Contents

1. SCOPE OF WORK

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

1.1 TIMELINE

Name	Start Date	End Date	Man-days
OWASP Juice Shop	OCT 1st, 2025	Nov 30th, 2025	20

1.2 TARGETS

The scope below was preview.owasp-juice.shop

ID	Asset Identifier (URL/IP)
1	https://preview.owasp-juice.shop/

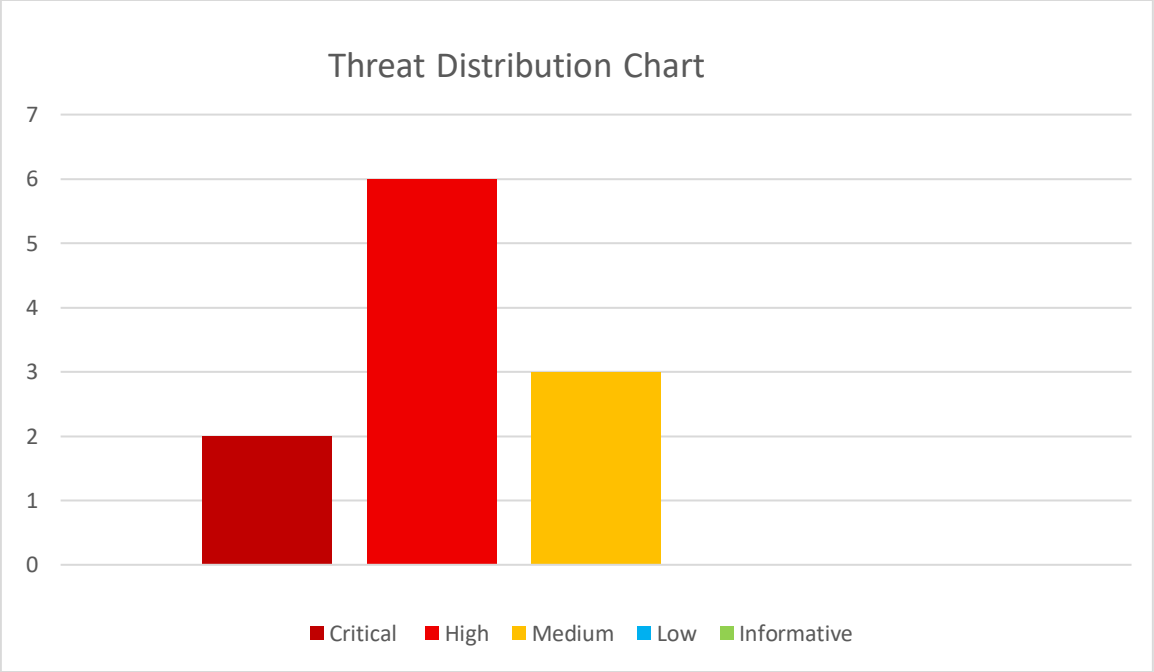
2. FINDING OVERVIEW

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

ID	Vulnerability Name	Severity
Vuln 01	SQL Injection	**Critical**
Vuln 02	Brute-Force Attack on Security Questions	**Critical**
Vuln 03	Payment Bypass in Check-out System	**High**
Vuln 04	XXE	**High**
Vuln 05	Broken Access Control	**High**
Vuln 06	Sensitive Data Exposure	**High**
Vuln 07	NoSQL injection	**High**
Vuln 08	Deprecated Interface	**High**
Vuln 09	XSS	**Medium**
Vuln 10	File Upload	**Medium**
Vuln 11	IDOR	**Medium**
Vuln 12	Admin Panel Path Disclosure via main.js	**Medium**

3. RISK OVERVIEW

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



4. POTENTIAL BUSINESS IMPACT

- Exploitation of the payment bypass vulnerability will result in direct revenue loss and inventory shrinkage as attackers successfully complete orders without transferring funds.
- The critical SQL injection flaw exposes the entire customer database to theft, subjecting the organization to severe regulatory fines and legal liability under data protection laws like GDPR.
- Mass defacement of product reviews via NoSQL injection will immediately erode consumer confidence and permanently damage the brand's reputation in the marketplace.
- The exposure of confidential internal documents and acquisition plans provides competitors with strategic intelligence that can be used to undermine the company's market position.
- Unrestricted brute-force attacks and administrative account takeovers allow malicious actors to seize full control of the platform, potentially locking out legitimate administrators and disrupting operations.
- Insecure Direct Object References (IDOR) allow unauthorized access to private customer data, leading to a loss of user privacy and potential class-action litigation.
- The XML External Entity (XXE) vulnerability exposes the internal server infrastructure to reconnaissance and Denial of Service attacks that could cause prolonged system downtime.
- Persistent Cross-Site Scripting (XSS) vulnerabilities jeopardize customer safety by enabling attackers to steal user session cookies or launch phishing campaigns directly through the trusted website.
- The compromise of administrative accounts and subsequent data manipulation will require expensive forensic investigations and disaster recovery efforts, draining operational budgets.
- Inconsistent transaction data caused by fraudulent orders will corrupt business analytics and inventory records, leading to poor decision-making and logistical errors.

5. TECHNICAL VULNERABILITIES DETAILS

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

5.1 SQL Injection

Severity	Critical
CVSS	CVSS 10
Affected Assets	• Authentication system & Administrator account
Reference	**https://cwe.mitre.org/data/definitions/89.html**

DESCRIPTION

The application's login endpoint is critically vulnerable to SQL injection via the email parameter. The application fails to properly sanitize user input before incorporating it into database queries. By injecting malicious SQL syntax, an attacker can manipulate the query logic to bypass password verification entirely.

IMPACT

- Full Account Takeover (Administrative)
- Unauthorized Access to User Data
- Catastrophic Security Breach
- Complete Authentication Failure

RECOMMENDATIONS

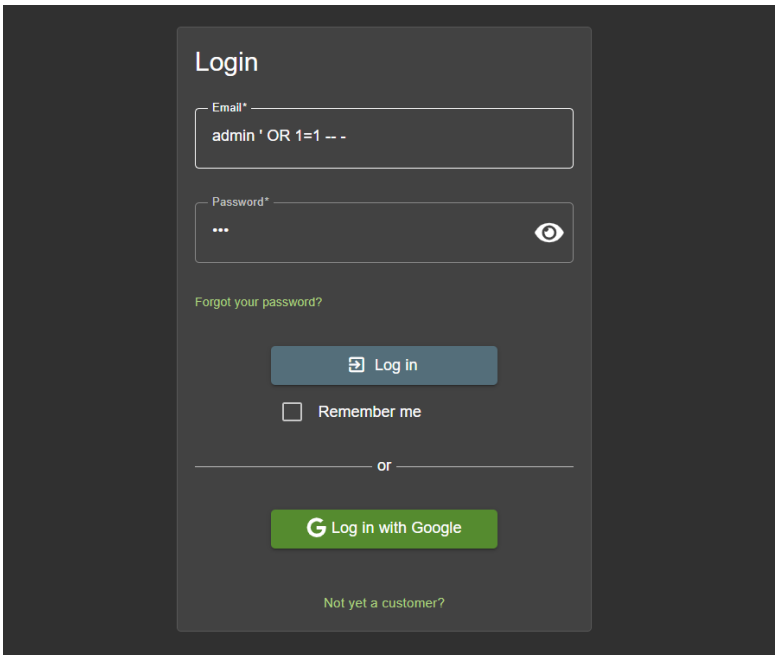
- Mandate Parameterized Queries: This is the most effective defense. Ensure all database queries use prepared statements to strictly separate code (SQL) from data (user input).
- Adopt Secure ORMs: Utilize an Object-Relational Mapper (ORM) that handles database interactions securely by default, preventing raw SQL concatenation.
- Input Validation and Sanitization: strictly validate user inputs against expected formats (e.g., valid email structure) and sanitize characters that pose security risks.
- Web Application Firewall (WAF): Deploy a WAF to detect and block common SQL injection patterns. Note that this is a defense-in-depth measure and does not replace secure coding.
- Centralize Data Access: funnel authentication logic through vetted stored procedures or a centralized data access layer to enforce consistent security controls.
- Generic Error Handling: Ensure error messages are user-friendly but non-verbose to prevent leaking database structure details to attackers.

D.T

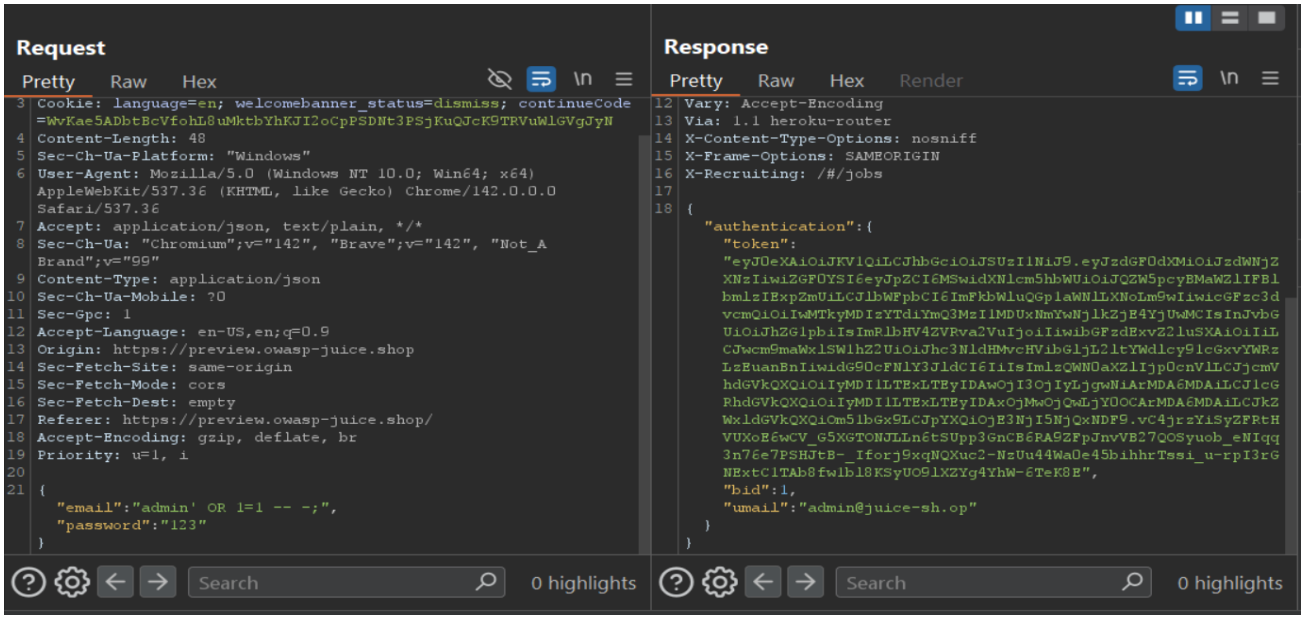
STEPS TO REPRODUCE

SCENARIO 1:

- Navigate to the login page: https://preview.owasp-juice.shop/#/login

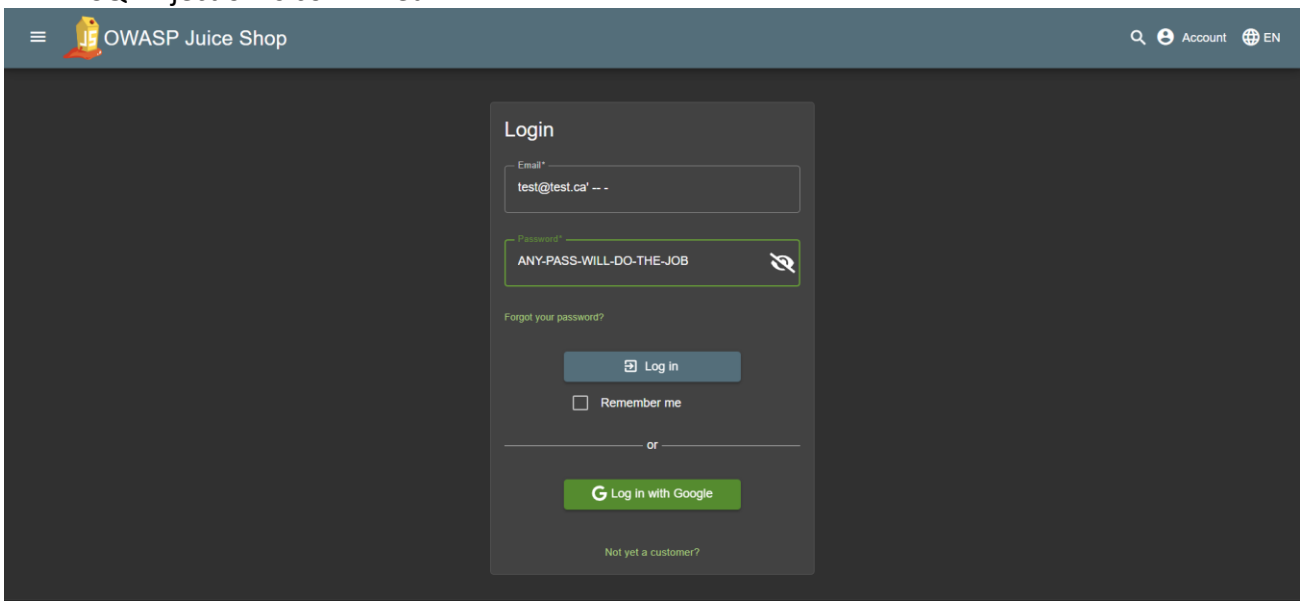


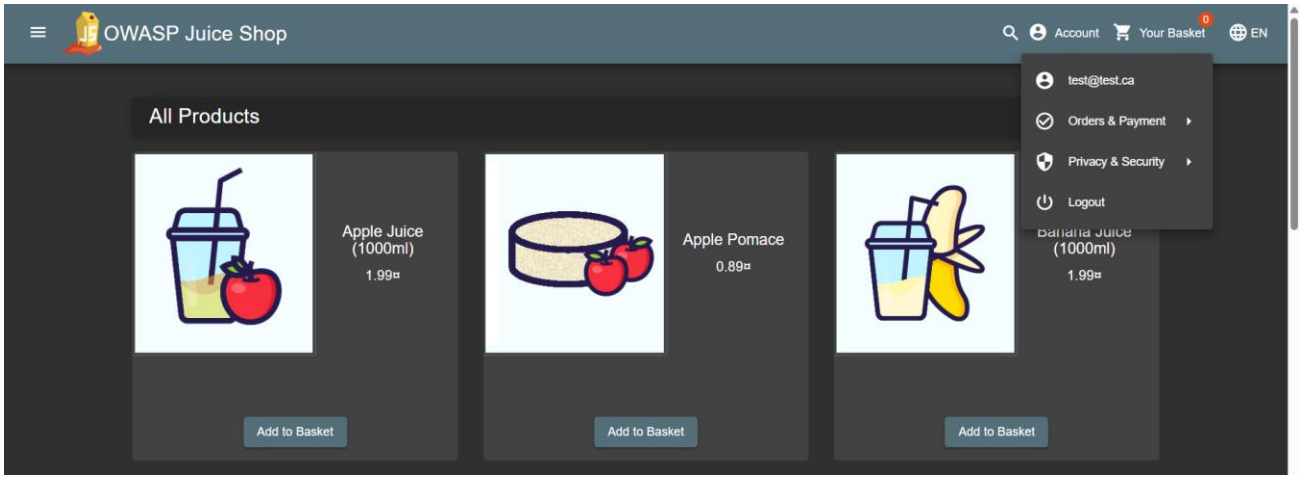
- Send a POST request to /rest/login:
- If the response returns an authentication token ("token": "...") and user fields showing {"bid": 1, "email": "admin@juice-sh.op"}, the authentication was bypassed and you are logged in as the first database user (likely the administrator).



