FINAL PROJECT

(PENETRSTION TESTING)

Vulnerability 1

Vulnerability Name: DOM-Based Cross-Site Scripting (DOM XSS)

Vulnerability Category: Cross-Site Scripting (XSS)

CVSS Score: 6.1 (Medium)

Juice Shop Challenge Difficulty: 1 Star

Description

DOM-Based Cross-Site Scripting (DOM XSS) occurs when client-side JavaScript processes attacker-controlled input and inserts it into the Document Object Model (DOM) without properly validating or sanitizing the input. In this vulnerability, an attacker can inject JavaScript code through an iframe payload containing a JavaScript URI.

The vulnerable functionality allows the injected payload to be interpreted and executed within the victim's browser context.

Affected Functionality or Endpoint

The affected functionality is the client-side functionality of the OWASP Juice Shop application that processes the attacker-controlled input used in the DOM XSS challenge.

Proof of Concept (PoC)

The vulnerability was successfully demonstrated using the following payload:

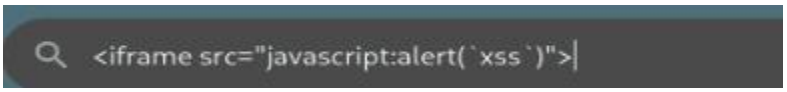
```
<iframe src="javascript:alert(\xss`)">`
```

When the payload is processed by the vulnerable functionality, the JavaScript code executes in the browser and displays an alert containing the text xss.

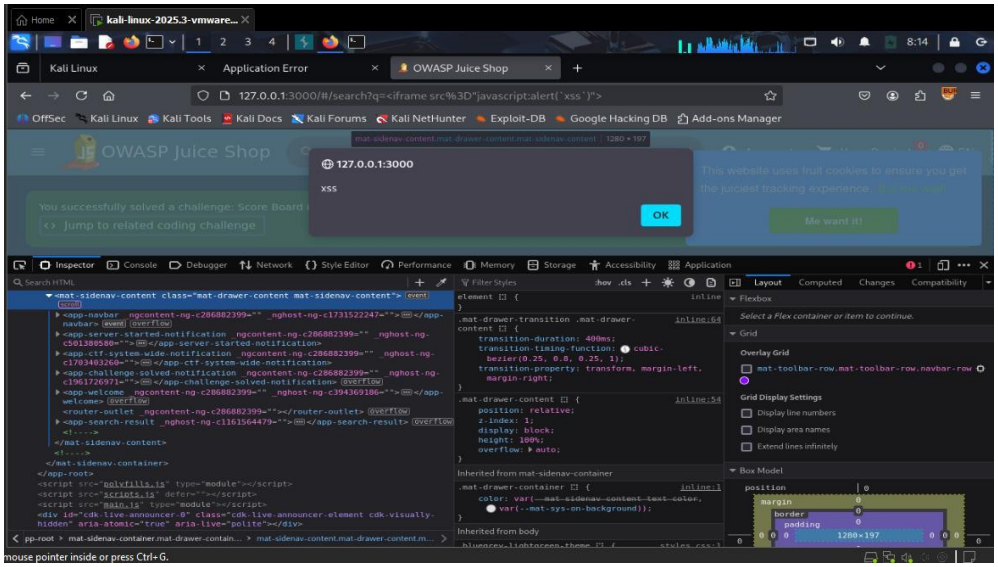
Screenshots / Evidence

Screenshots will be provided showing:

1. The injected payload.



2. The resulting JavaScript alert.



Security Impact

Successful exploitation of DOM XSS allows an attacker to execute arbitrary JavaScript in a victim's browser within the application's security context. Depending on the application's functionality and security controls, this could potentially be used to manipulate page content, perform actions on behalf of the victim, or access information available to client-side JavaScript.

How to Fix the Vulnerability

User-controlled input should be properly validated and safely handled before being inserted into the DOM. The application should avoid dangerous DOM manipulation methods and JavaScript URI schemes when processing untrusted input. Appropriate output encoding and sanitization should also be implemented, together with a restrictive Content Security Policy where applicable

Vulnerability 2 Administrator Login Injection

Vulnerability Name: Admin Login

Vulnerability Category: SQL Injection

CVSS Score: 9.8 (Critical)

Juice Shop Challenge Difficulty: 2 Stars

Description

The login functionality is vulnerable to an injection attack that allows an attacker to manipulate the application's authentication query or logic. By supplying specially crafted input in the login fields, the authentication mechanism can be bypassed.

This allows an unauthenticated attacker to successfully log in using the administrator's account without possessing the legitimate administrator credentials.

Affected Functionality or Endpoint

The affected functionality is the application's login/authentication functionality, specifically the administrator login process.

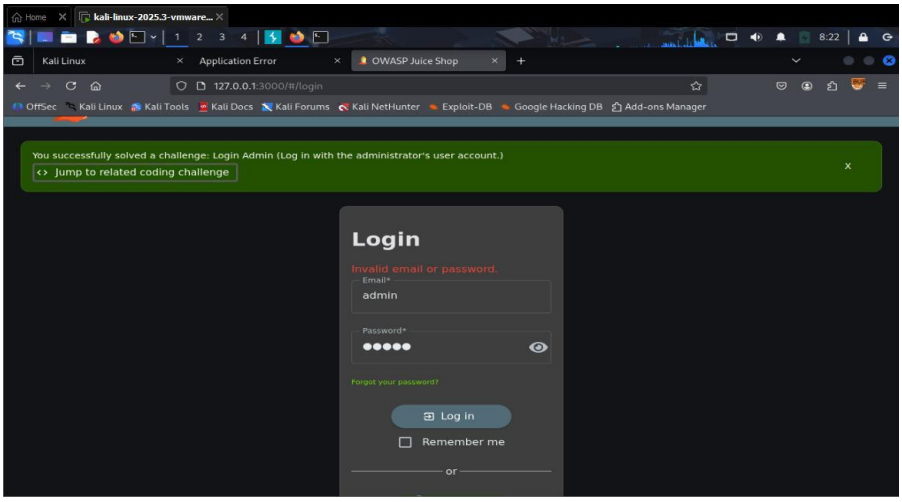
Proof of Concept (PoC)

A specially crafted injection payload was submitted through the login functionality. The application accepted the manipulated input and authenticated the attacker as the administrator without requiring the legitimate administrator password.

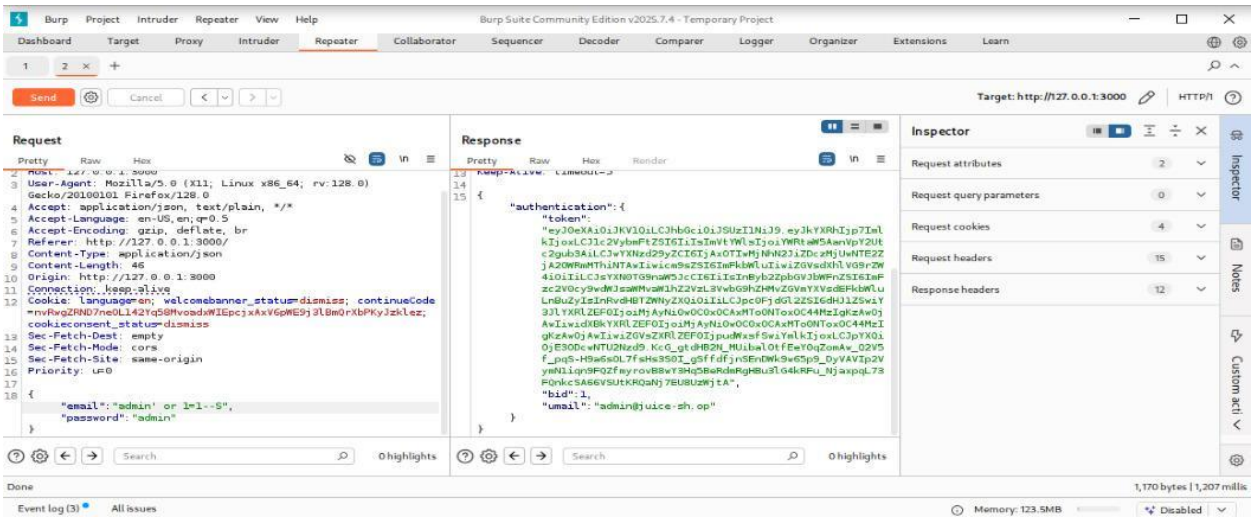
Screenshots / Evidence

Screenshots will be provided showing:

1. The login page before exploitation.
2. The injected login input.



3. Successful authentication as the administrator.
4. Relevant Burp Suite request/response evidence, if available.



Security Impact

This vulnerability can result in complete authentication bypass. An attacker who successfully exploits the issue may gain administrator-level access to the application and perform privileged actions. This could lead to unauthorized access, modification of application data, and compromise of other users' information.

How to Fix the Vulnerability

The application should use parameterized queries or prepared statements instead of dynamically constructing database queries from user input. User input should also be properly validated, and authentication logic should be implemented securely. Error messages should not reveal information about the underlying database or authentication process.

Vulnerability 3 IDOR – Basket Access

Vulnerability Name: View Basket

Vulnerability Category: Broken Access Control /IDOR

CVSS Score: 7.7 (High)

Juice Shop Challenge Difficulty: 2 Stars

Description

The application does not properly enforce authorization when accessing shopping basket information. By modifying the identifier associated with a basket request, an authenticated user can access another user's shopping basket.

This represents a Broken Access Control vulnerability and can also be classified as an Insecure Direct Object Reference (IDOR), because the application relies on a user-controlled object identifier without sufficiently verifying that the requesting user is authorized to access the referenced resource.

Affected Functionality or Endpoint

The affected functionality is the shopping basket functionality and the endpoint responsible for retrieving a user's basket.

Proof of Concept (PoC)

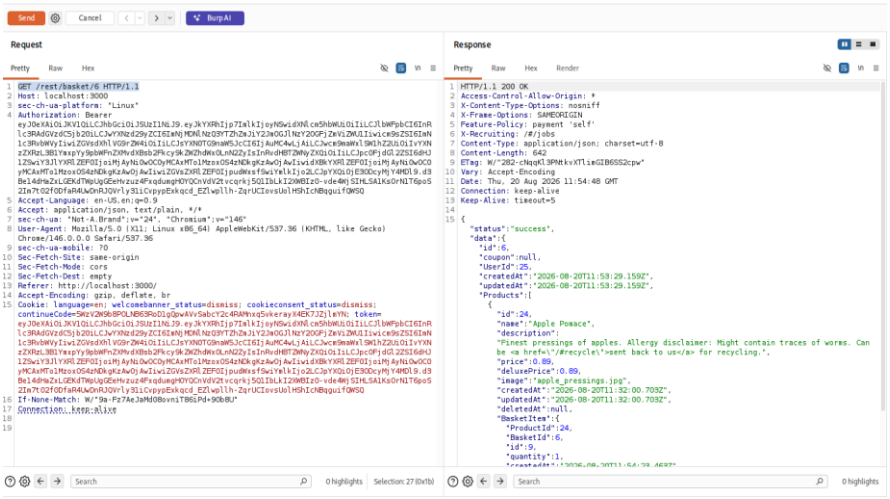
After accessing the user's own shopping basket, the basket identifier in the relevant request was modified to reference another user's basket.

The application returned the other user's basket information instead of rejecting the request, demonstrating that authorization was not properly enforced.

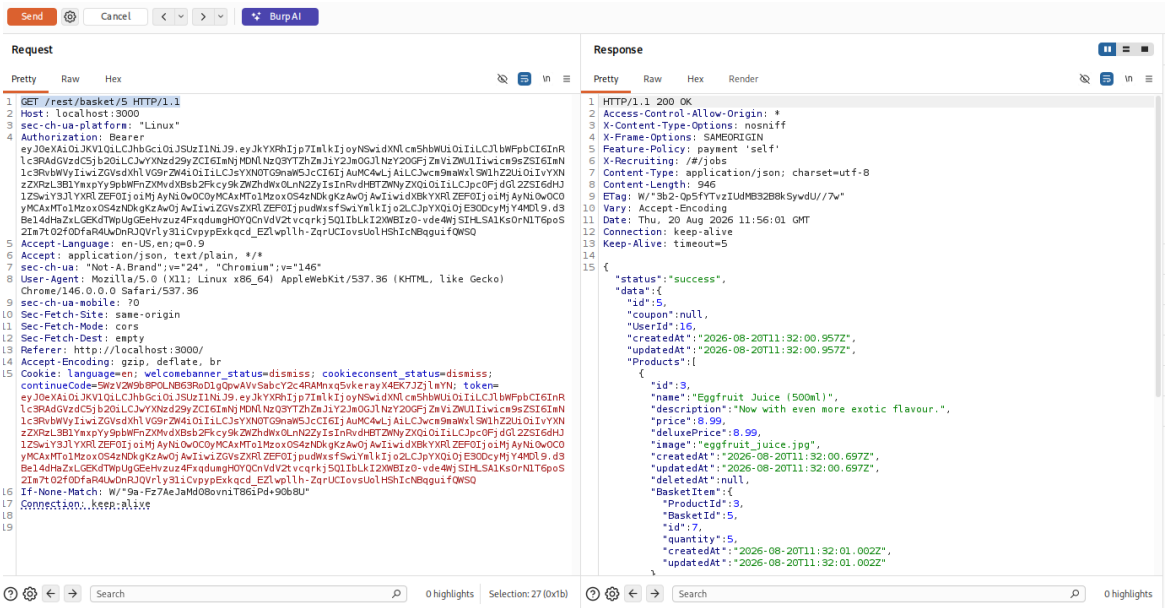
Screenshots / Evidence

Screenshots will be provided showing:

1. The original request for the user's own basket.



The modified request and response showing another user’s basket.



Security Impact

An attacker could potentially access private information belonging to other users. Depending on the information stored in the basket, this may expose products, quantities, user-related information, or other sensitive application data.

The vulnerability demonstrates that authentication alone is insufficient because the application fails to verify authorization for the requested resource.

How to Fix the Vulnerability

The server should perform an authorization check for every basket request. The application must verify that the authenticated user owns or is otherwise authorized to access the requested basket before returning its contents. Authorization checks must be performed server-side rather than relying on user-controlled identifiers.

Vulnerability 4 Price Manipulation

Vulnerability Name: Payback Time

Vulnerability Category: Business Logic Vulnerability

CVSS Score: 7.7 (High)

Juice Shop Challenge Difficulty: 3 Stars

Description

The application's payment/order functionality does not properly validate or enforce the expected business rules for order values. By manipulating the relevant input associated with an order, it is possible to create an order that results in an unintended financial outcome for the attacker.

The vulnerability demonstrates insufficient server-side validation of values used during the ordering or payment process.

Affected Functionality or Endpoint

The affected functionality is the shopping cart/order and payment processing functionality of the OWASP Juice Shop application.

