Owasp Juice shop Pentesting Report

-Shamiur Rahman Sefat



1.Zero star (improper input validation)

Description: Capture the feedback request in burp then change the rating to 0.

Risk: \*

Proof of concept:



Solution:

**Input Validation**: Ensure that the application validates ratings before accepting them. For a star rating system, you typically want to restrict the ratings to a predefined range (e.g., 1 to 5 stars).

**Error Handling**:

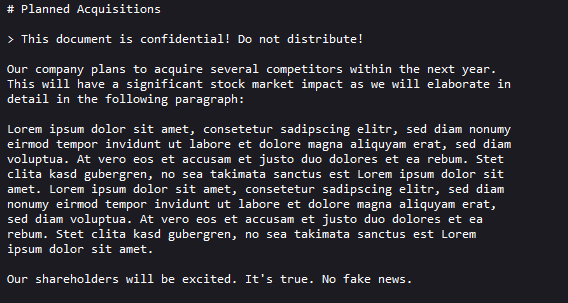
* Implement proper error handling to manage cases where invalid data might be submitted. This helps in maintaining application stability and providing meaningful feedback to users.

2.Confidential document (Sensitive Data Exposure)

Description: herokuapp.com/ftp/acquisitions.md

Risk: \*

Proof of concept:



Solution:

**Access Control**:

* **Ensure Proper Authorization**: Verify that users have the appropriate permissions before allowing access to confidential documents. Implement role-based access control (RBAC) to enforce this.

**Encrypt Sensitive Data**: Ensure that confidential documents are encrypted both at rest and in transit. This helps protect the data from unauthorized access.

**Audit Logging**:

* **Monitor Access**: Implement logging to track who accesses confidential documents and when. This can help in auditing and detecting unauthorized access attempts.

3.Dom XSS(XSS)

Description: giving input <iframe src="javascript:alert('xss')"> in the search bar

Risk: \*

Proof of concept:



Solution:

**Example of vulnerable code:**

document.getElementById('output').innerHTML = userInput;

**Example of fixed code:**

const sanitizedInput = DOMPurify.sanitize(userInput);

document.getElementById('output').innerHTML = sanitizedInput;

**->Avoid Dangerous JavaScript Methods**

Be cautious with methods like eval(), setTimeout(), setInterval(), document.write(), and location.href if they involve user input. These methods can easily be exploited in a DOM XSS attack.

4.Error Handling (Security Misconfiguration)

Description: herokuapp.com/rest/shakir

Risk: \*

Proof of concept:



Solution:

**Graceful Degradation:** Handle errors gracefully by displaying user-friendly messages instead of raw error details.

**Avoid Overexposure**: Don’t expose detailed errors to users; log them server-side and show generic messages.

**Centralized Handling**: Use a centralized mechanism (like middleware) to uniformly manage errors across the application.

**Input Validation**: Rigorously validate inputs to prevent errors caused by invalid or malicious data.

**Fallback Mechanisms**: Implement fallback options for critical operations if something fails.

**Asynchronous Handling**: Properly handle errors in asynchronous code using try/catch with async/await or .catch() with promises.

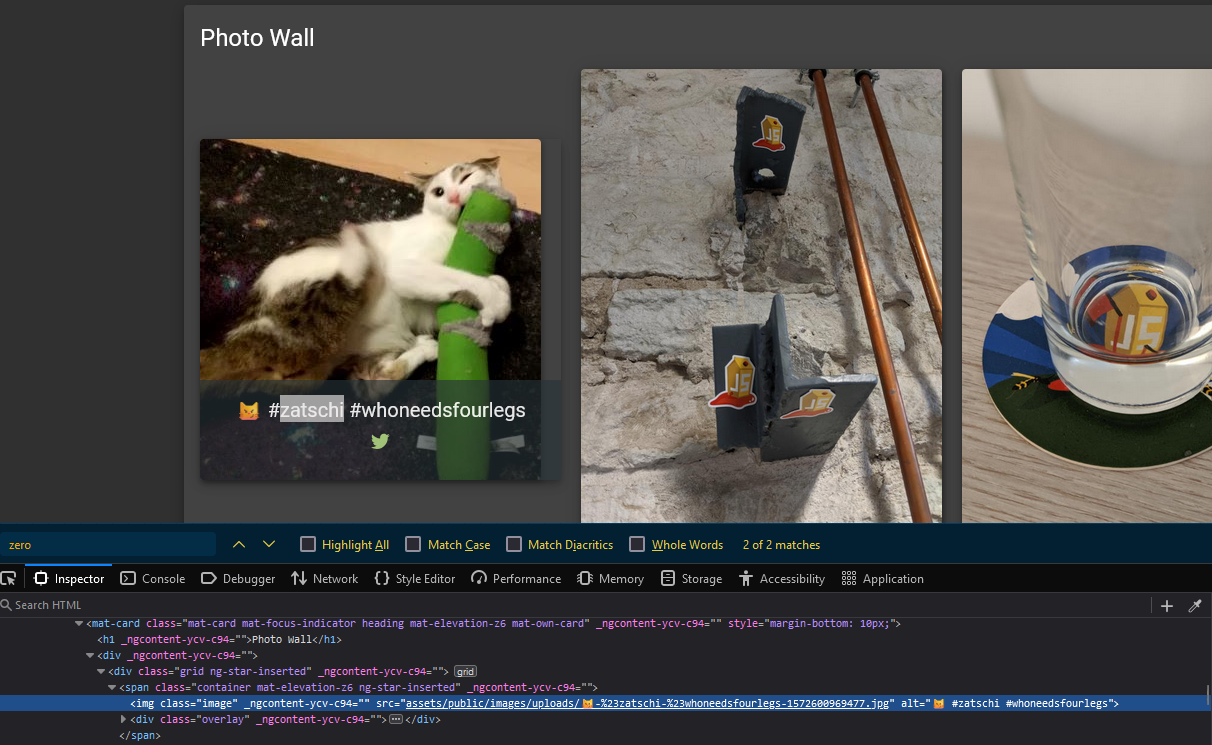
**Logging**: Log errors with context, categorizing them by severity for easier diagnosis.

5.Missing encoding(improper input validation)

Description: Go to the photo wall and then inspect that page. you’ll find the cat we see below is not visible bcz of missing encoding of special character # in a html tag.we need to replace that # tag with %23.

Risk: \*

Proof of concept:



Solution:

Ensure that all user-generated content is properly encoded before being inserted into HTML, URLs, or script contexts to prevent Cross-Site Scripting (XSS) attacks. Specifically, use the appropriate encoding functions (e.g., HTML, JavaScript, URL encoding) based on where the data will be used.

