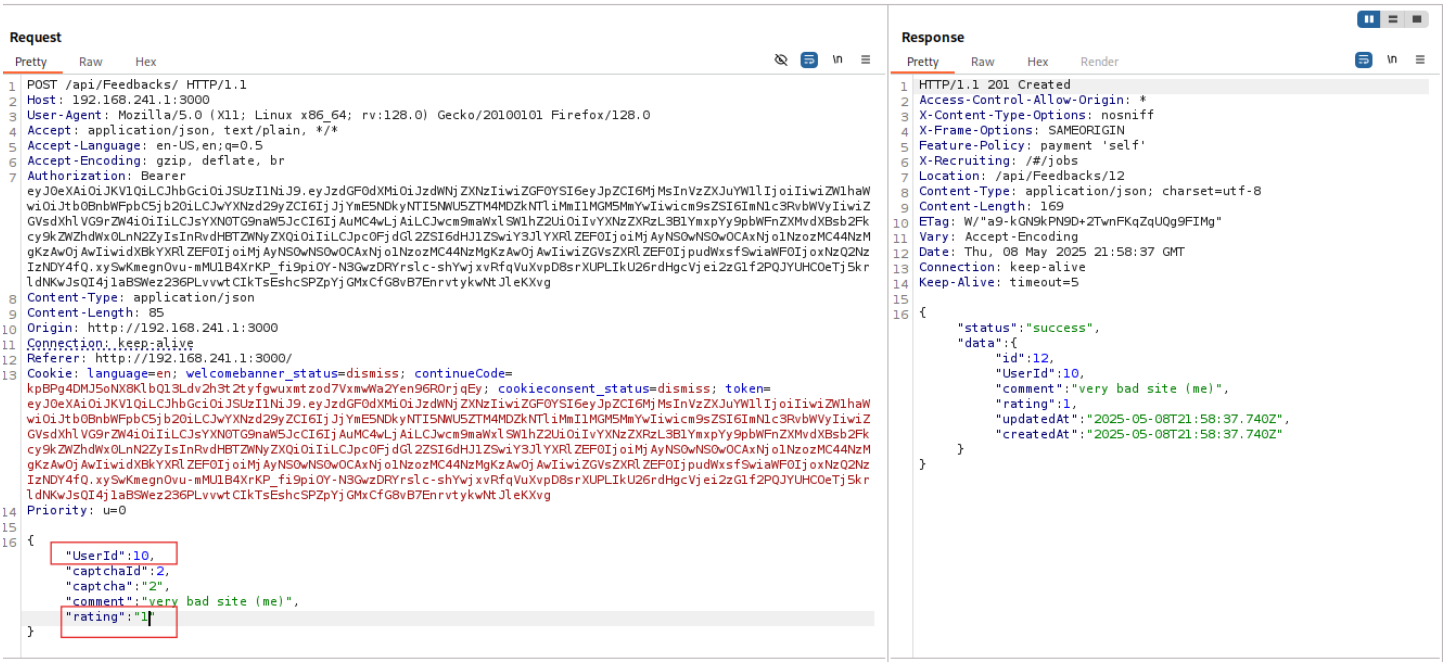
1.Navigate to the Customer Feedback form on http://192.168.241.1:3000/#/contact , and intercept the request.

2.Send a POST request to http://192.168.241.1:3000/api/Feedbacks



3.Observe the 201 Created response from the server.

4.so we can just change the UserId attribute in the request and it will send a feedback with other user's name



Request

Pretty Raw Hex

```
POST /api/Feedbacks/ HTTP/1.1
Host: 192.168.241.1:3000
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.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYWllIjoiIiwiZW1haWwiOiJtb0BnbWFpbC5jb20iLCJwYXNzd29yZCI6IjJjYmE5NDkyNTI5NWU5ZTM4MDZkNTliMmI1MGM5MmYwIiwicm9sZSI6ImNlc3RvbWVyIiwiZGVsdXhlVG9rZW4iOiIiLCJsYXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJlZSwiY3JlYXRlZEF0IjoiMjAyNS0wNS0wOCAxNjolNzozMC44NzM gKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNS0wNS0wOCAxNjolNzozMC44NzMgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzQ2NzIzNDY4fQ.xySwKmegnOvu-mMUlB4XrKP_fi9piOY-N3GwzDRYrslc-shYwjxvRfqVuXvpD8srXUPLIkU26rdHgcVjei2zG1f2PQJYUHCOeTj5krldNKwJsQI4j1aBSWez236PLvvwtCIkTsEshcSPZpYjGMxCfG8vB7Enrvtykw NtJleKXvg
Content-Type: application/json
Content-Length: 85
Origin: http://192.168.241.1:3000
Connection: keep-alive
Referer: http://192.168.241.1:3000/
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
kpBPg4DMJ5oNX8KlbQl3Ldv2h3t2tyfgwuxmtzod7VxmwWa2Yen96ROrjqEy; cookieconsent_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYWllIjoiIiwiZW1haWwiOiJtb0BnbWFpbC5jb20iLCJwYXNzd29yZCI6IjJjYmE5NDkyNTI5NWU5ZTM4MDZkNTliMmI1MGM5MmYwIiwicm9sZSI6ImNlc3RvbWVyIiwiZGVsdXhlVG9rZW4iOiIiLCJsYXN0TG9naW5JcCI6IjAuMC4wLjAiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJlZSwiY3JlYXRlZEF0IjoiMjAyNS0wNS0wOCAxNjolNzozMC44NzM gKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNS0wNS0wOCAxNjolNzozMC44NzMgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzQ2NzIzNDY4fQ.xySwKmegnOvu-mMUlB4XrKP_fi9piOY-N3GwzDRYrslc-shYwjxvRfqVuXvpD8srXUPLIkU26rdHgcVjei2zG1f2PQJYUHCOeTj5krldNKwJsQI4j1aBSWez236PLvvwtCIkTsEshcSPZpYjGMxCfG8vB7Enrvtykw NtJleKXvg
Priority: u=0

{
    "UserId":10,
    "captchaId":2,
    "captcha":"2",
    "comment":"very bad site (me)",
    "rating":"1"
}
```

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/12
Content-Type: application/json; charset=utf-8
Content-Length: 169
ETag: W/"a9-kGN9kPN9D+2TwnFKqZqUQg9FIMg"
Vary: Accept-Encoding
Date: Thu, 08 May 2025 21:58:37 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "status":"success",
    "data":{
        "id":12,
        "UserId":10,
        "comment":"very bad site (me)",
        "rating":1,
        "updatedAt":"2025-05-08T21:58:37.740Z",
        "createdAt":"2025-05-08T21:58:37.740Z"
    }
}
```

Mitigation:

- **Server-Side Authentication and Authorization:** The server-side application must securely determine the identity of the user submitting the feedback based on their active session or authentication token. The UserId should be derived from the authenticated user context and not be directly taken from user-provided input (either via a form field or API parameter).

7) Vulnerability: NoSQL Injection in Product Review API

Executive Summary:

The PATCH /rest/products/reviews endpoint is vulnerable to NoSQL injection due to unsanitized input passed to a MongoDB query. The backend function updateProductReviews accepts an id field from the client, which can be manipulated to include MongoDB operators like $ne. This allows attackers to alter the behavior of the database query and target multiple records.

Impact

An authenticated user can inject a crafted payload to update **all product reviews** in the application, regardless of ownership. This could be abused for mass defacement, spam injection, or data corruption.

Severity :

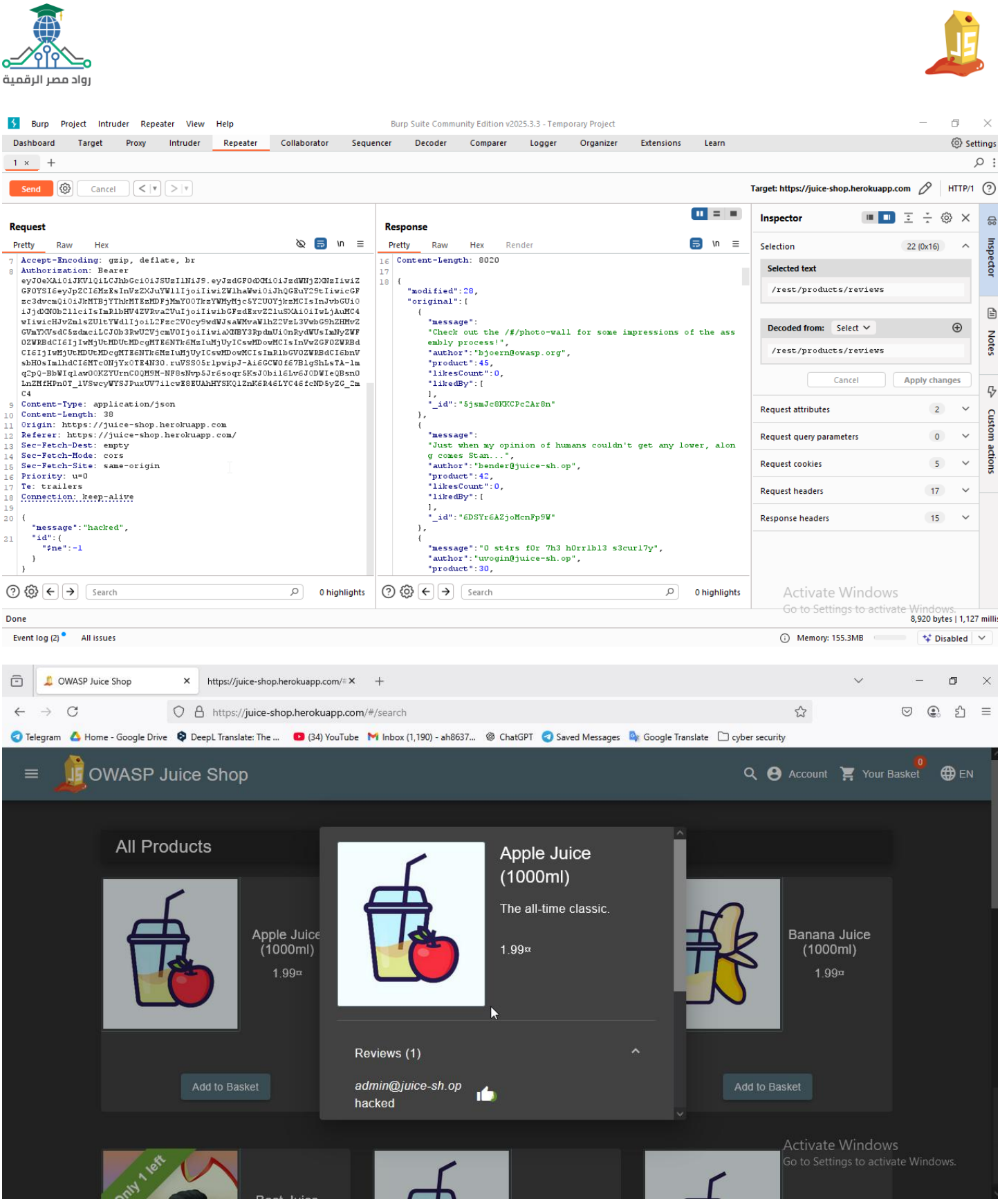
High

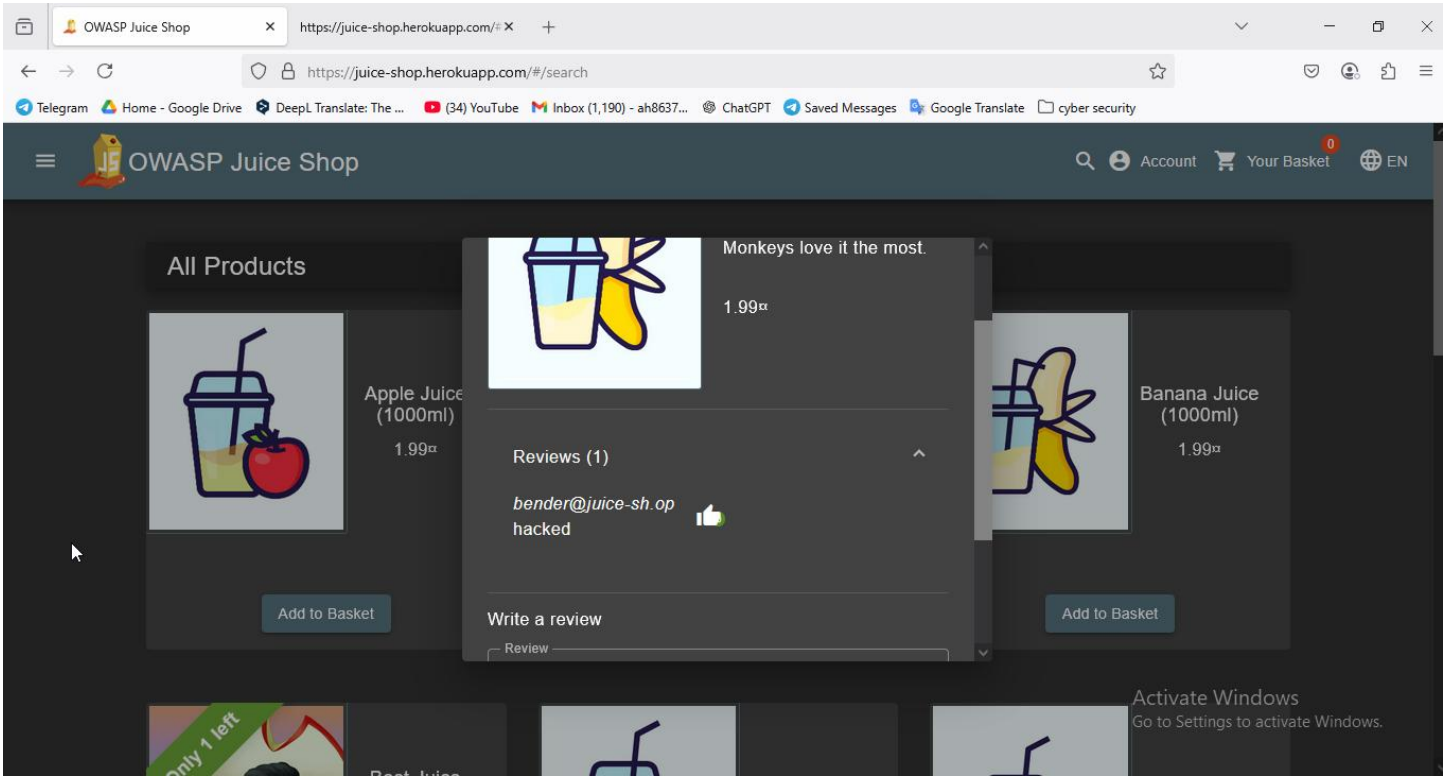
This vulnerability allows mass modification of user-generated content with minimal effort and no elevated privileges.

Proof of Concept (PoC)

Request (PATCH to /rest/products/reviews):