Proof of Concept (PoC)

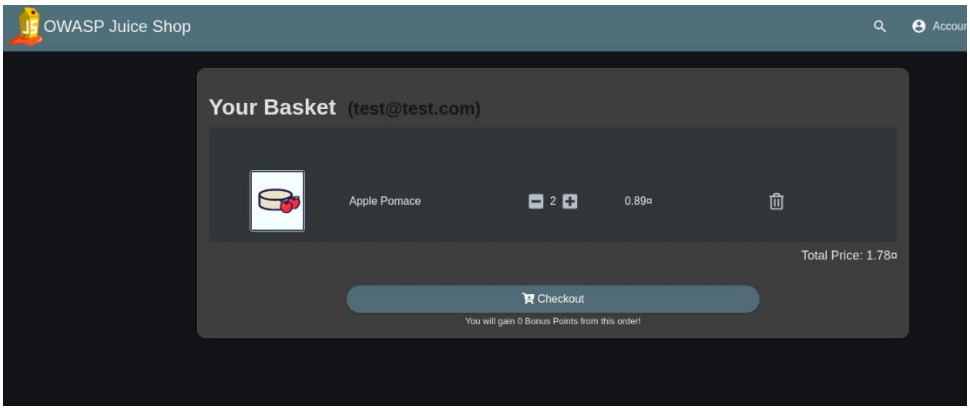
The order-related input was manipulated so that the resulting order generated an unintended positive financial balance for the attacker.

The application accepted the manipulated values instead of rejecting the invalid transaction, demonstrating that the server did not properly validate the order's financial logic.

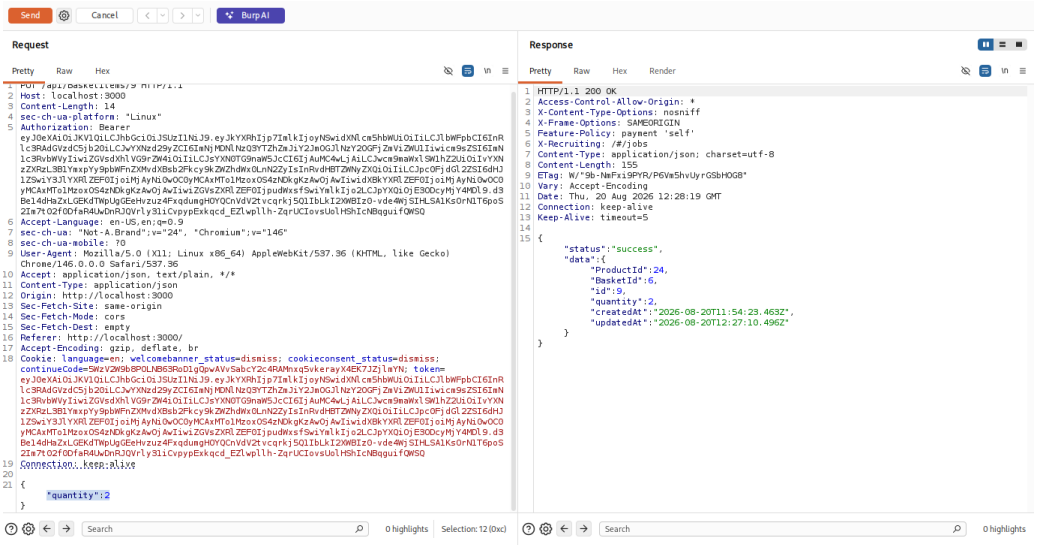
Screenshots / Evidence

Screenshots will be provided showing:

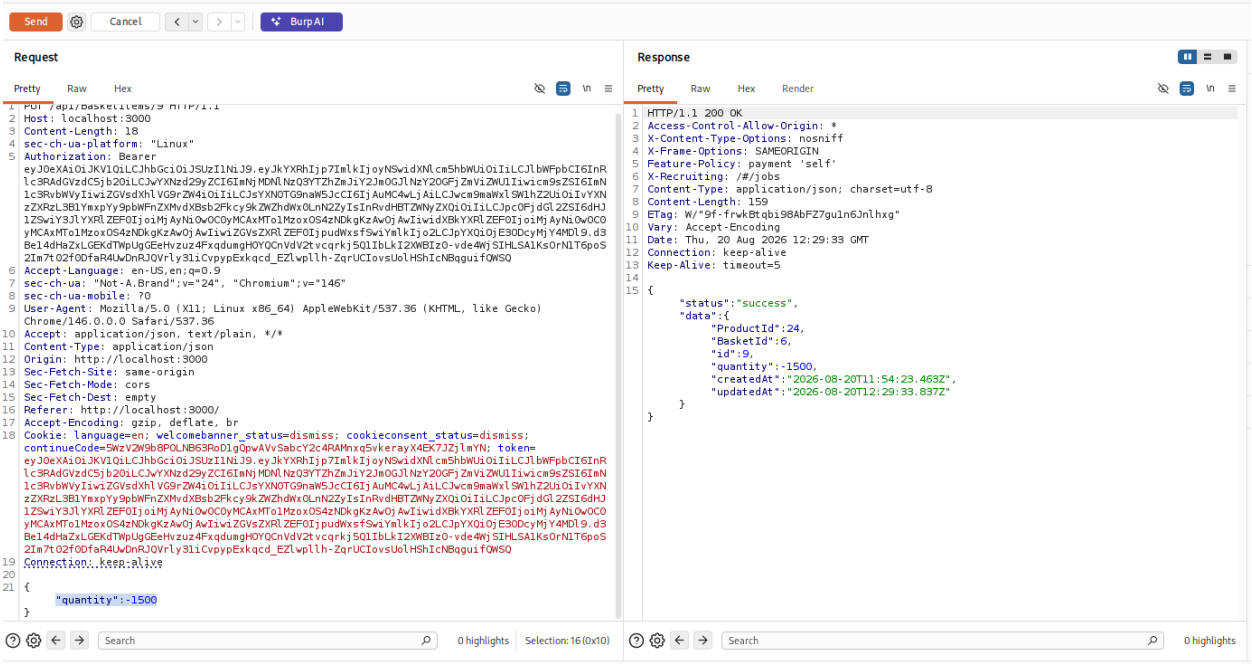
1. The original order/cart state.



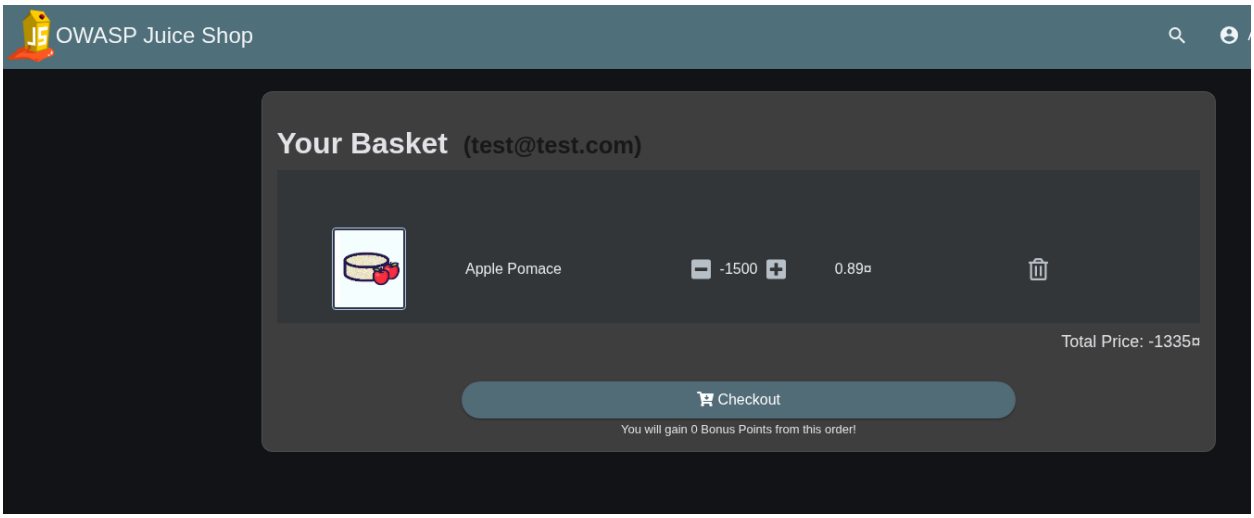
2. Original Request and Response



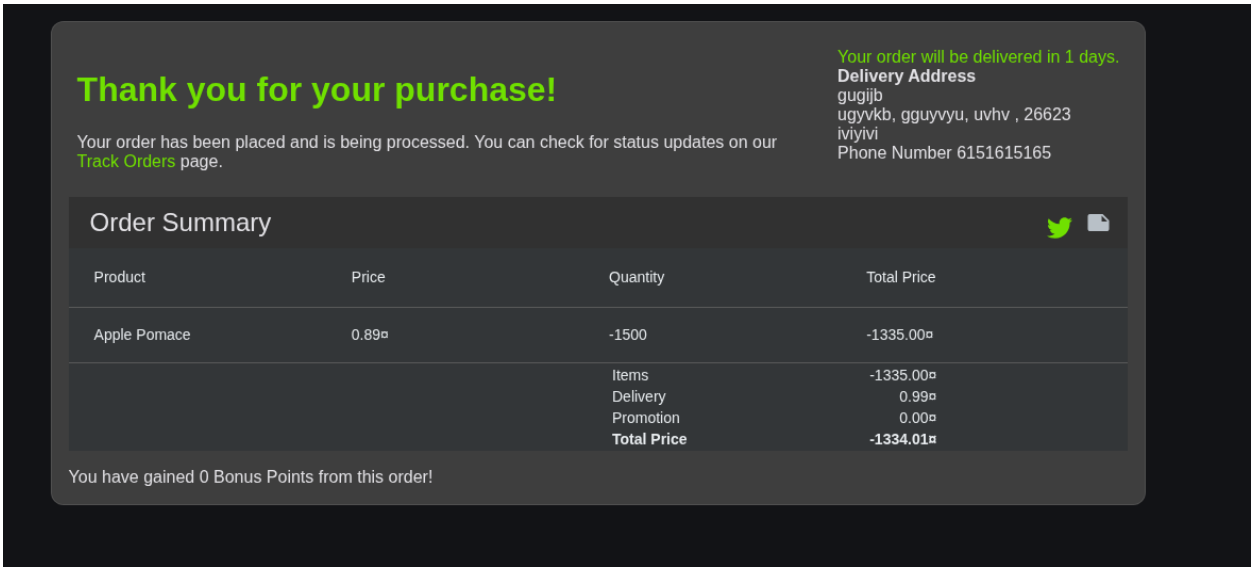
3. Manipulated Request and Response.



4. Cart Total After Price Manipulation.



5. Successful Checkout with Manipulated Price.



Security Impact

Successful exploitation could allow an attacker to manipulate financial transactions and obtain an unauthorized financial benefit. In a real-world e-commerce application, similar flaws could result in financial loss, fraudulent transactions, incorrect account balances, and significant business impact.

How to Fix the Vulnerability

All financial values and transaction calculations must be generated and validated on the server side. The application should never trust price, quantity, discount, credit, or other financially significant values supplied by the client. Business rules should be enforced server-side, and transactions should be rejected whenever calculated values fall outside the expected conditions.

Vulnerability 5 Password Reset Bypass

Vulnerability Name: Bjoern's Favorite Pet

Vulnerability Category: Broken Authentication

CVSS Score: 7.4 (High)

Juice Shop Challenge Difficulty: 3 Stars

Description

The password recovery mechanism relies on a security question as part of the authentication process. The security question and its answer can be used to reset the password of Bjoern's OWASP Juice Shop account.

The vulnerability demonstrates weaknesses in the account recovery mechanism because knowledge of the original security-question answer is sufficient to initiate a password reset without requiring a stronger form of identity verification.

Affected Functionality or Endpoint

The affected functionality is the **Forgot Password / password recovery** mechanism.

Proof of Concept (PoC)

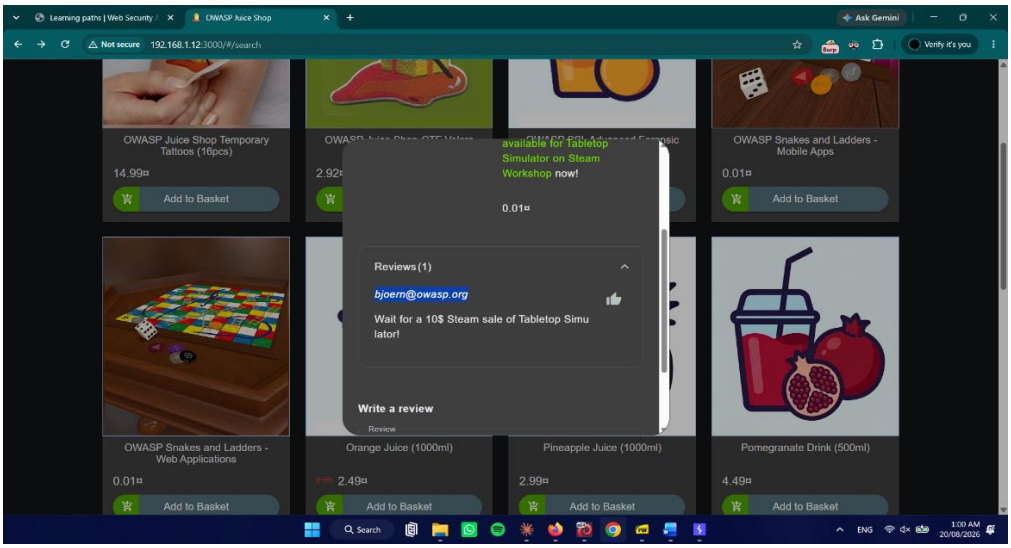
The password recovery functionality was accessed for Bjoern's account. The original answer to the account's security question was supplied, allowing the password reset process to be completed.

The successful password reset demonstrates that the account recovery mechanism can be abused when the security-question answer is known or successfully guessed.