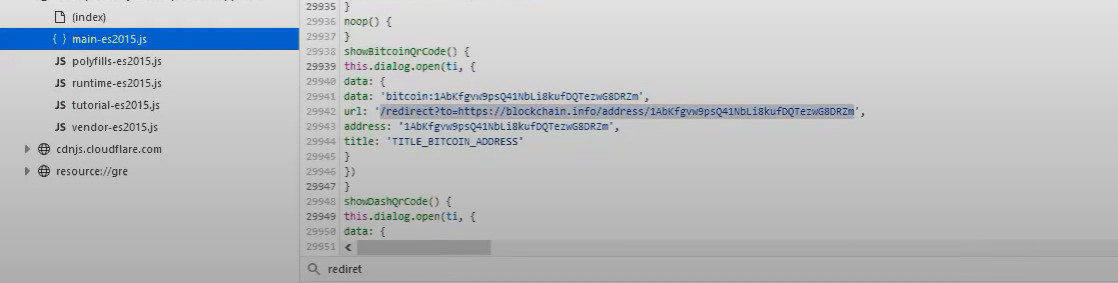
6.Outdated whitelist (Unvalidated Redirects)

Description: Debugger->main java script source code->

find the word ‘redirect’ and you’ll get some outdated crypto path->attach them with herokuapp.com/

Risk: \*

Proof of concept:



Solution:

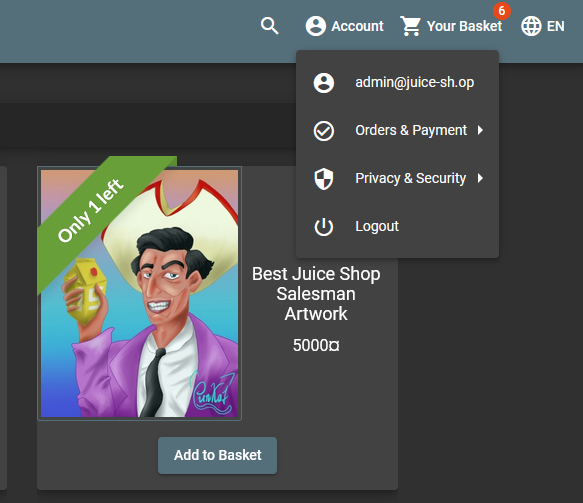
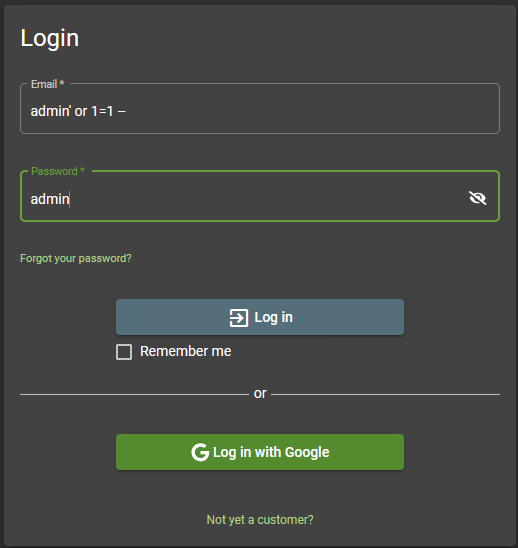
Replace the outdated whitelist approach with a more flexible allowlist, ensuring it includes all current and necessary entries. Regularly update the list to reflect new trusted sources or entities, and consider transitioning to a more secure validation method, such as input validation or content security policies, depending on the context.

7.Admin Login (SQL Injection)

Description: We can login as admin by inject some basic malicious sql code. We got the admin account cz might be admin account was the first acc in the database.

Risk: \*\*

Proof of concept:



Solution:

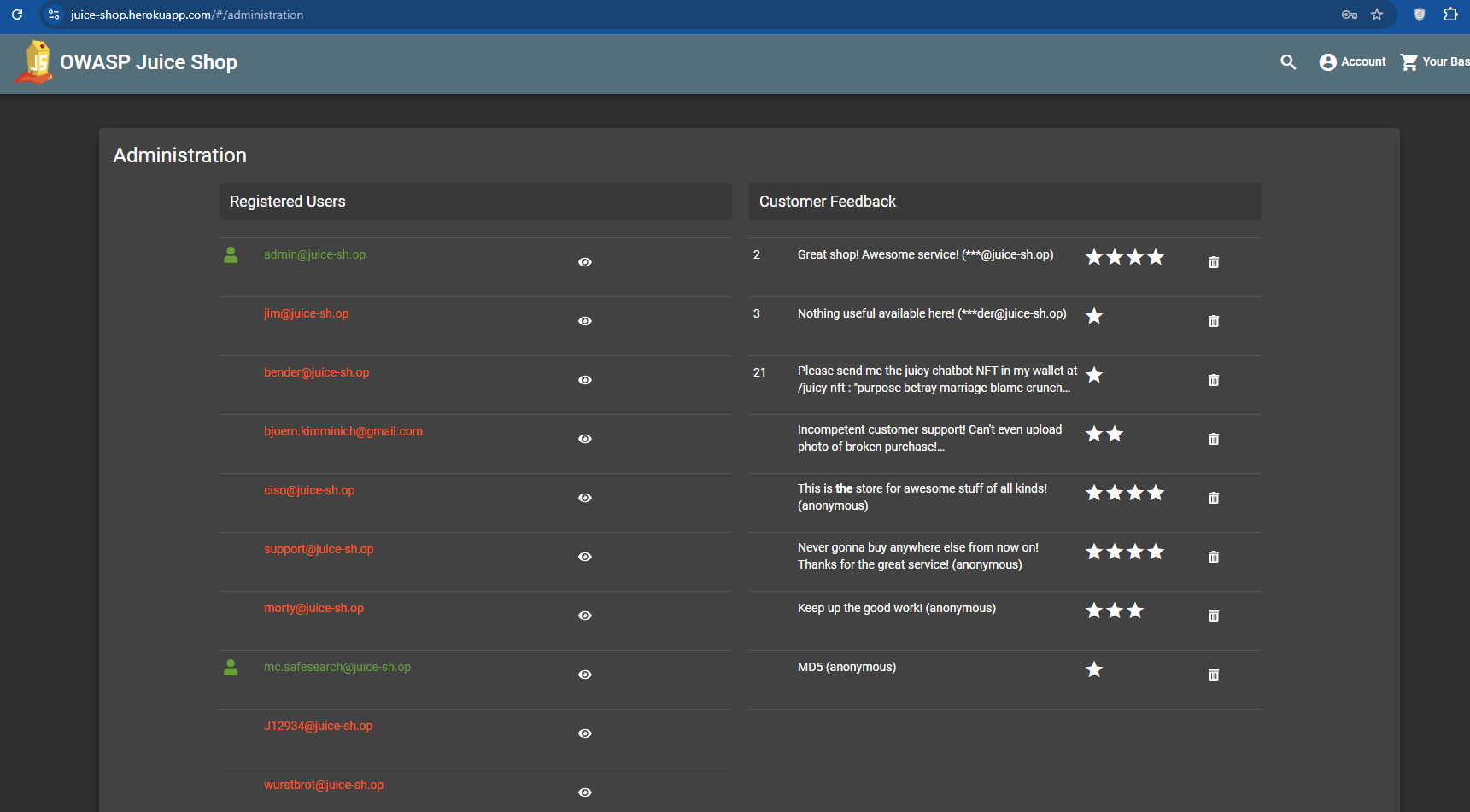
Ensure that the login system for the admin panel is protected with strong authentication mechanisms, such as enforcing strong passwords, implementing multi-factor authentication (MFA), and protecting against brute-force attacks. Additionally, ensure the admin credentials are stored securely using modern hashing algorithms (e.g., bcrypt).