```
{
  "id": { "$ne": -1 },
  "message": "Hacked "
}
```


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.3.3 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Send Cancel
Target: https://juice-shop.herokuapp.com
HTTP/1
Request
Response
Inspector
Selection
Selected text
/rest/products/reviews
Decoded from:
Request attributes
Request query parameters
Request cookies
Request headers
Response headers
Done
Event log (2)
All issues
Memory: 155.3MB
Disabled
OWASP Juice Shop
https://juice-shop.herokuapp.com/#/search
Account
Your Basket
EN
All Products
Apple Juice (1000ml)
The all-time classic.
1.99¤
Reviews (1)
admin@juice-sh.op
hacked
Banana Juice (1000ml)
Add to Basket
Activate Windows
Go to Settings to activate Windows.



Mitigation

- **Validate Input:** Ensure that the id field only accepts primitive, expected values (e.g., integers or ObjectIDs). Do not allow query operators in user input.
- **Use Safe Query Construction:** Cast IDs to appropriate types and avoid using user input directly in queries.
- **Ownership Enforcement:** Implement logic to confirm that the user can only update their own review by matching both userId and reviewId.

Sanitize User Input: Use libraries or ORMs that auto-sanitize input before passing to queries.

8) Vulnerability: Broken Access Control via HTTP Parameter Pollution in Basket Functionality

Executive Summary:

The /api/BasketItems/ endpoint fails to properly enforce access control, allowing an authenticated user to manipulate other users' baskets. The vulnerability is exploited using HTTP Parameter Pollution (HPP). Although directly modifying the basketId to another user's ID did not work, repeating the parameter with both the attacker's own basket ID and another (e.g., basketId=2&basketId=3) resulted in unauthorized actions being performed on the other user's basket. The server processes only the last or a specific occurrence of the repeated parameter, allowing access control to be bypassed.

Impact

An attacker can add products to other users' shopping baskets without authorization. This can be used to confuse users, tamper with their purchases, or manipulate business logic that depends on basket contents (e.g., discounts, promotions, auto-checkouts).

Severity :

High

The vulnerability breaks fundamental access control and affects user-specific resources. It may lead to privilege escalation or unauthorized actions that affect user experience and trust.

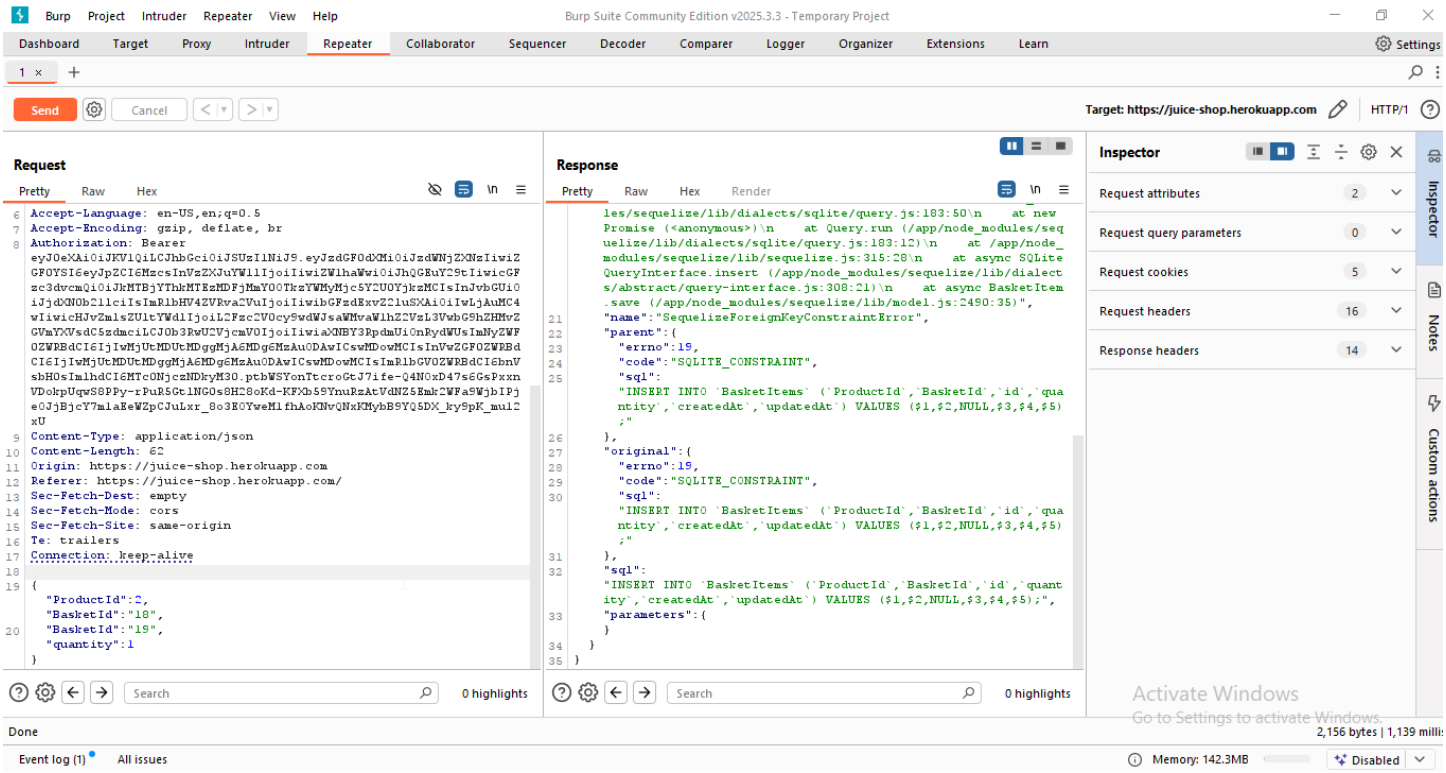
Proof of Concept (PoC)

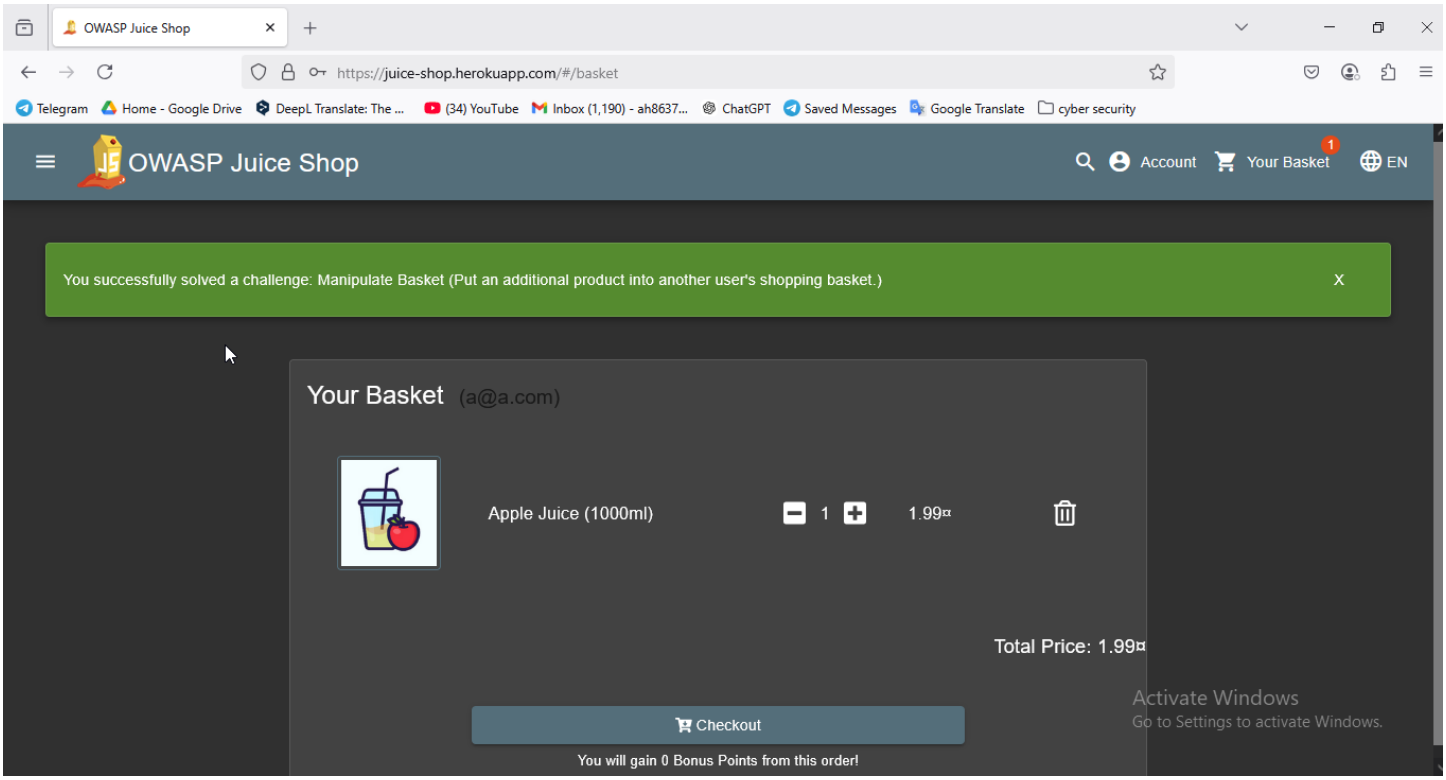
Steps:

1. Add an item to your own basket via /api/BasketItems/ using your valid basketId.
2. Capture the request in Burp Suite.
3. Modify the request to include:

   {"ProductId":2,"BasketId":"18",

   "BasketId":"19","quantity":1}


Burp Project Intruder Repeater View Help
Burp Suite Community Edition v2025.3.3 - Temporary Project
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn
Settings
Send Cancel
Target: https://juice-shop.herokuapp.com
HTTP/1
Request
Pretty Raw Hex
Response
Pretty Raw Hex Render
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 16
Response headers 14
Inspector
Notes
Custom actions
Search
0 highlights
Activate Windows
Go to Settings to activate Windows.
Done
2,156 bytes | 1,139 millis
Event log (1)
All issues
Memory: 142.3MB
Disabled


OWASP Juice Shop
https://juice-shop.herokuapp.com/#/basket
Telegram
Home - Google Drive
DeepL Translate: The ...
(34) YouTube
Inbox (1,190) - ah8637...
ChatGPT
Saved Messages
Google Translate
cyber security
OWASP Juice Shop
Account
Your Basket
EN
You successfully solved a challenge: Manipulate Basket (Put an additional product into another user's shopping basket.)
Your Basket (a@a.com)
Apple Juice (1000ml)
1
1.99¤
Total Price: 1.99¤
Checkout
You will gain 0 Bonus Points from this order!
Activate Windows
Go to Settings to activate Windows.

Result:

The item is added to basket ID 3, not your own, confirming unauthorized access via parameter pollution.

Mitigation

- Reject duplicate parameters on the server side. Only a single occurrence of a sensitive parameter like basketId should be allowed per request.
- Enforce strict server-side access control by validating that the basketId belongs to the authenticated user.
- Use secure coding practices that do not rely on client-supplied identifiers for authorization.
- Implement logging and alerting for anomalies like cross-user basket modifications.

9) Vulnerability: Sensitive Data Exposure via Unprotected Access Log File

Executive Summary:

The application exposes internal log files to unauthenticated users through an unprotected endpoint. By brute-forcing directories using a tool like ffuf, an attacker can discover and access a log file used by the support team. This file contains sensitive internal information, such as user actions, system events, or potentially even IP addresses and other metadata, violating the principle of least privilege and secure data handling.

Impact

An attacker can retrieve internal logs which may include sensitive operational information, user behavior, server messages, and other system insights. This could lead to further exploitation, including:

- Gaining knowledge of internal endpoints or hidden functionality
- Mapping application structure for targeted attacks
- Identifying administrator actions or support workflows

Severity :

Medium to High

While this vulnerability does not directly expose user credentials or execute code, it leaks internal application data that could support more advanced attacks or reconnaissance.

Proof of Concept (PoC)

Tools Used:

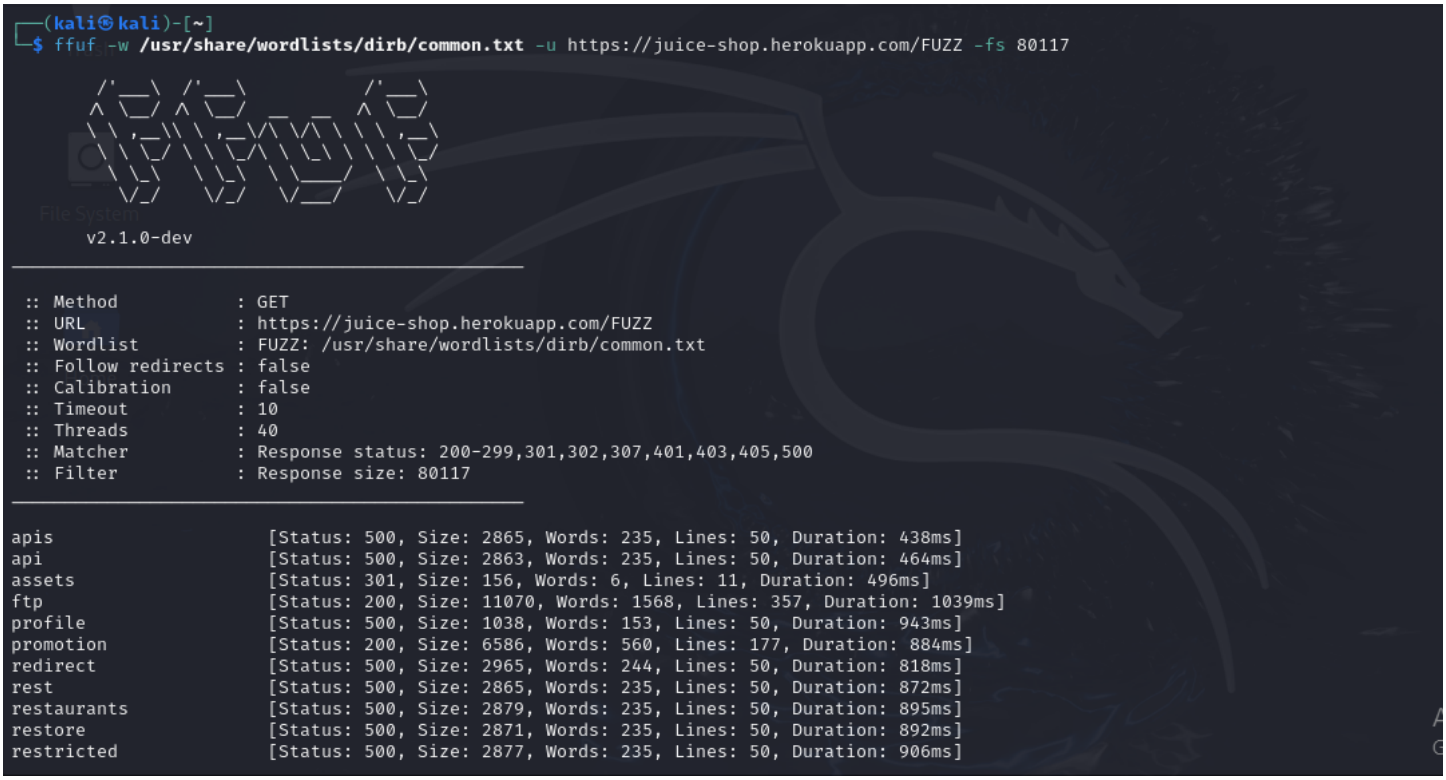
- ffuf (Fast web fuzzer)

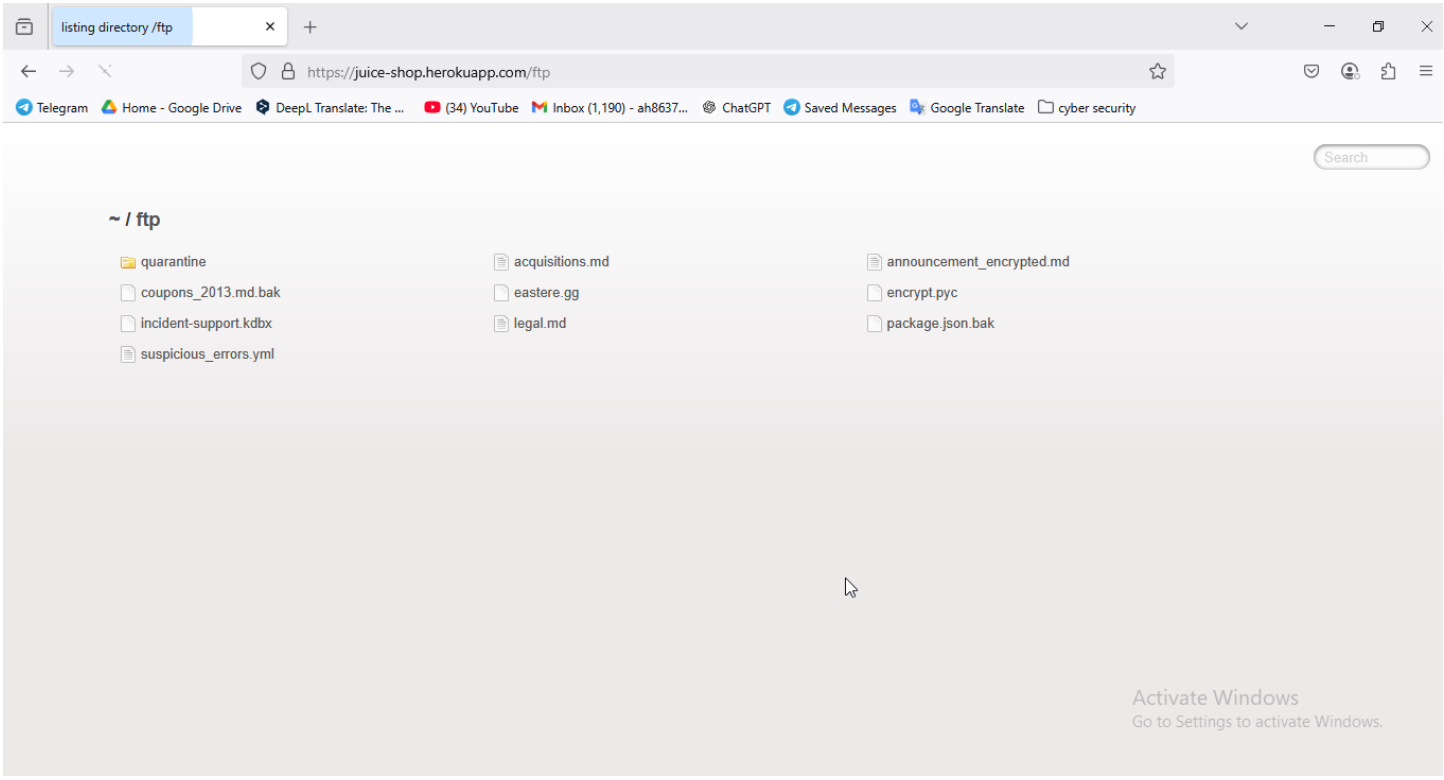
Steps:

1. Use ffuf to brute-force directories on the target domain:

ffuf -u https://juice-shop.herokuapp.com/FUZZ -w /path/to/wordlist.txt

2. Discover the access log file (e.g., /support/logs/access.log).
3. Navigate to the discovered URL in a browser or via curl.