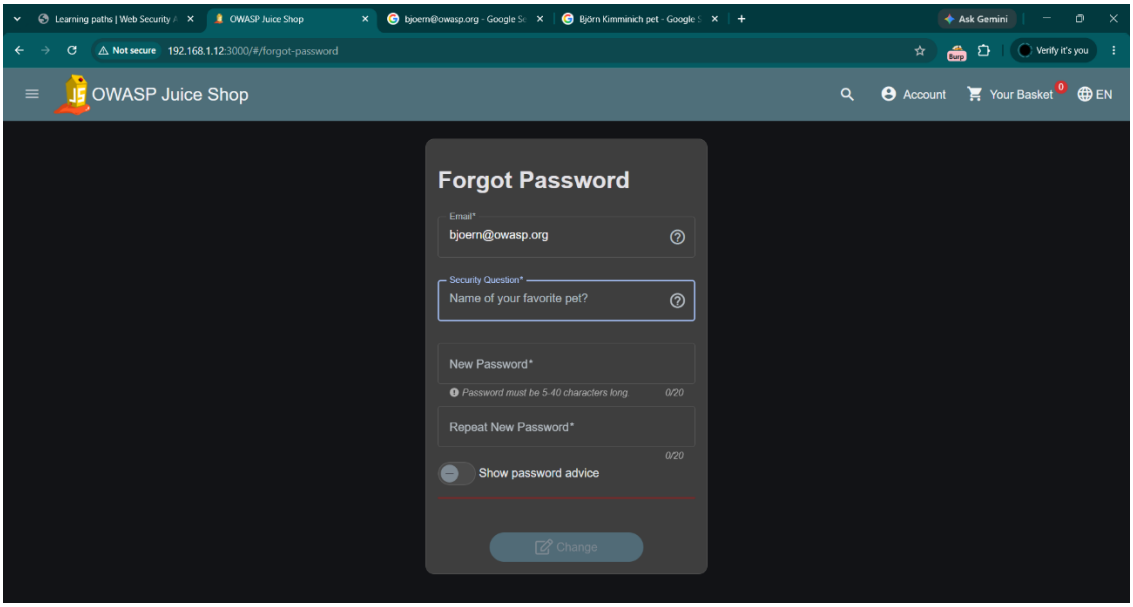
Screenshots / Evidence

Screenshots will be provided showing:

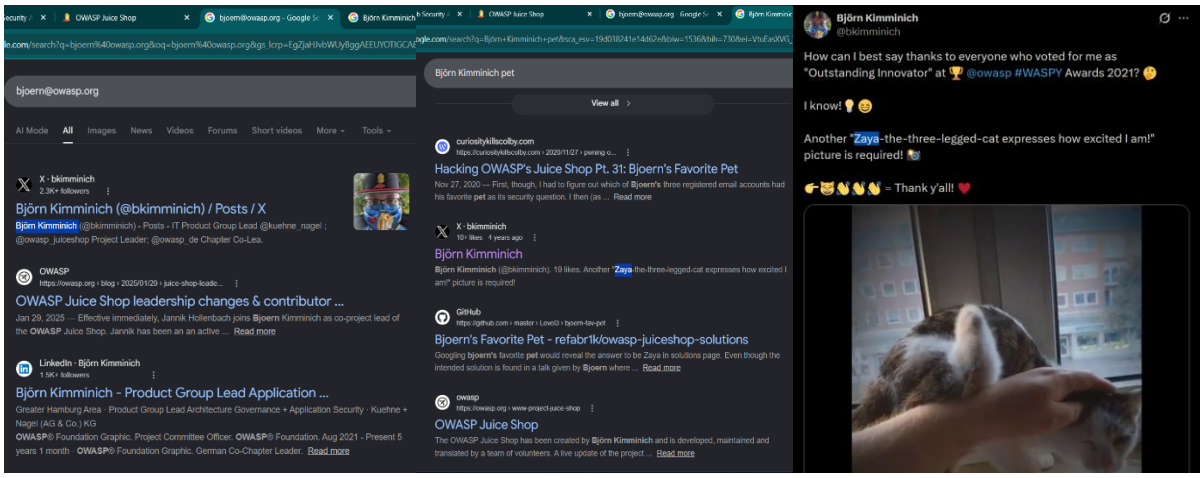
1. Finding Bjoern's email in a review.



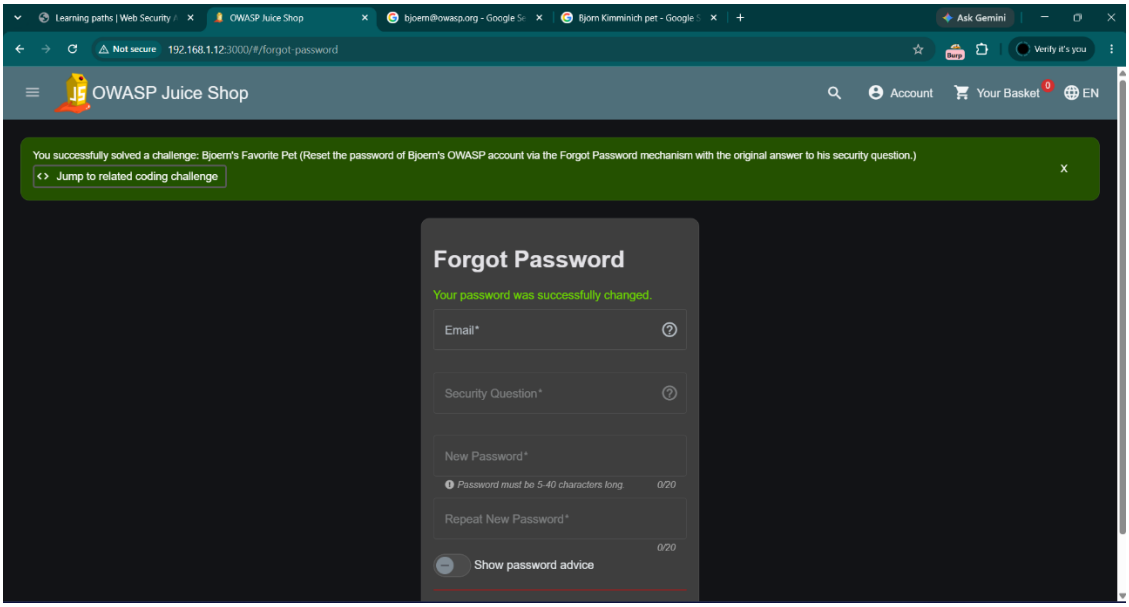
2. Figuring out Bjoern's security question by entering his email in the forgot password function.



3. Quick Google Search to find the answer.



4. Supplying the answer (Zaya) and Successful password reset.



Security Impact

An attacker who obtains or guesses the security-question answer could potentially take over the associated account. Account takeover may allow the attacker to access private information, perform actions as the victim, or make unauthorized changes to the account.

How to Fix the Vulnerability

Security questions should not be relied upon as the primary mechanism for account recovery because their answers may be publicly discoverable or easily guessed. A stronger password-reset mechanism should use a randomly generated, short-lived, single-use reset token delivered through a verified communication channel. Additional protections such as rate limiting and monitoring should also be implemented.

Vulnerability 6 Poison Null Byte File Access

Vulnerability Name: Poison Null Byte

Vulnerability Category: Path Traversal / Insecure File Handling

CVSS v3.1 Base Score: 7.5 (High)

Juice Shop Challenge Difficulty: 4 Stars

Description

The application exposes an /ftp directory that is reachable from a link on the About Us / Terms of Use page. This directory lists a number of files, including backup and legacy files that are not meant to be publicly downloadable. The server enforces a file-extension allow list on this directory, only permitting .md and .pdf files to be downloaded directly; requesting a file with any other extension (such as .bak) returns an error stating that only .md and .pdf files are allowed.

This restriction can be bypassed using a Poison Null Byte. A single URL-encoded null byte (%00) was not sufficient on its own, since the browser/URL layer decodes it once before the request reaches the application. Encoding the percent sign as well, producing %2500, ensured the null byte survived that first decoding step and was only resolved to an actual null byte once the server processed the filename internally. By appending %2500 followed by an allowed extension (.md) to the end of the restricted filename, the server's extension check inspects the trailing ".md" and treats the request as valid, while the underlying file system call still resolves the null-terminated string and returns the original restricted file. This is a classic Improper Input Validation flaw: the validation layer and the file-access layer interpret the same string differently, and the null byte is used to desynchronize the two.

Affected Functionality or Endpoint

The affected functionality is the /ftp file listing and download endpoint, specifically its file-extension validation logic. The vulnerability was demonstrated against a restricted backup file discovered in this directory (a legacy coupon file with a .bak extension).

Proof of Concept (PoC)

The /ftp directory was first located via a link on the About Us page (the "Check out our terms of use" link resolves to a file under /ftp), revealing that the directory itself is browsable.

A restricted file inside /ftp (an old coupons backup file with a .bak extension) was requested directly and rejected by the server with an "Only .md and .pdf files are allowed!" error.

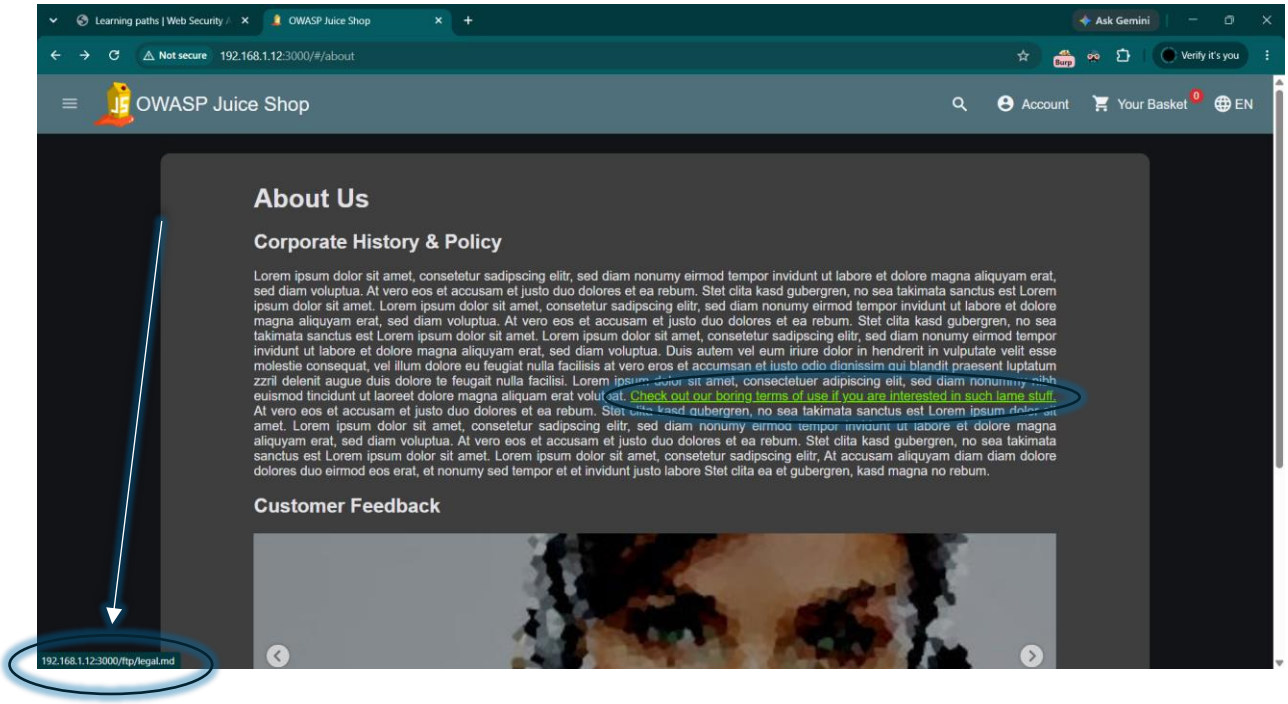
The request was then modified to append the double-encoded null byte followed by an allowed extension, in the form: /ftp/coupons_2013.md.bak%2500.md

The server accepted this request as valid because it saw the trailing ".md", but the operating system's file handling terminated the filename at the null byte, causing the original .bak file to be returned to the attacker.

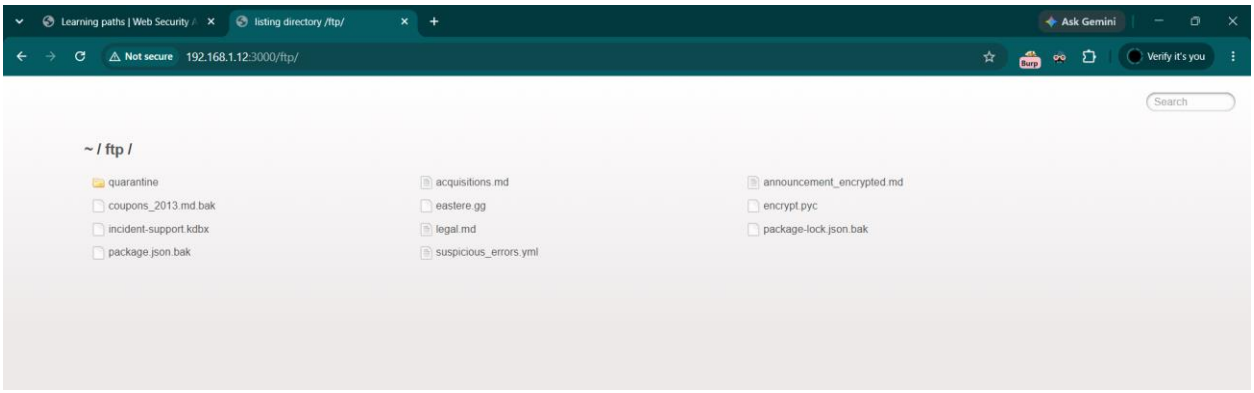
Screenshots / Evidence

Screenshots will be provided showing:

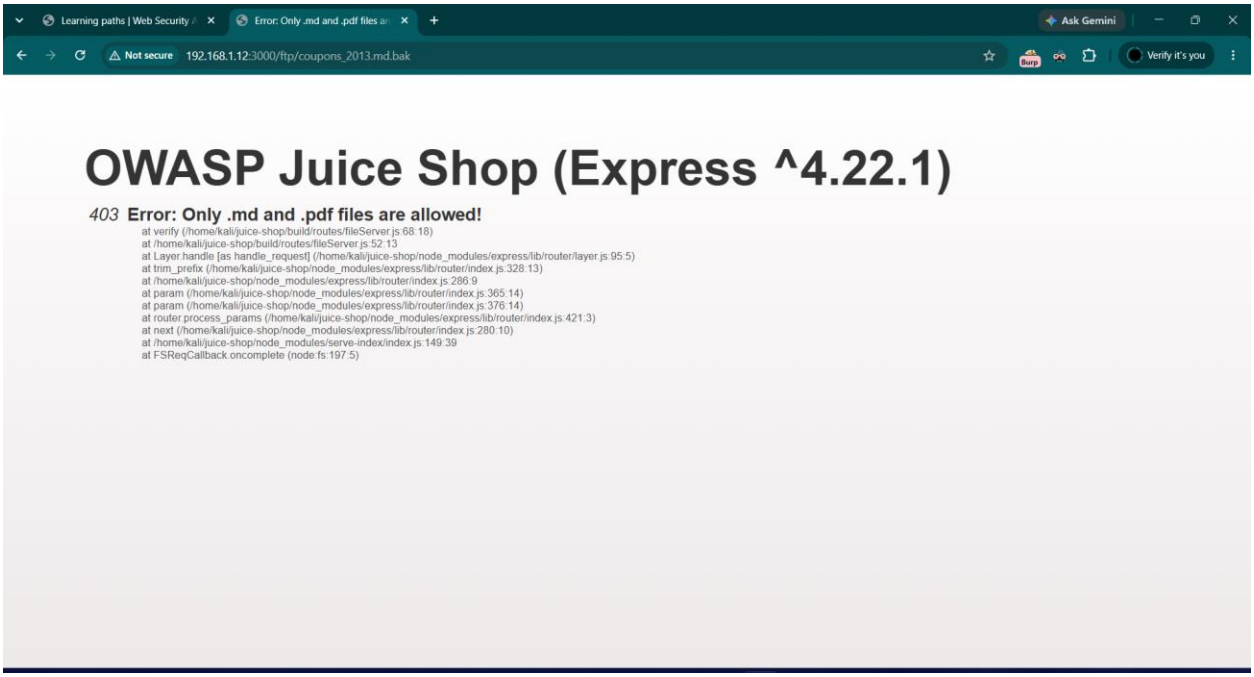
1. The About Us page with the "Check out our terms of use" link showing the /ftp URL.