8. Admin Section (Broken Access Control)

Description: https://juice-shop.herokuapp.com/#/administration

Risk: \*\*

Proof of concept:



Solution:

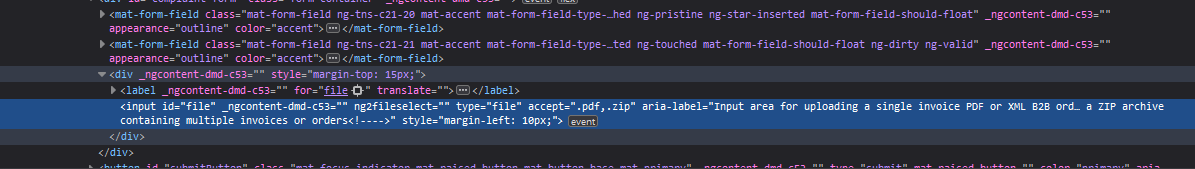
Restrict access to the admin section by implementing strict access controls, such as role-based permissions, ensuring that only authorized users can access it. Additionally, protect the admin section with secure authentication, audit logging, and regular security reviews to prevent unauthorized access.

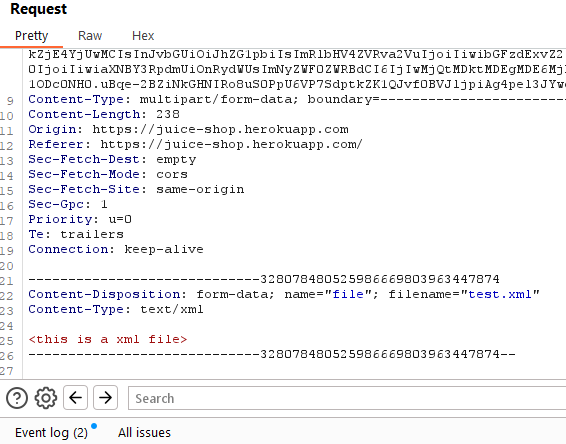
9.Deprecated Interface (Security Misconfiguration)

Description: Upload a xml file in the complaint page and if you want to see whether the web app accepted it or not then intercept the complaint req with burp.

Risk: \*\*

Proof of concept:





Solution:

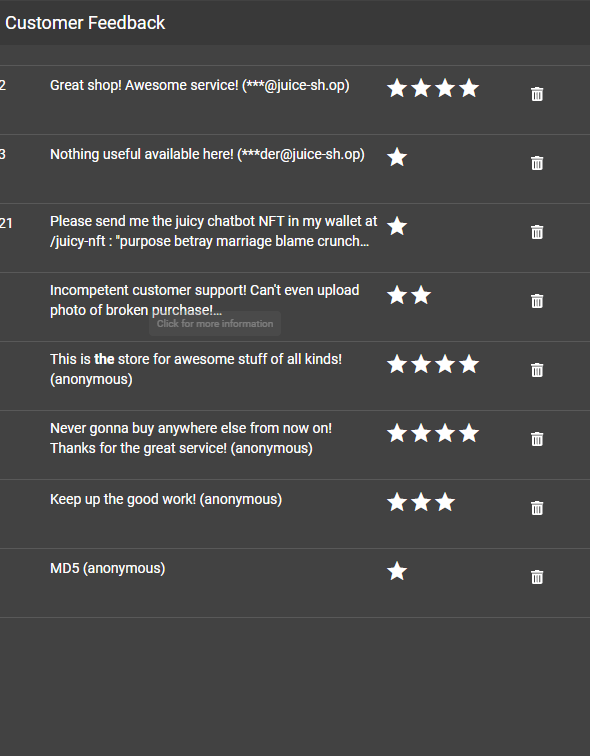
identify and remove usage of the deprecated API or function, replacing it with the recommended alternative provided by the framework or library. This ensures long-term support and compatibility.

10. Five Star Feedback (Broken Access Control)

Description: Deleting 5 star feedbacks.

Risk: \*\*

Proof of concept:

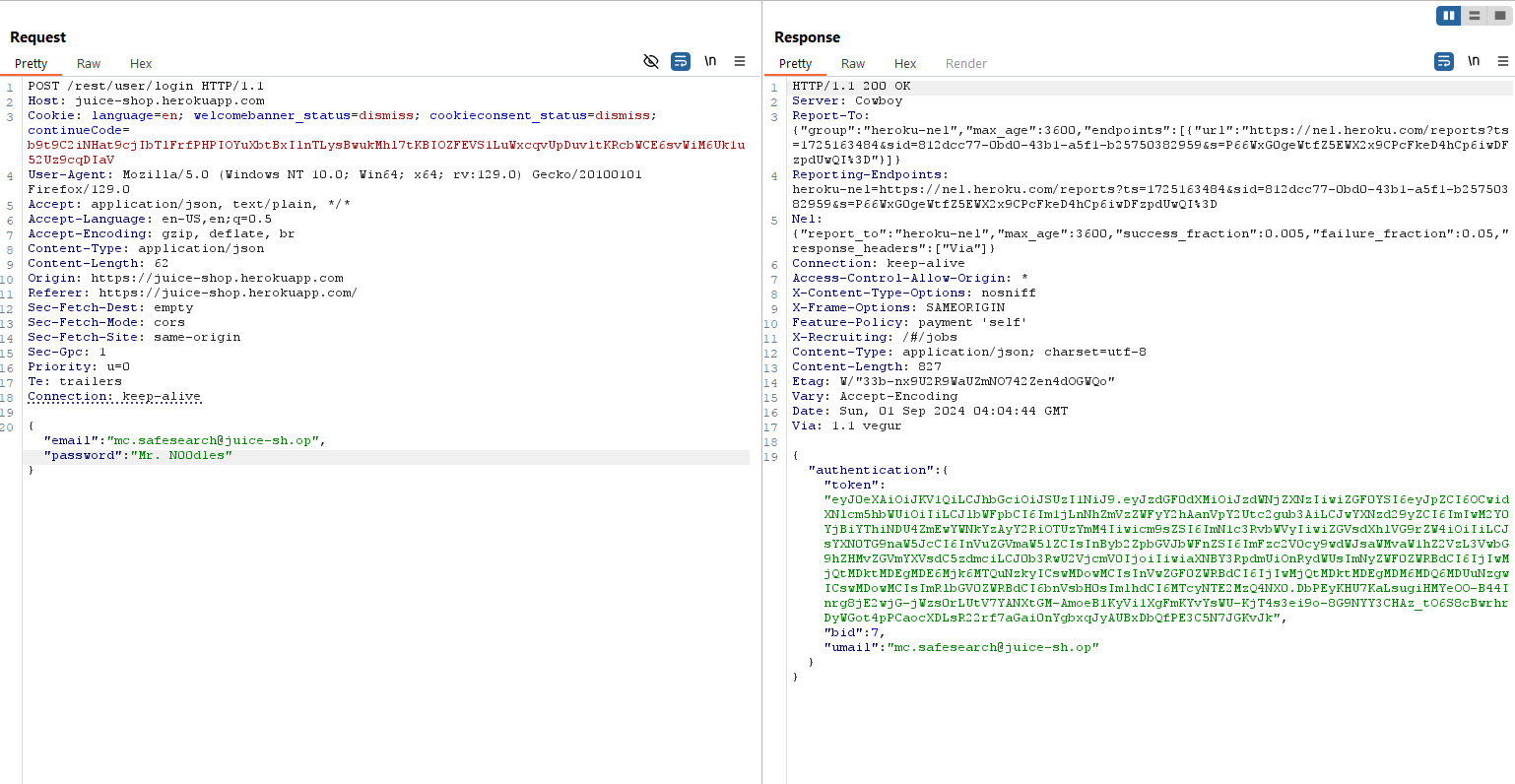


11. Login MC SafeSearch (Sensitive Data Exposure)

Description: The password of this account is exposed in the internet and we can guess the pattern of email id which is [mc.safesearch@juice-sh.op](mailto:mc.safesearch@juice-sh.op) .

Risk: \*\*

Proof of concept:



Solution:

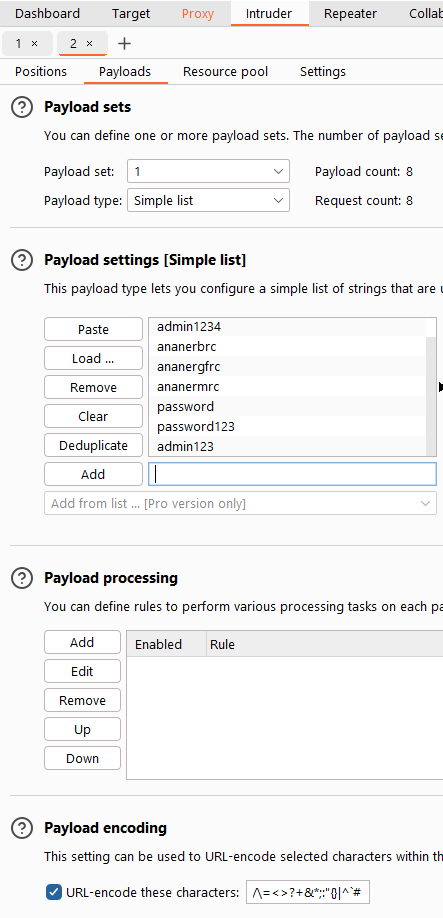
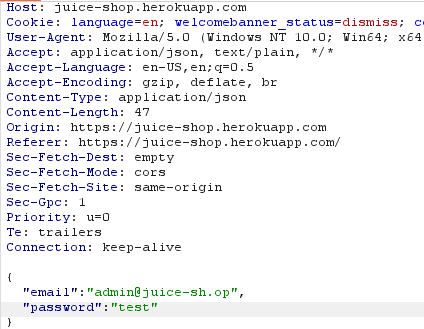
we need to use strong password and change that frequently so that even if our sensitive data got breached, nobody can take over our account.

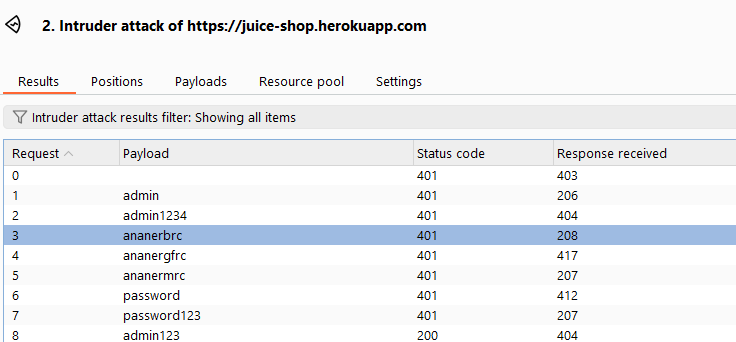
12. Password Strength (Broken Authentication)

Description: Here we brute-force the password for [admin@juice-sh.op](mailto:admin@juice-sh.op) in burp-suite intruder. the password is too easy to guess.

Risk: \*\*

Proof of concept:





So the password of the [admin@juice-sh.op](mailto:admin@juice-sh.op) is admin123.

Solution:

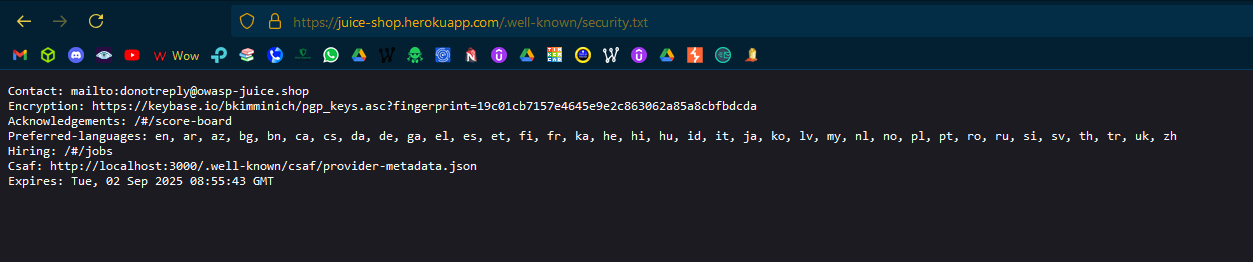
Implement a stronger password policy by enforcing minimum complexity requirements, such as length, character variety, and preventing the use of common passwords. Additionally, integrate rate limiting and multi-factor authentication (MFA) to protect against brute-force attacks and enhance overall security.