```
┌──(kali㉿kali)-[~]
└─$ ffuf -w /usr/share/wordlists/dirb/common.txt -u https://juice-shop.herokuapp.com/support/FUZZ -fs 80117

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : https://juice-shop.herokuapp.com/support/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response size: 80117
________________________________________________

logs                    [Status: 200, Size: 8889, Words: 1483, Lines: 346, Duration: 270ms]
Logs                    [Status: 200, Size: 8889, Words: 1483, Lines: 346, Duration: 267ms]
routes                  [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 198ms]
rpc                     [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 190ms]
:: Progress: [3519/4614] :: Job [1/1] :: 71 req/sec :: Duration: [0:09:15] :: Errors: 1000 ::
```

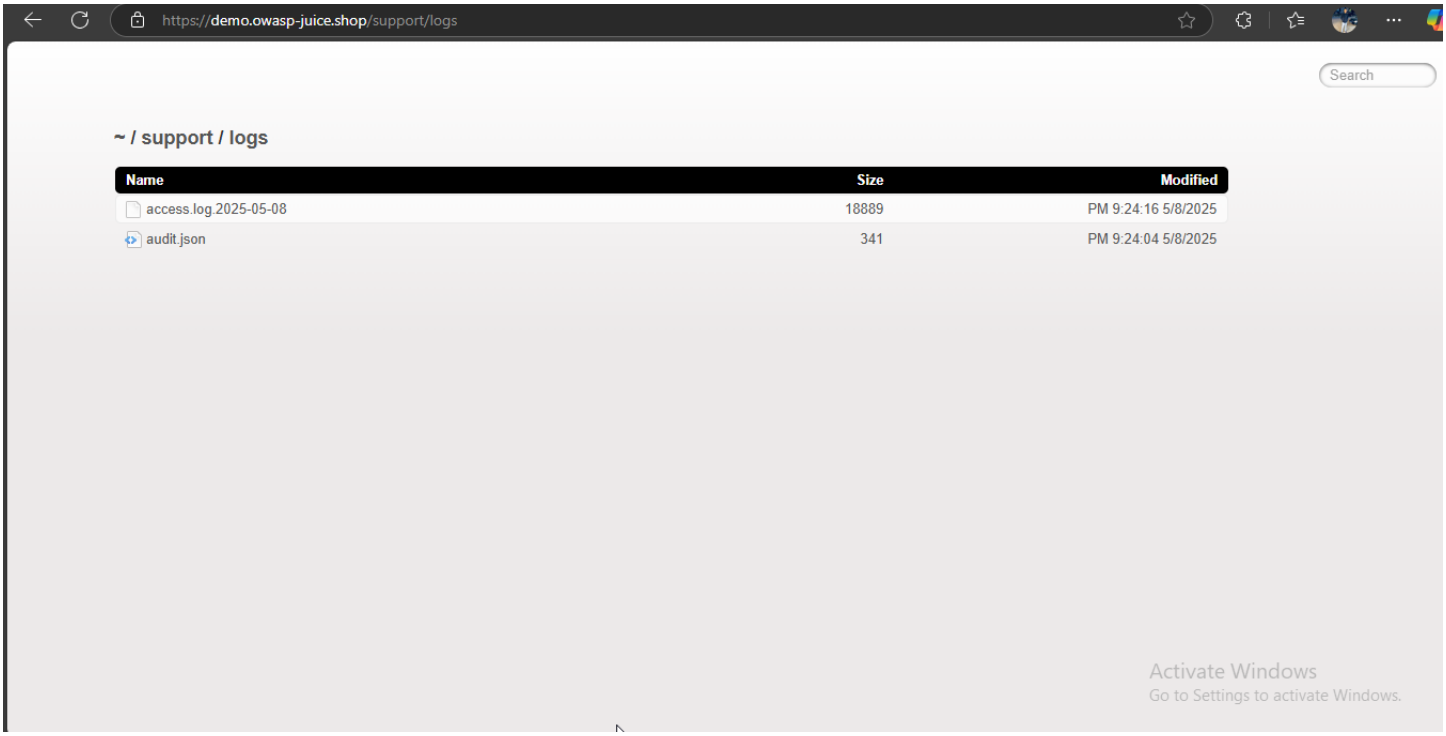
```
┌──(kali㉿kali)-[~]
└─$ ffuf -w /usr/share/wordlists/dirb/common.txt -u https://juice-shop.herokuapp.com/support/FUZZ -fs 80117

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v2.1.0-dev
________________________________________________

 :: Method           : GET
 :: URL              : https://juice-shop.herokuapp.com/support/FUZZ
 :: Wordlist         : FUZZ: /usr/share/wordlists/dirb/common.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200-299,301,302,307,401,403,405,500
 :: Filter           : Response size: 80117
________________________________________________

logs                    [Status: 200, Size: 8889, Words: 1483, Lines: 346, Duration: 270ms]
Logs                    [Status: 200, Size: 8889, Words: 1483, Lines: 346, Duration: 267ms]
routes                  [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 198ms]
rpc                     [Status: 200, Size: 0, Words: 1, Lines: 1, Duration: 190ms]
:: Progress: [3519/4614] :: Job [1/1] :: 71 req/sec :: Duration: [0:09:15] :: Errors: 1000 ::
```



Result:

The server returns a plaintext log file containing sensitive operational data from the support team.

Mitigation

- Restrict access to sensitive files like logs via authentication and authorization.
- Do not store sensitive operational files within the web root or serve them over HTTP.
- Apply server-level controls (e.g., .htaccess rules, Nginx/Apache configs) to deny access to .log, .bak, .env, and similar sensitive file types.
- Implement proper logging practices by storing logs outside of publicly accessible directories and rotating them securely.

10) Vulnerability: Access to Forgotten Developer Backup via Null Byte Injection

Executive Summary:

The application attempts to restrict file access to .md and .pdf files. However, due to improper input validation and reliance on file extension filtering, it is possible to bypass this protection using a **null byte injection** technique. The attacker accessed forgotten developer backup files like coupons_2013.md.bak and package.json.bak by appending a null byte (%00) followed by a valid extension (e.g., .md). The server likely checks only the portion of the filename before the null byte for validation, but the underlying file system or interpreter reads the full filename, granting access.

Additionally, the file was discovered using directory brute-forcing with ffuf, and the file itself contained sensitive information such as legacy coupon codes and website configuration details.

Impact

The attacker gains access to backup files not intended for public consumption. These files may contain:

- Outdated or valid coupon codes
- Development notes or logic
- Application structure and metadata (e.g., from package.json)
  Such information can assist further attacks, including business logic abuse or source code analysis.

Severity :

Medium to High

While this does not lead directly to code execution, exposing internal backups significantly weakens the application's security posture.

Proof of Concept (PoC)

Tools Used:

- ffuf

- Web browser or curl

Steps:

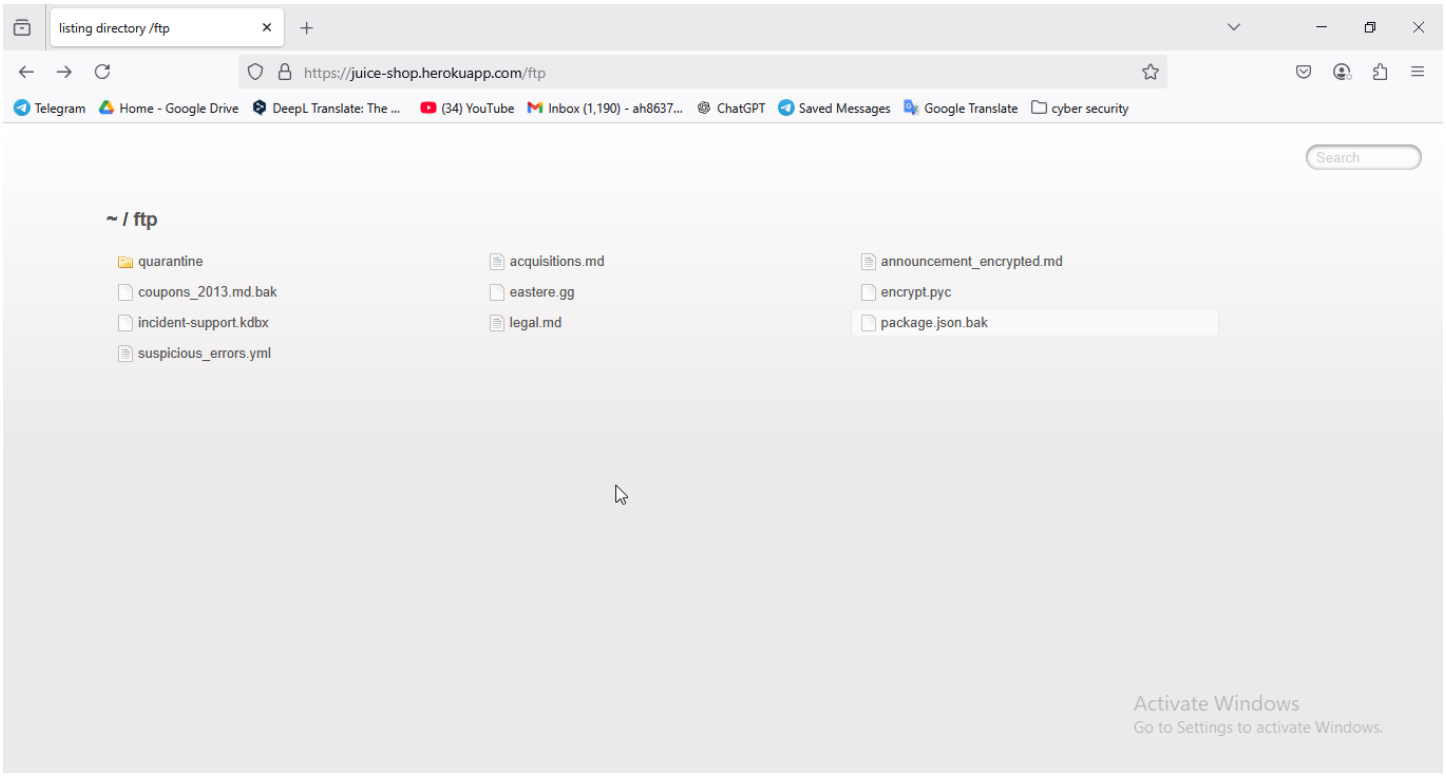
1. Use ffuf to discover hidden files:

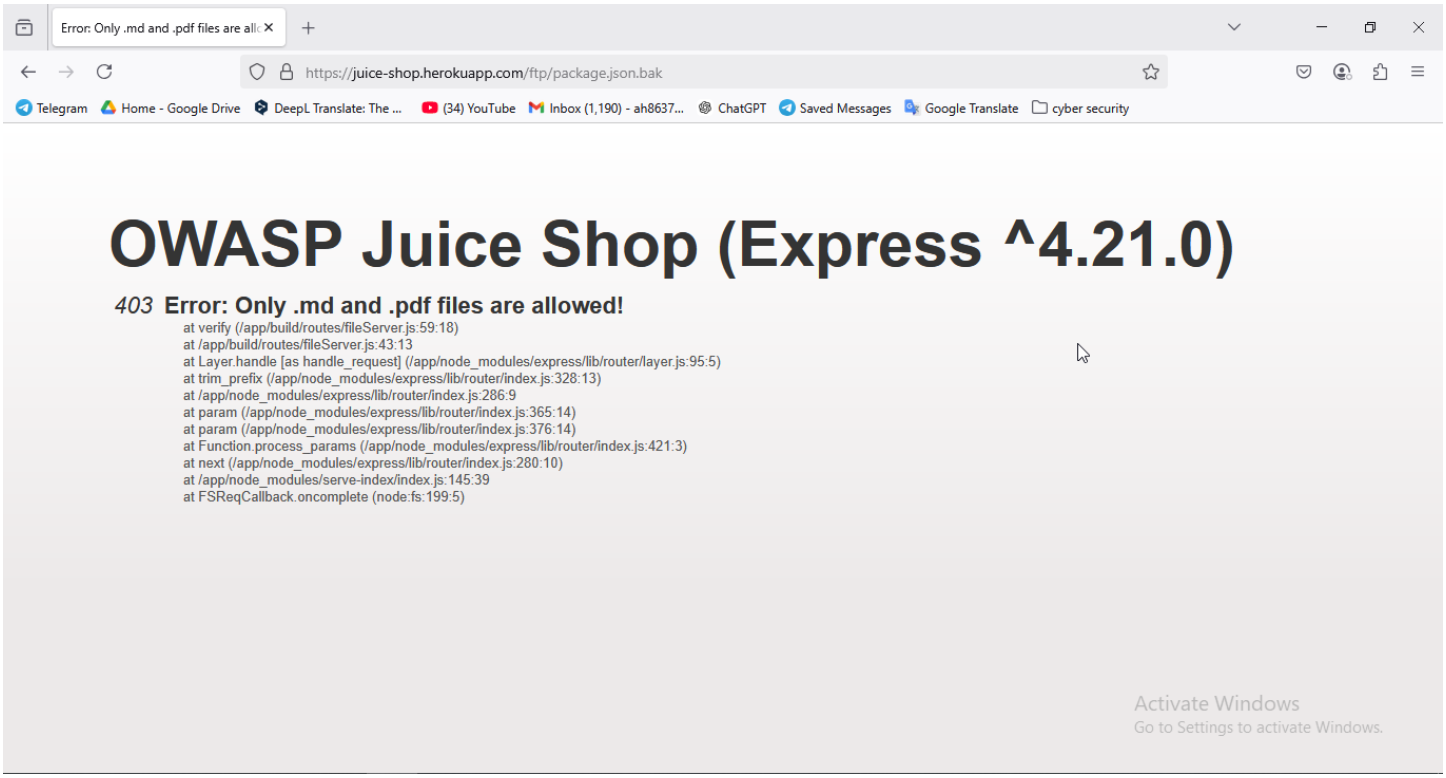
ffuf -https://juice-shop.herokuapp.com/ftp/FUZZ -w /path/to/wordlist.txt

2. Discover a file such as: https://juice-shop.herokuapp.com/ftp/coupons_2013.md.bak
3. Direct access to this file returns a 403 error due to file extension restrictions.
4. Bypass filter using null byte injection:

https://juice-shop.herokuapp.com/ftp/coupons_2013.md.bak%00.md

(Ensure %00 is URL-encoded properly.)





Result:

Server serves the .bak file containing sensitive internal information.

Mitigation

- **Strict File Extension Handling:** Validate file extensions server-side using safe methods that do not rely on simple string matching. Reject filenames containing null bytes or other encoding tricks.
- **Avoid Serving Backup Files:** Backup or temporary files should not be placed within publicly accessible directories.
- **Web Server Configuration:** Configure the server to deny access to any .bak, .zip, .tar, .env, or other common backup/config file extensions.

- **Sanitize Input:** Remove or reject null bytes and other control characters from user-supplied input before processing.

11) Vulnerability: Improper Input Validation in File Upload – Bypassing Allowed File Types

Executive Summary:

The file upload functionality intended to restrict users to uploading specific file types fails to properly validate the file's content and MIME type. While the server likely performs client-side or superficial validation (e.g., checking extensions), it accepts uploads based solely on manipulated request headers. By intercepting the file upload request using Burp Suite, the attacker modified both the file's content and its Content-Type to that of an allowed type, successfully uploading an unauthorized .xml file. This indicates that the server lacks proper server-side validation of file types.

Impact

An attacker can upload disallowed file types such as .xml, which may lead to:

- Storage of unauthorized content on the server
- Attacks like XXE (XML External Entity) if the server processes uploaded XML
- Information disclosure, denial of service, or other logic abuse if unsafe files are used in later workflows

Severity :

Medium

While it doesn't directly lead to code execution or stored XSS in this case, the failure to enforce strict upload rules introduces the risk of further exploitation, especially if the server ever parses uploaded files.

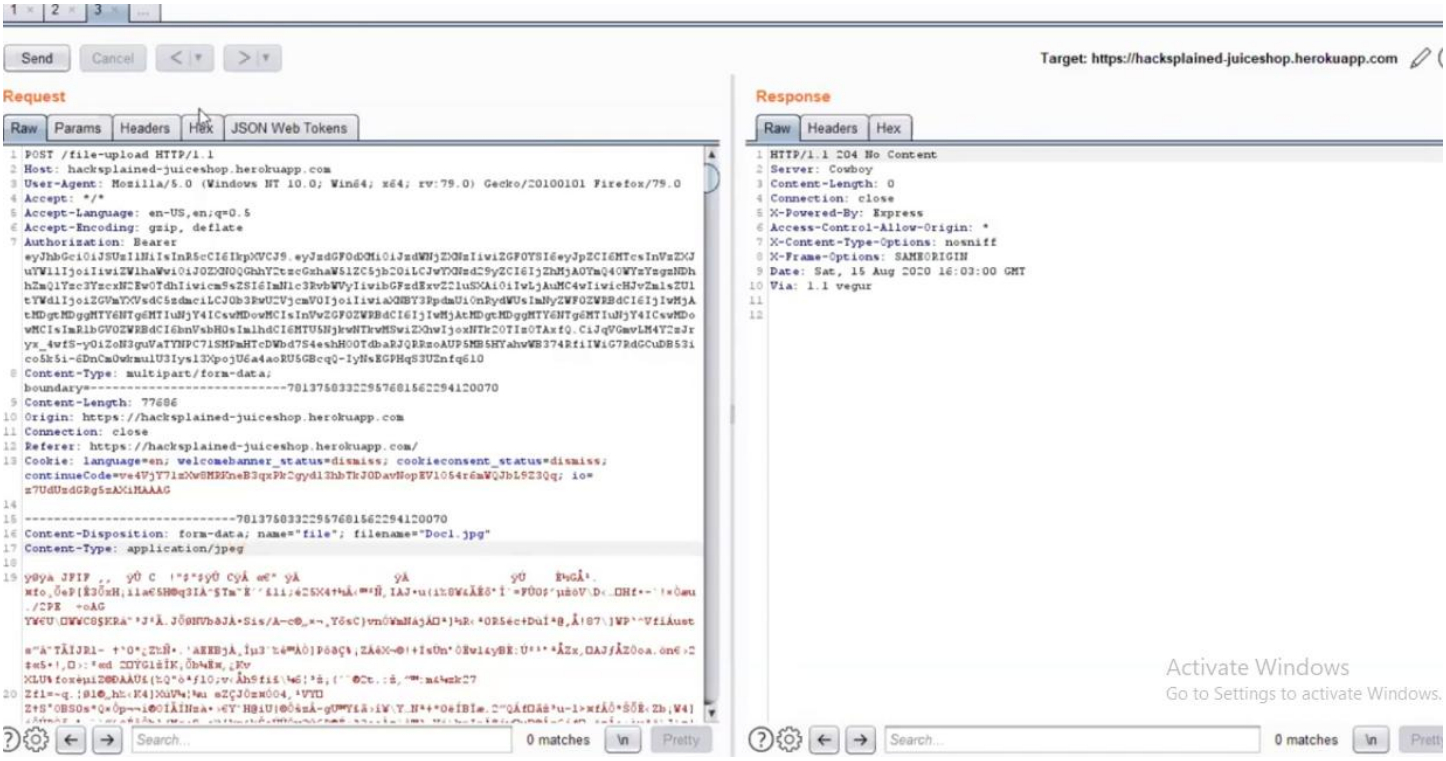
Proof of Concept (PoC)

Tools Used:

- Burp Suite

Steps:

1. Authenticate and access the file upload functionality.
2. Upload a file, intercept the request in Burp Suite.
3. Modify:
    - File extension to .jpeg
    - Content-Type to application/xml or another acceptable type if required
4. Forward the modified request.



Result:

The server accepts and stores the .jpg file, bypassing intended restrictions.

Mitigation

- **Server-Side Validation of File Type:** Use a whitelist of allowed MIME types and file extensions, validated server-side.
- **Content Inspection:** Perform content-based validation (magic number or signature checking) to verify file type regardless of extension or MIME header.

- **Restrict Executable or Dangerous Formats:** Block files like .xml, .html, .js, and others unless specifically required by business logic.
- **Use File Upload Libraries:** Employ secure and tested libraries for file uploads that automatically handle content checking and type enforcement.
- **Sanitize File Names and Paths:** Prevent directory traversal or injection via uploaded file names.

12) Vulnerability: XML External Entity (XXE) via Improper File Upload Validation

Executive Summary:

The application allows uploading .xml files due to weak input validation during file upload. The server accepts and processes the uploaded XML files without disabling external entity resolution, which introduces an **XXE vulnerability**. By uploading a crafted XML payload containing an external entity reference, an attacker can force the server to read internal files, such as /etc/passwd, and include their contents in the XML response (or leak it through error messages or other channels, depending on the parsing behavior).

Impact

This vulnerability allows:

- **File Disclosure** (e.g., reading /etc/passwd, config files)
- **Denial of Service** (via XML bomb payloads)
- **Server-Side Request Forgery (SSRF)** in some cases
- Further exploitation depending on the parser (e.g., RCE in legacy Java/XML libraries)

Severity :

Critical

Full server-side file read through upload and parsing of attacker-controlled XML. It can expose sensitive configuration files and potentially lead to deeper compromise.

Proof of Concept (PoC)

Tools Used:

- Burp Suite

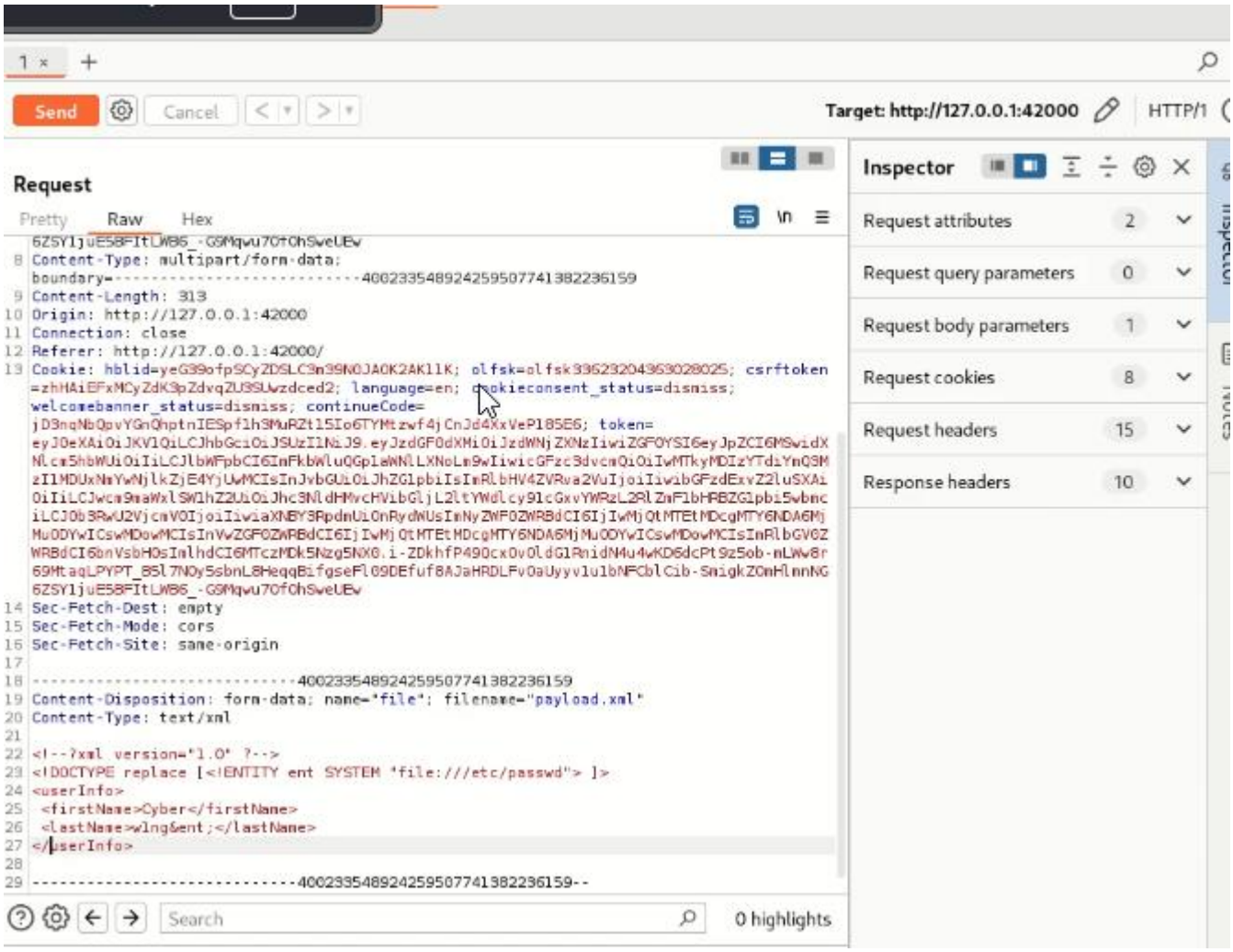
Steps:

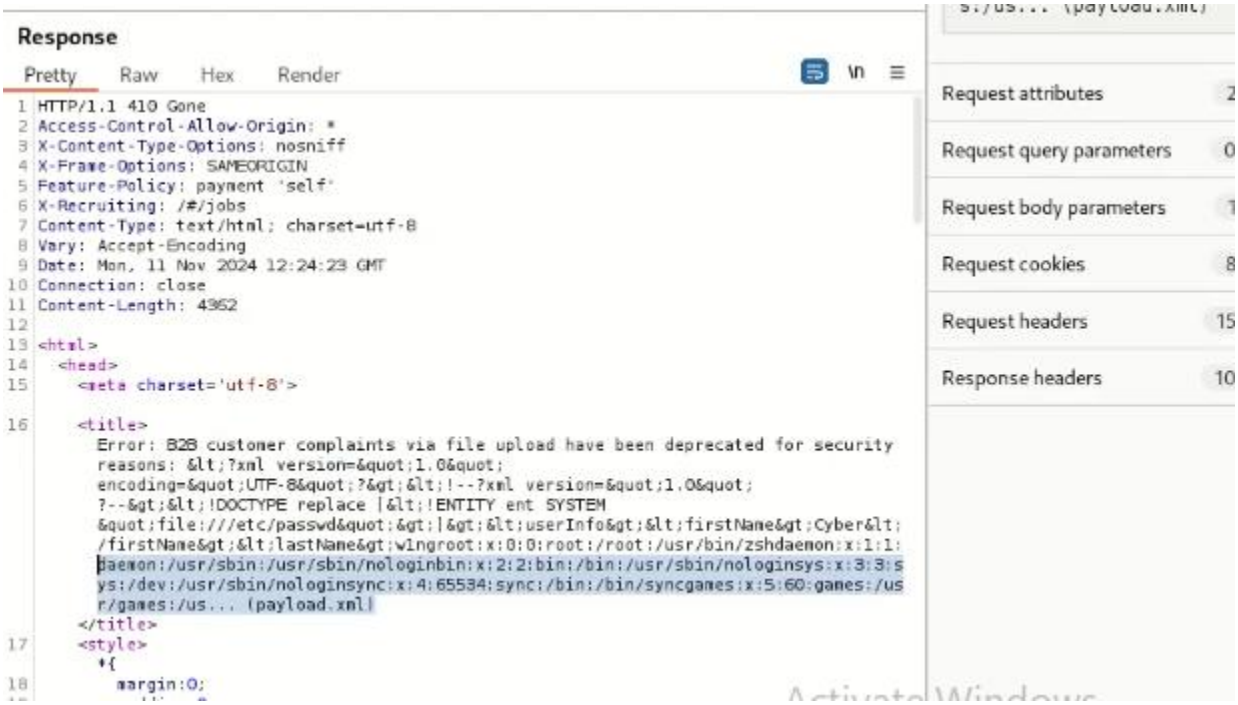
1. Intercept the file upload request.
2. Modify the file to be an .xml file with the following payload:

xml