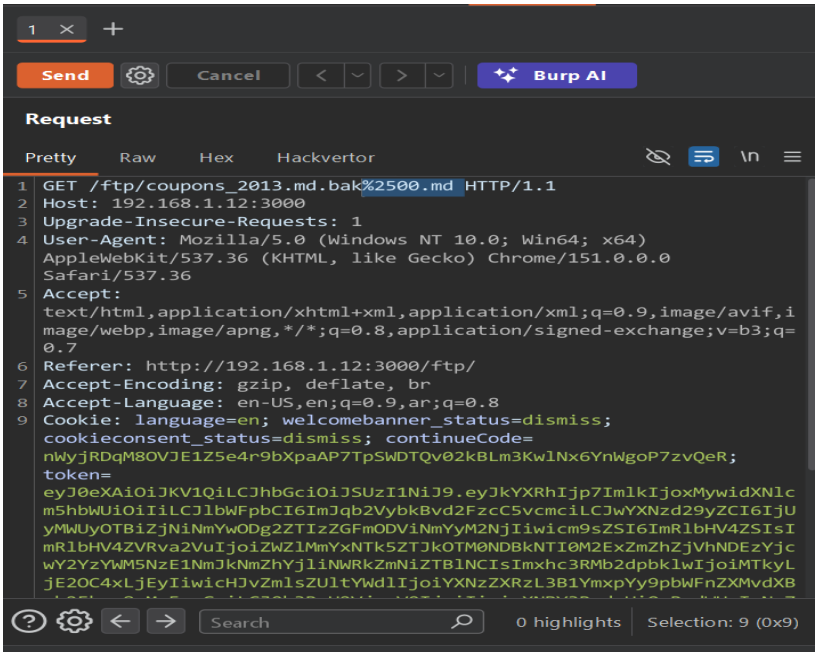
2. The /ftp directory listing itself, showing it's publicly browsable.



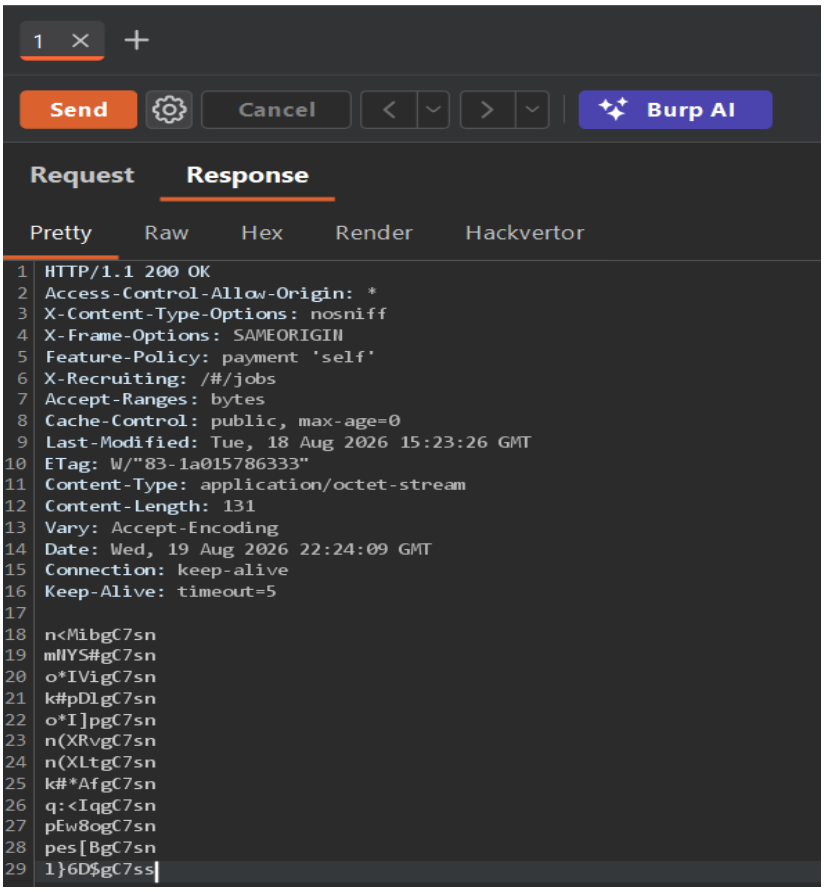
3. The rejected request and 403-style error when requesting the coupons_2013.md.bak file directly.



4. The modified request containing the %2500.md payload (%00 didn't work so we used %2500 instead because it's URL encoded).



5. The successful response returning the contents of the restricted backup file.



Security Impact

This vulnerability allows an attacker to bypass file-extension restrictions and retrieve files that were never intended to be publicly accessible, including backups, legacy data, and potentially other sensitive artifacts left on the server. In this case it exposed an old coupon backup file, which in turn enabled further exploitation (see Finding 7). In a real-world deployment, this class of flaw could expose source code backups, credentials, configuration files, or other confidential data.

How to Fix the Vulnerability

File-serving endpoints should validate the requested filename and extension using safe, canonicalized path handling rather than simple string/suffix checks, and should reject any filename containing null bytes or other control characters before it reaches the file system layer. Ideally, arbitrary user-controlled file paths should not be used to resolve files on disk at all; a safer approach is to map requested resources to an internal identifier or allow list of known, intentionally published files. Legacy, backup, and temporary files should also not be stored in a publicly reachable directory in the first place.

Vulnerability 7 Predictable Coupon Encoding

Vulnerability Name: Forged Coupon

Vulnerability Category: Cryptographic Issues (Insecure / Predictable Encoding of Sensitive Values)

CVSS Score: 7.7 (High)

Juice Shop Challenge Difficulty: 6 Stars

Description

OWASP Juice Shop generates discount coupon codes by encoding a formatted string (a month/year identifier combined with a discount percentage) rather than by issuing a random, server-tracked, single-use token. Because the encoding scheme is a reversible, publicly known algorithm rather than a cryptographically strong or unpredictable value, an attacker who recovers one legitimate coupon and identifies the pattern used to build it can construct new, arbitrary coupon codes of their own choosing — including codes for discounts far higher than any coupon the store actually issued.

This is classified as a Cryptographic Issue: the weakness is not in the discount logic itself, but in the use of a predictable/reversible encoding in place of a proper cryptographic control (such as a signed or randomly generated, server-validated token) to protect a value that has real financial impact.

Affected Functionality or Endpoint

The affected functionality is the discount coupon generation and redemption mechanism used at checkout, and the historical coupon data that was exposed via the Poison Null Byte vulnerability described in Finding 6.

Proof of Concept (PoC)

A backup file containing a list of historical, encoded coupon codes was recovered from the /ftp directory using the Poison Null Byte technique described in Finding 6.

The recovered coupon strings did not resemble plaintext, so several standard encoding schemes were tested against them using Burp Suite's Decoder tool (including Base64 and other common encodings). None produced a readable result. A web search for the encoding format used by OWASP Juice Shop's coupon system identified Z85 as the scheme in use. Decoding a recovered coupon string using Z85 confirmed this, revealing a plaintext value in the format MONTHYEAR-PERCENT (a month, year, and discount percentage).

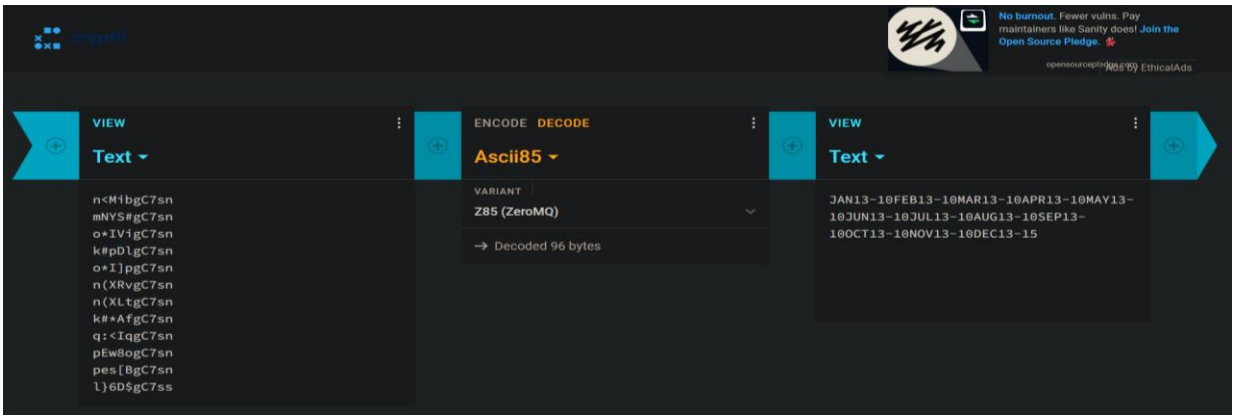
Using this pattern and the latest coupon I found on the website's BlueSky Page, a new string representing the current month/year and a 90% discount was constructed and encoded using the same scheme, producing a coupon code that was never actually issued by the store.

The forged coupon code was submitted at checkout and was accepted by the application, applying a 90% discount to the order.

Screenshots / Evidence

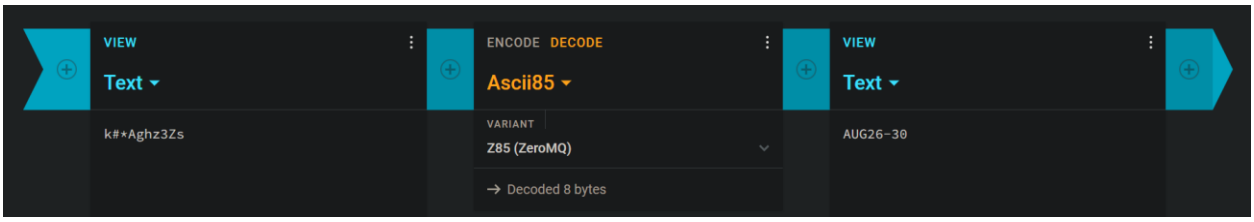
Screenshots will be provided showing:

1. The recovered historical coupon list from the backup file in the last response and its decoding (Z85) showing the underlying (M/Y-%) pattern.

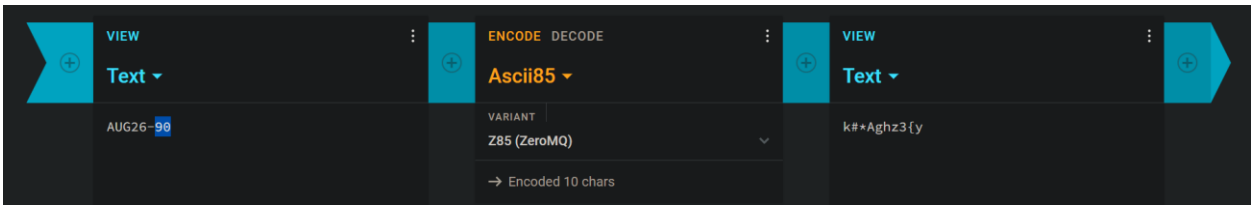


2. The last coupon found on OWASP Juice Shop BlueSky page.

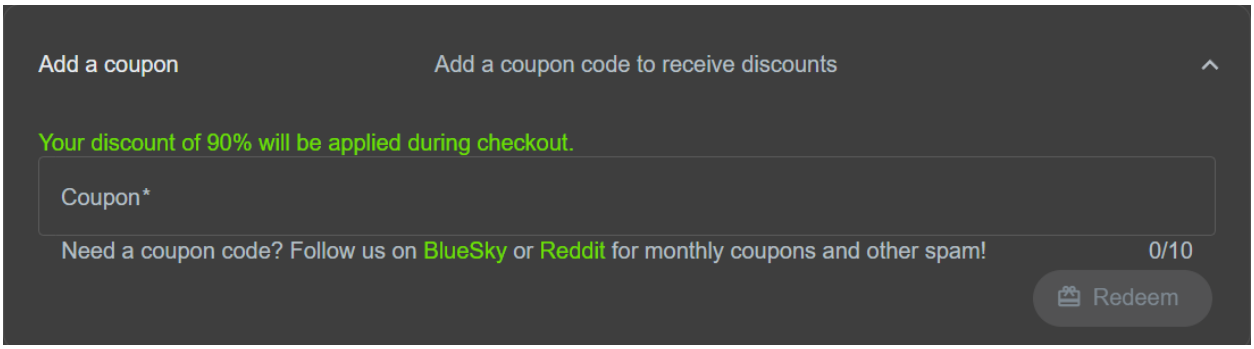
3.Decoding it and using it to forge a new coupon with a bigger discount.



4. The newly forged coupon code constructed from that pattern and encoding it to be able to use it.



5. The redeem request applying the forged coupon and the resulting discount.



Security Impact

Successful exploitation allows an attacker to generate valid, high-value discount codes at will, without ever being issued one by the store. This results in direct financial loss for the business, since discounts can be applied to any order without limit or authorization. More broadly, the vulnerability shows that discount codes are validated based on being correctly formatted rather than on being an unpredictable value verifiably issued and tracked server-side.

How to Fix the Vulnerability

Coupon codes should be generated using a cryptographically secure random value (or a securely signed token) rather than a deterministic encoding of predictable fields such as month, year, and discount percentage. Every issued coupon should be stored server-side along with its validity period and remaining redemptions, and redemption should be validated by look-up against that store rather than by decoding the submitted code. Historical or backup files containing coupon data should not be reachable from a publicly exposed directory, which also reinforces the need to remediate Finding 6.

Vulnerability 8 Changing Password

Vulnerability Name: Change Bender's Password

Vulnerability Category: Broken Authentication

CVSS Score: 8.1 (High)

Juice Shop Challenge Difficulty: 5 Stars

Description

The application contains a Broken Authentication vulnerability in its password modification endpoint **(/rest/user/change-password).** When an authenticated user attempts to change their password, the server is supposed to require and validate three parameters: current, new, and repeat. However, the backend application fails to properly validate the existence or validity of the **current** password field.