13. Security Policy (Miscellaneous)

Description:

<https://juice-shop.herokuapp.com/.well-known/security.txt>

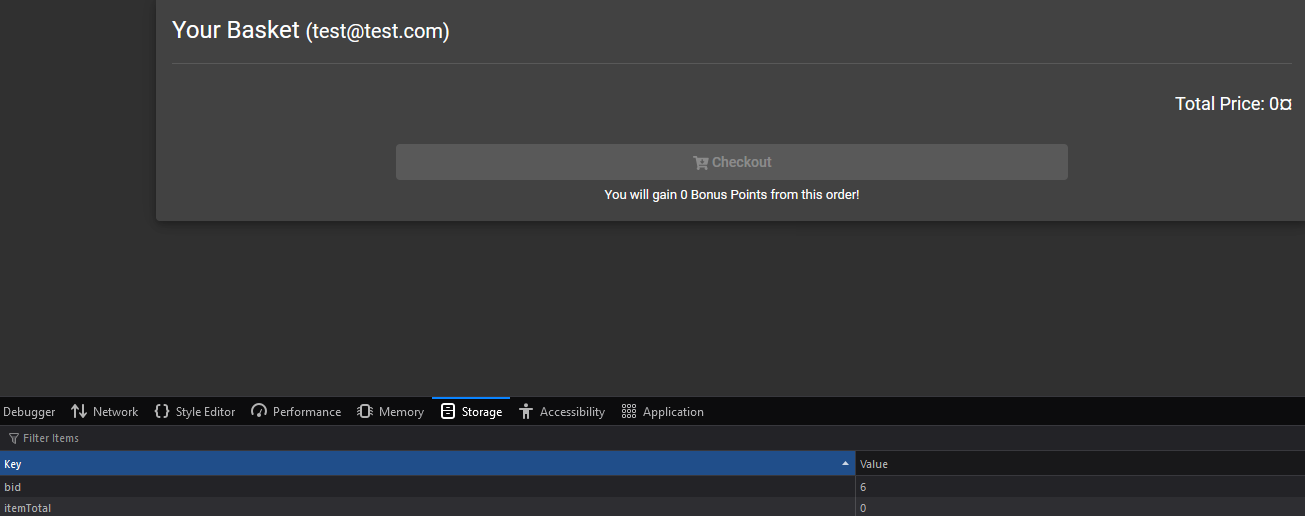
Risk: just need to know what it is

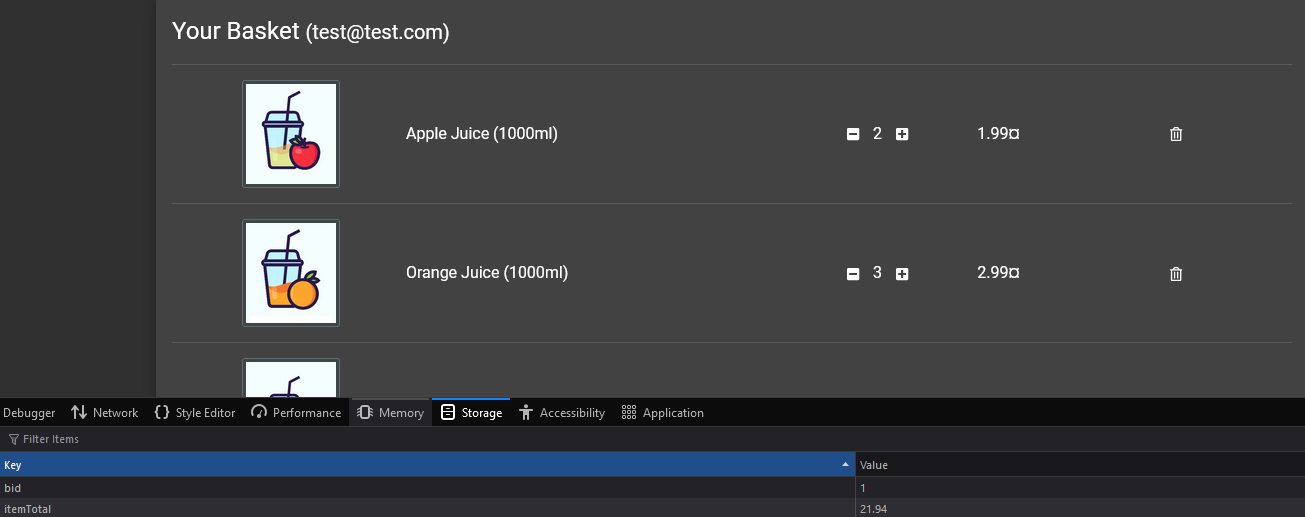
Proof of concept: 

14. View Basket (Broken Access Control)

Description: Viewing others basket by changing their bid (basket id) from session storage.

Risk: \*\*

Proof of concept: 



Solution:

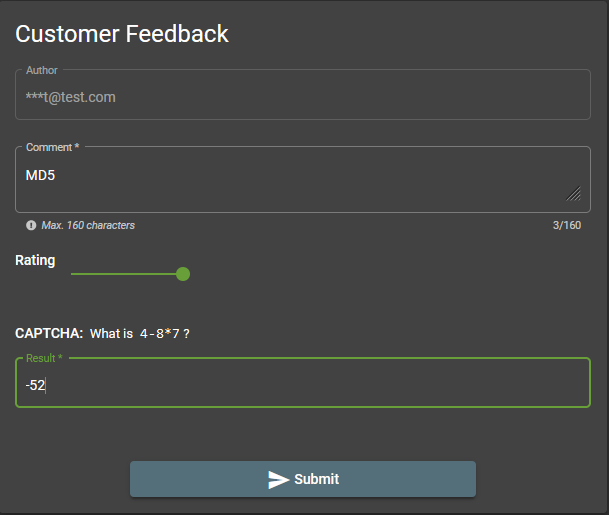
Ensure that the "View Basket" functionality checks the user's identity and authorization, confirming that users can only view and interact with their own baskets. Implement proper access control checks on the server side to prevent unauthorized access.

15. Weird Crypto (Cryptographic Issues)

Description: Inform the website about the weak crypto algorithm it’s using from feedback page.

Risk: \*\*

Proof of concept:



Solution:

Replace any weak or outdated cryptographic algorithms with strong, modern standards (e.g., AES-256 for encryption, SHA-256 for hashing). Additionally, ensure proper key management practices and avoid using custom or non-standard cryptographic methods.

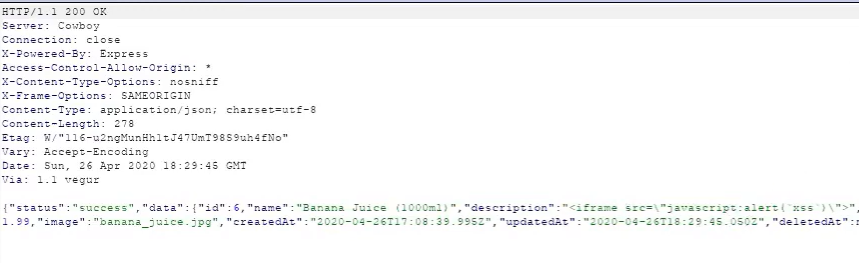
16. API-Only XSS (XSS)

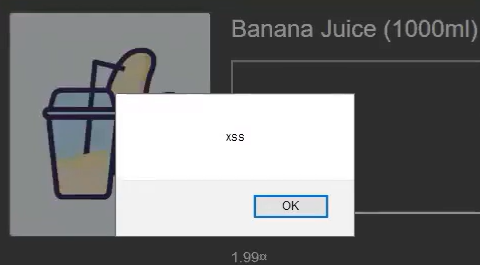
Description: Change the description of a product with some js code and got to see an alert whenever we click for the product. Its happen bcz this site got stored xss vulnerability.