```
<?xml version="1.0"?>
<!DOCTYPE foo [
 <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<root>
 <data>&xxe;</data>
</root>
```

3. Set the content type to application/xml.
4. Upload and trigger parsing (e.g., via viewing, download, or processing endpoint).


Send
Cancel
Target: http://127.0.0.1:42000
HTTP/1
Request
Pretty
Raw
Hex
Content-Type: multipart/form-data;
boundary=--------------------------400233548924259507741382236159
Content-Length: 313
Origin: http://127.0.0.1:42000
Connection: close
Referer: http://127.0.0.1:42000/
Cookie: hblid=yeG39ofpSCyZDSLC3n39N0JAOK2AK1lK; olfsk=olfsk33623204363028025; csrftoken
=zhHAiBFxMCyZdK3pZdvqZU3SUwzdced2; language=en; cookieconsent_status=dismiss;
welcomebanner_status=dismiss; continueCode=
jD3nqNbQpvYGnQhptnIESpf1h3MuRZtl5Io6TYMtzwf4jCnJd4XxVeP185E6; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MSwidX
NlcmShbWUiOiIiLCJlbWFpbCI6InFkbWluQGpleWNlLXNoLm9wIiwicGFzc3dvcmQiOiIwMTkyMDIzYTdiYmQ3M
zIlMDUxNmYwNjlkZjE4YjUwMCIsInJvbGUiOiJhZGlpbiIsInRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAi
OiIiLCJwcm9maWxlSW1hZ2UiOiJhc3NldHMvcHVibGljL2ltYWdlcy91cGxvYWRzL2RlZmF1bHRBZGlpbi5vbmc
iLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjQtMTEtMDcgMTY6NDA6Mj
MuODYwICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjQtMTEtMDcgMTY6NDA6MjMuODYwICswMDowMCIsInRlbGV0Z
WRBdCI6bnVsbHOsImlhdCI6MTczMDk5NzgSNX0.i-ZDkhfP490cxOvOldG1RnidNMu4wKD6dcPt9z5ob-mLWv8r
69MtaqLPYPT_8Sl7NOy5sbnL8HeqqBifgseFl09DEfuf8AJaHRDLFvOaUyyv1u1bNFCblCib-SnigkZOnHlnnNG
6ZSY1juE5BFItLWB6_-GSMqwu7OfOhSveUEv
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
--------------------------400233548924259507741382236159
Content-Disposition: form-data; name="file"; filename="payload.xml"
Content-Type: text/xml
<!--?xml version="1.0" ?-->
<!DOCTYPE replace [<!ENTITY ent SYSTEM "file:///etc/passwd">]>
<userInfo>
<firstName>Cyber</firstName>
<lastName>wlng&ent;</lastName>
</userInfo>
--------------------------400233548924259507741382236159--
Search
0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request body parameters 1
Request cookies 8
Request headers 15
Response headers 10



Result:

The contents of /etc/passwd are returned in the response or stored within the app.

Mitigation

- **Disable External Entity Parsing:** Configure the XML parser to disallow external entities (e.g., disable DOCTYPE, DTDs, and entity resolution).
- **Use Secure Parsers:** Use modern libraries with XXE protection enabled by default (e.g., defusedxml in Python, SecureDocumentBuilderFactory in Java).
- **Reject XML Uploads if Not Needed:** If XML upload is not a business requirement, block it entirely.
- **Input Validation and Sanitization:** Validate file type both by MIME and content signature. Sanitize and inspect uploaded files.

13) Vulnerability: Payment Bypass in Check-out System

Executive Summary:

The application contains a critical vulnerability in the order checkout process that allows an attacker to bypass payment by manipulating the `PaymentId` parameter. By setting `PaymentId` to `null` in the checkout request, the attacker can complete a purchase without being charged, resulting in financial losses for the platform. This vulnerability exists because of insufficient server-side validation of payment status before order processing.

Impact:

This vulnerability allows attackers to complete purchases without making any actual payment, leading to:

- Direct financial losses for the platform
- Potential abuse by fraudulent users
- Undermining of the business mode

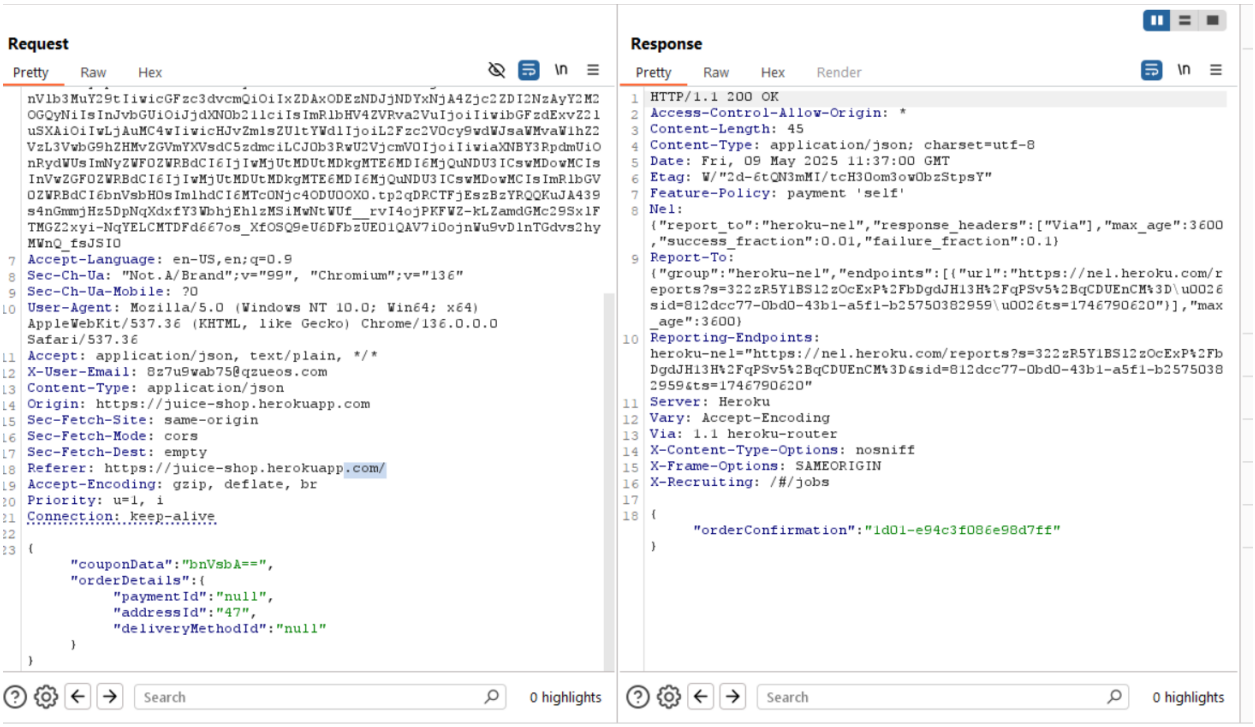
Severity:

High

- Exploitation is straightforward requiring only basic knowledge of HTTP request interception
- No authentication bypass required (assumes user is already authenticated)
- Direct financial impact on the business

PoC (Steps to Reproduce):

1. Add items to your shopping cart on the e-commerce platform and proceed to checkout.
2. Initiate the checkout process and intercept the HTTP request that contains the `PaymentId` using an intercepting proxy like Burp Suite.
3. Modify the `PaymentId` parameter in the intercepted request, setting its value to `null`.

1. 

3.Forward the modified request to the server and complete the checkout process.

4.Confirm that the order is processed and completed without any payment being made.

Mitigation

- Implement strict server-side validation of the `PaymentId` parameter to ensure it is linked to a successful payment transaction before processing any orders.
- Add robust server-side checks to confirm that the payment process is completed and verified before finalizing the order.
- Implement transaction verification with payment processor before order completion.
- Log and monitor all checkout requests to detect and investigate anomalies, such as orders with a null or invalid `PaymentId`.
- Consider implementing a payment workflow that prevents manipulation of payment status

14) Vulnerability: DOM Cross-Site Scripting