An attacker can exploit this logic flaw by intercepting the HTTP request using a web proxy (e.g., Burp Suite) and removing the current parameter entirely. The server processes the request as valid, bypassing **current-password** verification and allowing an unauthorized password update for the target user **(Bender).**

Affected Functionality or Endpoint

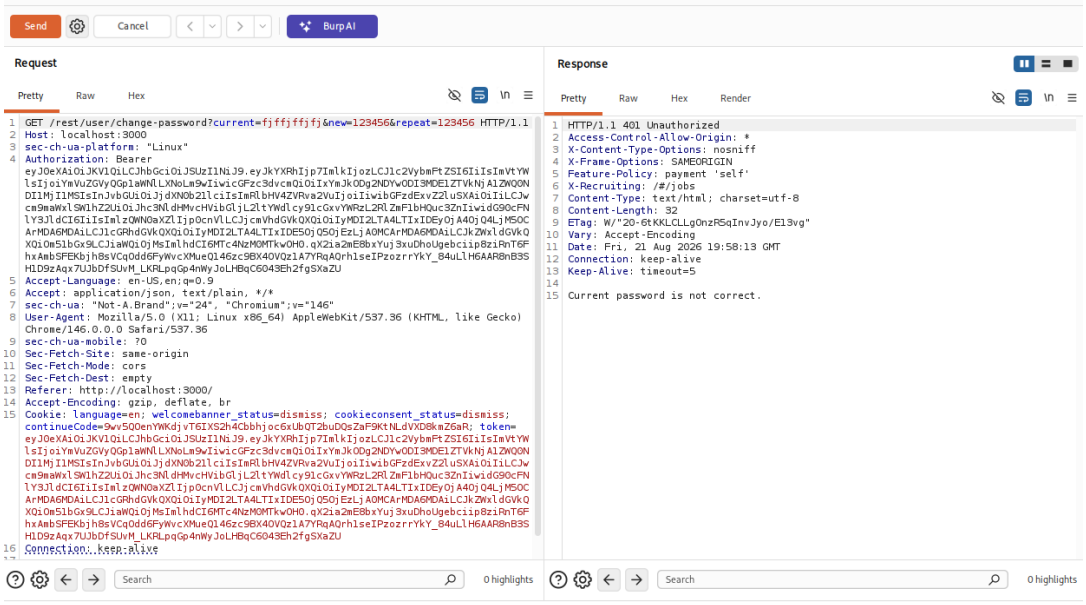
- **Endpoint:** /rest/user/change-password
- **HTTP Method:** GET (or POST/PUT via API endpoint depending on implementation version)
- **Vulnerable Parameter:** current

Proof of Concept (PoC)

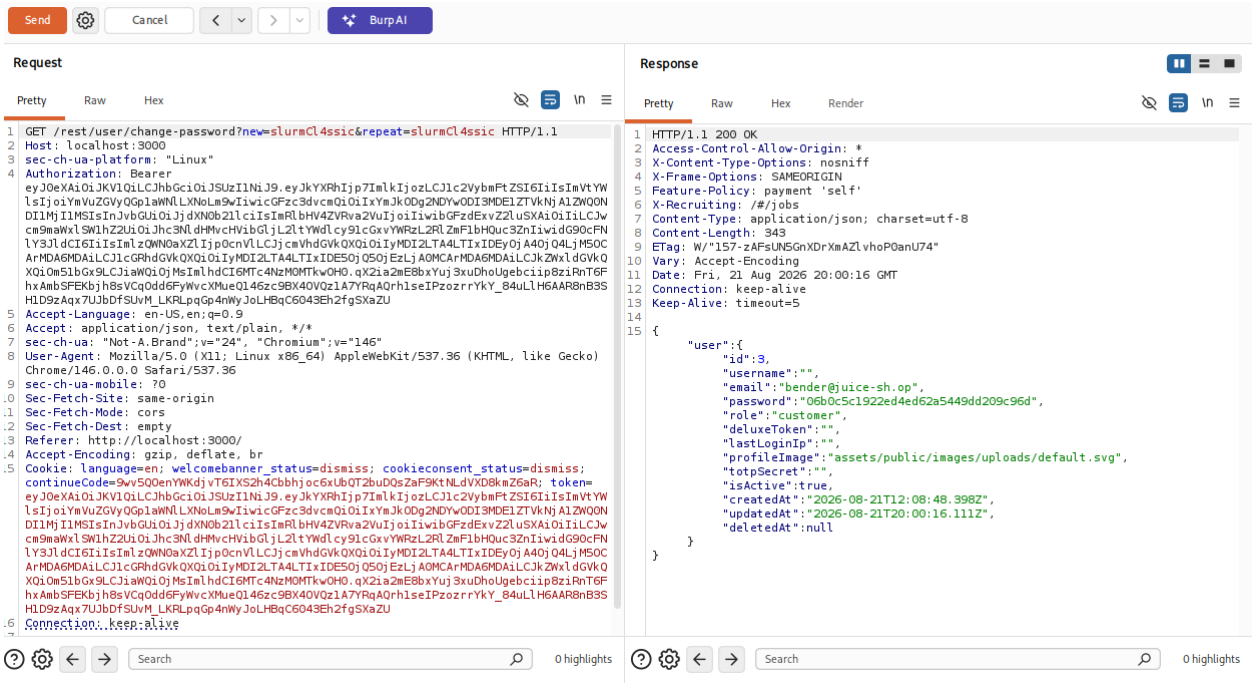
1. **Log in** to the target application as Bender (e.g., using stolen credentials, session hijack, or SQL Injection on the login form).
2. Navigate to **User Profile / Change Password** section.
3. Intercept the password change request using Burp Suite Proxy.
4. Send the captured request to **Burp Repeater**.
5. Remove the current parameter from the query string entirely.
6. Execute the modified HTTP request. The server returns a 200 OK status with a successful response payload.
7. Log out and log back in using bender@juice-sh.op and the new password slurmCl4ssic to confirm account takeover.

Screenshots / Evidence

1. Burp Suite HTTP history log displaying the intercepted password change request.



2. Modified HTTP request with the removed current parameter and the corresponding HTTP 200 OK response payload.



Security Impact

- **Account Takeover (ATO):** Allows attackers who gain temporary session access (e.g., via Cross-Site Scripting, Session Hijacking, or local access) to permanently lock out the legitimate user by changing their password without knowing the original secret.
- **Authentication Bypass:** Circumvents a core security defense designed to ensure password modifications are authorized by the actual account owner.

How to Fix the Vulnerability

- **Strict Server-Side Validation:** Ensure the backend explicitly verifies that the current password field is present, non-empty, and matches the stored hash in the database before processing any password change request.
- **Use Secure Methods for Sensitive Actions:** Use POST or PUT HTTP methods with standard request body parameters rather than relying on URL query parameters (GET requests) for credential updates.
- **Session Invalidation:** Revoke all existing active JWT tokens or sessions upon a successful password change, requiring re-authentication.

Vulnerability 9 Unauthorized Username Modification

Vulnerability Name: CSRF – Unauthorized Username Modification

Vulnerability Category: Cross-Site Request Forgery (CSRF)

CVSS Score: 8.1 (High)

Juice Shop Challenge Difficulty: 3 Stars

Description

Cross-Site Request Forgery (CSRF) is a vulnerability that allows an attacker to cause a victim's browser to send an unwanted request to a web application where the victim is already authenticated.

During testing of the OWASP Juice Shop application, the /profile endpoint was found to accept a POST request that changes the authenticated user's username.

The request does not contain a CSRF token, and the application relies on the user's authentication cookie. This can allow a malicious webpage to submit a forged request on behalf of an authenticated user.

The primary root cause is the absence of a dedicated CSRF protection mechanism, such as a unique and unpredictable CSRF token associated with state-changing requests. The /profile endpoint accepts the state-changing request without requiring a CSRF token.

Affected Functionality or Endpoint

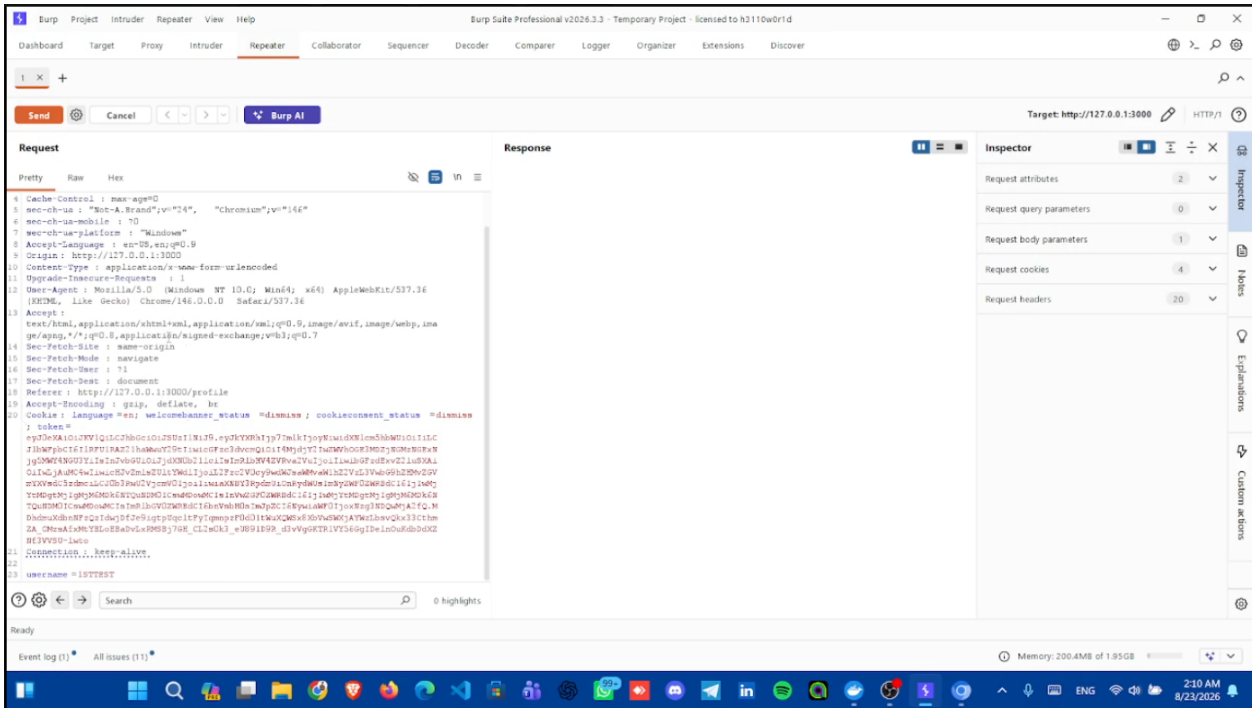
- Endpoint: /profile
- HTTP Method: POST
- Vulnerable Parameter: username (no CSRF token required)

Proof of Concept (PoC)

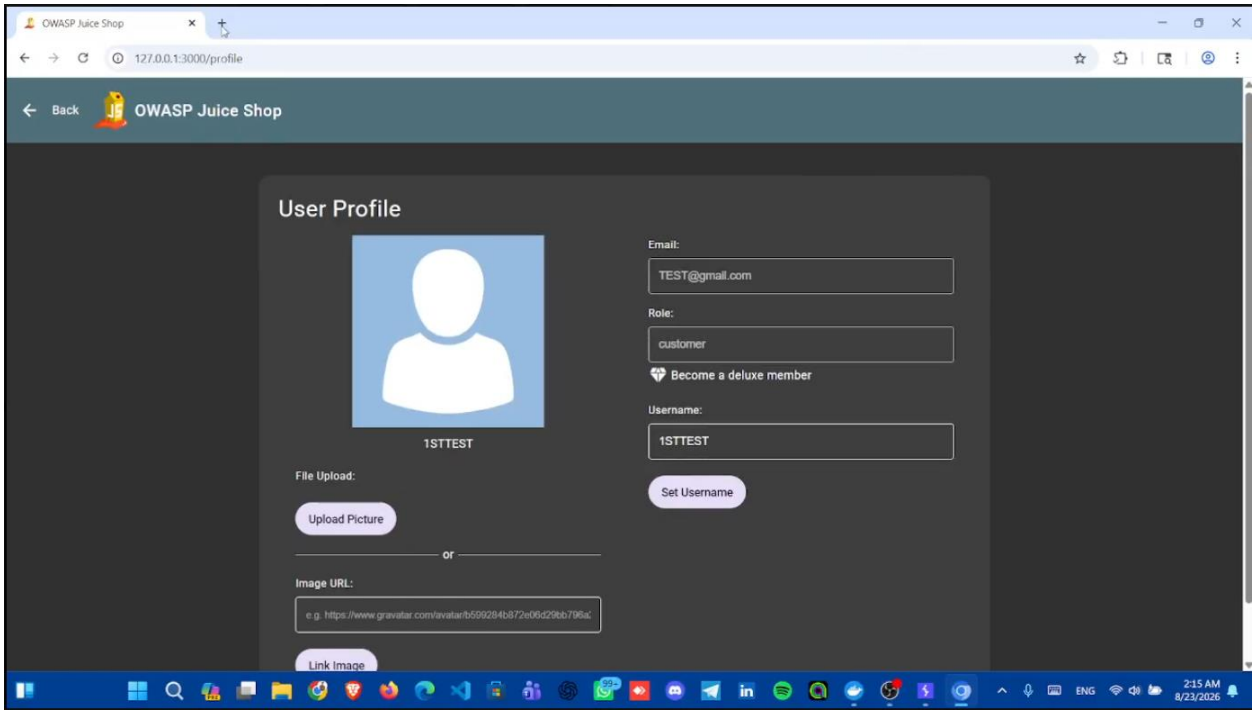
1. **Start the OWASP Juice Shop application using Docker.**
2. **Open https://juice-shop.herokuapp.com/**
3. **Log in to a valid user account.**
4. **Open Burp Suite → Proxy → HTTP History.**
5. **Modify the username through the application's profile functionality.**
6. **Capture the following request: POST /profile**
7. **Confirm that the request contains: username=<value>**
8. **Observe that no CSRF token is included in the request.**
9. **Create an HTML page containing the CSRF form shown above.**
10. **Open the malicious HTML page while the victim is authenticated.**
11. **Submit the form.**
12. **Verify whether the username is changed to CSRFTEST.**

Screenshots / Evidence

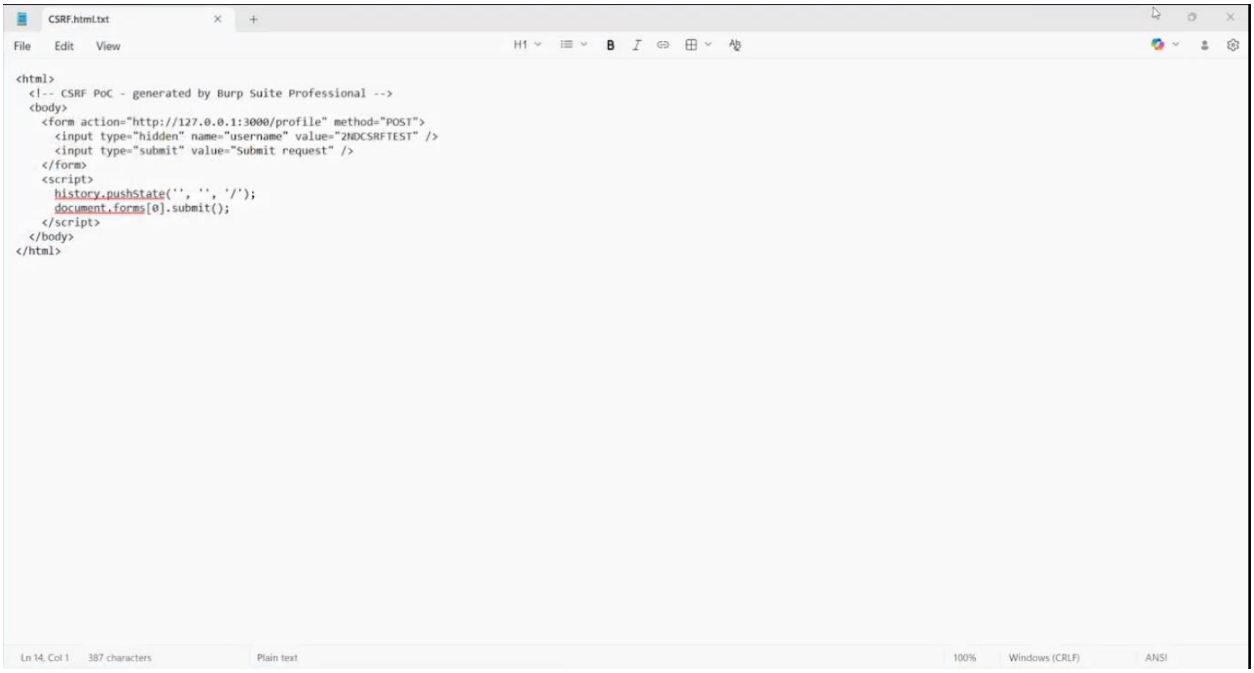
1. request displaying the intercepted username-change request, confirming no CSRF token is present.