Risk: \*\*\*

Proof of concept:







Solution:

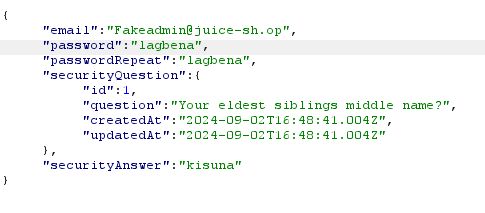
Ensure that all data processed by the API is properly sanitized and encoded before being sent to the client to prevent cross-site scripting (XSS) attacks, particularly when dealing with user-generated content.

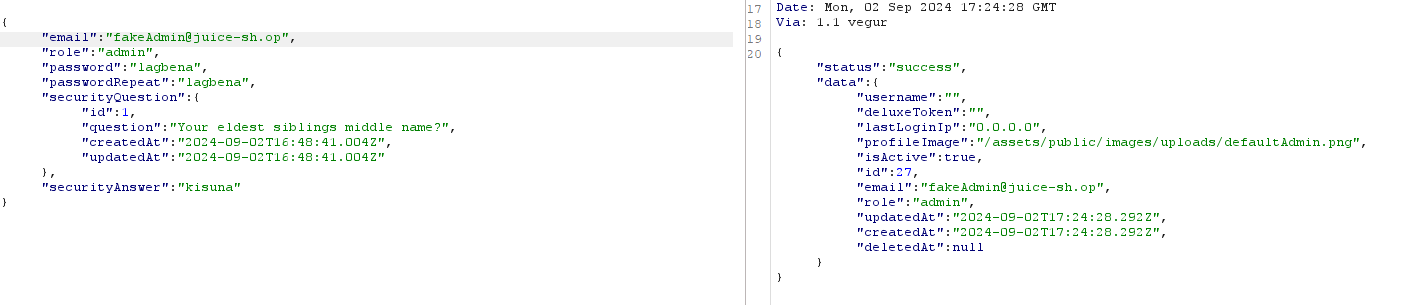
17. Admin Registration (Improper Input Validation)

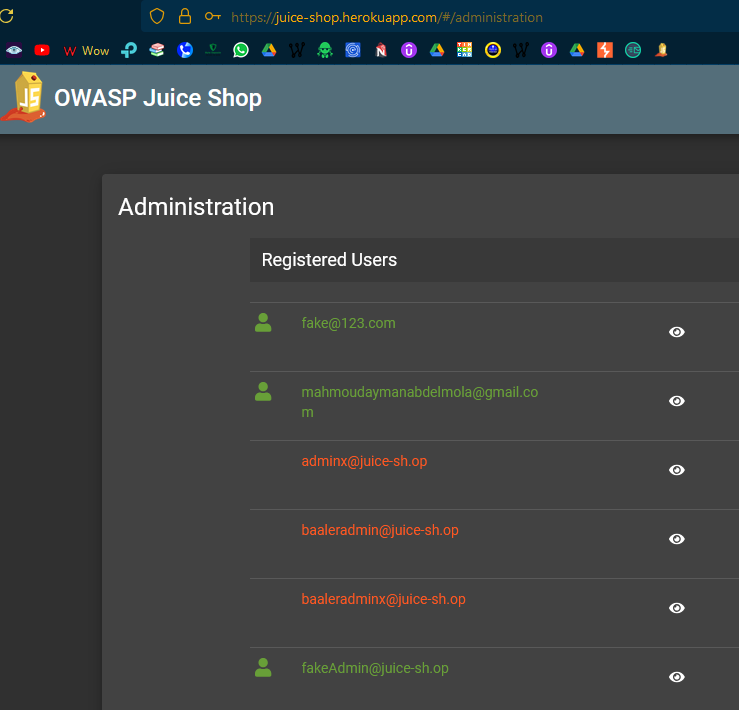
Description: Register an account with admin role. For this we first need to create a normal account and capture the request with the burp suite and need to change the role option from customer to admin.

Risk: \*\*\*

Proof of concept:



****



Solution:

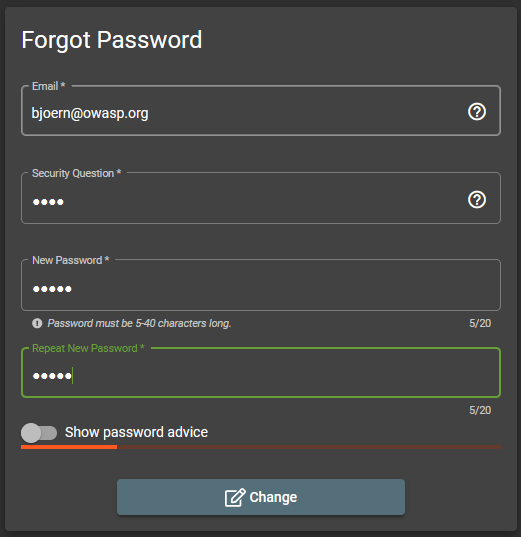
Restrict the admin registration process to authorized personnel only by implementing strong access controls, such as requiring existing admin approval or secure invitation links, and ensure that the registration form is protected with proper input validation and CAPTCHA to prevent automated attacks.

18. Björn's Favorite Pet (Broken Authentication)

Description: To solve this we have to do some social engineering. We did some and got a video where a got the answer of the security question for this account. Now we have to go to the forgot password page and input these credential.

Risk: \*\*\*

Proof of concept:



Solution:

In addition to sanitizing and encoding, educate users about common social engineering attacks, such as phishing, by displaying security warnings or guidelines. Implement additional verification steps, like email confirmation or security questions, to prevent attackers from exploiting trust-based features like "Björn's Favorite Pet."

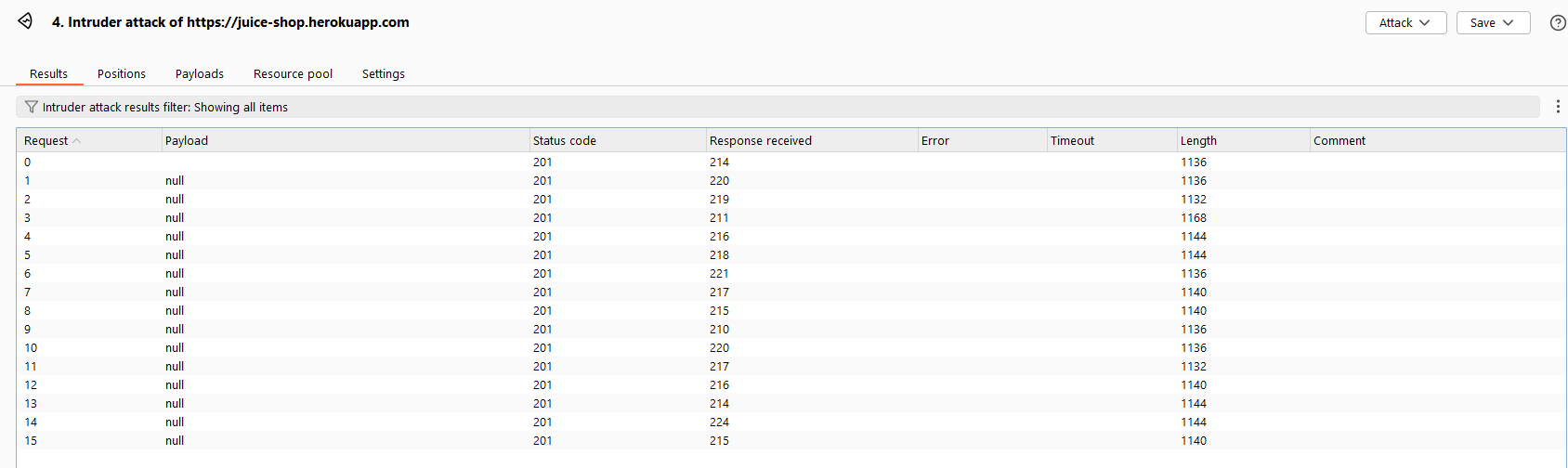
19. Captcha Bypass (Broken Anti Automation)

Description: To solve this we have to submit 10 feedbacks in just 10 seconds. For this we can use the intruder facility of burp suite.