Executive Summary:

The website has a security issue called DOM XSS in its search feature. When users type in the search box, the application takes that text and puts it directly into the webpage without checking if it contains harmful code. This happens in the `main.js` file (line 7067) where it uses the unsafe `innerHTML` method

Impact:

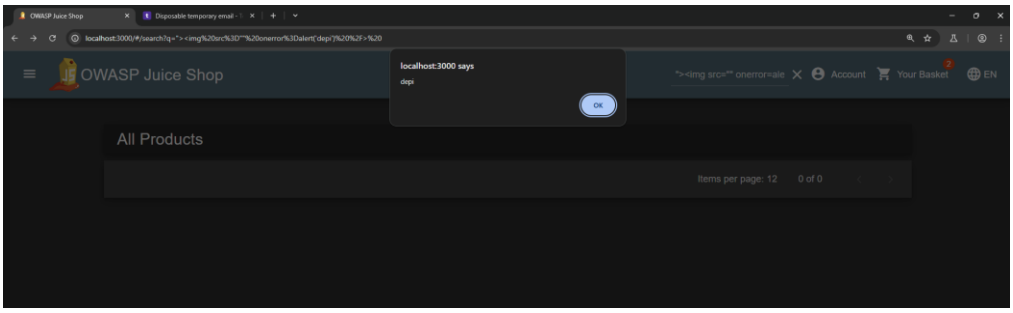
attacker can trick users into running malicious JavaScript code by:

- Stealing their login session
- Taking their private data
- Performing actions as if they were the user
- Changing what they see on the page
- Sending them to fake websites

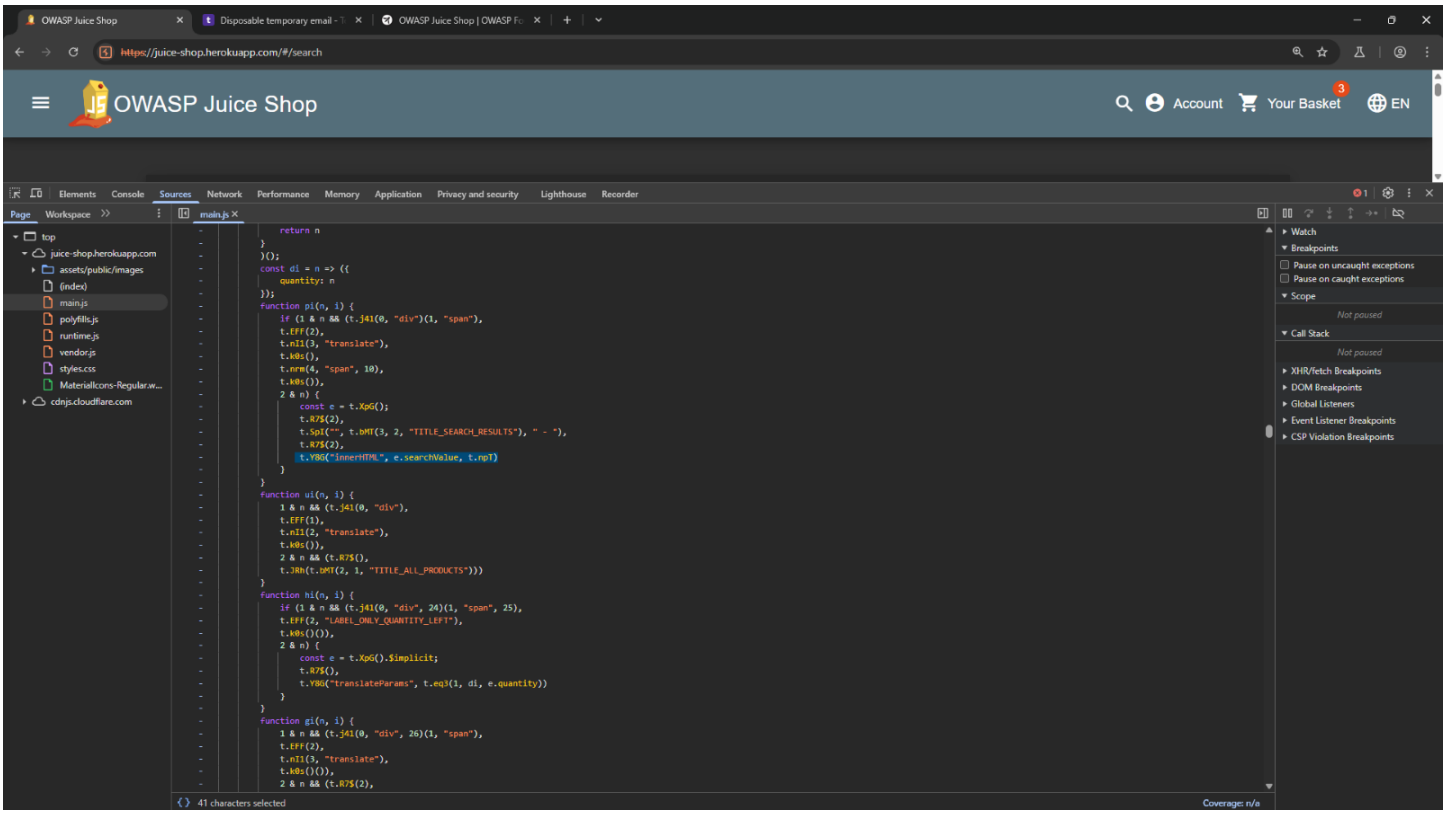
Severity:

High

- Exploitation requires user interaction (victim must visit a crafted URL)
- Affects client-side security rather than server infrastructure
- Potential for session compromise and data theft
- **PoC (Steps to Reproduce)**
- Go to the website's main page
- Add this code to the search parameter in the URL:
- `"><img src="" onerror=alert('depi') />`
- When this URL loads, an alert saying "depi" will pop up, proving the vulnerability works



PoC (Steps to Reproduce):



Mitigation

- Check and clean user input before using it
- Don't use unsafe methods like `innerHTML` - use safer ones like `textContent`
- Encode special characters based on where they'll be used (HTML, JavaScript, URL, etc.)
- Treat all user input as potentially dangerous
- Add a Content Security Policy (CSP) to block unwanted scripts

15) Vulnerability: SQL Injection IN Login Form

Executive Summary:

The login functionality in the OWASP Juice Shop application is vulnerable to SQL injection attacks. By adding SQL syntax to the email field, an attacker can bypass the password verification entirely and log in as any user without knowing their password. In this case, I successfully logged in as the user "jim@juice-sh.op" by manipulating the email field with SQL comment characters.

Impact:

This vulnerability allows attackers to:

- Gain unauthorized access to any user account without knowing passwords
- Bypass authentication mechanisms completely
- Access sensitive user information and functionality
- Potentially escalate to admin privileges depending on targeted users

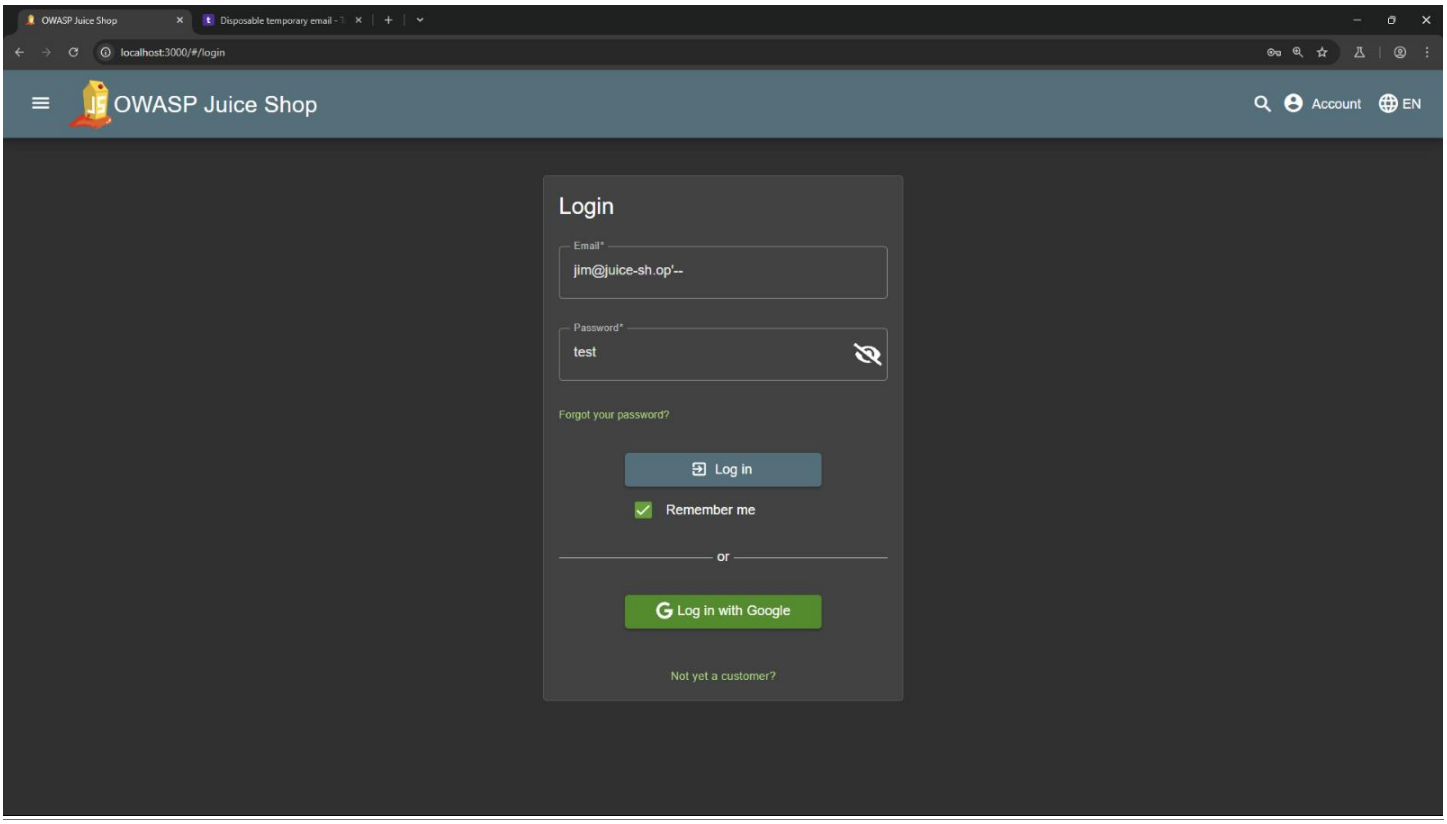
Severity:

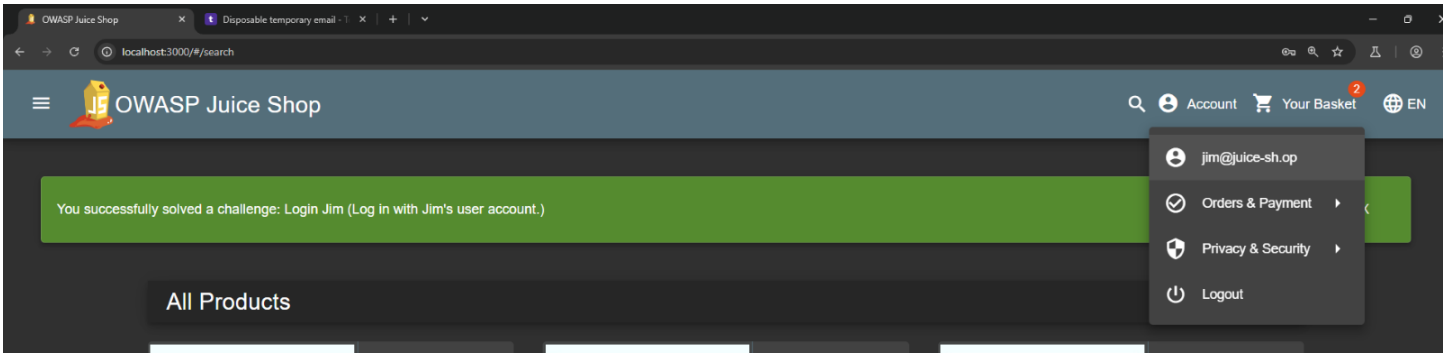
High

- No special privileges required to exploit
- Simple to execute with basic SQL injection knowledge
- Completely bypasses authentication
- Provides full access to user accounts

PoC (Steps to Reproduce):

1. Navigate to the login page of the application (localhost:3000/#/login)
2. In the email field, enter a valid username followed by SQL injection characters: `jim@juice-sh.op'--`





1. Enter any value in the password field (or leave it blank)
2. Click the "Log in" button
3. Observe that you are successfully logged in as user Jim
4. As shown in the screenshots, the application displays "You successfully solved a challenge: Login Jim"

Mitigation:

- Implement prepared statements (parameterized queries) to prevent SQL injection
- Use an ORM (Object-Relational Mapping) framework that handles SQL securely
- Apply input validation to reject suspicious characters in username fields
- Implement proper error handling to prevent SQL error messages from revealing information
- Consider using stored procedures for database operations related to authentication

16) Vulnerability: Security Question Brute Force in Password Reset

Executive Summary:

The application's password reset functionality is vulnerable to brute force attacks against security questions. An attacker can use automated tools like Burp Suite Intruder to systematically guess answers to security questions without any rate limiting or account lockout mechanisms. In this case, the attacker successfully discovered that "Samuel" was the correct answer to Jim's security question, allowing unauthorized password to reset.

Impact:

This vulnerability allows attackers to:

- Reset passwords for any user whose security question answer can be guessed
- Gain unauthorized access to user accounts
- Bypass the intended account recovery process
- Take over accounts without leaving evidence visible to legitimate users

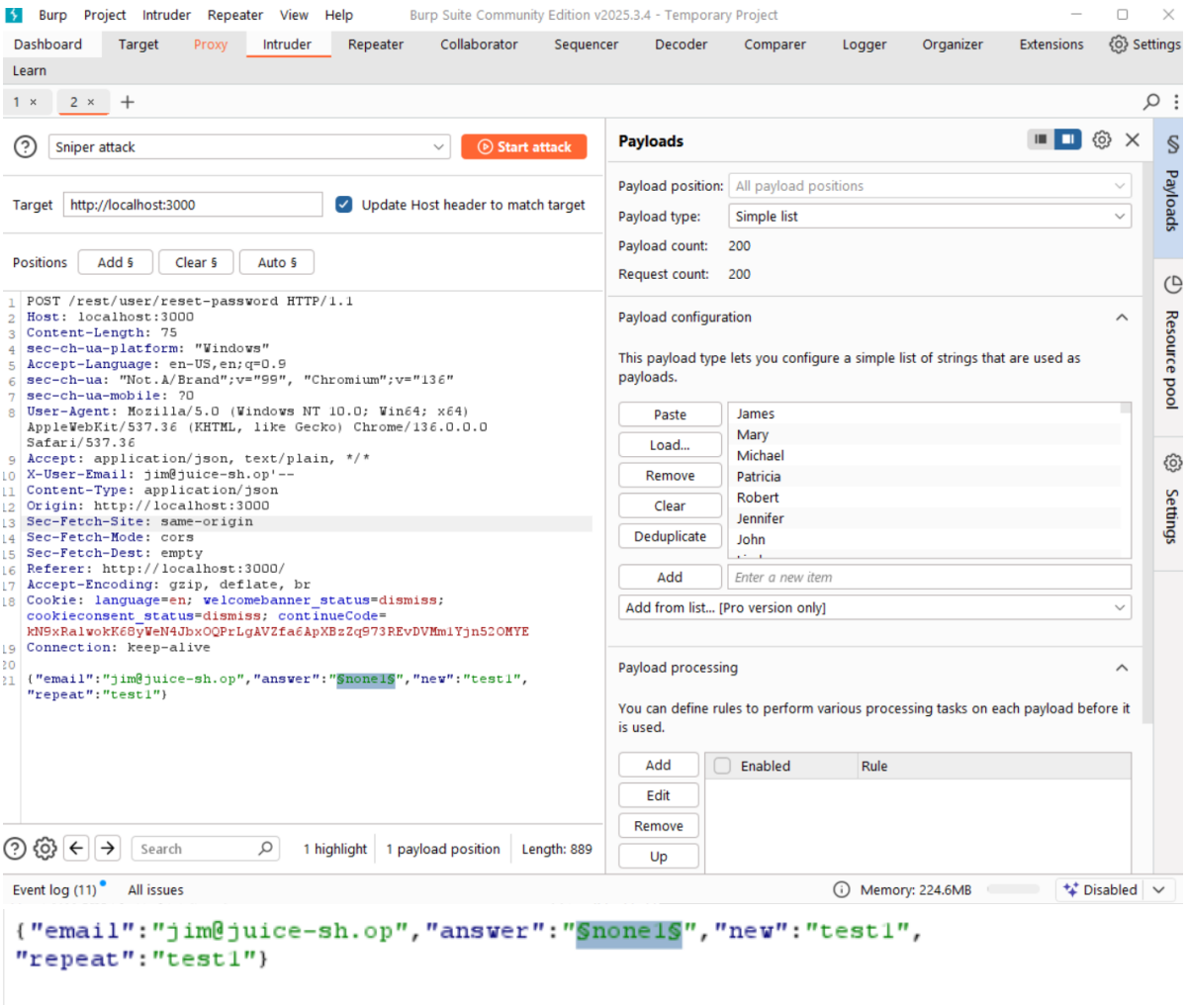
Severity:

High

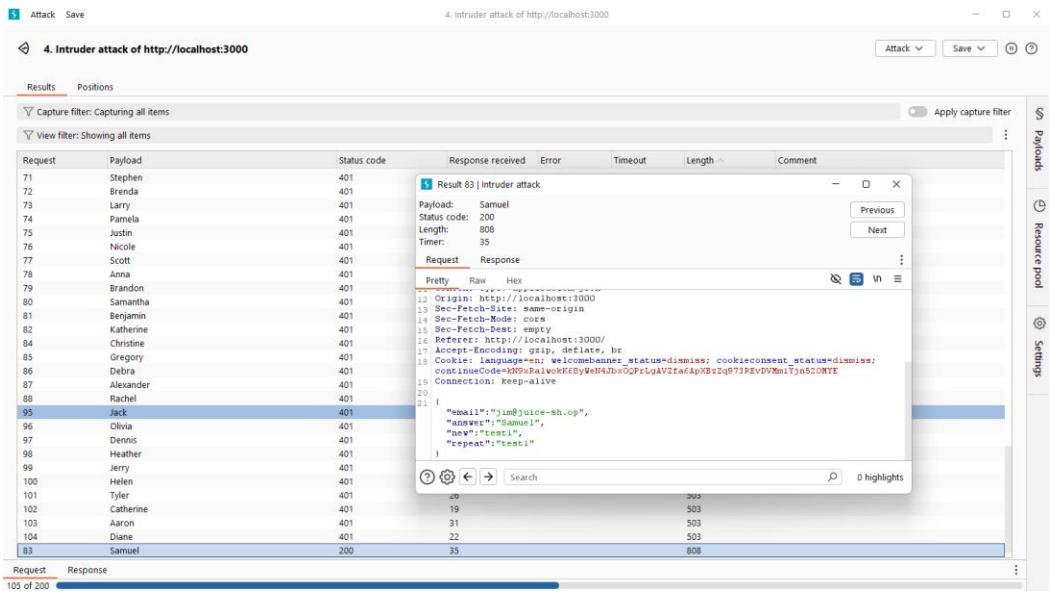
- Requires knowing a valid username/email (jim@juice-sh.op)
- No advanced technical skills needed beyond using common penetration testing tools
- Provides complete account takeover
- No lockout or throttling mechanisms to prevent automated attempts

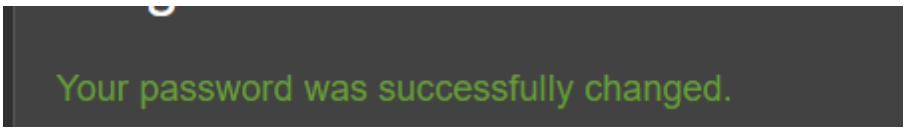
PoC (Steps to Reproduce):

1. Identify a valid user email address (jim@juice-sh.op)
2. Navigate to the password reset functionality
3. Request a password reset for the target email
4. Use Burp Suite Intruder to intercept the password reset request:



{"email":"jim@juice-sh.op","answer":"$none1$","new":"test1","repeat":"test1"}