SCENARIO 2:

- Navigate to https://preview.owasp-juice.shop/#/login
- In the email field submit: test@test.ca'-- -
- Submit the form; if authentication is bypassed and you are logged in as the specified user, the SQL injection is confirmed.





5.2 Brute-Force Attack on Security Questions

Severity	Critical
CVSS	CVSS 9.4
Affected Assets	• Password Reset Mechanism & Account Recovery System
Reference	**https://cwe.mitre.org/data/definitions/307.html** **https://cwe.mitre.org/data/definitions/640.html**

DESCRIPTION

The application's password reset functionality is vulnerable to brute-force attacks against the security question answers. There are no effective rate limiting or account lockout mechanisms in place following failed attempts to answer the security question.

This allows an attacker to automate the guessing of the security question answer for any valid user via simple tooling (e.g., Burp Suite Intruder). If the security question is weak or the answer is common/short (e.g., a short numerical ID), an attacker can easily bypass the intended account recovery controls.

IMPACT

The successful exploitation of this vulnerability can lead to:

- Unauthorized Password Reset: Enables an attacker to initiate and complete the password reset process for any account whose security-question answer can be successfully guessed.
- Account Takeover: Directly grants unauthorized access to user accounts, allowing the attacker to view and modify the user's data and settings.
- Circumvention of Security Controls: The attacker bypasses the intended account-recovery controls and gains access without needing the legitimate user's current password.
- Low Traceability: Account takeover can occur without obvious failure traces visible to the legitimate user until the account is already compromised.

RECOMMENDATIONS

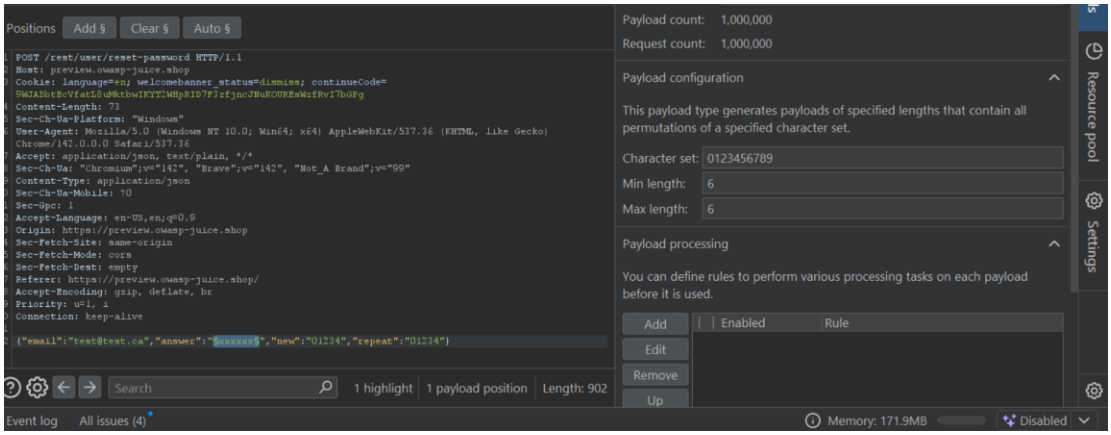
- Implement strict rate limiting on password reset attempts (e.g., a maximum of 5 attempts per hour per user/IP) to mitigate brute-force attacks.
- Introduce CAPTCHA or similar human verification after a small number of failed security question attempts to prevent automated guessing.
- Strengthen authentication mechanisms by using more complex, non-guessable security questions or replacing them entirely with multi-factor authentication.

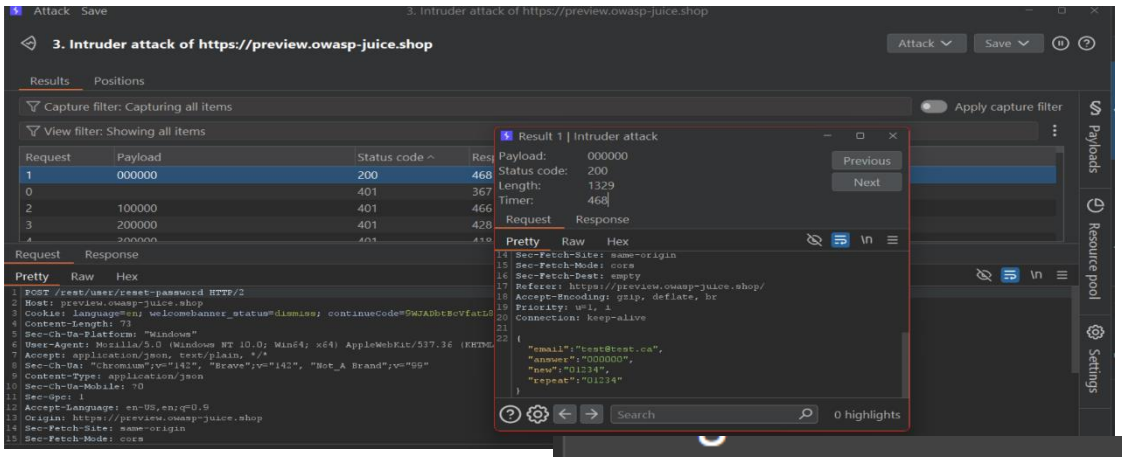
- Apply temporary account lockouts following multiple failed attempts at answering security questions to slow down or deter automated attacks.
- Enable detailed logging and alerting for unusual password reset activity, such as multiple failed attempts or repeated resets from the same IP address.
- Require additional identity verification (e.g., email or SMS confirmation, device-based validation) when password resets originate from new devices or locations.

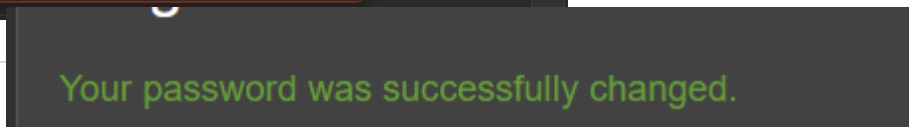
STEPS TO REPRODUCE

SCENARIO 1:

- Identify a valid user email address (test@test.ca)
- Navigate to the password reset functionality
- Request a password reset for the target email
- Use Burp Suite proxy to intercept the password reset request and send it to the intruder:







5.3 Payment Bypass in Check-out System

Severity	High
CVSS	CVSS 8.6
Affected Assets	• abuse by fraudulent users
Reference	**https://cwe.mitre.org/data/definitions/285.html**

DESCRIPTION

Multiple API endpoints across the application exhibit Broken Access Control flaws, allowing authenticated users to perform unauthorized actions against resources belonging to other users. This vulnerability primarily occurs because the application fails to adequately verify that the user ID provided in the request or derived from the session matches the owner of the resource being accessed or modified (known as a missing Authorization Check).

IMPACT

This flaw allows attackers to complete purchases without making any payment, that leads to:

• financial losses for the store.

• Potential abuse by fraudulent users

• Undermining of the business mode

RECOMMENDATIONS

• 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.

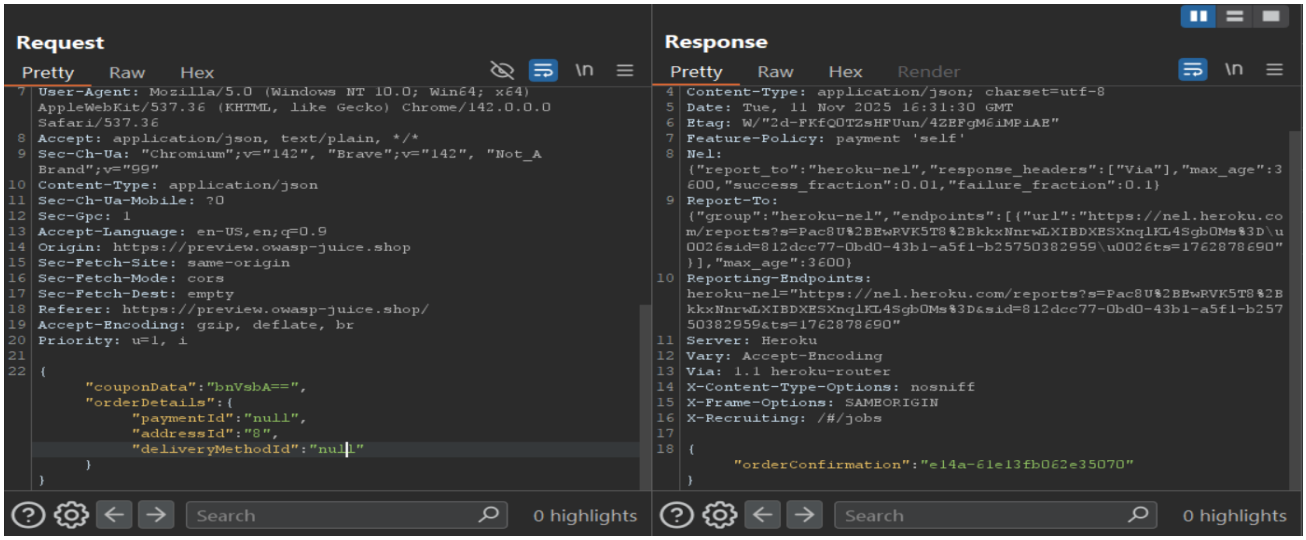
• Implement transaction verification with payment processor before order completion.

• 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

STEPS TO REPRODUCE

- Place products in your shopping cart and click through to the checkout page.
- 2. Using an intercepting proxy such as Burp Suite, capture the outgoing checkout request that includes the PaymentId parameter.
- Edit the captured request by changing the PaymentId value to null, then submit the modified request and observe whether the order completes without payment.



- Forward the modified request to the server and finish the checkout flow.
- verify that the order is accepted and marked complete despite no payment being recorded.

5.4 XML External Entity (XXE) via Improper File Upload Validation

Severity	High
CVSS	CVSS 8.6
Affected Assets	· https://juice-shop.com/complain (File upload endpoint) · Backend XML parser accepting unvalidated input. · Application server and internal storage containing XML-processed files.
Reference	XML External Entity Prevention Cheat Sheet

DESCRIPTION

The /complain upload endpoint accepts only .zip and .pdf files, but validation is weak. By intercepting the upload request and modifying the file type and headers, the tester was able to upload a crafted XML file. The server's parser processes XML input without disabling external entity references, allowing an attacker to include malicious entities that can read local files or access internal network resources (XXE vulnerability).

IMPACT

- Potential disclosure of local server files (sensitive configuration, credentials), or SSRF to internal services.
- Possible DoS via entity expansion if parser permits recursive expansion.
- If the parser runs with elevated privileges, the impact may include sensitive system data disclosure and further compromise.

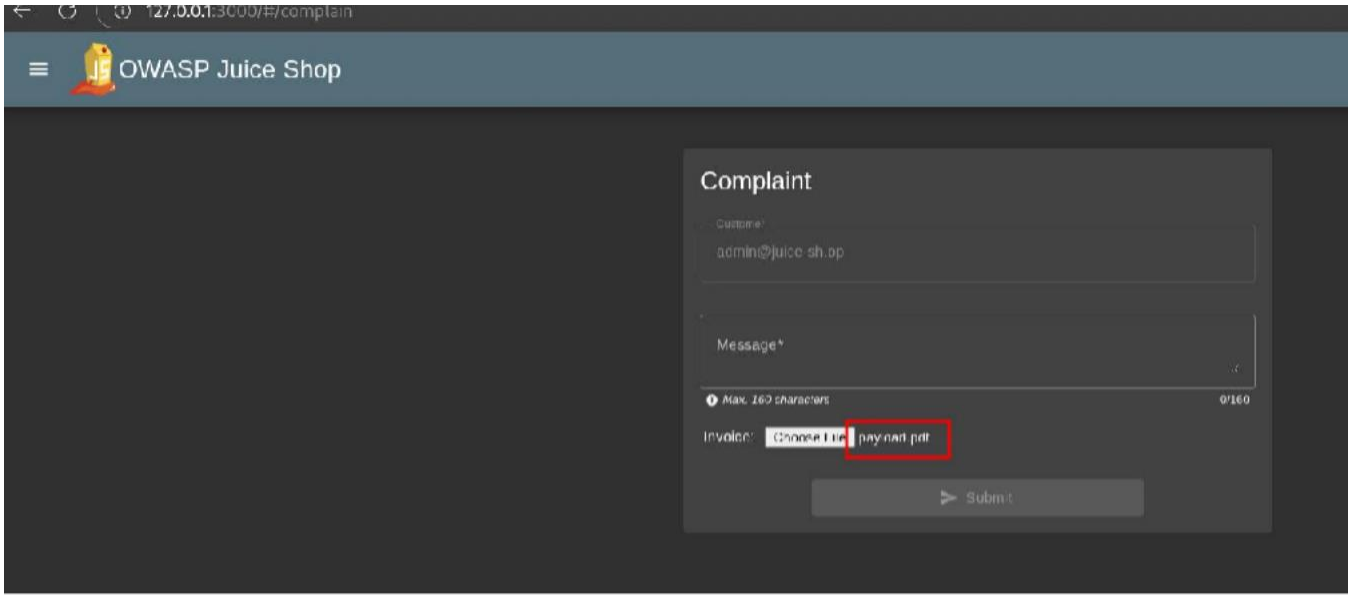
RECOMMENDATIONS

- Disable DTDs and external entity processing in the XML parser.
- Strictly validate and whitelist allowed file types and MIME types.
- Parse files in a restricted environment with limited permissions.
- Set limits on file size and parsing time to prevent DoS attacks.
- Keep XML libraries updated with the latest security patches.