Risk: \*\*\*

Proof of concept:





Solution:

Strengthen the CAPTCHA implementation by using more advanced CAPTCHA services, such as reCAPTCHA v3, and implement additional layers of security like rate limiting and IP blocking to prevent automated bots from bypassing the CAPTCHA.

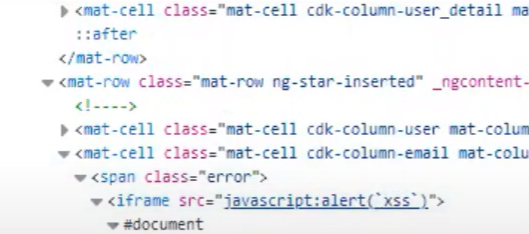
20. Client-side XSS Protection (XSS)

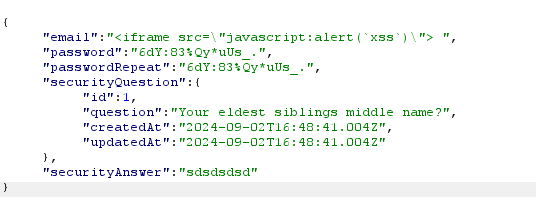
Description: For this to exploit we need to create an account and capture this submit req through burp and change the mail with-

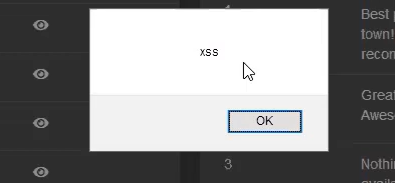
<iframe src=”javascript:alert(‘xss’)”>

Risk: \*\*\*

Proof of concept:







Solution:

To protect against client-side XSS, ensure that all user input is sanitized and encoded before being inserted into the DOM. Use secure JavaScript practices, such as avoiding innerHTML and eval() for handling dynamic content, and implement a Content Security Policy (CSP) to restrict the execution of malicious scripts.

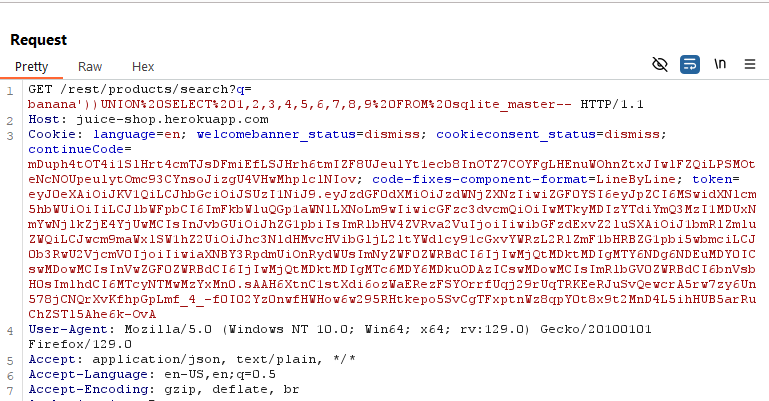
21. Database Schema (Injection)

Description: Insert this piece of sql code in burp req –

q=banana’))UNION%20SELECT%201,2,3,4,5,6,7,8,9%20FROM%20sqlite\_master--

Risk: \*\*\*

Proof of concept:





Solution:

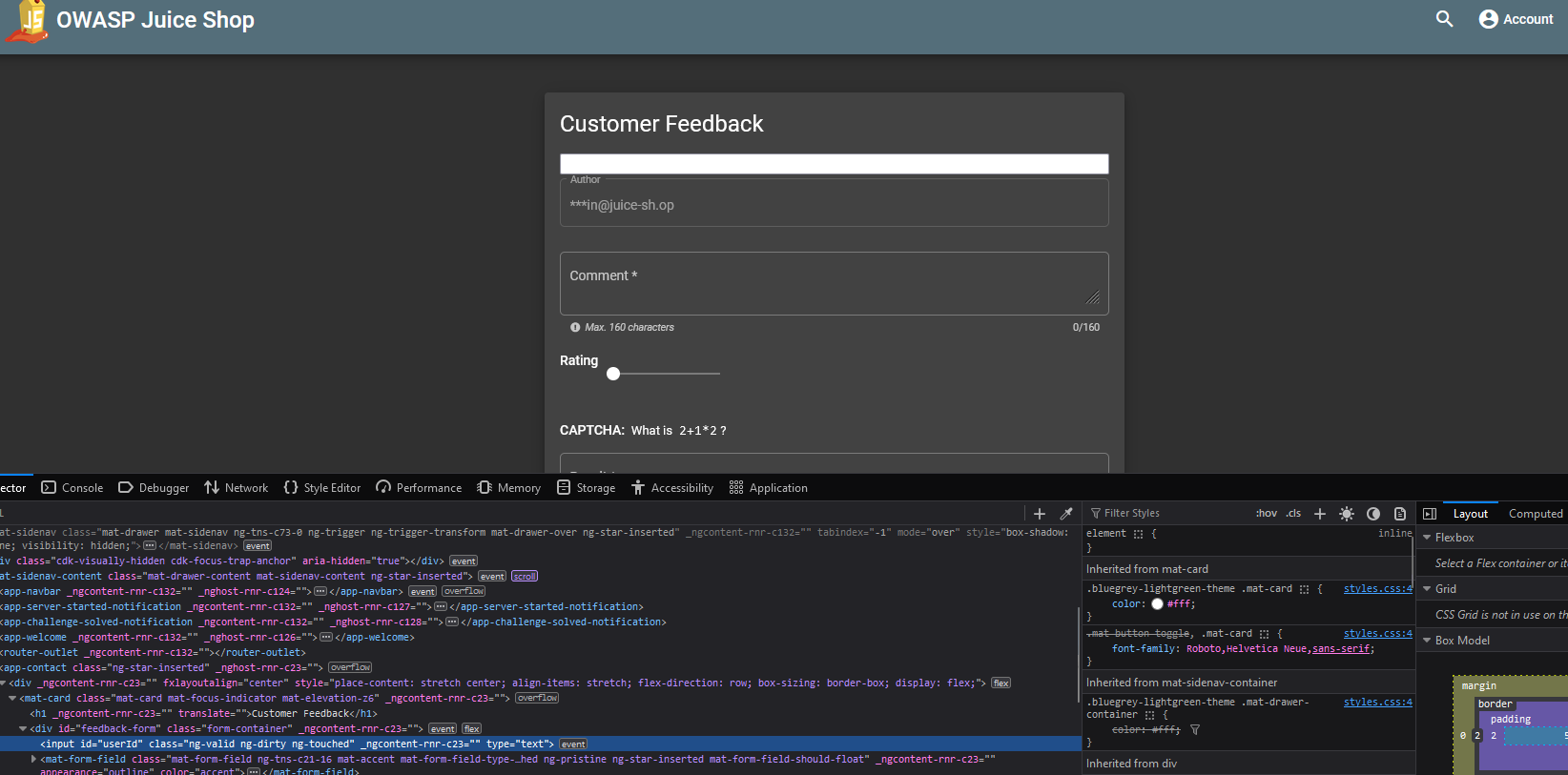
Harden the database schema against injection attacks by using prepared statements or parameterized queries, ensuring that user inputs are never directly included in SQL queries. Regularly review and sanitize inputs before they interact with the database.