```
POST /rest/user/reset-password HTTP/1.1
Host: localhost:3000
Content-Type: application/json

{"email":"jim@juice-sh.op","answer":"PAYLOAD_HERE","new":"test1","repeat":"test1"}
```

1. Configure Burp Intruder with a list of common names as payloads
2. Start the attack and monitor responses
3. Identify successful attempts (status code 200) - in this case "Samuel" was the correct answer
4. Use the discovered answer to reset the password to "test1"

Mitigation :

- Implement strict rate limiting on password reset attempts (e.g., max 5 attempts per hour)
- Add CAPTCHA or other human verification after a small number of failed attempts
- Use stronger security questions or implement multi-factor authentication
- Consider temporary account lockouts after multiple failed security answer attempts
- Log and alert on unusual password reset activity patterns
- Require additional verification when changing passwords from new devices/locations

17) Vulnerability: HTTP-Header XSS

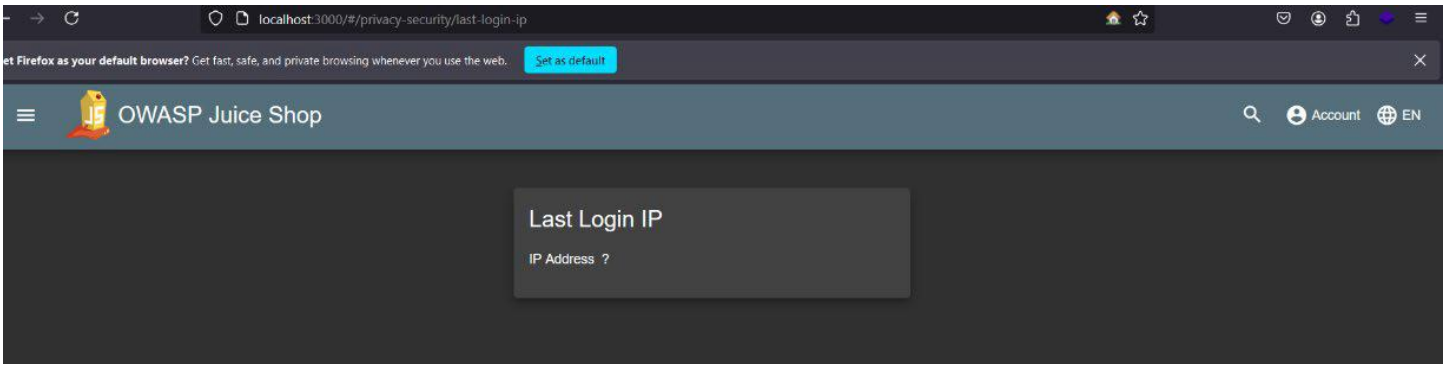
Executive Summary:

HTTP-Header XSS occurs when user-controlled input is reflected in HTTP headers without proper sanitization. This vulnerability allows attackers to inject malicious scripts via headers, which can be executed in the user's browser.

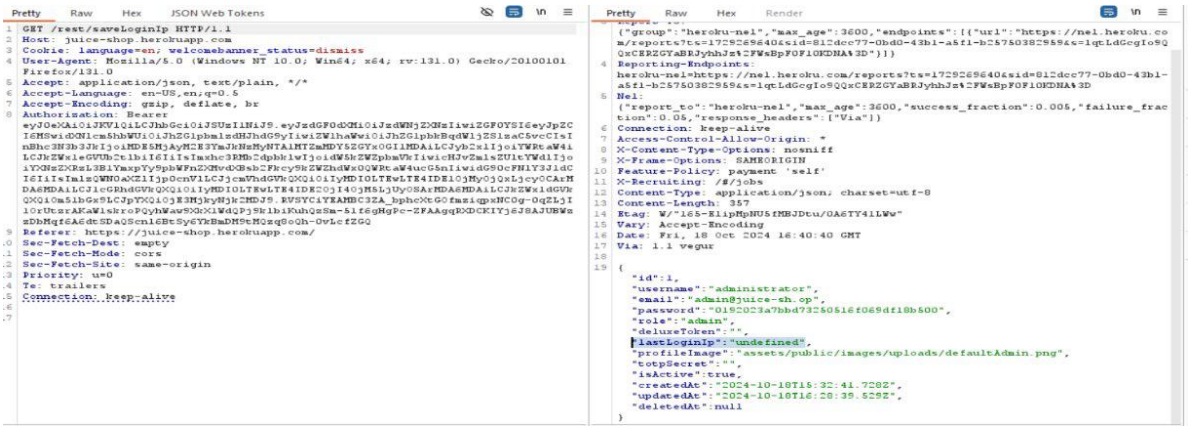
Impact:

- If an attacker successfully exploits this vulnerability, they could:
- - Steal sensitive information like passwords or alert(document.cookie) to steal client's session cookie .
- - Users may be tricked into entering personal information on fake forms.
- - Scripts can redirect users to harmful sites or initiate downloads.
- - Session Hijacking Attackers could impersonate users by stealing their session data.
-

PoC (Steps to Reproduce):



log in to the application. After that, log out to capture the request to the /rest/saveLoginIp endpoint, which is generated during the logout process. When capture the request observe "lastloginIP"

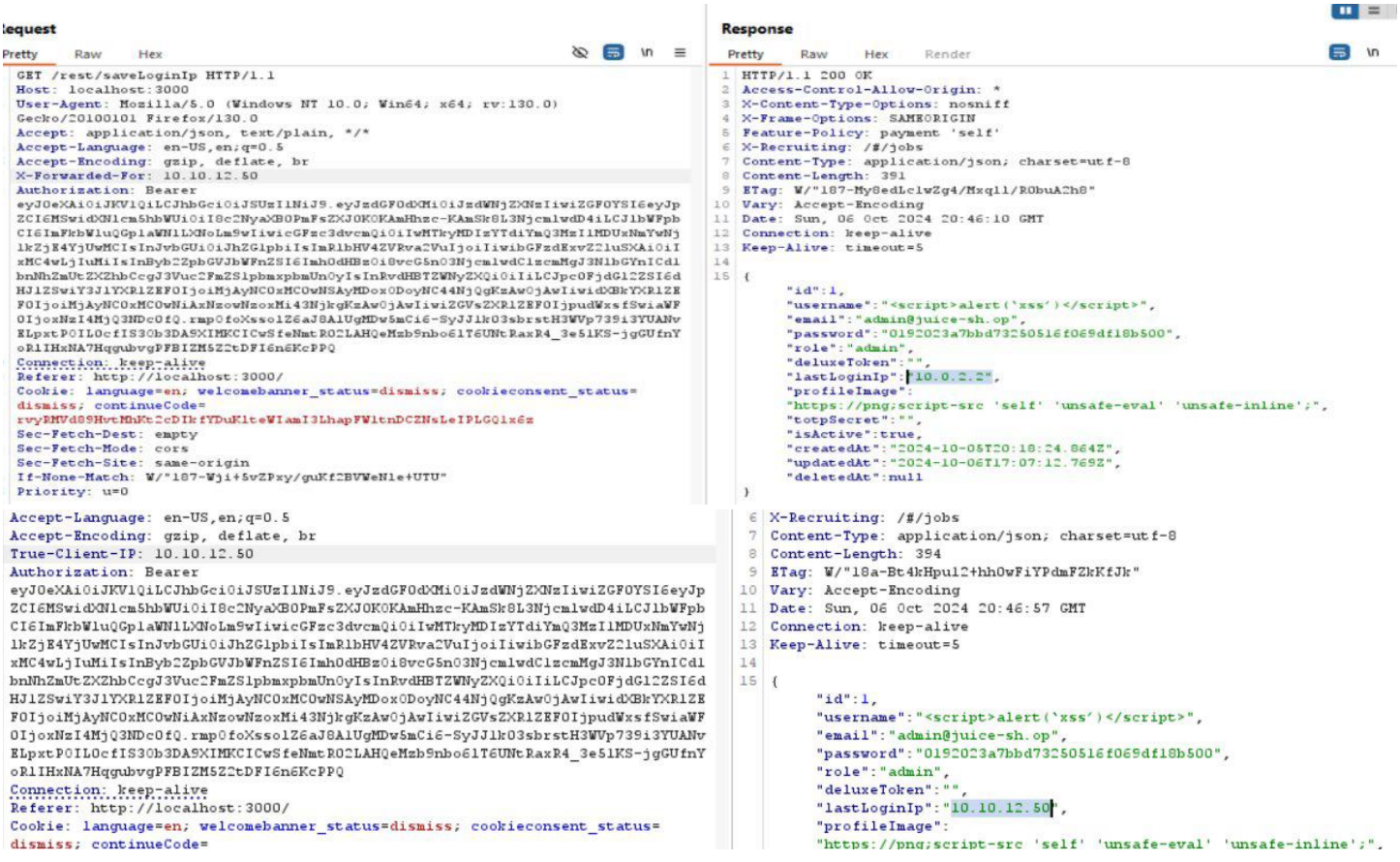


go to search about spoofing client Ip

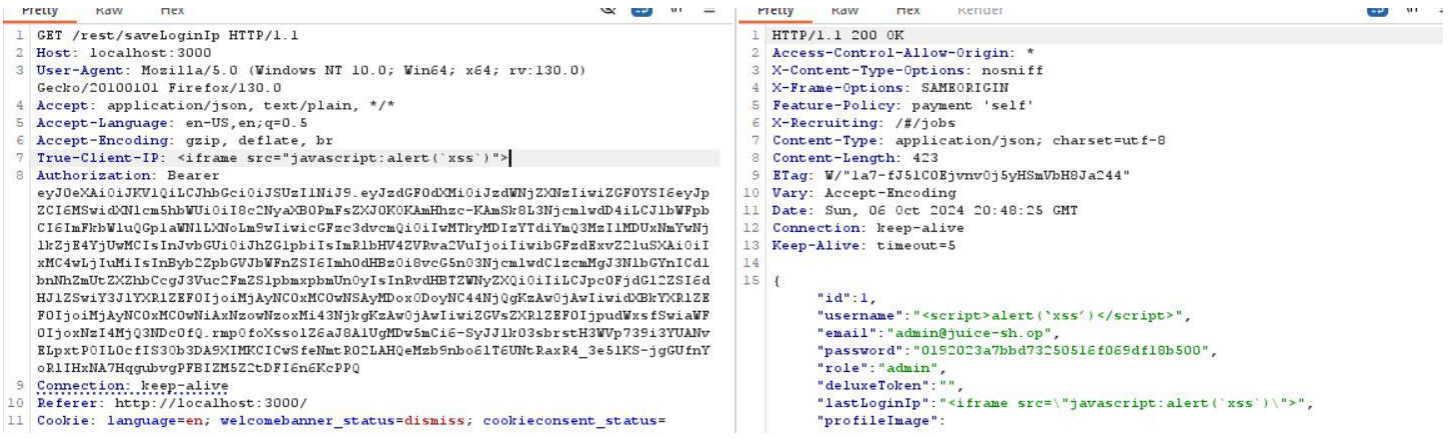
Remediation: Spoofable client IP address

HTTP request headers such as X-Forwarded-For, True-Client-IP, and X-Real-IP are not a robust foundation on which

When using the first “X-Forwarded-For” header (didn’t work)



So let’s inject the following payload: <iframe src="javascript:alert('xss')">