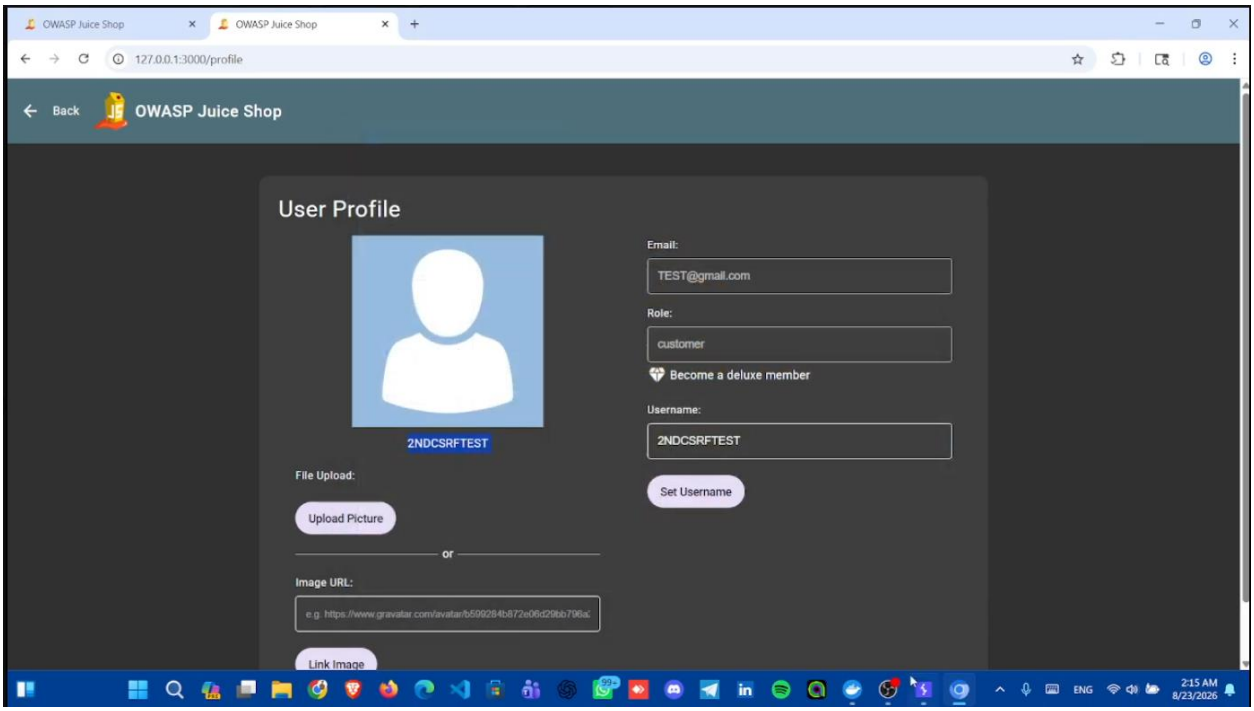
2. confirming the server accepted and processed the request.



3. The malicious CSRF HTML page opened in the browser, containing the hidden form pre-filled with the forged username value.



4. The victim's profile page after the attack, showing the username successfully changed to "2NDCSRFTEST" without any intentional action by the victim.



Security Impact

If successfully exploited, an attacker could potentially cause an authenticated user to perform unauthorized profile modifications without intentionally requesting the action.

Depending on the application's other state-changing endpoints, the same weakness could potentially affect additional account or profile operations.

The vulnerability requires the victim to be authenticated and to visit or interact with an attacker-controlled page. However, successful exploitation can result in an unauthorized modification of the victim's account information.

How to Fix the Vulnerability

The application should implement a dedicated CSRF protection mechanism, such as a unique, unpredictable, per-session (or per-request) CSRF token that is validated on every state-changing request, including the /profile endpoint.

Session cookies should also be configured with the SameSite attribute (Strict or Lax) to reduce the browser's willingness to send them on cross-site requests. Sensitive actions should additionally verify the Origin/Referer header as a defense-in-depth measure.

Vulnerability 10 Unrestricted File Upload

Vulnerability Name: Upload Type – File Extension Restriction Bypass

Vulnerability Category: File Upload Vulnerability

CVSS Score: 4.3 (Medium)

Juice Shop Challenge Difficulty: 3 Stars

Description

file upload mechanism of the "Complaint" feature allows users to easily bypass file type restrictions. While the client-side interface correctly blocks unauthorized extensions (permitting only .pdf and .zip), the backend API fails into it.

An attacker can bypass this defense mechanism by initially uploading a benign file to pass the frontend checks, intercepting the resulting POST request using an interception proxy, and tampering with the filename parameter. This enables the successful upload of all file types, such as executable PHP scripts, which the server accepts and stores without validation.

Affected Functionality or Endpoint

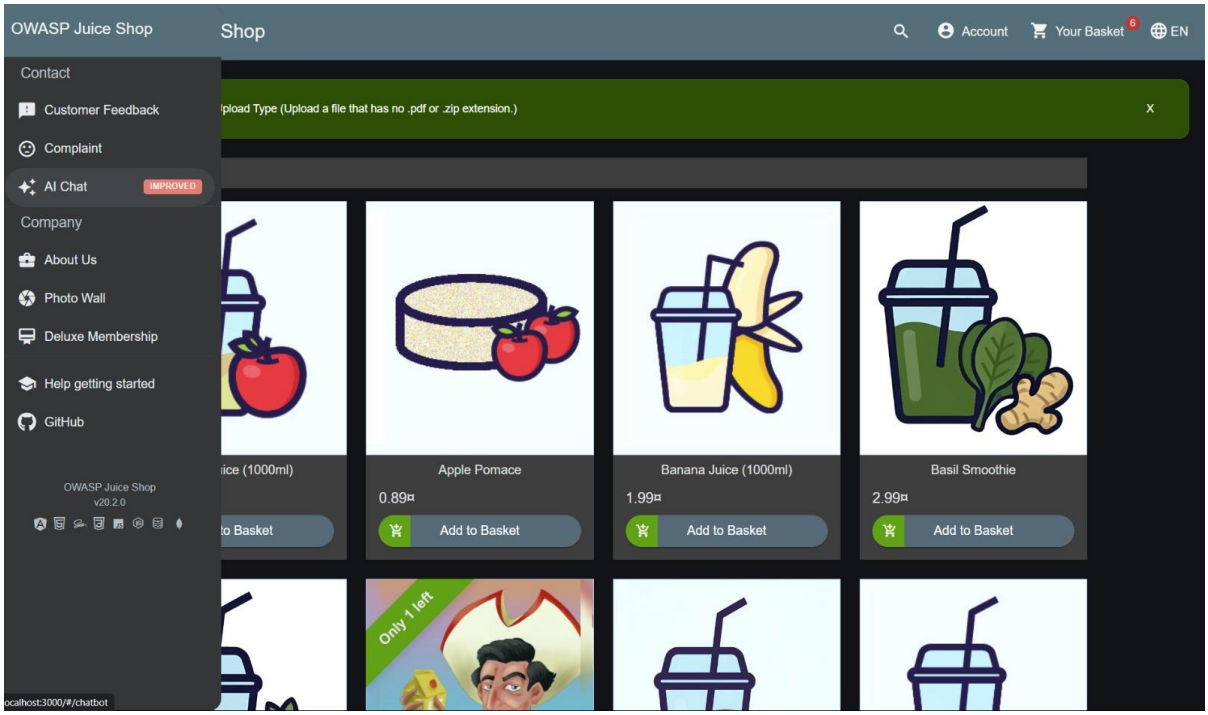
- Endpoint: /api/Complaints
- HTTP Method: POST
- Vulnerable Parameter: The filename and Content-Type attributes

Proof of Concept (PoC)

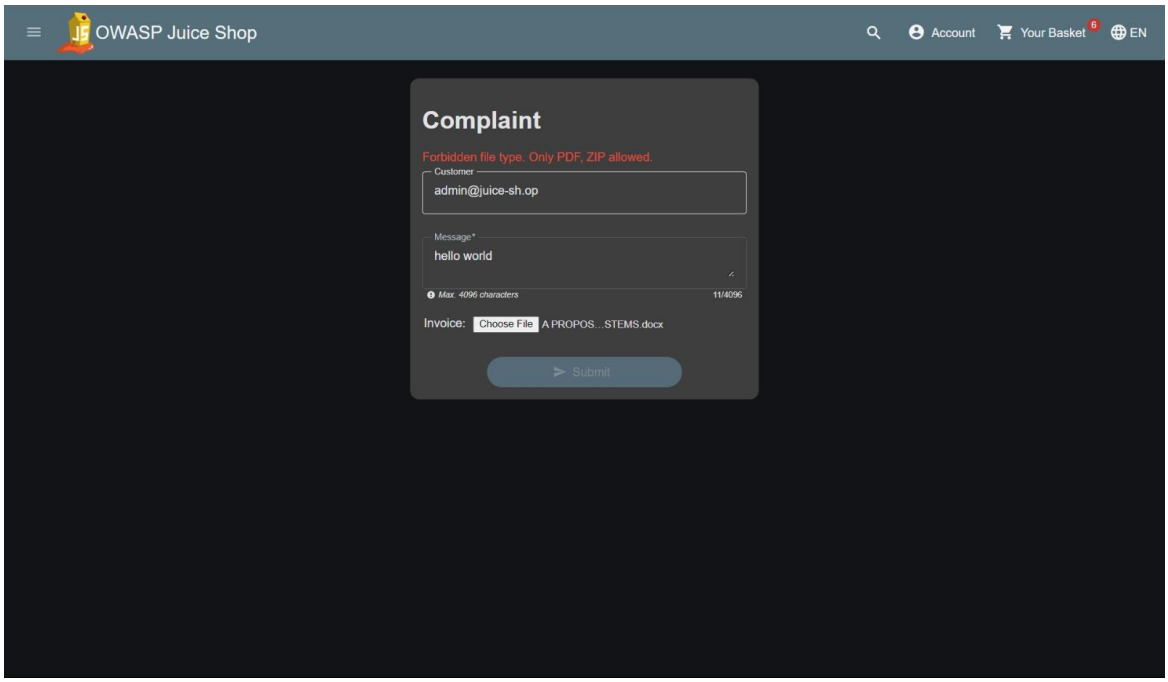
1. **Log in** to the https://juice-shop.herokuapp.com/
2. Navigate To **Complaint** section.
3. Do required input fields (Customer and Message).
4. Select a legitimate file, such as .pdf, to satisfy the frontend validation requirements.
5. Prepare a web proxy (such as Burp Suite) to capture the outbound HTTP POST request before it reaches the server.
6. Modify the filename value, altering it from ".pdf" to any file type you want such as ".php"
7. Adjust the Content-Type directive to the appropriate setting for the file you uploaded (application/x-httpd-php in my example).