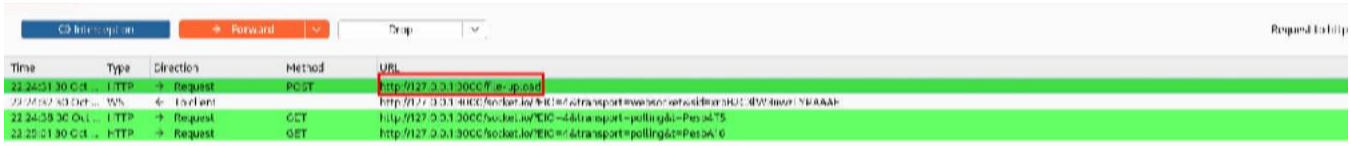
D.T

STEPS TO REPRODUCE

- Upload a pdf file:



- Intercept the file upload request.

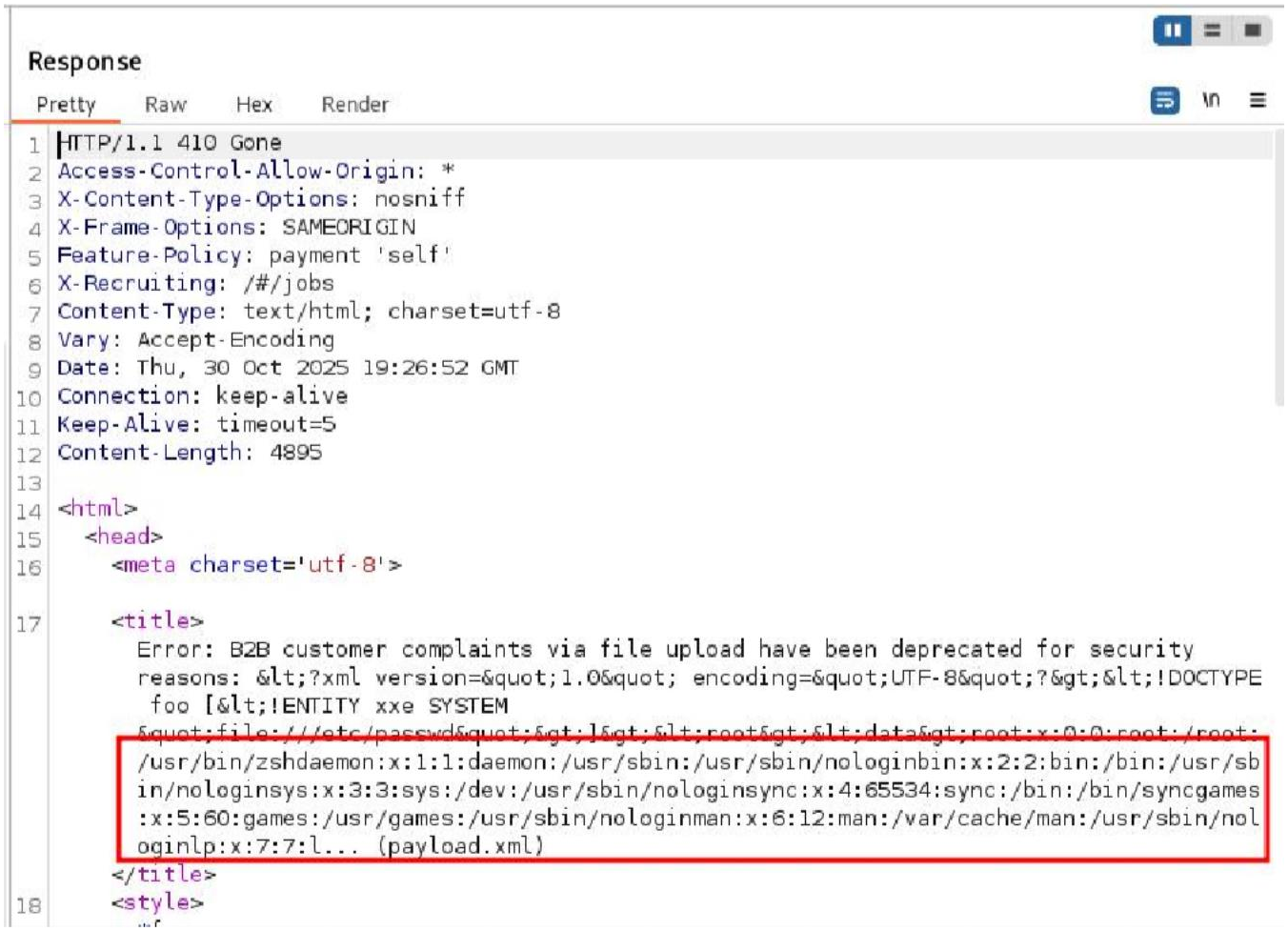


- Modify the file to be an .xml file with following payload:

```
------WebKitFormBoundarySY5kB7ErBCK7MTRO
Content-Disposition: form-data; name="file"; filename="payload.xml"
Content-Type: text/xml

<?xml version="1.0"?>
<!DOCTYPE foo [
 <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<root>
 <data>&xxe;</data>
</root>
------WebKitFormBoundarySY5kB7ErBCK7MTRO--
```

- It will return with passwd content.



5.5 Broken Access Control

Severity	High
CVSS	CVSS 7.7
Affected Assets	• /api/BasketItems/ • /api/Feedbacks
Reference	https://owasp.org/Top10/A01_2021-Broken_Access_Control/

DESCRIPTION

Multiple API endpoints across the application exhibit Broken Access Control flaws, allowing authenticated users to perform unauthorized actions against resources belonging to other users. This vulnerability primarily occurs because the application fails to adequately verify that the user ID provided in the request or derived from the session matches the owner of the resource being accessed or modified (known as a missing Authorization Check).

IMPACT

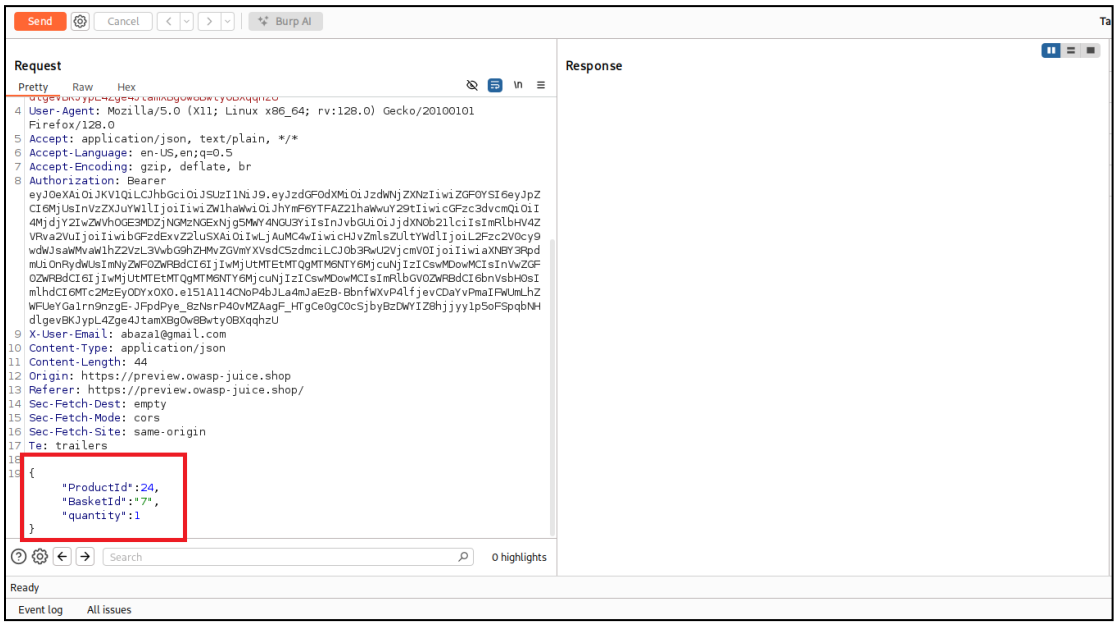
- Unauthorized Write Access: Attackers can tamper with the data of other users (e.g., adding products to another user's cart).
- Data Integrity Compromise: The ability to submit forged feedback damages the trustworthiness and integrity of system-wide data (ratings, reviews).
- User Confusion and Business Logic Manipulation: Tampering with user baskets can disrupt the purchasing process and skew business metrics or promotions.

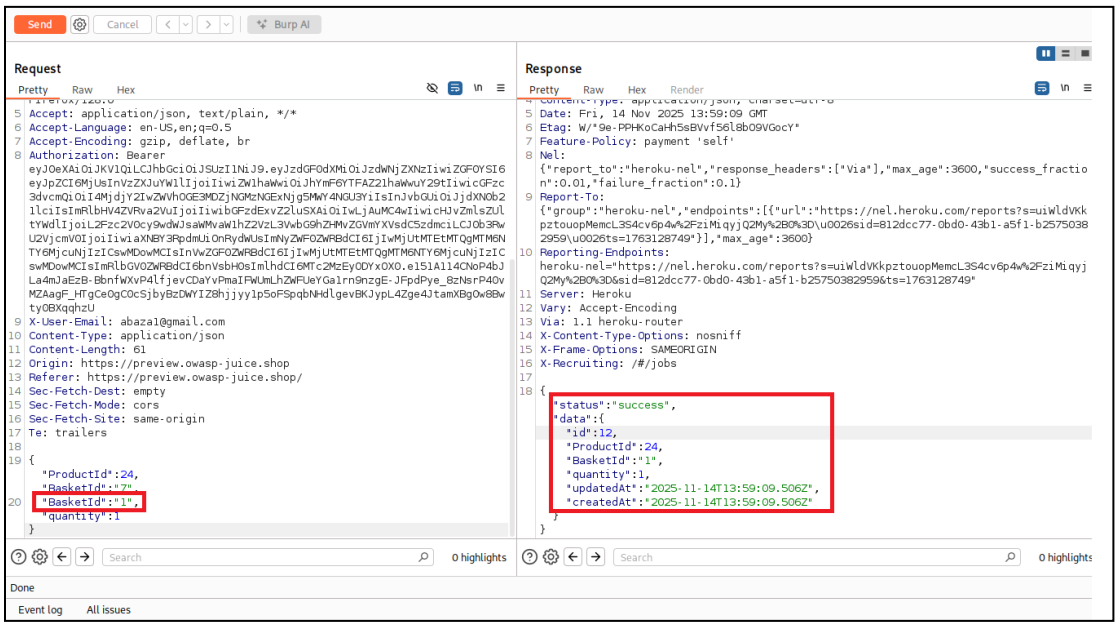
RECOMMENDATIONS

- **Strict Server-Side Authorization:** The server **must** implement robust checks to ensure that the user making the request is the **sole owner** of the resource (e.g., the basketId or the userId for the feedback). This check must be performed on the server after authentication.
- **Derive Identifiers Internally:** Never trust client-supplied identifiers for authorization. Sensitive IDs (like userId or basketId) must be retrieved **only** from the authenticated user's session token.
- **Reject Duplicate Parameters:** Implement strict input validation to **reject** or safely handle requests containing duplicate sensitive parameters (like basketId) to mitigate HTTP Parameter Pollution (HPP).
- **Logging and Alerting:** Implement logging for cross-user resource access attempts and unusual activity patterns.

STEPS TO REPRODUCE

SCENARIO 1:

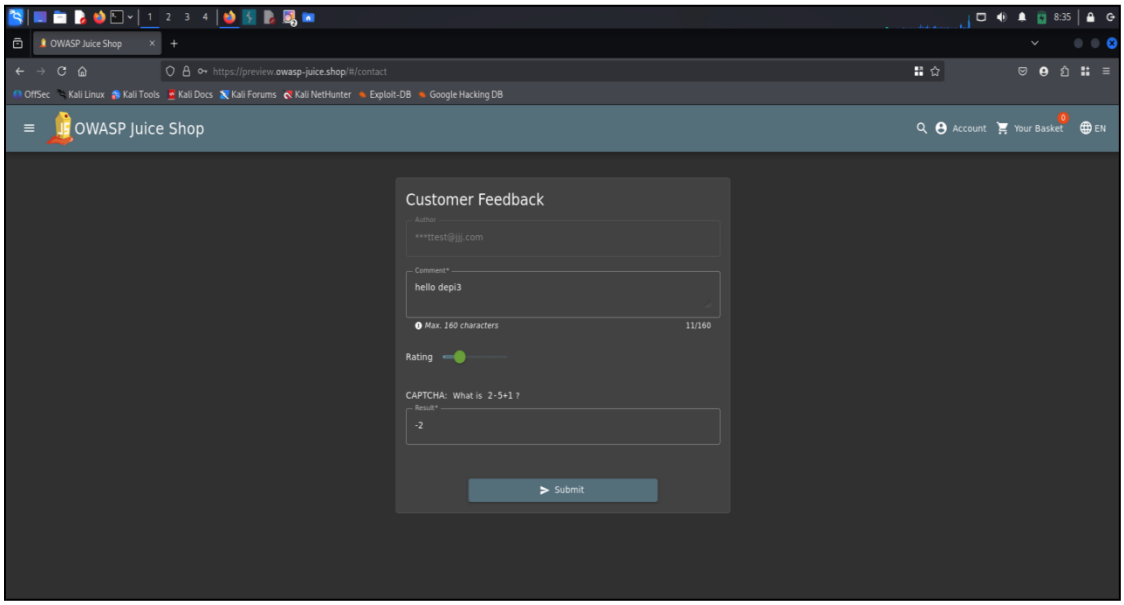
- Add an item to your own basket using your valid basketId.
- Capture the request in Burp Suite.
- 

- Modify the request to include: “BasketId”:”1”
- 

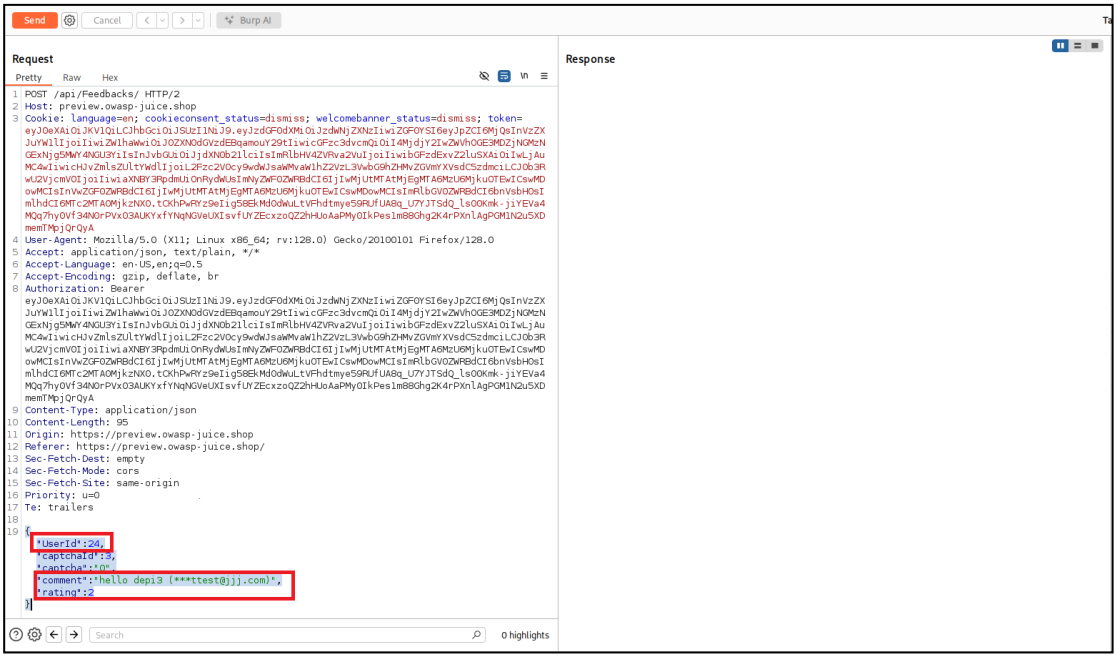
- Result
- The item is added to basket ID 1, not your own, confirming unauthorized access via parameter pollution.