```
GET /rest/saveLoginIp HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0)
Gecko/20100101 Firefox/130.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
True-Client-IP: <iframe src="javascript:alert(`xss`)">
Authorization: Bearer
eyJ0eXAi0iJKV1QiLCJhbGci0iJSUzI1NiJ9.eyJzdGF0dXMi0iJzdWNjZXNzIiwiZGF0YSI6eyJp
ZCI6MSwidXNlcm5hbWUi0iI8c2NyaXB0PmFsZXJ0K0KAmHhzc-KAmSk8L3NjcmlwdD4iLCJlbWFpb
CI6ImFkbWluQGplaWNlLXNoLm9wIiwicGFzc3dvcmQi0iIwMTkyMDIzYTdiYmQ3MzI1MDUxNmYwNj
lkZjE4YjUwMCIsInJvbGUi0iJhZG1pbiIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAi0iI
xMC4wLjIuMiIsInByb2ZpbGVJbWFnZSI6Imh0dHBz0i8vcG5n03NjcmlwdC1zcmMgJ3NlbGYnICdl
bnNhZmUtZXZhbCcgJ3Vuc2FmZSlpbmxpbmUn0yIsInRvdHBTZWNyZXQi0iIiLCJpc0FjdGl2ZSI6d
HJlZSwiY3JlYXRlZEF0IjoiMjAyNC0xMC0wNSAyMDox0DoyNC44NjQgKzAw0jAwIiwidXBkYXRlZE
F0IjoiMjAyNC0xMC0wNiAxNzowNzoxMi43NjkgKzAw0jAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF
0IjoxNzI4MjQ3NDc0fQ.rmp0foXssolZ6aJ8AlUgMDw5mCi6-SyJJlr03sbrstH3WVp739i3YUANv
ELpxtP0IL0cfIS30b3DA9XIMKCICwSfeNmtR02LAHQeMzb9nbo6lT6UNtRaxR4_3e5lKS-jgGUfnY
oRlIHxNA7HqgubvgPFBIZM5Z2tDFI6n6KcPPQ
Connection: keep-alive
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Content-Length: 423
ETag: W/"1a7-fJ51C0Ejvnv0j5yHSmVbH8Ja244"
Vary: Accept-Encoding
Date: Sun, 06 Oct 2024 20:48:25 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "id":1,
    "username":"<script>alert(`xss`)</script>",
    "email":"admin@juice-sh.op",
    "password":"0192023a7bbd73250516f069df18b500",
    "role":"admin",
    "deluxeToken":"",
    "lastLoginIp":"<iframe src=\"javascript:alert(`xss`)\">",
    "profileImage":
```

You successfully solved a challenge: HTTP-Header XSS (Perform a persisted XSS attack with <iframe src="javascript:alert(`xss`)"> through an HTTP header.)

18) Vulnerability: API-only XSS

Executive Summary:

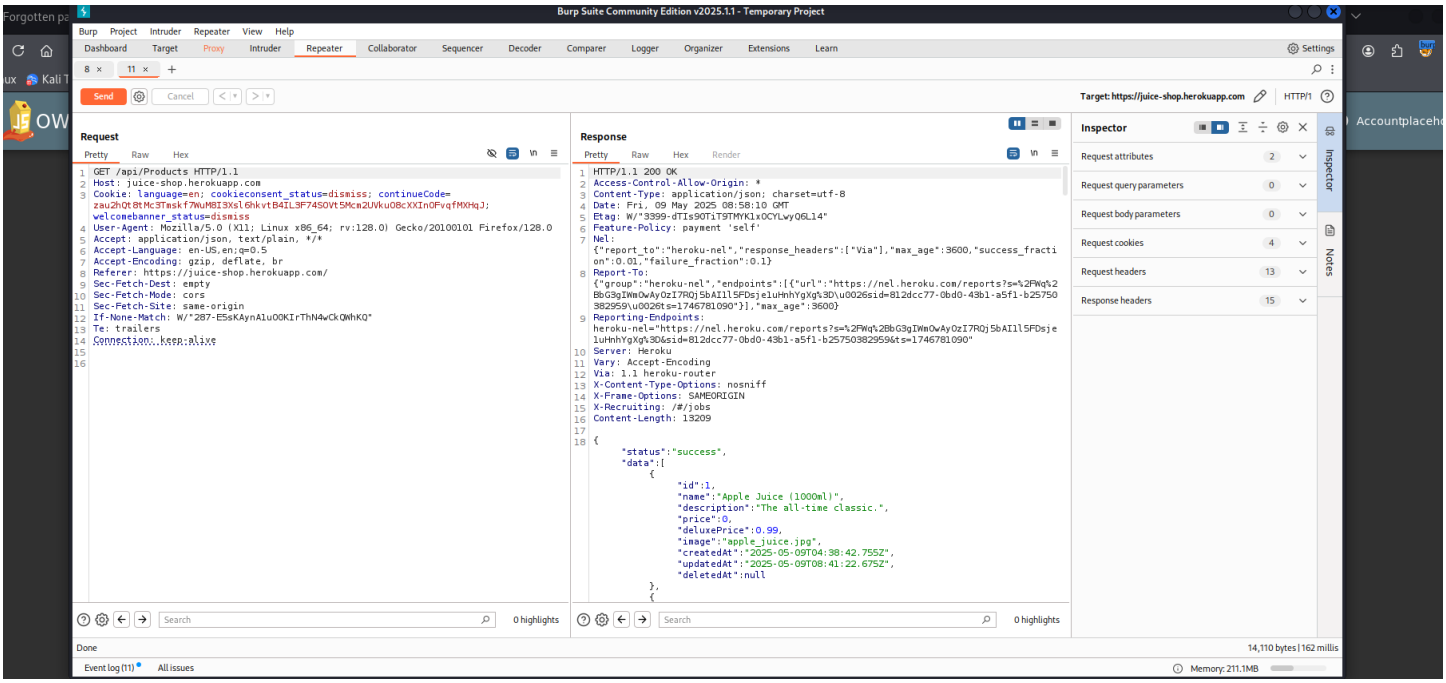
This report identifies a Stored Cross-Site Scripting (XSS) vulnerability in the Juice Shop application that is triggered through an API-based request. Specifically, the vulnerability is found in the API endpoint /api/Products/6, where an authenticated user is able to inject JavaScript payloads via the product description field. This malicious script is stored on the server and is executed when another user accesses the product page. If exploited, this XSS attack can allow attackers to steal session cookies, hijack user sessions, or perform other malicious actions in the context of an authenticated user.

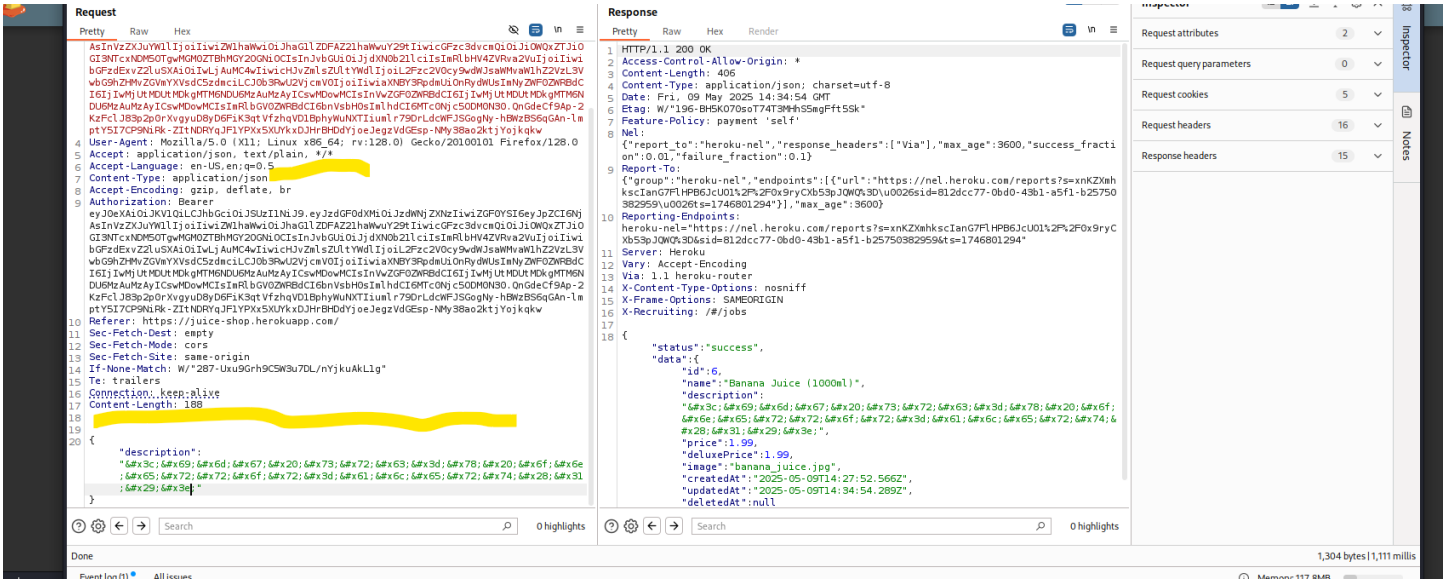
Impact:

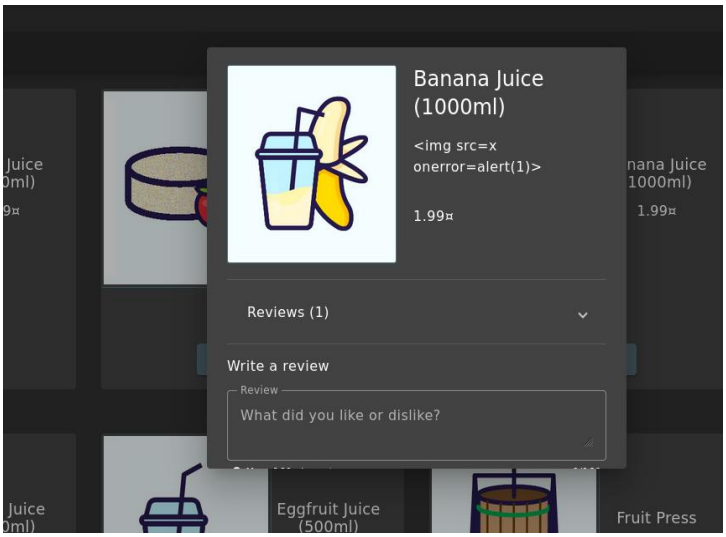
The impact of this vulnerability can be severe:

- **Session Hijacking**: Attackers can inject scripts that steal session cookies or tokens, leading to full account compromise.
- **Data Theft**: Malicious scripts could access sensitive data such as personal user information or payment details stored in the browser.
- **Loss of Trust**: If users are targeted by this attack, it may lead to loss of user trust and damage to the reputation of the application.

Severity:
High
The vulnerability is classified as high severity because it allows attackers to inject malicious JavaScript code via an API endpoint. This could lead to significant security risks such as session hijacking, data theft, and unauthorized actions performed in the context of authenticated users.
PoC (Steps to Reproduce): (insert screen shots here)


Burp Suite Community Edition v2025.1.1 - Temporary Project
Request
GET /api/Products HTTP/1.1
Host: juice-shop.herokuapp.com
Response
HTTP/1.1 200 OK
Inspector


Request
Response
Inspector



Mitigation:

To prevent this type of attack, the application should:

1. **Sanitize Inputs**: Ensure that any user-provided input via API requests (such as the `description` field in the product API) is properly sanitized and encoded before being stored or rendered.
2. **Content Security Policy (CSP)**: Implement a Content Security Policy to restrict the execution of potentially dangerous scripts from untrusted sources.
3. **HTTP Headers**: Use headers like `X-XSS-Protection` to mitigate reflected XSS. Ensure that cookies are marked with the `HttpOnly` flag to prevent access via JavaScript.
4. **Use Safe Libraries/Frameworks**: Utilize libraries that help automatically escape user input when rendering it on the client side (e.g., React or Angular's built-in sanitization features).
5. **Regular Security Audits**: Conduct routine penetration testing and security audits to identify potential vulnerabilities like this one

19) Vulnerability: payback time (improper input validation)

Executive Summary:

A vulnerability was discovered in the product addition functionality, where the system fails to properly validate user-supplied input when adding items to the basket. By manipulating the quantity value in the request headers—specifically setting it to a negative number—a user can trick the system into deducting money instead of charging, leading to a negative total and potential abuse of the payment system.

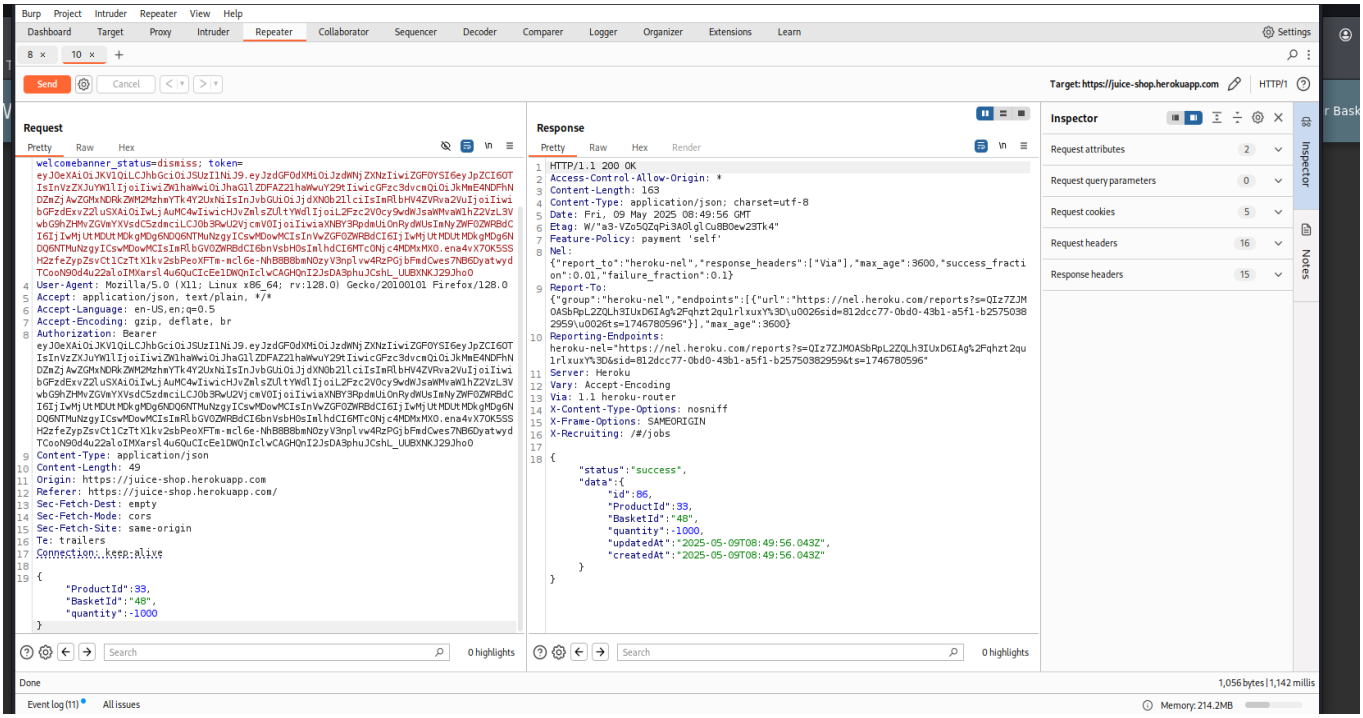
Impact:

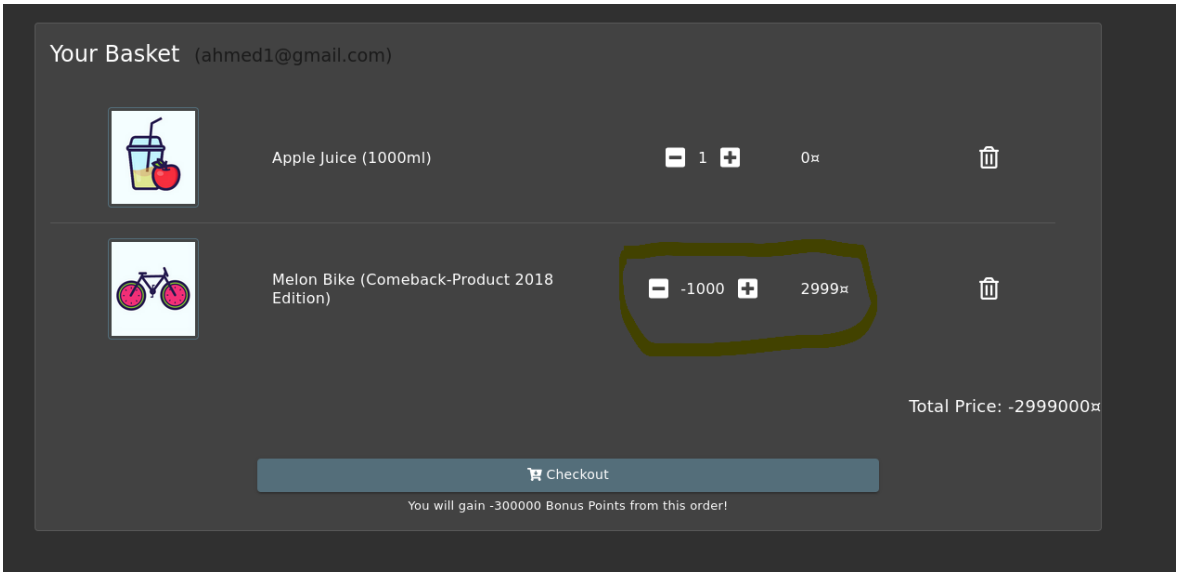
This flaw allows an attacker to:

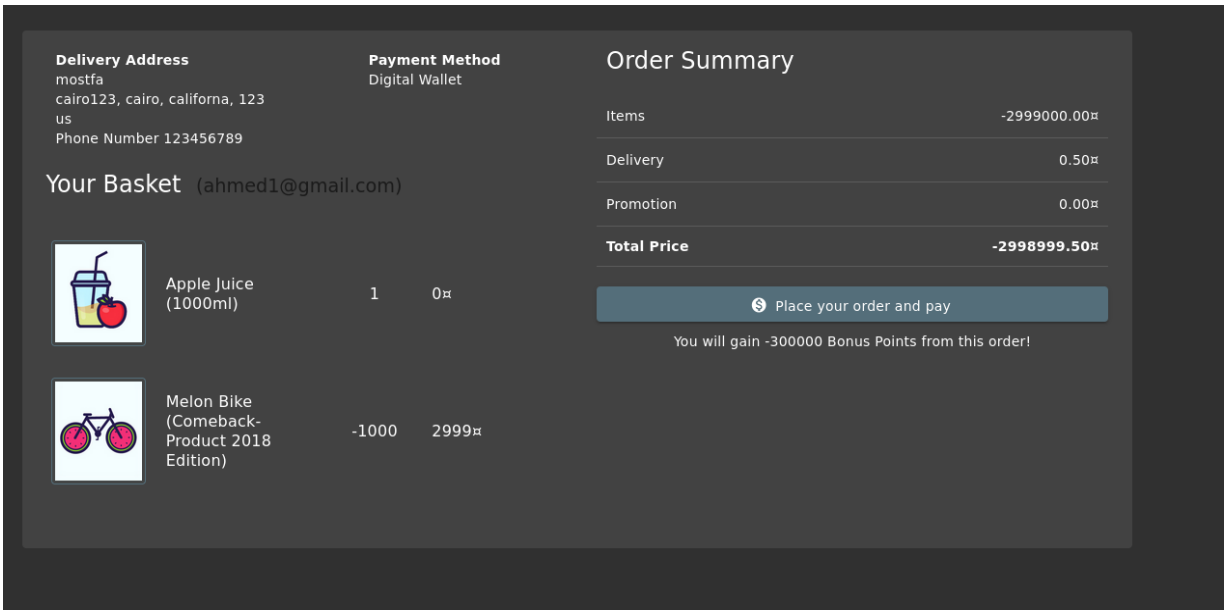
- **Exploit the basket to generate a negative total**, essentially making the store pay the user.
- **Bypass intended purchase flow** and compromise financial integrity.
- **Abuse the system repeatedly** to gain free or even financially beneficial transactions.

Severity:
High
This vulnerability directly affects the business logic and financial transactions of the application.

PoC (Steps to Reproduce): (insert screen shots here)


Burp Project Intruder Repeater View Help
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn Settings
Send Cancel
Target: https://juice-shop.herokuapp.com HTTP/1
Request
Pretty Raw Hex
welcomebanner_status=dismiss; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6OTIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJhaGl1ZDFAZ21haWwuY29tIiwicGFzc3dvcmQiOiJkMmE4NDFhNDZmZjAwZGMxNDRkZWM2MzhmYTk4Y2UxNiIsInJvbGUiOiJjdXN0b21lciIsInRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdC5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDUtMDkgMDg6NDQ6NTMuNzgyICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDUtMDkgMDg6NDQ6NTMuNzgyICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTc0Njc4MDMxMX0.ena4vX7OK5SSH2zfeZypZsvCt1CzTtXlkv2sbPeoXFTm-mcl6e-NhBBB8bmNOzyV3nplvw4RzPGjbFmdCwes7NB6Dyatwyd
TCooN9Od4u22aloIMXarsl4u6QuCIcEe1DWQnIclwCAGHQnI2JsDA3phuJCshL_UUBXNKJ29Jho0
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.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6OTIsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJhaGl1ZDFAZ21haWwuY29tIiwicGFzc3dvcmQiOiJkMmE4NDFhNDZmZjAwZGMxNDRkZWM2MzhmYTk4Y2UxNiIsInJvbGUiOiJjdXN0b21lciIsInRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIwLjAuMC4wIiwicHJvZmlsZUltYWdlIjoiL2Fzc2V0cy9wdWJsaWMvaW1hZ2VzL3VwbG9hZHMvZGVmYXVsdC5zdmciLCJ0b3RwU2VjcmV0IjoiIiwiaXNBY3RpdmUiOnRydWUsImNyZWF0ZWRBdCI6IjIwMjUtMDUtMDkgMDg6NDQ6NTMuNzgyICswMDowMCIsInVwZGF0ZWRBdCI6IjIwMjUtMDUtMDkgMDg6NDQ6NTMuNzgyICswMDowMCIsImRlbGV0ZWRBdCI6bnVsbH0sImlhdCI6MTc0Njc4MDMxMX0.ena4vX7OK5SSH2zfeZypZsvCt1CzTtXlkv2sbPeoXFTm-mcl6e-NhBBB8bmNOzyV3nplvw4RzPGjbFmdCwes7NB6Dyatwyd