Screenshots / Evidence

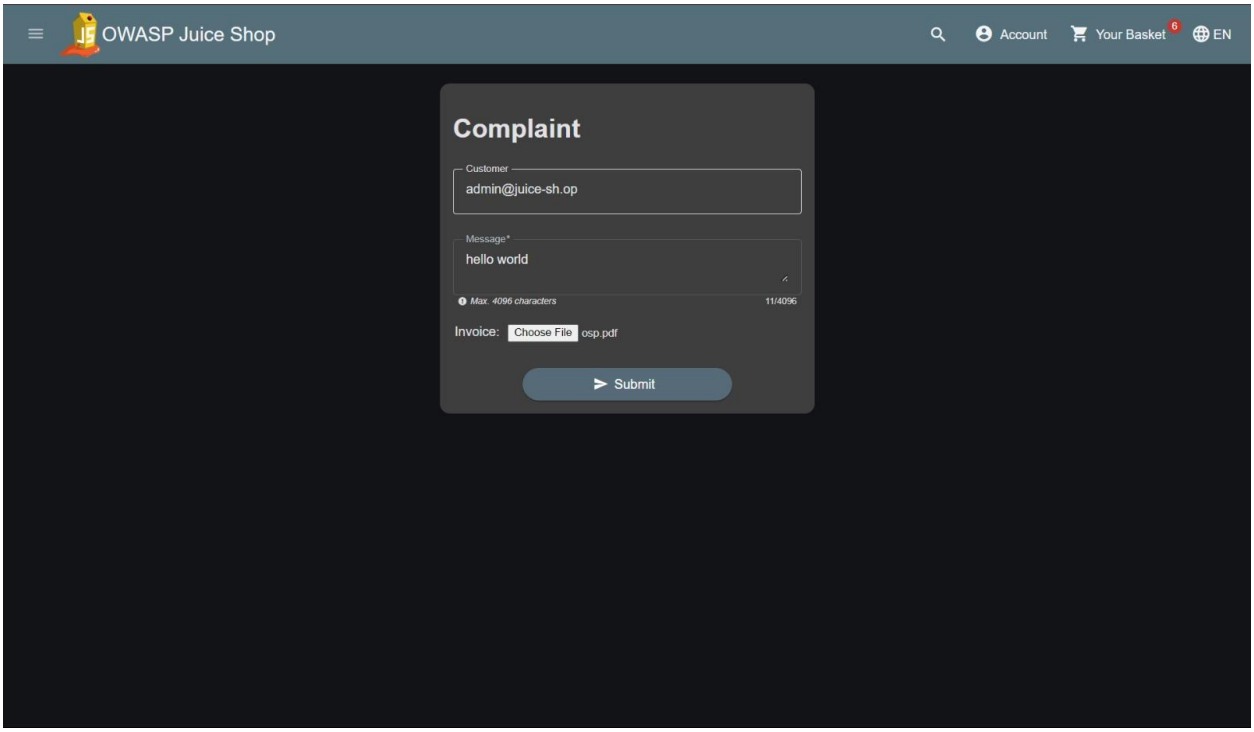
1. navigate to complaint section



2. Attempt to attach a restricted file format (e.g., .docx) to trigger client-side error message.



3. Now Upload legal file type like pdf



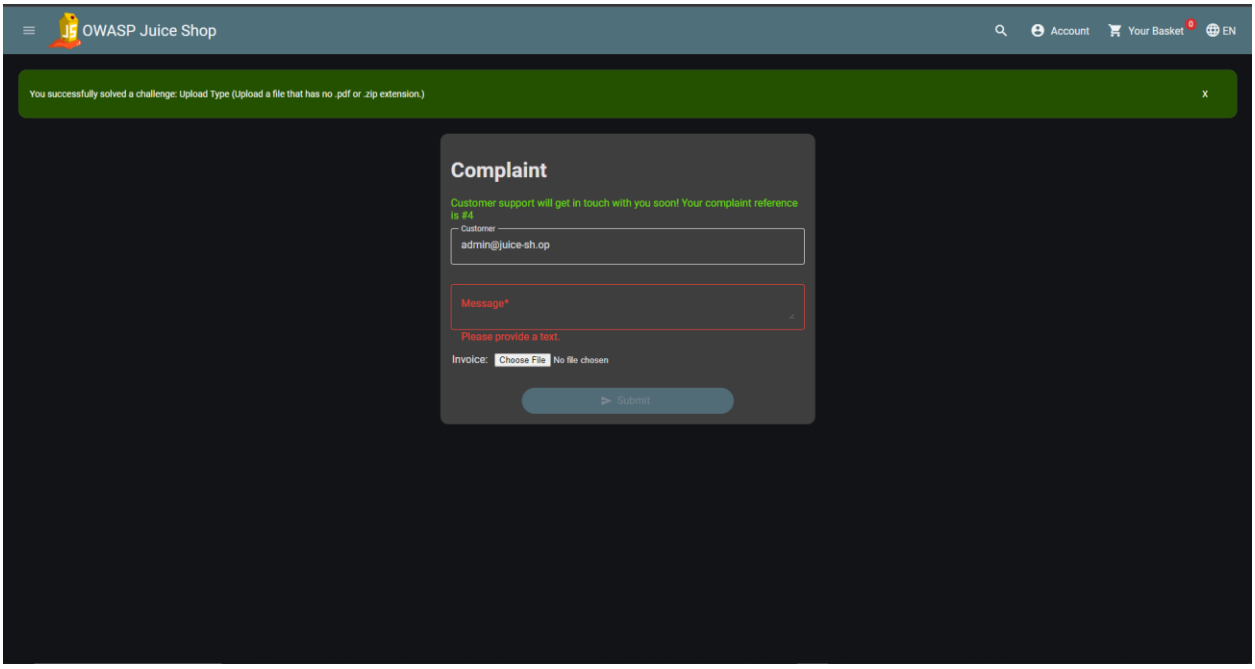
4. After Getting POST request in Burp Suite we can see filename and content type fields

```
20
21 ------WebKitFormBoundaryTmIWfxKwDl3TIz9d
22 Content-Disposition : form-data ; name="file"; filename="osp.pdf"
23 Content-Type : application/pdf
24
```

5. change them as you want to upload the file you want to upload it and send the request

```
21 ------WebKitFormBoundaryTmIWfxKwDl3TIz9d
22 Content-Disposition : form-data ; name="file"; filename="osp.php"
23 Content-Type : application/x-httpd-php
24
```

6. And The website accepted it



Security Impact

- Remote Code Execution (RCE): If the destination directory allows script execution, an attacker could upload a web shell. This leads to arbitrary command execution and potentially full compromise of the underlying web server.

- Malware Distribution & XSS: The upload directory can be weaponized to host malicious payloads or HTML documents containing Stored Cross-Site Scripting (XSS), compromising administrators or other users who review the complaints.

How to Fix the Vulnerability

- **Implement Backend Validation:** Discard frontend checks. The server must explicitly validate the incoming file against a strict, hardcoded whitelist of permitted extensions (only .pdf and .zip).
- **Secure Storage and Renaming:** Store all user-uploaded files outside of the public web root to prevent direct execution. Additionally, non-executable extension upon saving.

Vulnerability 11 Upload Size

Vulnerability Name: Upload Size – File Size Restriction Bypass

Vulnerability Category: Unrestricted File Upload

CVSS Score: 4.3 (Medium)

Juice Shop Challenge Difficulty: 3 Stars

Description

The complaint form only checks the invoice file size on the client side, with a limit of 100 kB. An attacker can choose a small PDF that passes this check, then use Burp Suite to intercept the upload request and replace the file with a much larger PDF.

Since the server does not check the file size again, it accepts and stores the oversized file.

Affected Functionality or Endpoint

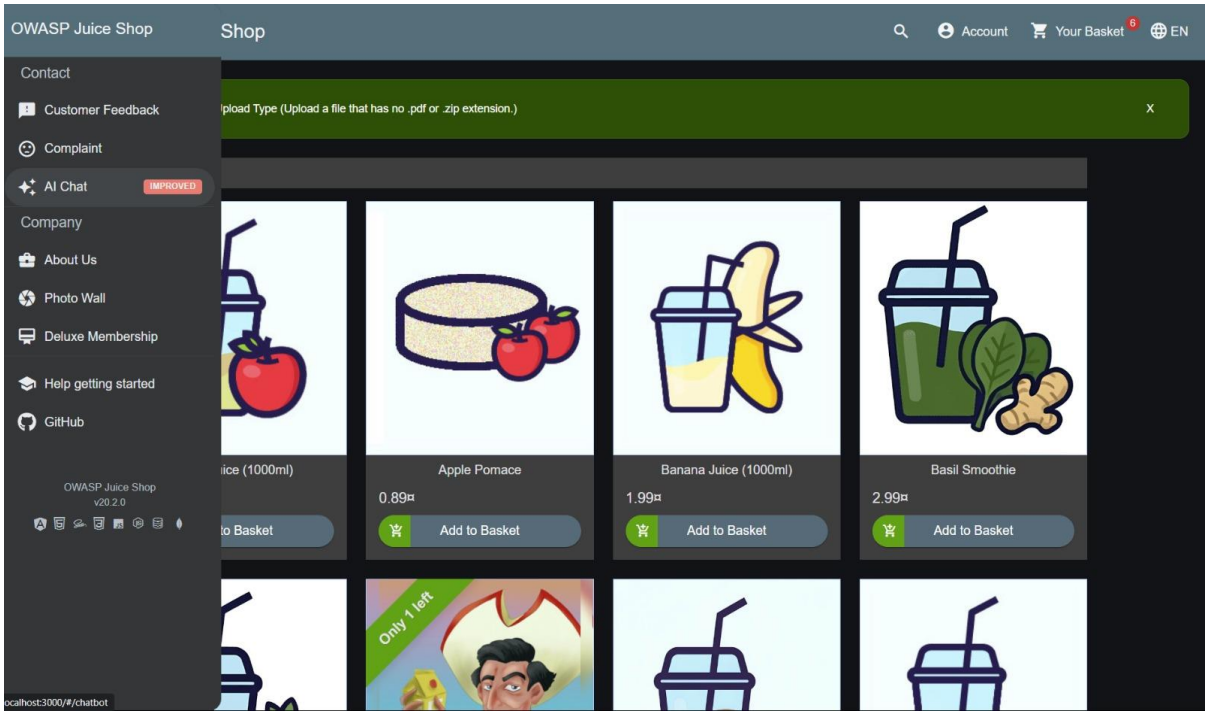
- **Endpoint:** POST /file-upload
- **Page:** /#/complain
- **Vulnerable Control:** Client-side-only file size validation (100 kB limit)

Proof of Concept (PoC)

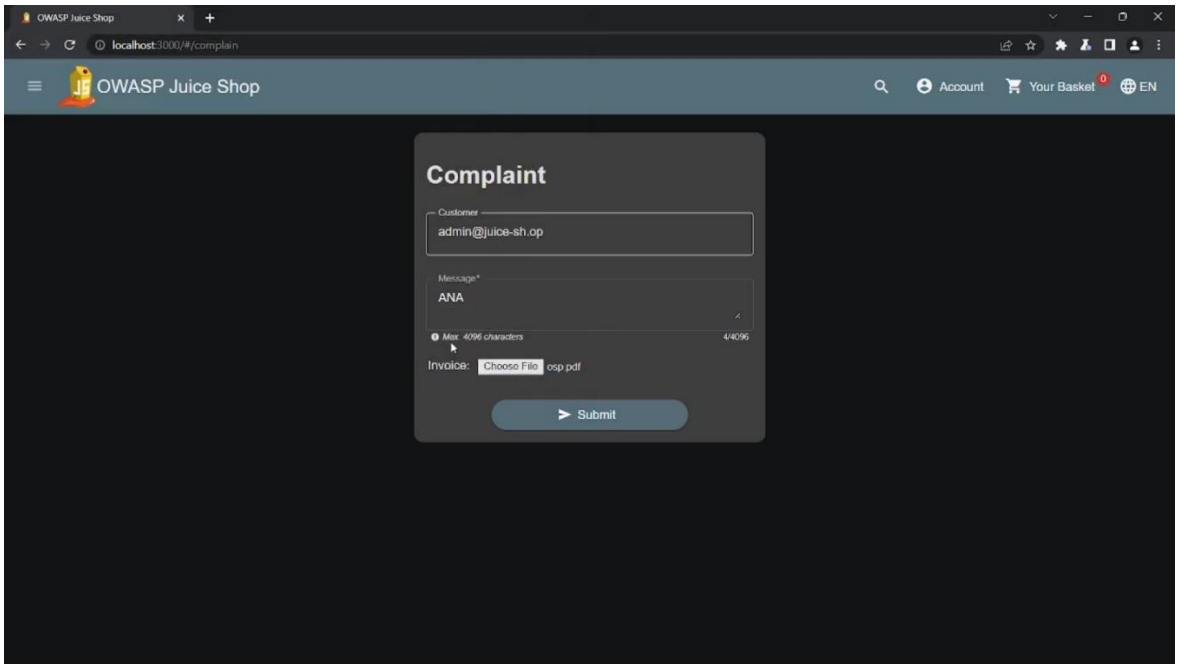
1. Navigate to the Complaint page and select a small PDF file (e.g. osp.pdf) that satisfies the 100 KB client-side limit.
2. Intercept the outbound request in Burp Suite before it reaches the server.
3. Replace the file's content in the request body with a larger PDF payload that exceeds 100 kB
4. The server accepts the oversized file.