SCENARIO 2:

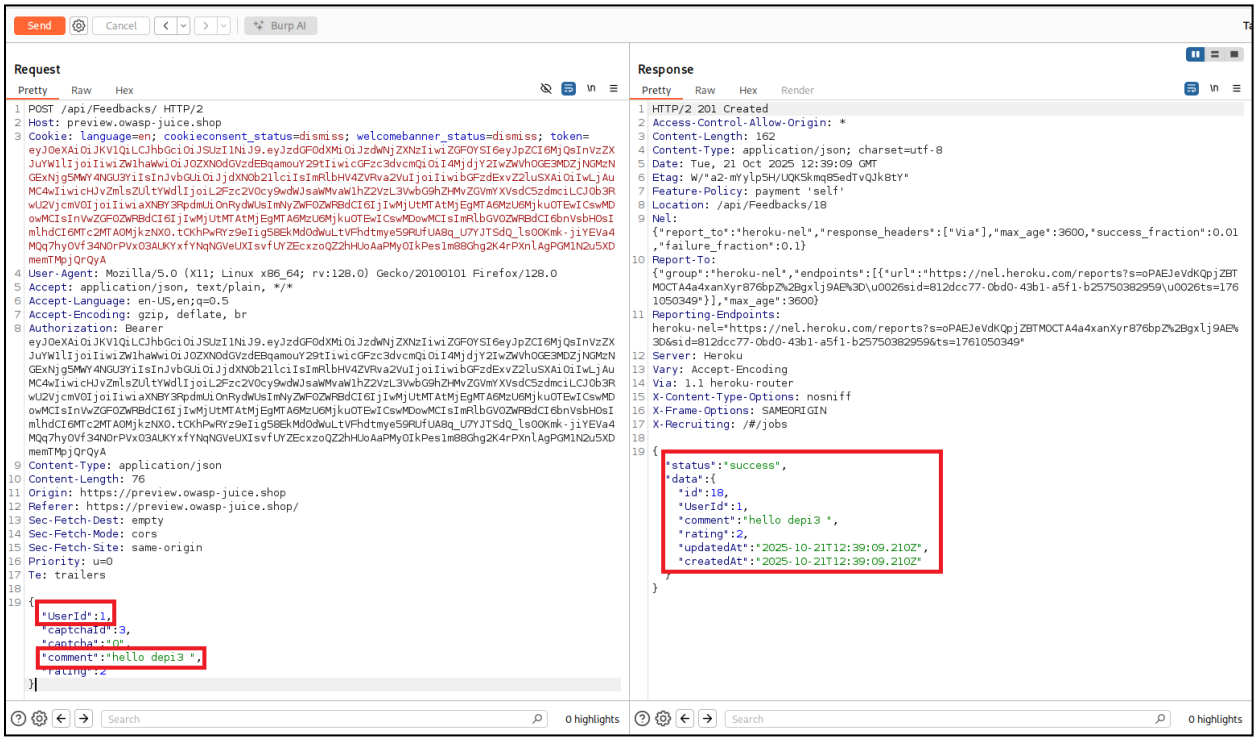
- Navigate to the Customer Feedback form on https://preview.owasp-juice.shop/#/contact
- Make a normal feedback process (can not change the author)



- Catch the request by Burp suite and send it to repeater



- so we can just change the UserId attribute and delete the mail address in the request and it will send a feedback with other user's name & without mail address



5.6 Sensitive Data Exposure

Severity	High
CVSS	CVSS 7.5
Affected Assets	• https://juice-shop.com/ftp/ • Internal file: acquisitions.md • Any file hosted in publicly accessible FTP directories
Reference	Sensitive Data Exposure Lab: File path traversal, validation of file extension with null byte bypass

DESCRIPTION

The application is vulnerable to Sensitive Data Exposure due to misconfiguration of public-facing directories, allowing unauthenticated access to internal and operational files.

IMPACT

- Unauthenticated Disclosure: Direct leakage of internal/confidential business documents, leading to potential competitive harm.
- System Compromise Risk: If documents contain secrets (credentials, API keys, PII), they become a direct source for system compromise or regulatory exposure.
- Reconnaissance: The easy discovery (e.g., via robots.txt) and exposed directory listing increase the likelihood and ease of attack.
- Session Hijacking & Pivoting: Exposed logs can reveal sensitive operational data, including session identifiers, user emails/identifiers, and internal endpoints. Attackers can use leaked tokens or identifiers for session hijacking or pivoting to privileged areas.
- Targeted Social Engineering: Exposure of support and operational data increases the chance of targeted phishing and reconnaissance against employees or users.

RECOMMENDATIONS

- Immediate Restrict access to the logs directory (remove public access; require authenticated admin access).
- Configure logging to never record secrets (Authorization headers, full request bodies, cookies) and to redact or truncate identifiers.
- Store logs outside the web document root and centralize logs in a secured log.
- Rotate and secure any tokens or credentials that were present in exposed logs.
- Implement log retention policies and monitoring/alerts for unusual log downloads.

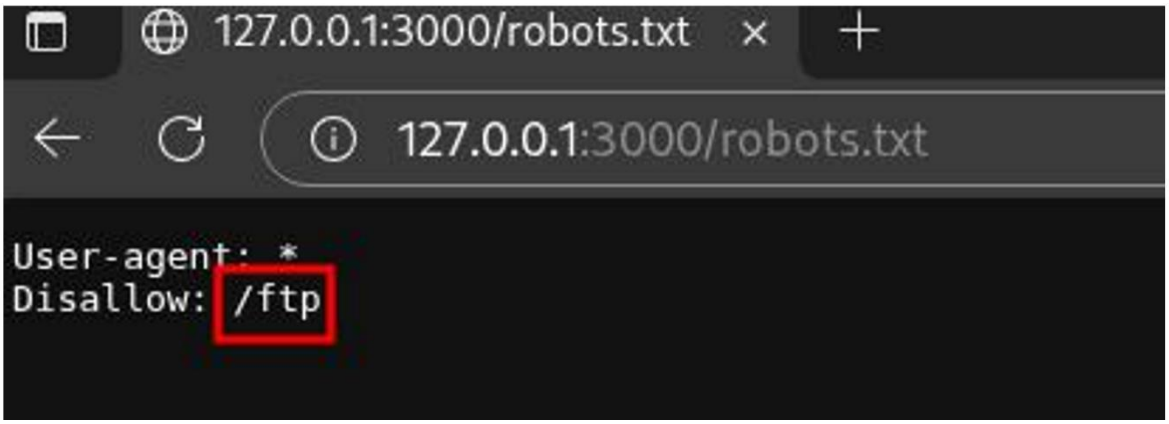
- Immediate Remove all confidential documents from the public FTP location.
- Disable anonymous FTP access; require authenticated access and strong credentials.

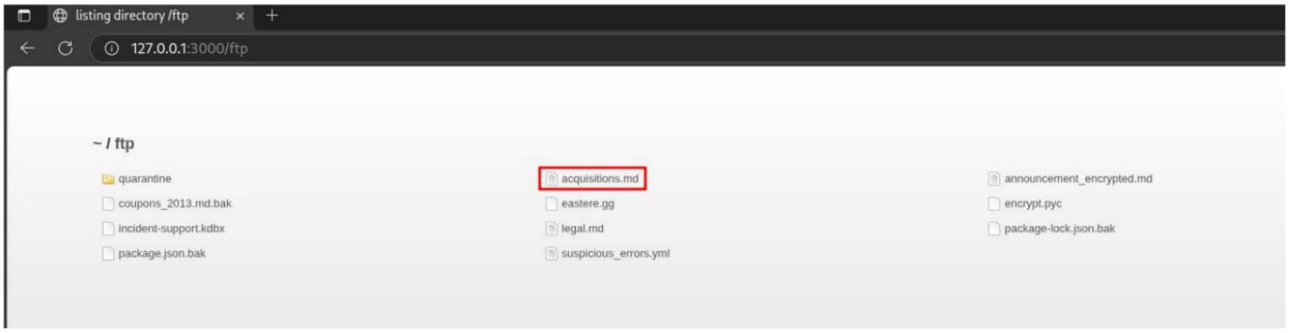
- Disable directory listing. ·
- Move internal documents to an authenticated, access-controlled storage.
- Remove references to internal resource locations from public files (do not list internal paths in robots.txt)

STEPS TO REPRODUCE

SCENARIO 1:

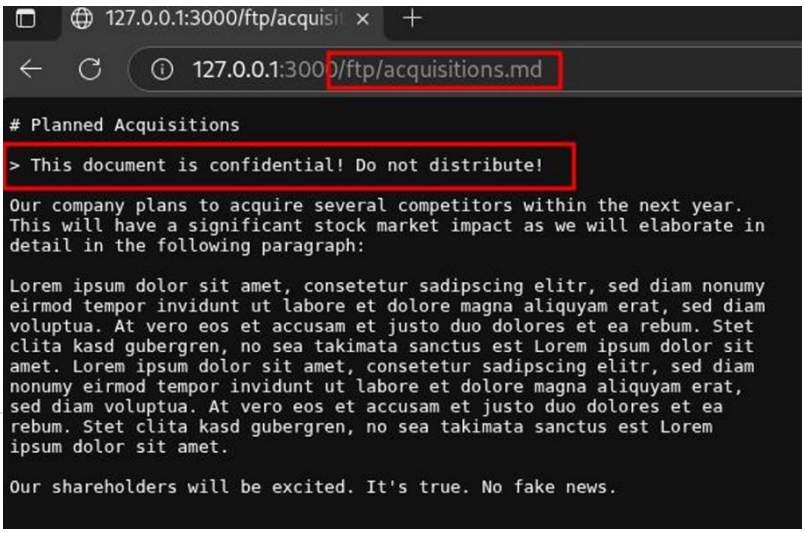
- Go to http://127.0.0.1:3000/robots.txt , after that we will find interesting directory.





- browse the directory by changing the URL into http://127.0.0.1:3000/ftp

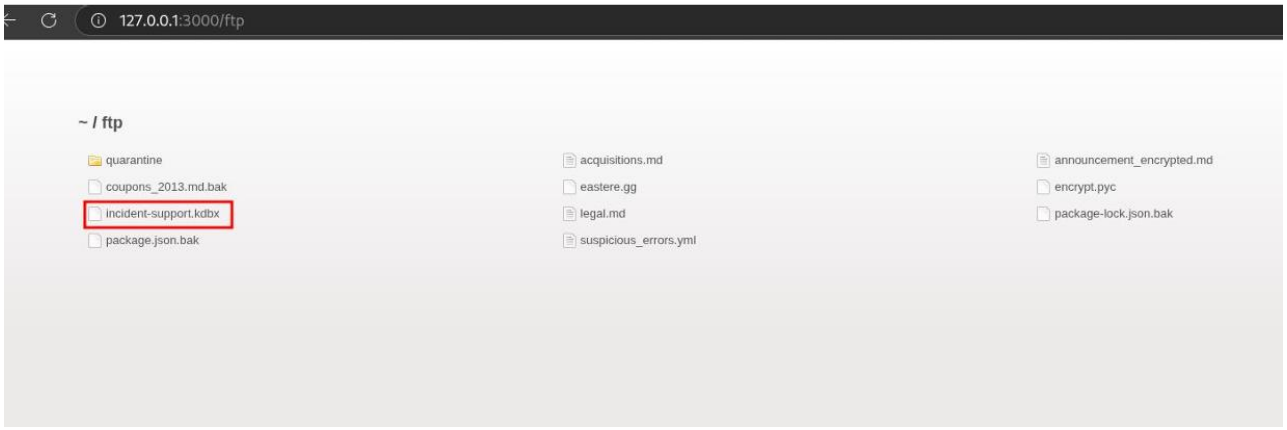
- You will find very interesting files like package.json.bak and acquisitions.md , open it



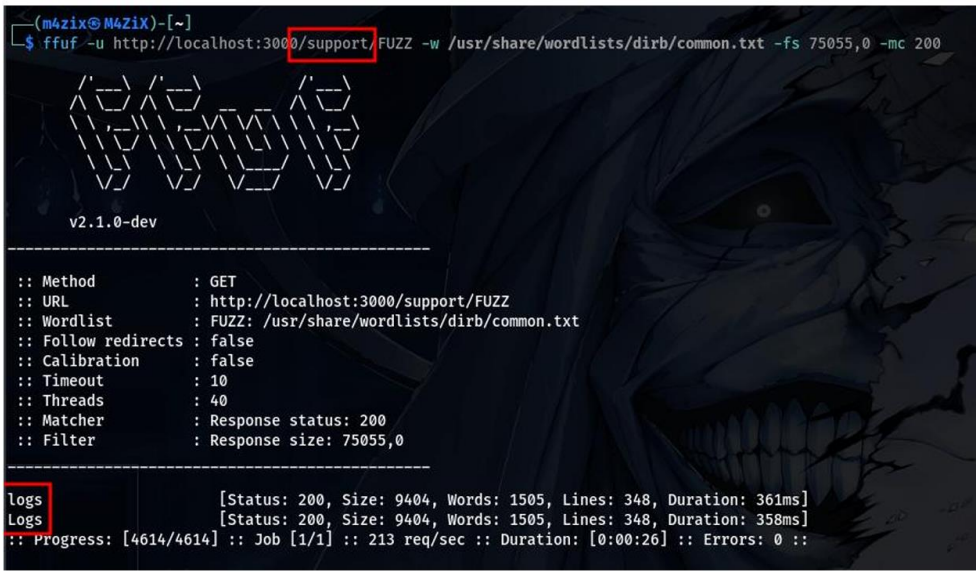
D.T

SCENARIO 2:

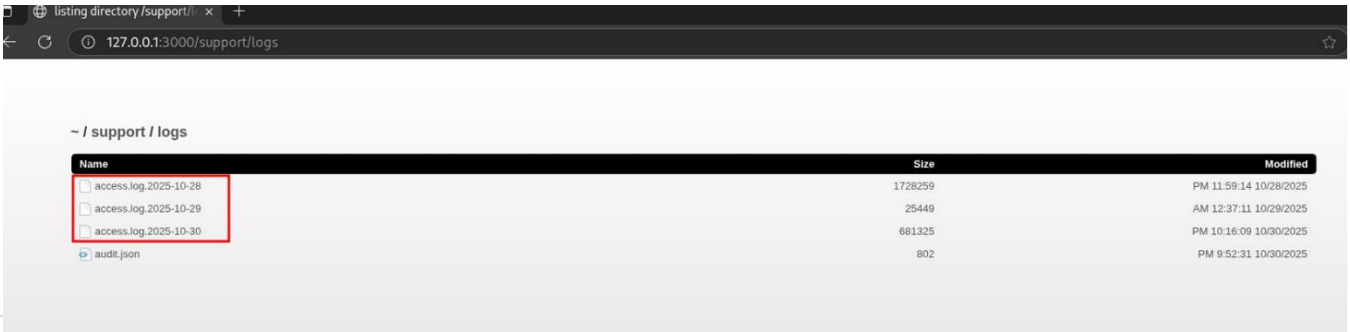
- Discover roles in the company by enumerate the app



- Use ffuf to brute-force directories on the target domain, find support/logs/ directory



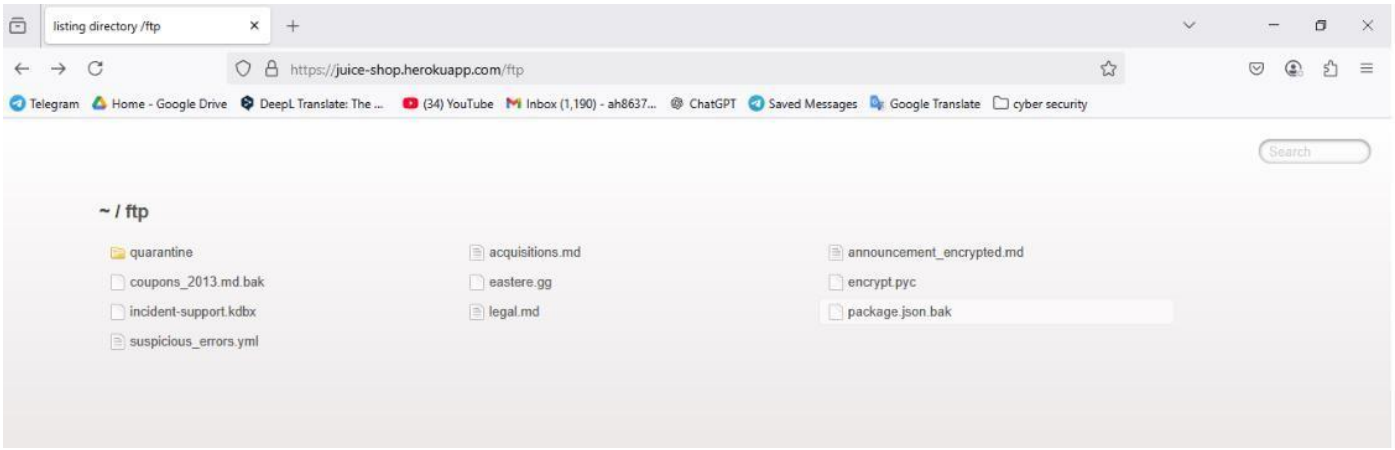
- Navigate to the discovered URL in a browser

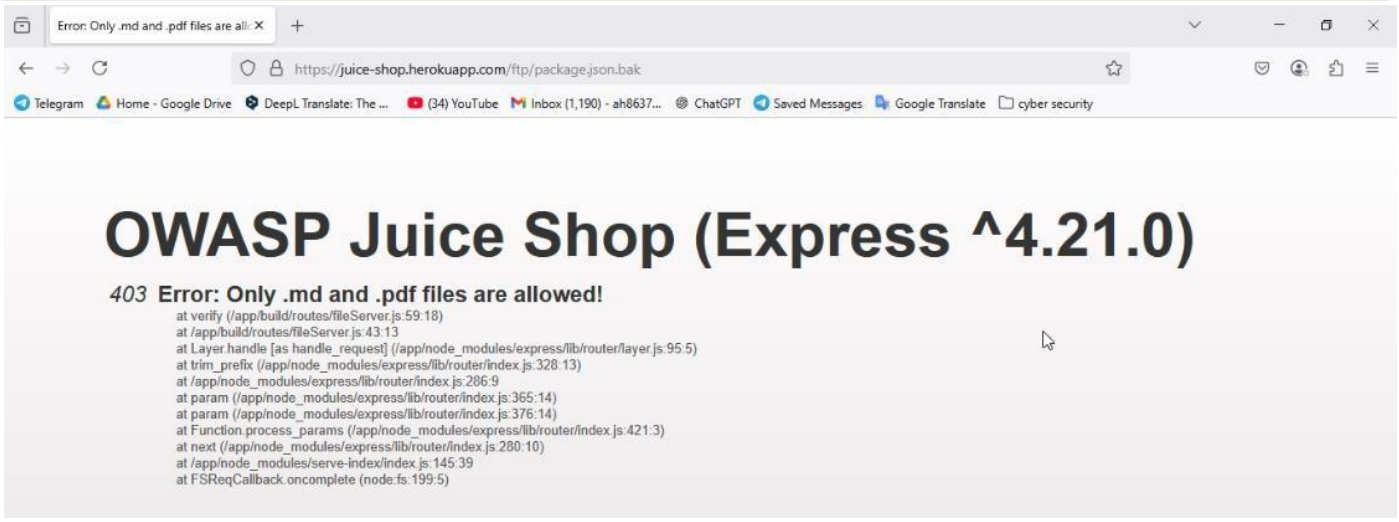


D.T

SCENARIO 3:

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





5.7 NoSQL Injection on Product Review Update

Severity	High
CVSS	CVSS 7.1
Affected Assets	• Database collections/tables • any user data exposed
Reference	NoSQL Security - OWASP Cheat Sheet Series

DESCRIPTION

The PATCH /rest/products/reviews endpoint is vulnerable to a NoSQL Injection flaw. The backend function updateProductReviews directly incorporates the user-supplied id field into a MongoDB query without adequate sanitization or validation.

An authenticated attacker can manipulate this id field to inject MongoDB operators, such as $ne (not equal). This alters the database query's intended behavior, changing it from targeting a single review to selecting and affecting multiple (or all) records.

IMPACT

An authenticated user can exploit this vulnerability to execute a mass update operation. By injecting a crafted payload, the attacker can update all product reviews in the application simultaneously, irrespective of who originally created them.

This can be abused for:

- Mass Defacement: Modifying the content of all reviews to malicious or unwanted text.
- Spam Injection: Injecting large volumes of spam across the application.
- Data Corruption: Altering or deleting data, leading to a loss of data integrity and system reliability.

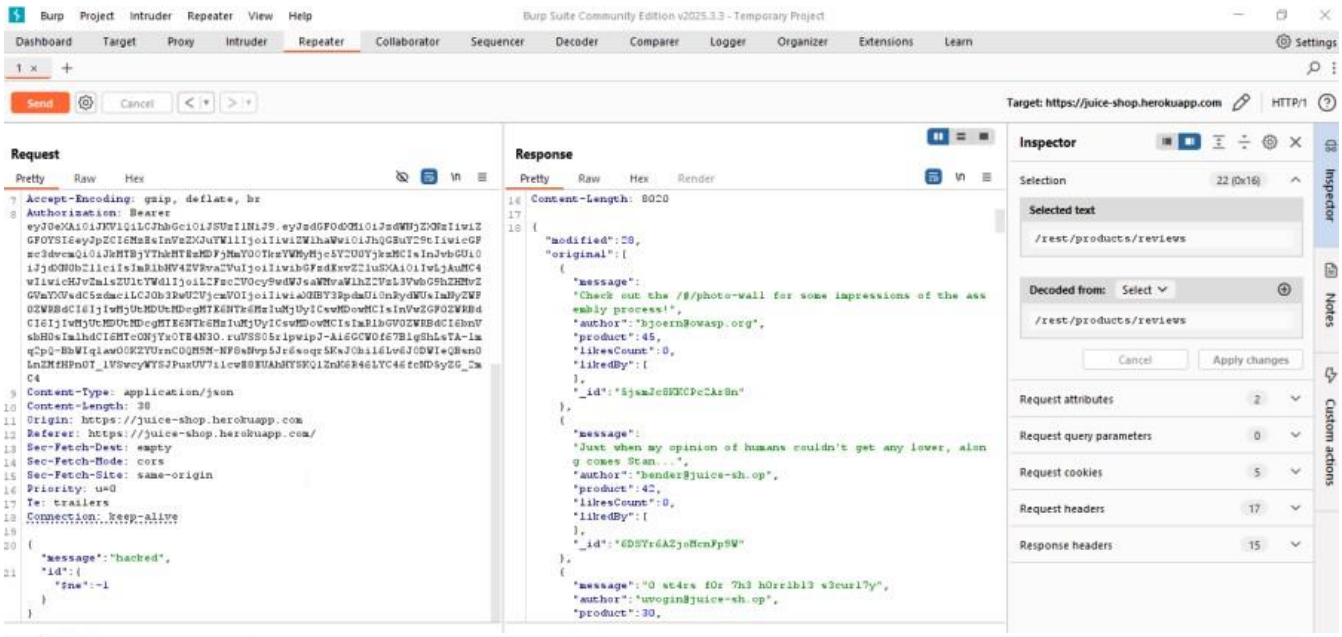
RECOMMENDATIONS

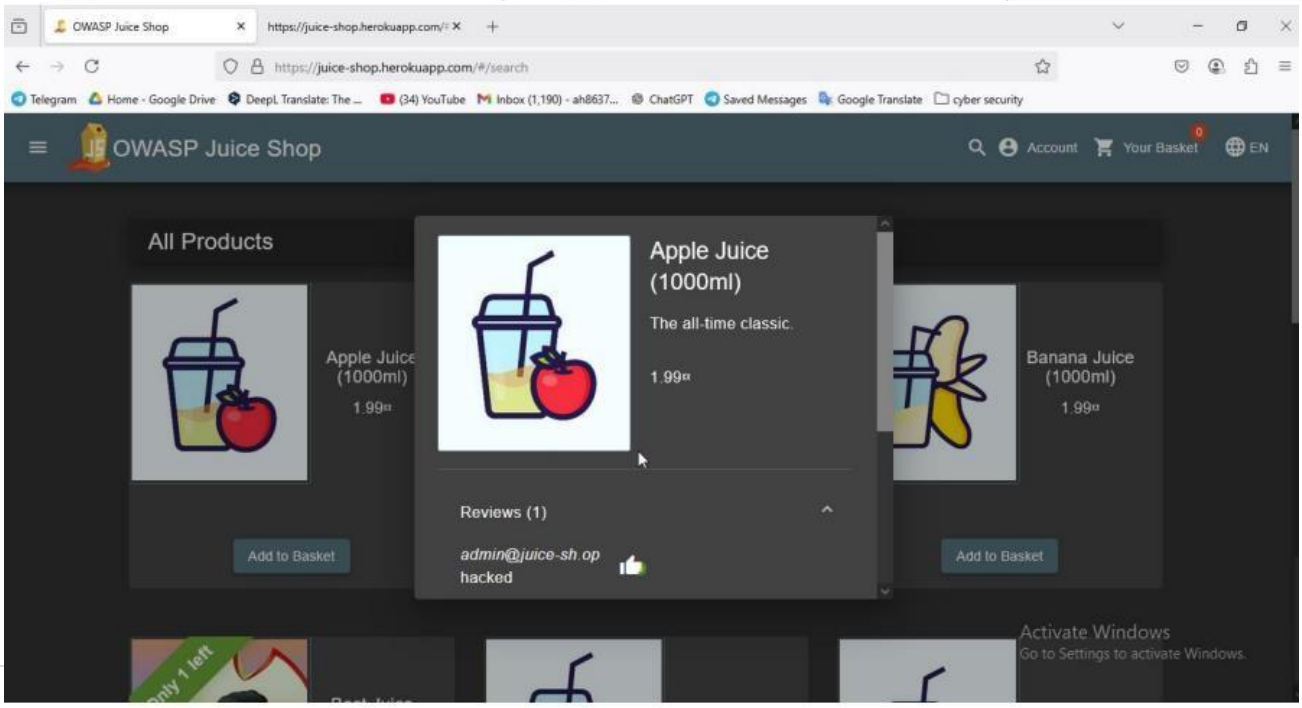
- Validate Input: Ensure that the id field only accepts primitive, expected values (e.g., integers or ObjectIDs). Do not allow query operators in user input.
- Use Safe Query Construction: Cast IDs to appropriate types and avoid using user input directly in queries.
- Ownership Enforcement: Implement 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

D.T

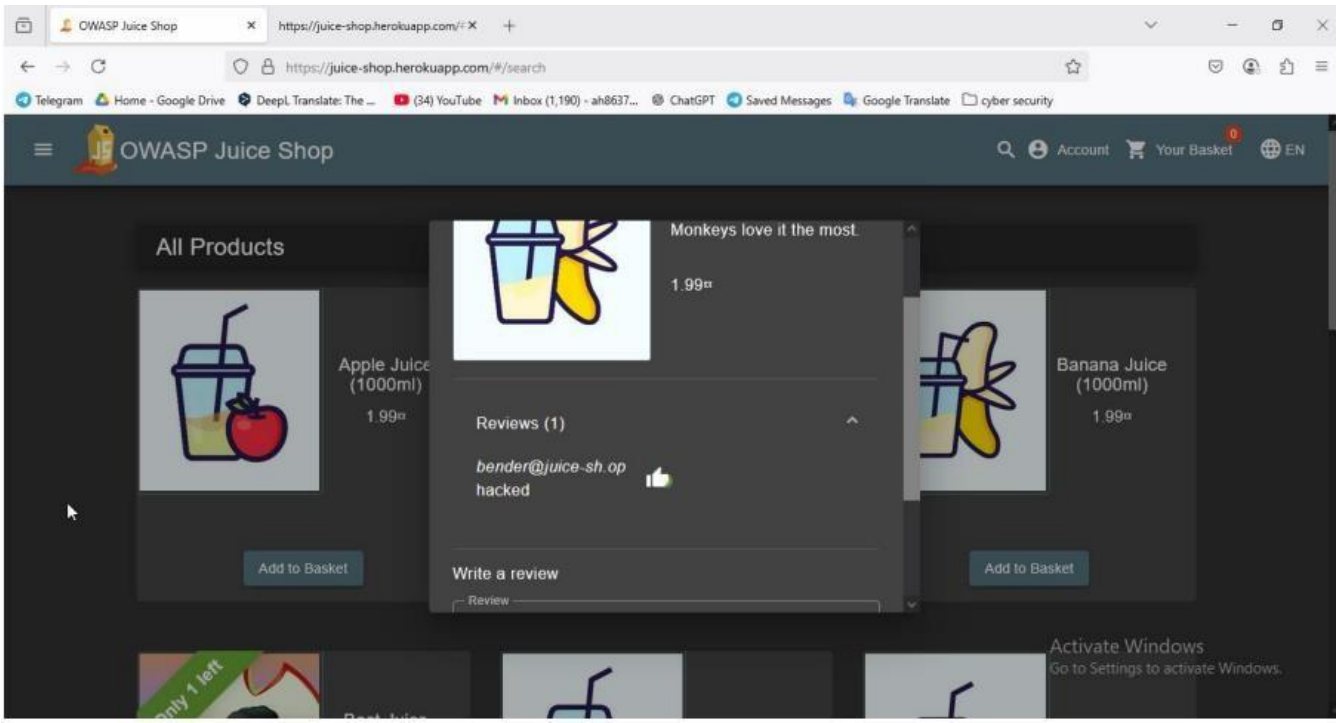
STEPS TO REPRODUCE

```
Request (PATCH to /rest/products/reviews):
{
"id": { "$ne": -1 },
 "message": "Hacked "
}
```





D.T


OWASP Juice Shop
https://juice-shop.herokuapp.com/#/search
OWASP Juice Shop
Account
Your Basket
EN
All Products
Monkeys love it the most.
1.99¤
Apple Juice (1000ml)
1.99¤
Reviews (1)
bender@juice-sh.op
hacked
Write a review
Review
Add to Basket
Banana Juice (1000ml)
1.99¤
Add to Basket
Only 1 left
Activate Windows
Go to Settings to activate Windows.