TCooN9Od4u22aloIMXarsl4u6QuCIcEe1DWQnIclwCAGHQnI2JsDA3phuJCshL_UUBXNKJ29Jho0
Content-Type: application/json
Content-Length: 49
Origin: https://juice-shop.herokuapp.com
Referer: https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Te: trailers
Connection: keep-alive
{
"ProductId":33,
"BasketId":"48",
"quantity":-1000
}
Search 0 highlights
Response
Pretty Raw Hex Render
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Length: 163
Content-Type: application/json; charset=utf-8
Date: Fri, 09 May 2025 08:49:56 GMT
Etag: W/"a3-VZo5QZqPi3A0lglCu8B0ew23Tk4"
Feature-Policy: payment 'self'
Nel:
{"report_to":"heroku-nel","response_headers":["Via"],"max_age":3600,"success_fraction":0.01,"failure_fraction":0.1}
Report-To:
{"group":"heroku-nel","endpoints":[{"url":"https://nel.heroku.com/reports?s=QIz7ZJMOASbRpL2ZQLh3IUxD6IAg%2Fqhzt2qu1rlxuxY%3D\u0026sid=812dcc77-0bd0-43b1-a5f1-b25750382959\u0026ts=1746780596"}],"max_age":3600}
Reporting-Endpoints:
heroku-nel="https://nel.heroku.com/reports?s=QIz7ZJMOASbRpL2ZQLh3IUxD6IAg%2Fqhzt2qu1rlxuxY%3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&ts=1746780596"
Server: Heroku
Vary: Accept-Encoding
Via: 1.1 heroku-router
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
X-Recruiting: /#/jobs
{
"status":"success",
"data":{
"id":86,
"ProductId":33,
"BasketId":"48",
"quantity":-1000,
"updatedAt":"2025-05-09T08:49:56.043Z",
"createdAt":"2025-05-09T08:49:56.043Z"
}
}
Search 0 highlights
Inspector
Request attributes 2
Request query parameters 0
Request cookies 5
Request headers 16
Response headers 15
Inspector
Notes
Done
1,056 bytes | 1,142 millis
Event log (11)
All issues
Memory: 214.2MB


Your Basket (ahmed1@gmail.com)
Apple Juice (1000ml)
1
0¤
Melon Bike (Comeback-Product 2018 Edition)
-1000
2999¤
Total Price: -2999000¤
Checkout
You will gain -300000 Bonus Points from this order!



Mitigation:

- Implement **strict input validation** on the backend to ensure quantity values are **positive integers only**.
- Reject any request with invalid or tampered data (e.g., negative or non-numeric quantities).
- Use **server-side validation only**, as client-side checks can be easily bypassed using tools like Burp Suite.

20) Vulnerability: **Deprecated Interface - Security Misconfiguration**

Executive Summary:

A security misconfiguration vulnerability was identified in the **complain page file upload** functionality. While the front-end interface restricts file uploads to **PDF and ZIP files**, a review of the underlying code (or by analyzing the request) reveals that **XML files are still accepted by the server**. This discrepancy between client-side validation and server-side logic allows attackers to upload unauthorized file types like XML, potentially exposing the application to further vulnerabilities such as **XML External Entity (XXE)** attacks.

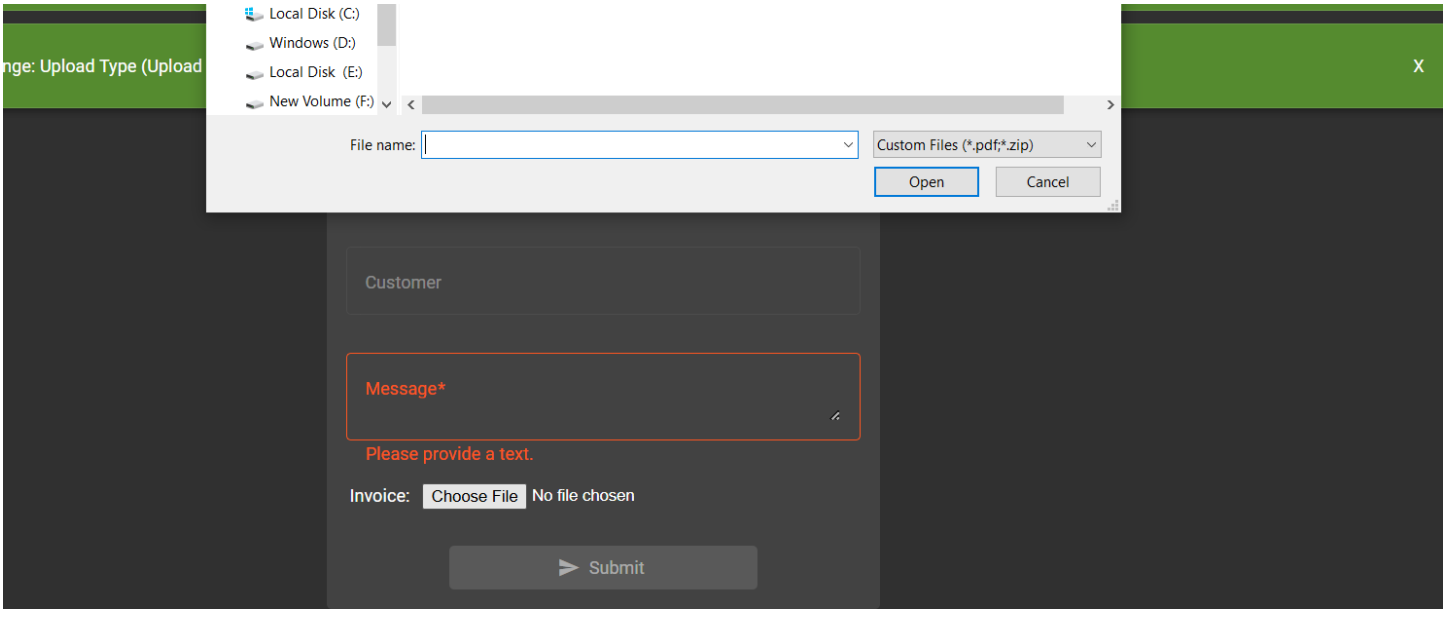
Impact:

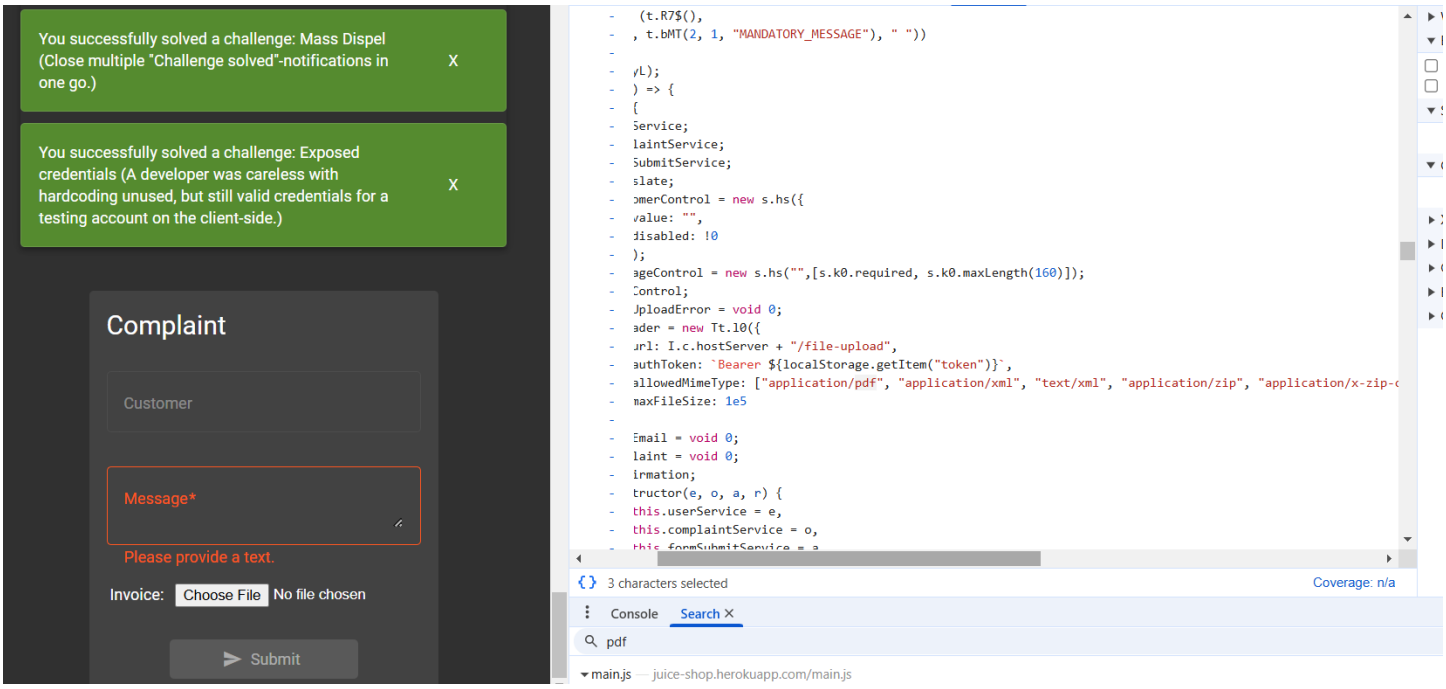
The impact of this vulnerability can be severe:

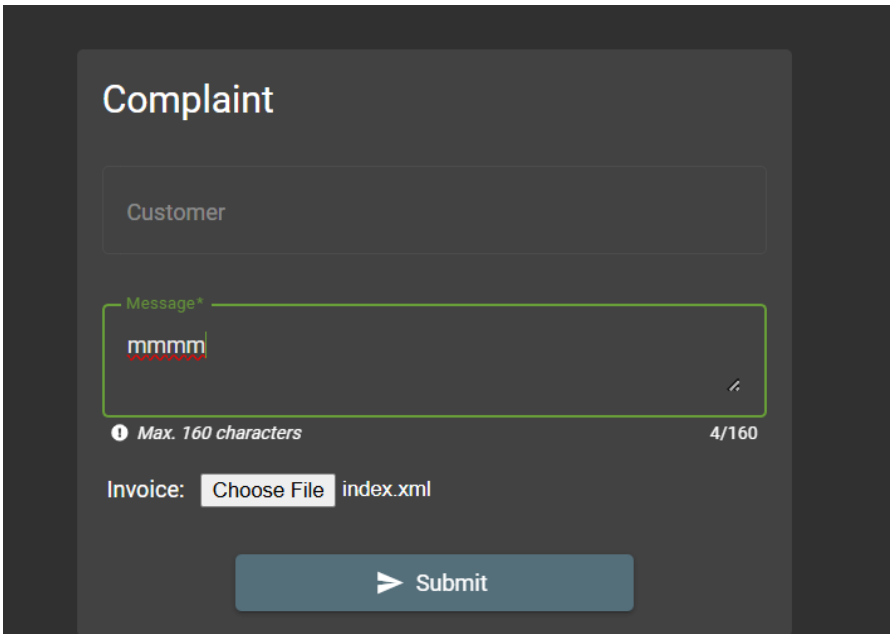
- Bypasses file type restrictions intended to secure the upload feature.
- May lead to XML-based attacks (e.g., XXE), file inclusion, or information disclosure.
- Indicates inconsistent validation between client-side and server-side controls, undermining trust in file handling mechanisms.

Severity:
Medium to High – Depending on whether the uploaded XML is parsed unsafely on the backend, this could lead to serious exploits like file disclosure or remote code execution.

PoC (Steps to Reproduce): (insert screen shots here)







Mitigation:

To prevent this type of attack, the application should:

- Implement strict server-side file type validation using MIME type and file signature checks.
- Remove support for unused or deprecated file types (e.g., XML) if not explicitly required.
- Apply a whitelist approach for file uploads and avoid relying solely on file extensions or client-side controls.

21) Vulnerability: **Missing Encoding**

Executive Summary:

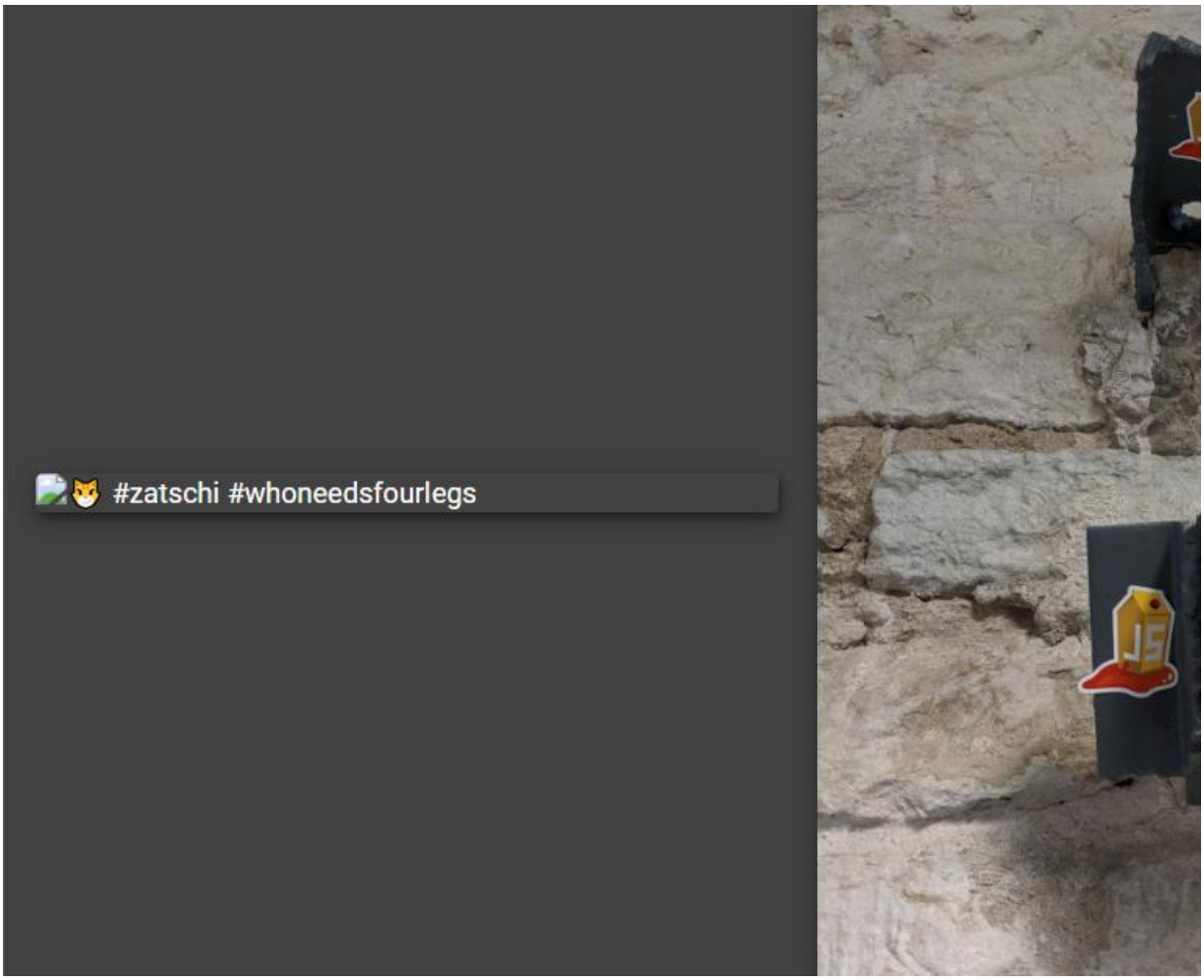
This report examines a security vulnerability related to improper error handling in URL processing. Specifically, the issue arises when special characters, like #, are not properly encoded in URLs. In the case of the # symbol, which is used to define fragments in URLs, if not correctly handled, it can lead to unexpected behavior or vulnerabilities. The attack can be mitigated by encoding the # character as %23 to avoid issues in URL parsing and to ensure the URL is interpreted correctly by the application.

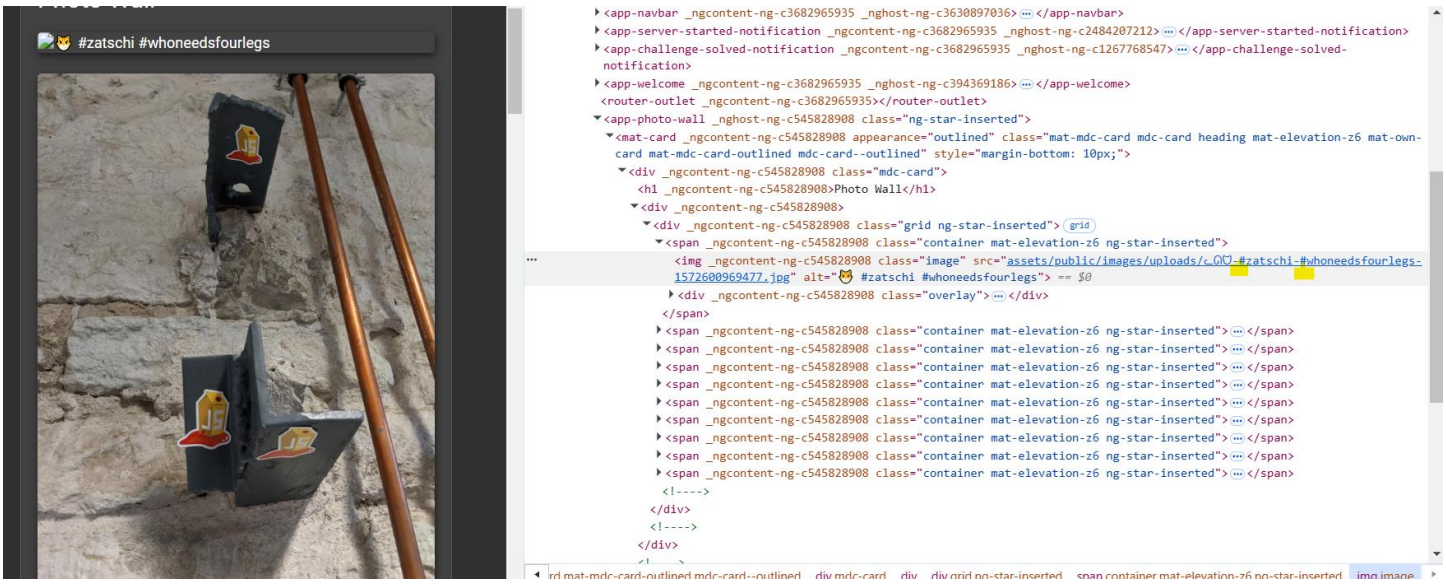
Impact:

The impact of this vulnerability can be severe:

- **Security Risks**: Attackers may exploit improperly handled URLs to inject malicious content, alter the flow of web requests, or gather sensitive information inadvertently exposed in error messages.
- **User Experience Issues**: Improper error handling can lead to unexpected application behavior, such as broken links, failed redirects, or the unintentional display of debug information, which can reduce the trust of users in the system's reliability.

Severity:
Medium
improper handling of special characters like # can lead to URL manipulation attacks
PoC (Steps to Reproduce): (insert screen shots here)

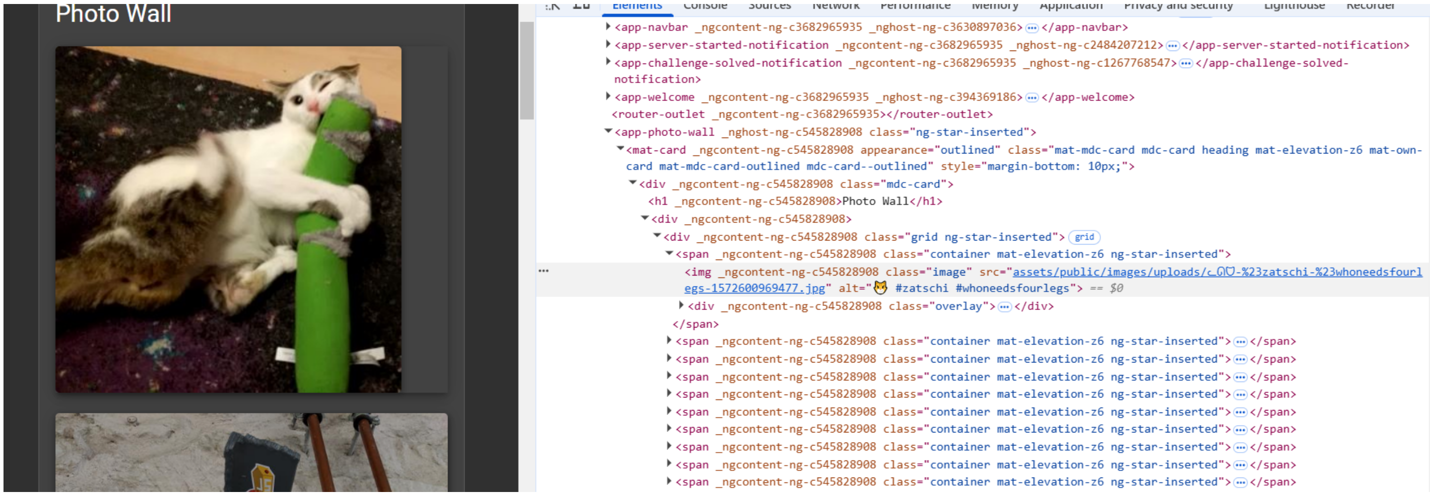

#zatschi #whoneedsfourlegs


#zatschi #whoneedsfourlegs
<app-navbar _ngcontent-ng-c3682965935 _nghost-ng-c3630897036> </app-navbar>
<app-server-started-notification _ngcontent-ng-c3682965935 _nghost-ng-c2484207212> </app-server-started-notification>
<app-challenge-solved-notification _ngcontent-ng-c3682965935 _nghost-ng-c1267768547> </app-challenge-solved-notification>
<app-welcome _ngcontent-ng-c3682965935 _nghost-ng-c394369186> </app-welcome>
<router-outlet _ngcontent-ng-c3682965935></router-outlet>
<app-photo-wall _nghost-ng-c545828908 class="ng-star-inserted">
<mat-card _ngcontent-ng-c545828908 appearance="outlined" class="mat-mdc-card mdc-card heading mat-elevation-z6 mat-own-card mat-mdc-card-outlined mdc-card--outlined" style="margin-bottom: 10px;">
<div _ngcontent-ng-c545828908 class="mdc-card">
<h1 _ngcontent-ng-c545828908>Photo Wall</h1>
<div _ngcontent-ng-c545828908>
<div _ngcontent-ng-c545828908 class="grid ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<img _ngcontent-ng-c545828908 class="image" src="assets/public/images/uploads/c-QO-#zatschi-#whoneedsfourlegs-1572600969477.jpg" alt="#zatschi #whoneedsfourlegs"> == $0
<div _ngcontent-ng-c545828908 class="overlay">

<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<span _ngcontent-ng-c545828908 class="container mat-elevation-z6 ng-star-inserted">
<!---->

<!---->

rd.mat-mdc-card-outlined.mdc-card--outlined div.mdc-card div div.grid.ng-star-inserted span.container.mat-elevation-z6.ng-star-inserted img.image



Mitigation:

Proper URL Encoding: Ensure that special characters like # are always encoded as %23 to prevent confusion between actual URL fragments and query parameters.

Robust Input Validation: Sanitize and validate all user inputs to prevent malicious data from entering the system.