Screenshots / Evidence

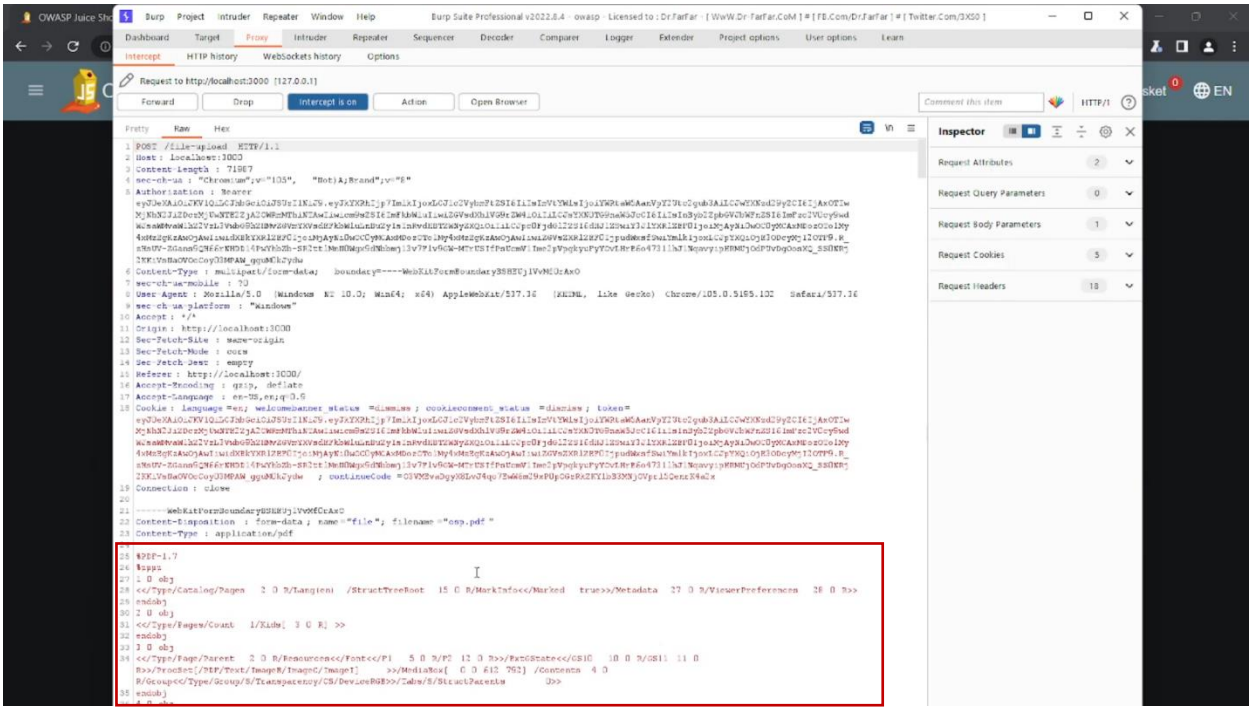
1. navigate to complaint section



2. Complaint form with a small PDF (osp.pdf) selected, passing the client-side size check.



3. We got content of osp.pdf that we uploaded it

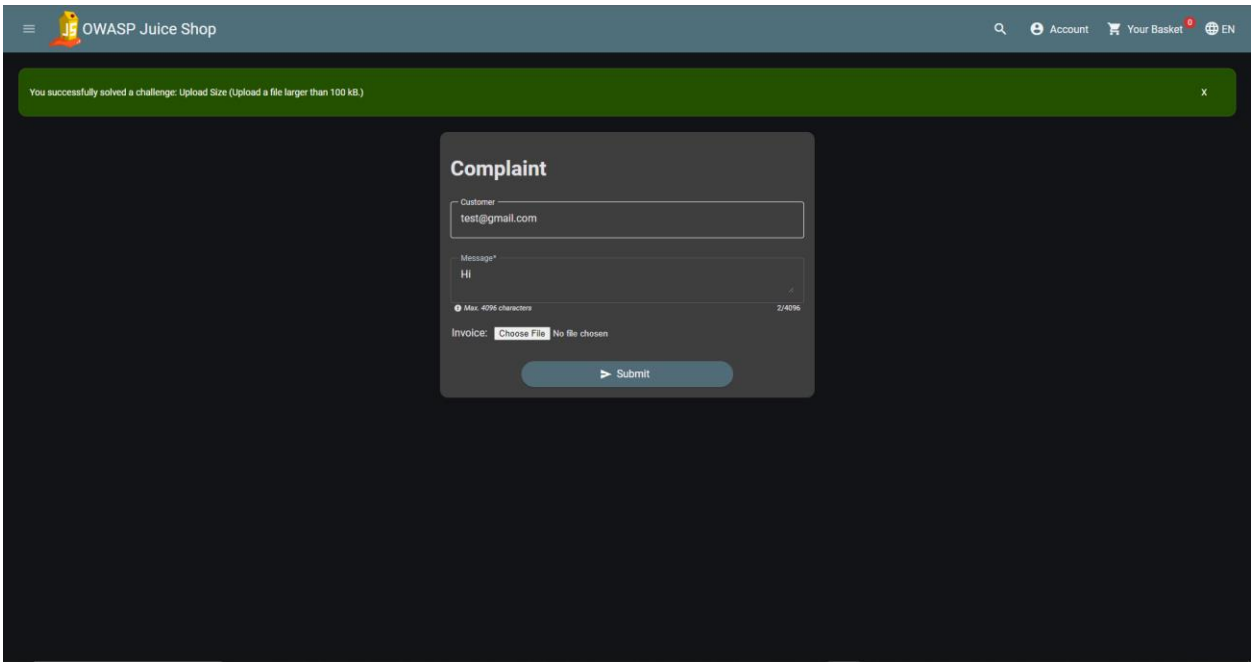


4. After changing content of PDF with another larger pdf body

```
%PDF-1.7
%µµµµ
1 0 obj
<</Type/Catalog/Pages 2 0 R/Lang(en) /StructTreeRoot 15 0 R/MarkInfo<</Marked true>>/Metadata 27 0
R/ViewerPreferences 28 0 R>>
endobj
2 0 obj
<</Type/Pages/Count 1/Kids[ 3 0 R] >>
endobj
3 0 obj
<</Type/Page/Parent 2 0 R/Resources<</Font<</F1 5 0 R/F2 12 0 R>>/ExtGState<</GS10 10 0 R/GS11 11 0
R>>/ProcSet[/PDF/Text/ImageB/ImageC/ImageI] >>/MediaBox[ 0 0 612 792] /Contents 4 0
R/Group<</Type/Group/S/Transparency/CS/DeviceRGB>>/Tabs/S/StructParents 0>>
endobj
4 0 obj
<</Filter/FlateDecode/Length 201>>
stream
x□□□M□@□ıú⊳cé Ì ª Ù□,¤fy□ÑÁ□yJú¤ý□Õ€□ùxg□□^y ª ;Á±₩¤□á-é®Å éF□Ò1I□â, ¹t7Í
□±□□V¸4R□ê¤H*)¼□aÕA
¶KF¤\R"□¸ª£Ÿ□□Lh¯ò Ú^òSÎ¥ØÅó□O Ï w□®1ù□~éGi□dBÛr¼□"□□=ª¥3Ë³□â_põÎ¾võ'ø ·□Àùz□Y□á□:AJ
endstream
endobj
5 0 obj
<</Type/Font/Subtype/Type0/BaseFont/BCDEEE+Aptos/Encoding/Identity-H/DescendantFonts 6 0 R/ToUnicode 23 0 R>>
endobj
6 0 obj
[ 7 0 R]
endobj
7 0 obj
<</BaseFont/BCDEEE+Aptos/Subtype/CIDFontType2/Type/Font/CIDToGIDMap/Identity/DW 1000/CIDSystemInfo 8 0
R/FontDescriptor 9 0 R/W 25 0 R>>
endobj
8 0 obj
<</Ordering(Identity) /Registry(Adobe) /Supplement 0>>
endobj
```

Search... 0 matches

5. We uploaded PDF file larger than 100KB Successfully



Security Impact

Since the file size is only checked on the client side, an attacker can bypass the limit and upload very large files. If this is abused repeatedly, it could use up server storage or slow down the application. It may also allow attackers to upload larger files than the application is supposed to accept.

How to Fix the Vulnerability

The server should check the file size itself before saving the file. The 100 kB limit should be enforced on the actual uploaded data, not just the file selected by the user. A maximum request size can also be configured on the web server or reverse proxy as an extra layer of protection.

Vulnerability 12 Forged Feedback

Vulnerability Name: Forged Feedback

Vulnerability Category: IDOR

CVSS Score: 7.7 (High)

Juice Shop Challenge Difficulty: 3 Stars

Description

Customer Feedback form submits a POST request to /api/Feedbacks/ containing a JSON body that includes a UserId field, in addition to the comment, rating, and CAPTCHA result. This UserId value is taken directly from client-supplied input.

By intercepting the request with Burp Suite and changing the UserId value from the attacker's own ID (e.g. 1) to an arbitrary other user's ID (e.g. 22), it is possible to submit feedback that is stored and displayed as if it were posted by that other user, without their knowledge or consent.

Affected Functionality or Endpoint

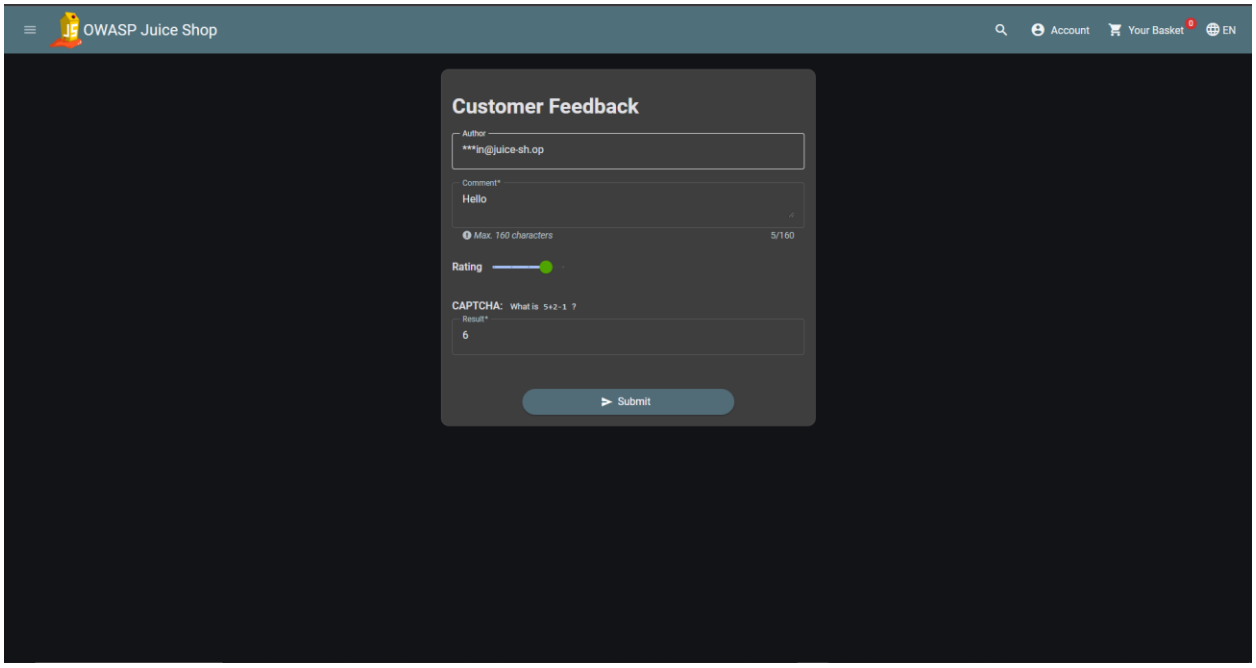
- Endpoint: POST /api/Feedbacks/
- Page: /#/contact
- Vulnerable Parameter: UserId

Proof of Concept (PoC)

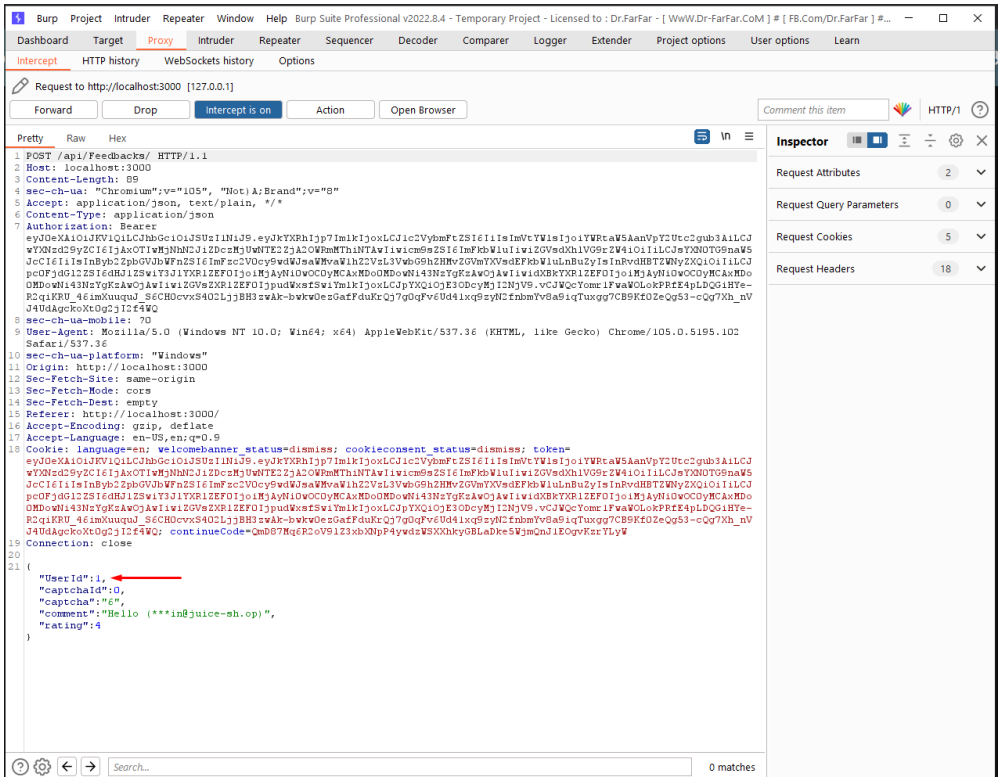
1. Fill in a comment, rating, and CAPTCHA, then submit the form while intercepting the request in Burp Suite.
2. Change the UserId value in the intercepted request to a different, arbitrary user ID .
3. Forward the modified request to the server.
4. The server accepts the request and stores the feedback under the forged UserId.

Screenshots / Evidence

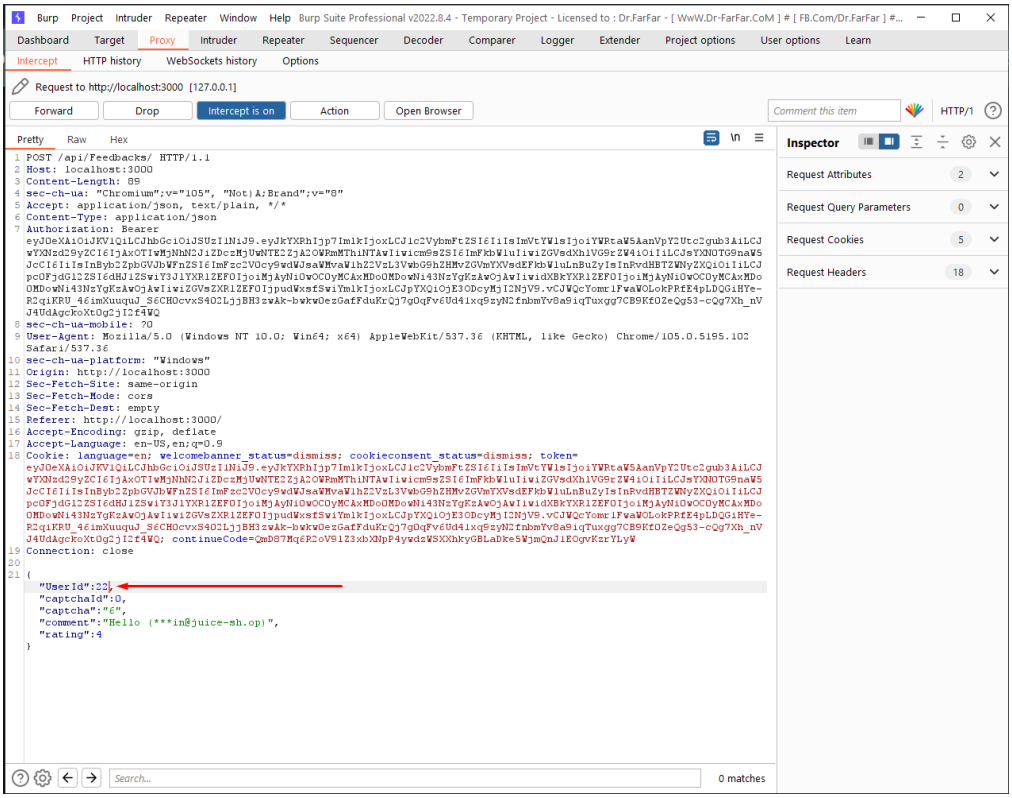
1. Writing Feedback with a specific user



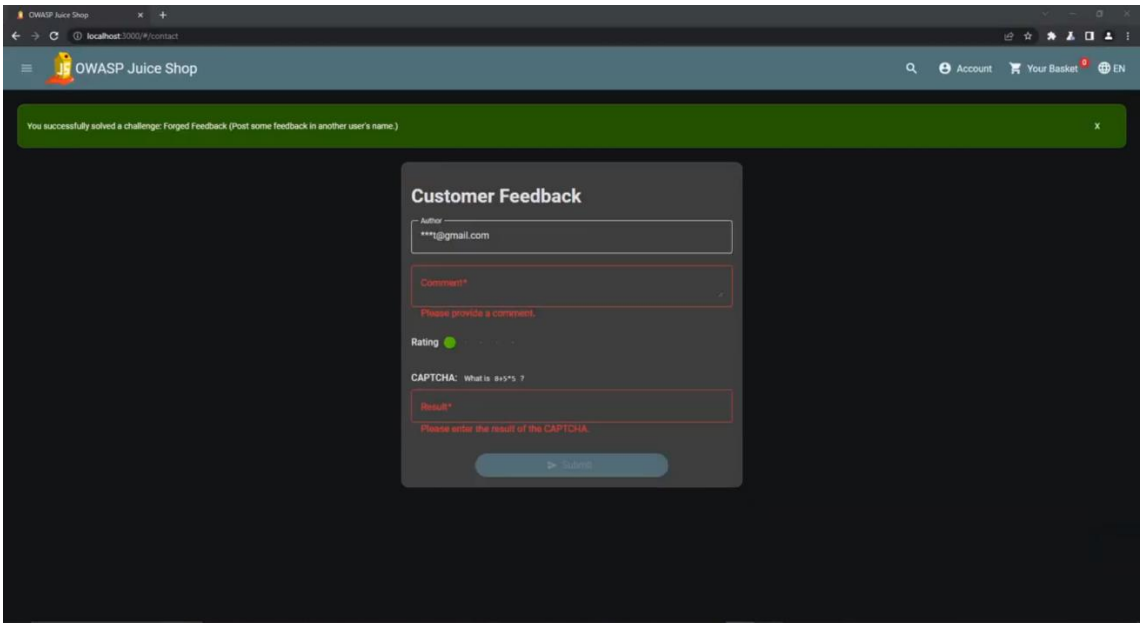
2. Here appear that User 1 is who sending the Feedback



3. Changing The UserId to another Valid User



4. Feedback has recorded by User 22 and the Vulnerability succussed



Security Impact

An attacker can submit feedback or comments using another user's ID, making it look like the comment came from that person.

This could be used to post offensive, false, or misleading content and damage someone's reputation.

It also reduces the trust and reliability of the feedback system. The same issue may exist in other parts of the application if they also trust user IDs sent by the client.

How to Fix the Vulnerability

The server should not trust the UserId sent by the user. It should get the correct UserId from the logged-in session.

The UserId in the request should be ignored or rejected. The server should also make sure users cannot change sensitive fields such as UserId or roles.