5.8 Deprecated Interface - Security Misconfiguration

Severity	High
CVSS	CVSS 7.1
Affected Assets	• File upload Function
Reference	https://pwning.owasp-juice.shop/companion-guide/latest/part2/security-misconfiguration.html

DESCRIPTION

A security misconfiguration vulnerability was identified in the complaint 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

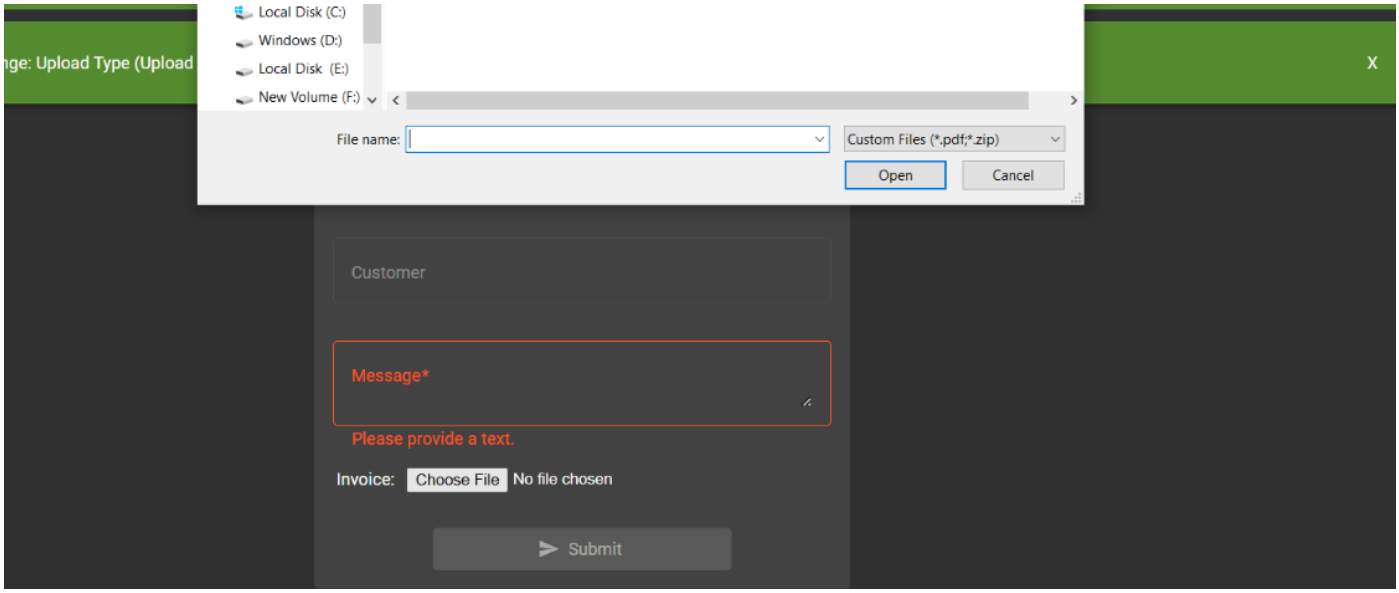
RECOMMENDATIONS

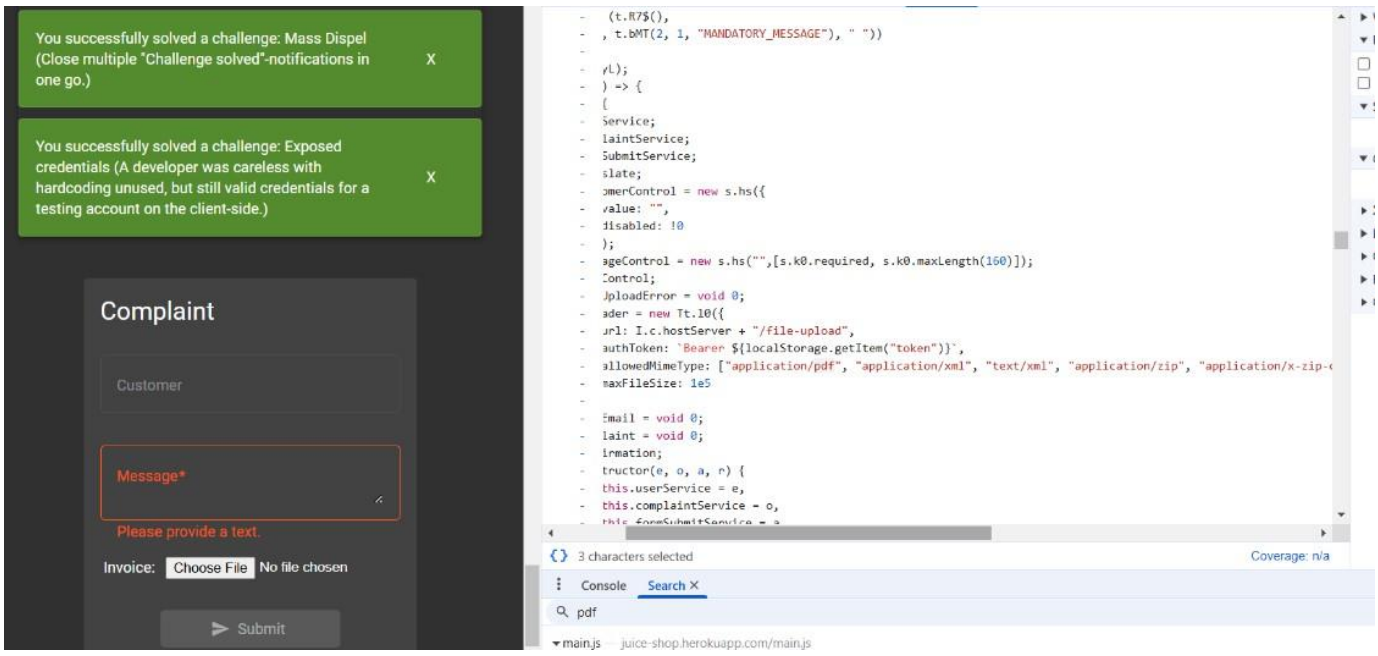
To prevent this type of attack, the application should:

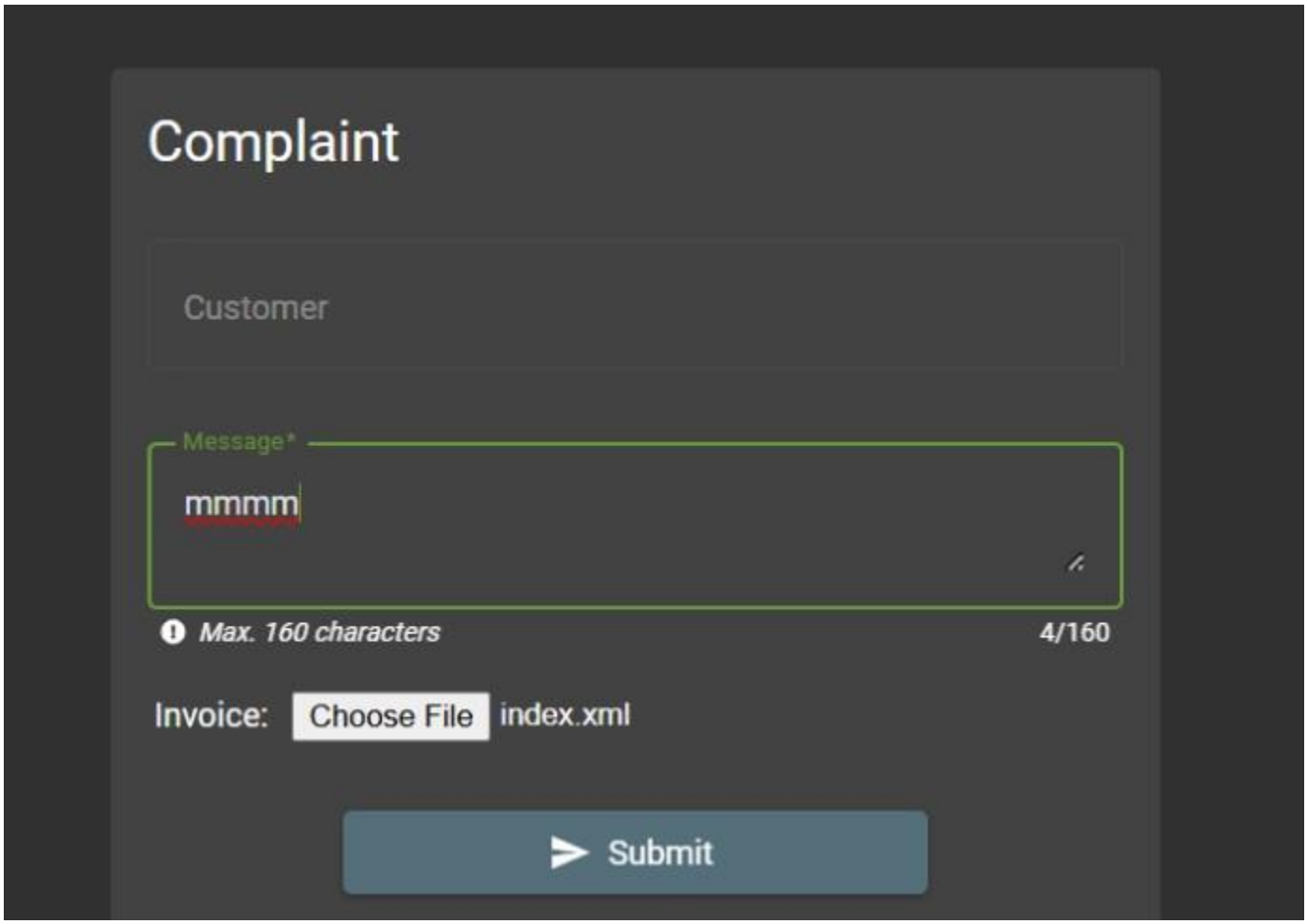
- Implement strict server-side file type validation using MIME type and file signature checks.
- Remove support for unused or deprecated file types (e.g., XML) if not explicitly required.
- Apply a whitelist approach for file uploads and avoid relying solely on file extensions or client-side

D.T

STEPS TO REPRODUCE






Complaint
Customer
Message*
mmmm
Max. 160 characters
4/160
Invoice:
Choose File
index.xml
Submit

5.9 XSS

Severity	High
CVSS	CVSS 6.3
Affected Assets	• search input field
Reference	https://owasp.org/www-community/attacks/DOM_Based_XSS

DESCRIPTION

The application is vulnerable to both DOM-Based XSS and Reflected XSS via HTTP Headers due to insufficient input validation and output encoding.

IMPACT

A successful XSS exploit, whether DOM-based or Reflected, allows an attacker to execute arbitrary scripts in the victim's browser, leading to severe consequences:

- Session Hijacking: Stealing the user's session cookies or authentication tokens.
- Data Theft: Capturing private user data displayed on the page.
- Impersonation: Performing actions on the application as if they were the legitimate user
- Phishing/Defacement: Modifying what the user sees on the page (e.g., displaying fake login forms) or redirecting them to external, malicious websites.
- Reputation Damage: Eroding user trust in the security of the application.

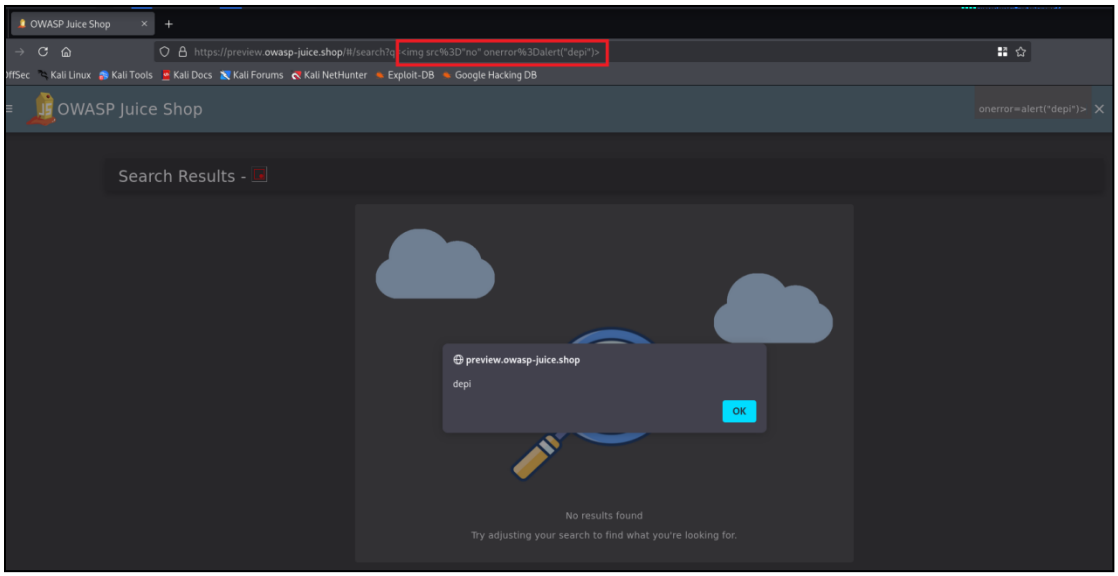
RECOMMENDATIONS

- **Input Validation & Output Encoding**
- **Secure Client-Side Development (DOM XSS)**
- **Header Handling (Reflected XSS)**
- **Defense-in-Depth**

STEPS TO REPRODUCE

SCENARIO 1:

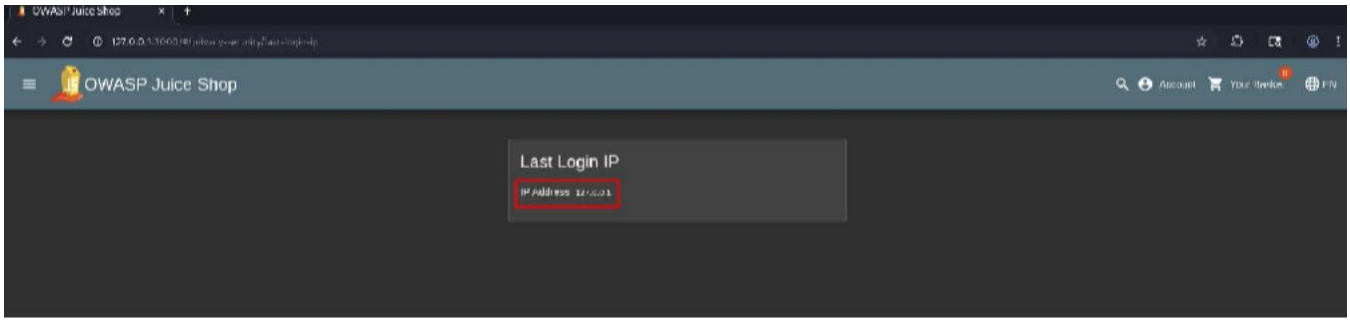
- Navigate to the main page of the website
- Locate the search input field
- Enter the following payload into the search box:
- <img src =”” onerror=alert(“depi”) >
- Submit the search request or press Enter



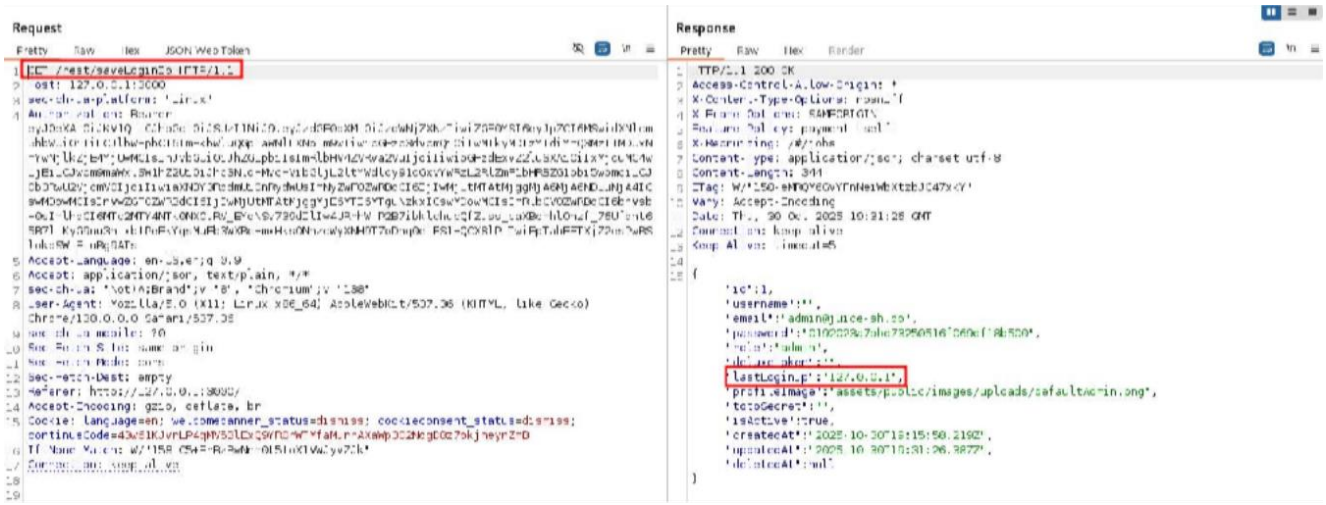
- Result
- The injected JavaScript payload is executed, triggering an alert box with the text "depi", confirming the presence of a DOM-Based Cross-Site Scripting (XSS) vulnerability.

SCENARIO 2:

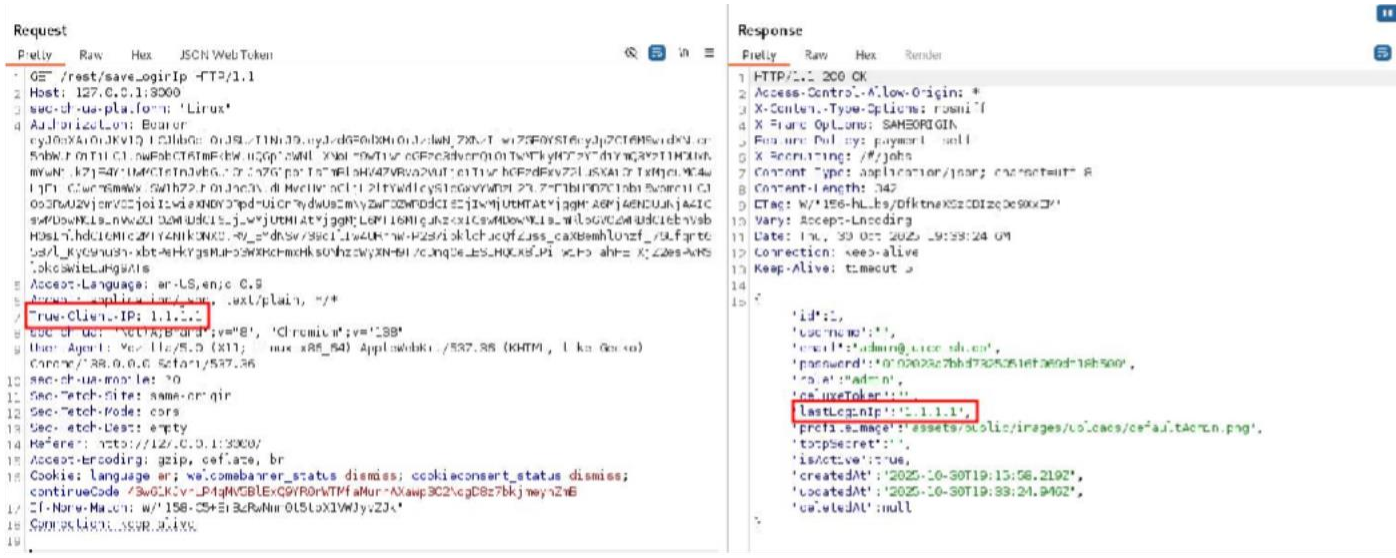
- log in to the application. After that, log out to capture the request to the /rest/saveLoginIp endpoint



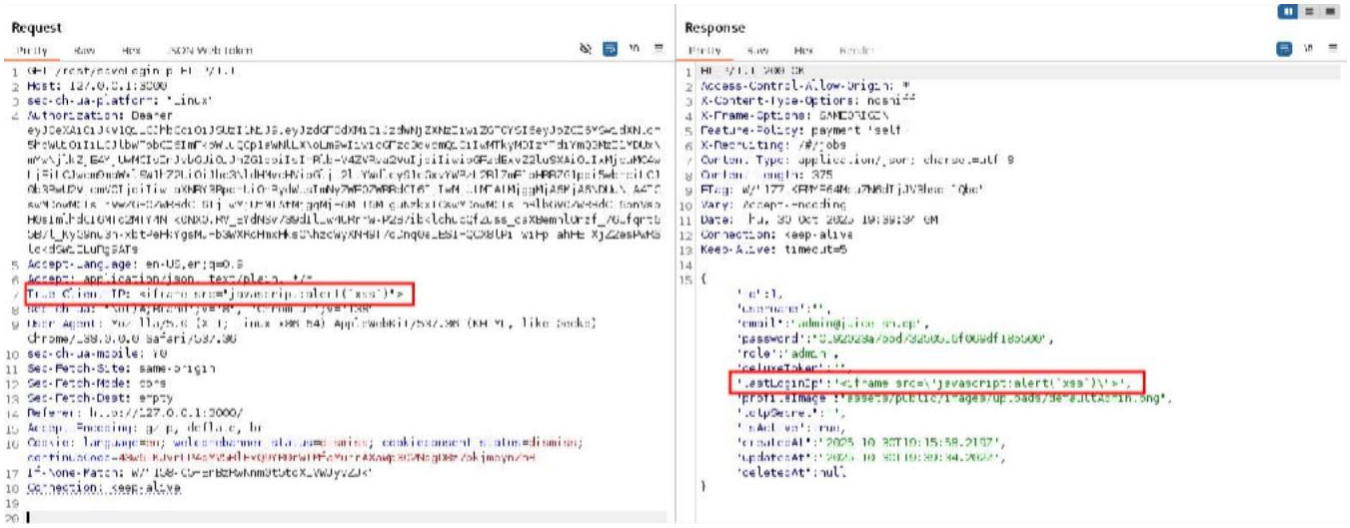
- Intercept the request then send it to the repeater to modify it.



- I searched about how to spoof client ip via header, i found X-Forwarder-For, True-Client-IP.
- I tried x-forwarder-for but it doesn't work, but true-client-ip worked.



- So let's inject our payload:



D.T

SCENARIO 2:

https://preview.owasp-juice.shop/api/products/5

Request

Pretty Raw Hex

```
1  GET /api/products HTTP/2
2  Host: preview.owasp-juice.shop
3  Cookie: language=en
4  Sec-Ch-Ua: "Not_A Brand";v="99", "Chromium";v="142"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "Windows"
7  Accept-Language: en-US,en;q=0.9
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
10 Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
   ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: none
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Accept-Encoding: gzip, deflate, br
16 Priority: u=0, i
17
```

Request

Pretty Raw Hex

```
1  PUT /api/products/5 HTTP/2
2  Host: preview.owasp-juice.shop
3  Cookie: language=en
4  Sec-Ch-Ua: "Not_A Brand";v="99", "Chromium";v="142"
5  Sec-Ch-Ua-Mobile: ?0
6  Sec-Ch-Ua-Platform: "Windows"
7  Accept-Language: en-US,en;q=0.9
8  Upgrade-Insecure-Requests: 1
9  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/142.0.0.0 Safari/537.36
0  Accept:
   text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,ima
   ge/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
1  Sec-Fetch-Site: none
2  Sec-Fetch-Mode: navigate
3  Sec-Fetch-User: ?1
4  Sec-Fetch-Dest: document
5  Accept-Encoding: gzip, deflate, br
6  Priority: u=0, i
7  Content-Type: application/json
8  Content-Length: 64
9
0  {
     "description":"<iframe src=\"javascript:alert('xss')\">"
   }
1
2
```

Response

Pretty Raw Hex Render

```
1  HTTP/2 200 OK
2  Access-Control-Allow-Origin: *
3  Content-Length: 235
4  Content-Type: application/json; charset=utf-8
5  Date: Sun, 30 Nov 2025 15:48:50 GMT
6  Etag: W/"eb-qHlcoY7Lt0BFU0nMzbx0LMr0aIw"
7  Feature-Policy: payment 'self'
8  Nel:
   {"report_to":"heroku-nel","response_headers":["Via"],"max_age":3600,"success_fr
   action":0.01,"failure_fraction":0.1}
9  Report-To:
   {"group":"heroku-nel","endpoints":[{"url":"https://nel.heroku.com/reports?s=BhH
   1d1BMytArAyJ8TqKHR93zwM1TiFNwNeZiiCKIypw%3D\u0026sid=812dcc77-0bd0-43b1-a5f1-b2
   5750382959\u0026ts=1764517730"}],"max_age":3600}
10 Reporting-Endpoints:
   heroku-nel="https://nel.heroku.com/reports?s=BhH1d1BMytArAyJ8TqKHR93zwM1TiFNwNe
   ZiiCKIypw%3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&ts=1764517730"
11 Server: Heroku
12 Vary: Accept-Encoding
13 Via: 1.1 heroku-router
14 X-Content-Type-Options: nosniff
15 X-Frame-Options: SAMEORIGIN
16 X-Recruiting: /#/jobs
17
18 {
     "status":"success",
     "data":{
       "id":5,
       "name":"Lemon Juice (500ml)",
       "description":"",
       "price":2.99,
       "deluxePrice":1.99,
       "image":"lemon_juice.jpg",
       "createdAt":"2025-11-30T08:48:13.445Z",
       "updatedAt":"2025-11-30T15:48:50.810Z",
       "deletedAt":null
     }
   }
```

5.10 File upload

Severity	Medium
CVSS	CVSS 6.3
Affected Assets	• File Upload Handler & Server-Side File Storage
Reference	https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

DESCRIPTION

The file upload feature fails to properly validate uploaded files on the server side. By intercepting the upload request and modifying both the file extension and the Content-Type header, it was possible to upload a file type that should be restricted. This demonstrates that the application relies on superficial client-controlled checks, allowing attackers to bypass file type restrictions and upload unauthorized files.

IMPACT

An attacker can upload disallowed file types (such as .xml), which may result in:

- **Malicious Content Storage:** Unauthorized or malicious content being stored on the server.
- **XXE Attacks:** XML External Entity attacks if the uploaded XML is later parsed by the application.
- **Security Risks:** Information disclosure, Server-Side Request Forgery (SSRF), Denial of Service (DoS), or other logic abuse during downstream processing of unsafe files.

RECOMMENDATIONS

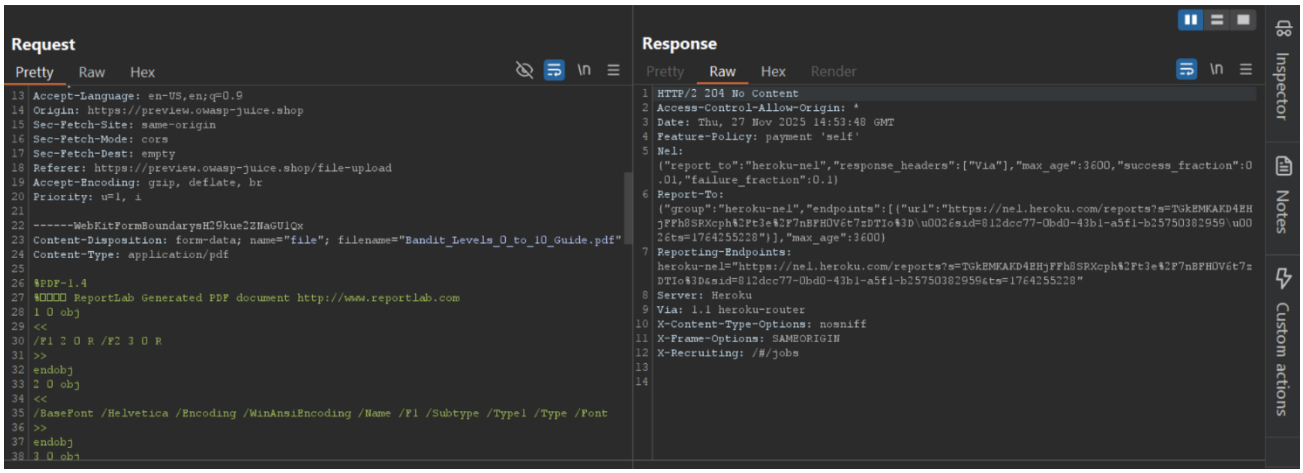
- Implement Server-Side Validation
- Validate File Content
- Block Dangerous Formats
- Use Secure Upload Libraries
- Sanitize File Names and Paths

STEPS TO REPRODUCE

1. Authenticate and navigate to the file upload functionality.

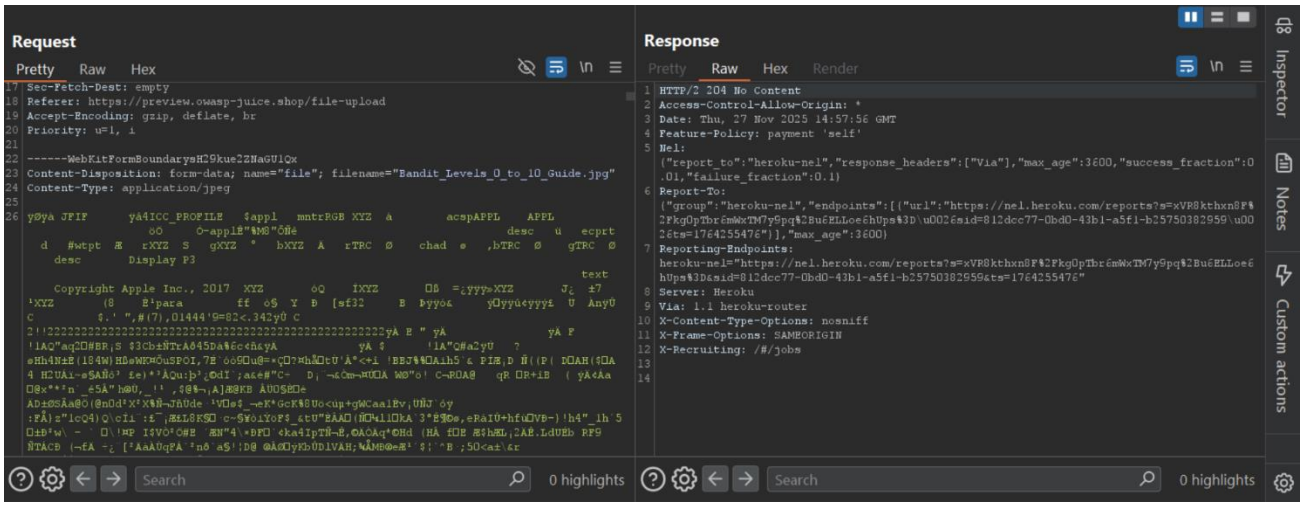
2. Upload any file while intercepting the request in Burp Suite.

D.T



3. Modify the captured request by changing:
• the file extension to .jpeg (or another allowed type)
• the content-type header to application/xml or another acceptable mime type

4. Forward the modified request to the server and observe that the upload is accepted despite containing a disallowed file type.



Result:
the server accepts and stores the file despite it containing a disallowed format, demonstrating that the intended file type restrictions can be bypassed.

5.11 Admin Panel Path Disclosure via main.js

Severity	Medium
CVSS	CVSS 5.3
Affected Assets	• `main.js` (public-facing JavaScript file)
Refrences	https://owasp.org/Top10/A05_2021-Security_Misconfiguration/

DESCRIPTION

During source code review of public-facing JavaScript files, the path to the Administrative Dashboard was found hardcoded inside `main.js`. Although access to the dashboard requires authentication, exposing such sensitive endpoints on the client-side increases the attack surface and provides unnecessary information to potential attackers.

IMPACT

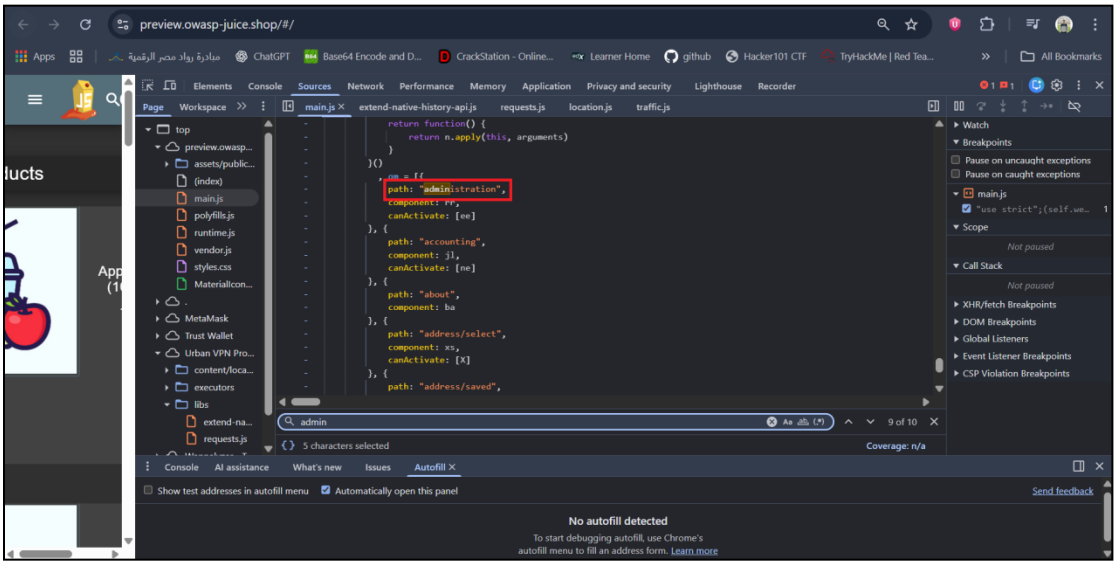
Exposing the administrative dashboard path (/adminstration) in a publicly accessible JavaScript file significantly aids attackers during reconnaissance, increases the attack surface, and enables targeted exploitation attempts. Although authentication is required, disclosing such sensitive information can facilitate brute-force attacks, access control testing, and exploitation chaining with other vulnerabilities, which may ultimately lead to full system compromise.

RECOMMENDATIONS

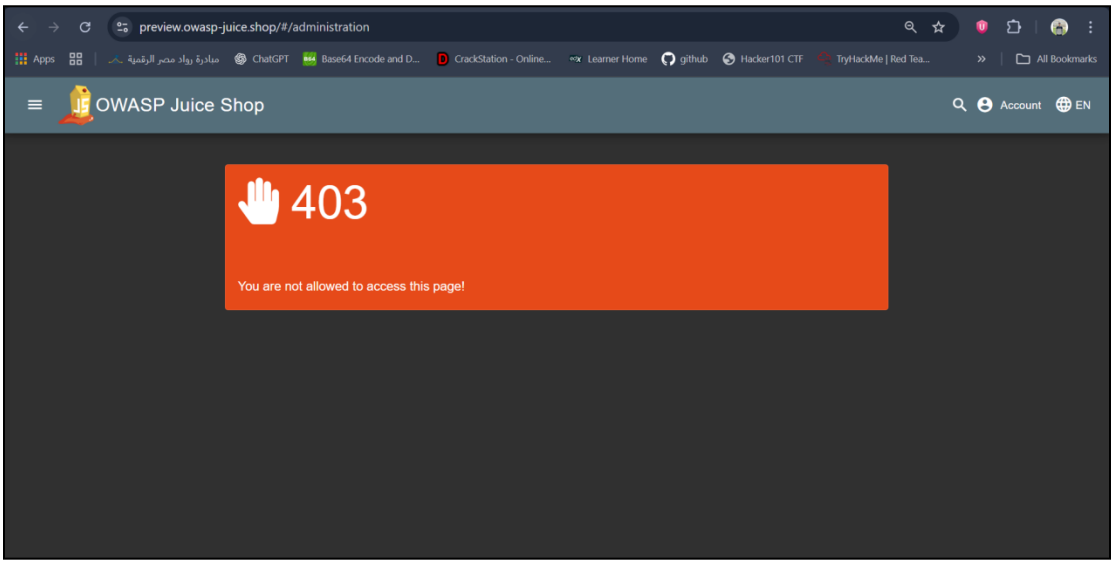
Remove the administrative panel path from all client-side JavaScript files and ensure that sensitive endpoints are only handled on the server side. Implement strict server-side access controls for all administrative routes, review the frontend build process to prevent similar exposures, and perform a code audit to identify and remove any additional sensitive information disclosed in client-side resources. Additionally, enable proper logging and monitoring for any access attempts to administrative endpoints.

STEPS TO REPRODUCE

1. **Inspect the client-side JavaScript files** by opening the browser's Developer Tools.
2. **Navigate to the Sources tab** to view all loaded JavaScript files.
3. **Locate and open the** main.js **file** from the list of resources.
4. **Search for sensitive admin-related endpoints** using the search feature



5. **Attempt to access the discovered endpoint** directly through the browser.



6. Observe that the endpoint exists but is only accessible when authenticated as an admin, confirming that the path is valid but improperly exposed in client-side code.

5.12 IDOR

Severity	High
CVSS	CVSS 5.0
Affected Assets	• /rest/basket
Reference	idor

DESCRIPTION

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

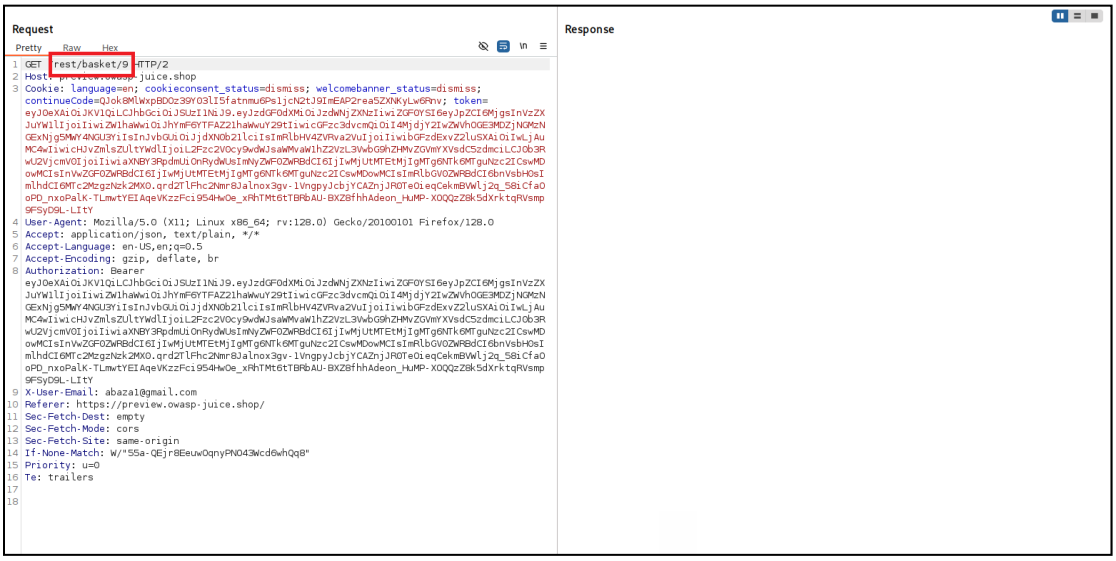
Exposure of another user's shopping basket contents can lead to the disclosure of sensitive information, such as the items they intend to purchase, quantities, and potentially any discounts applied. This can reveal shopping habits, personal preferences, and could be a precursor to further malicious activities.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

RECOMMENDATIONS

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).

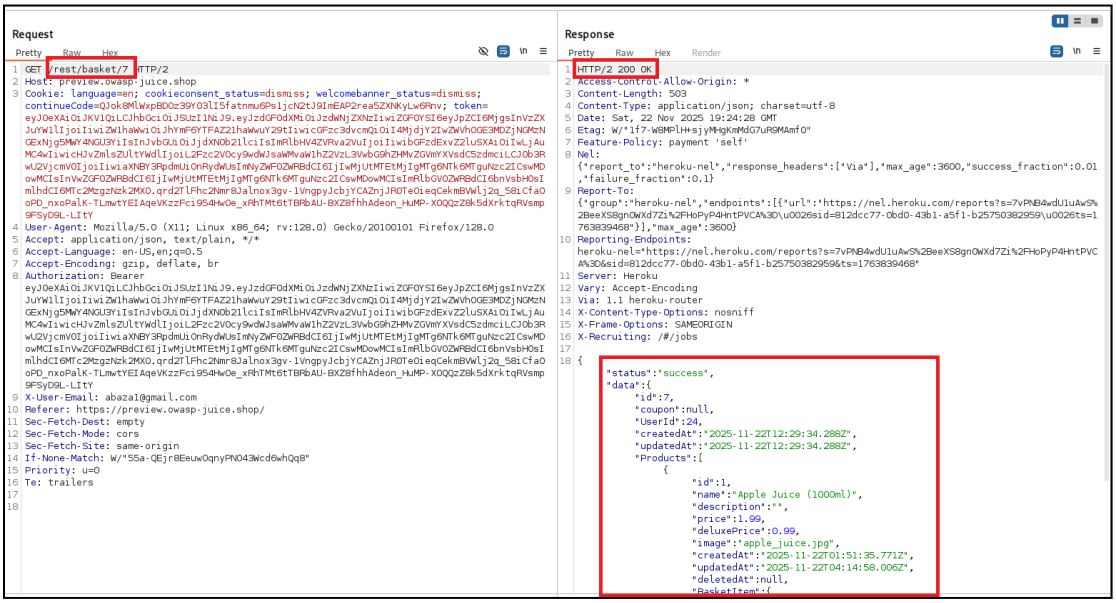
STEPS TO REPRODUCE

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
4. Intercept this request by burp suite



5. Note that the basket ID in the request.

6. I tried changing the ID to a random one.



7. 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.