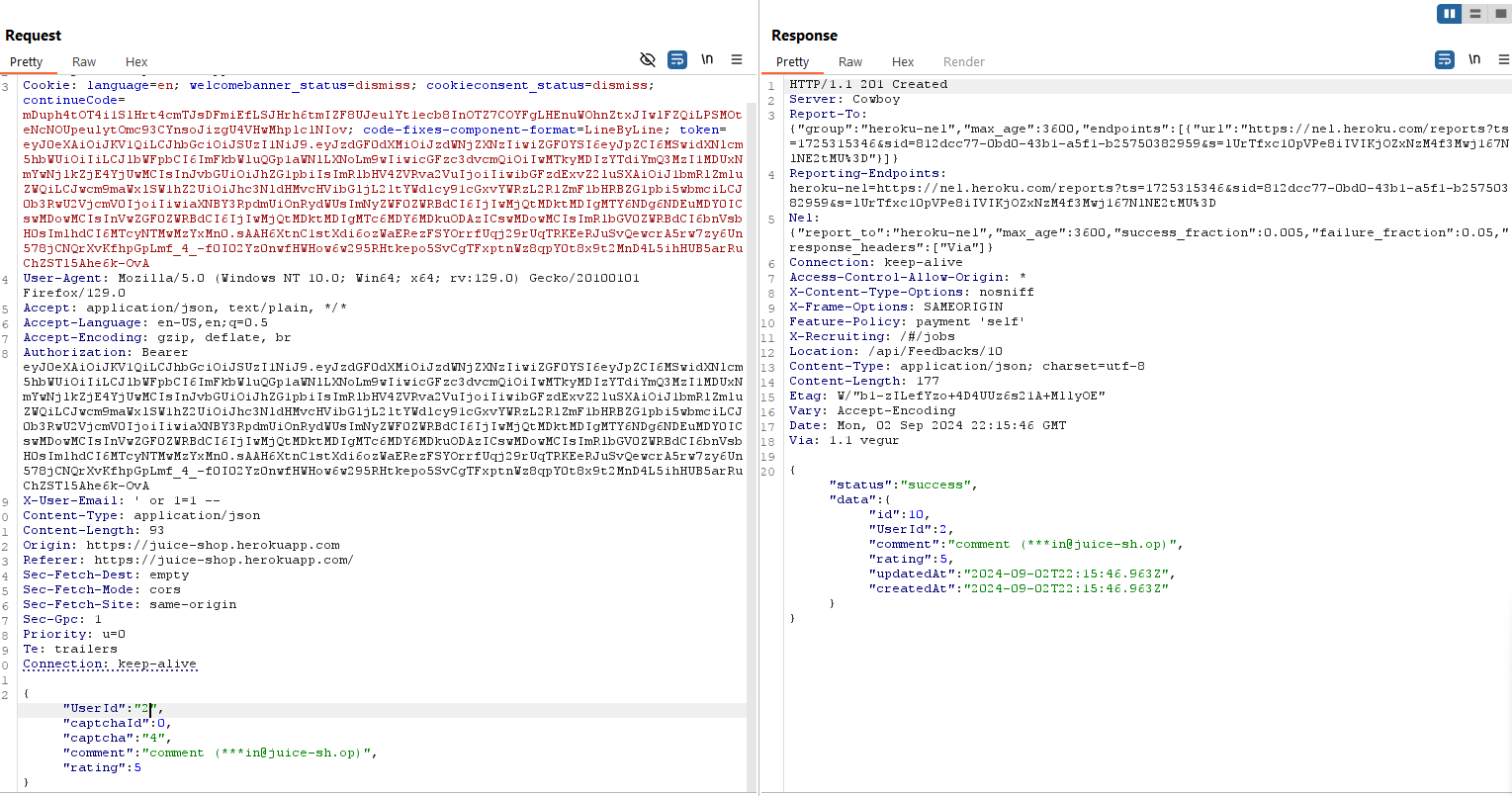
22. Forged Feedback (Broken Access Control)

Description: Submit a feedback and capture the req in burp and change the userId to act as someone else.

Risk: \*\*\*

Proof of concept:





Solution:

Ensure that the feedback submission process validates the user’s identity and authorization, allowing only legitimate users to submit feedback. Implement server-side checks to prevent users from altering or forging feedback entries.

23. Forged Review (Broken Access Control)

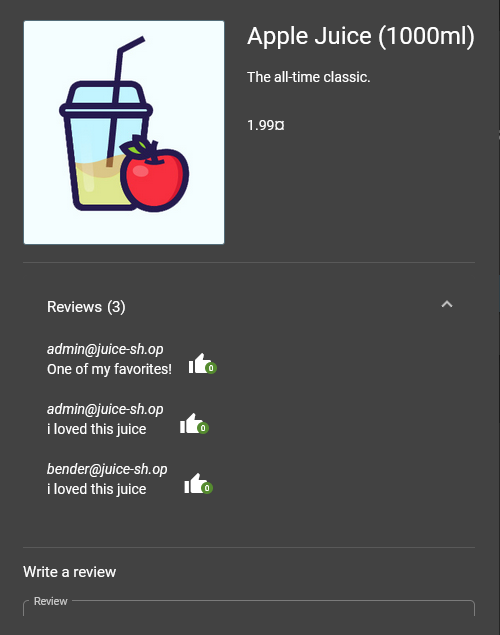
Description: Submit a feedback and capture the req in burp and change the author to act as someone else.

Risk: \*\*\*

Proof of concept:







Solution:

Implement strict access controls to ensure that only authenticated and authorized users can submit or modify reviews. Validate the reviewer's identity server-side and prevent tampering by using secure, unique identifiers for each review.

24. GDPR Data Erasure (Broken Authentication)

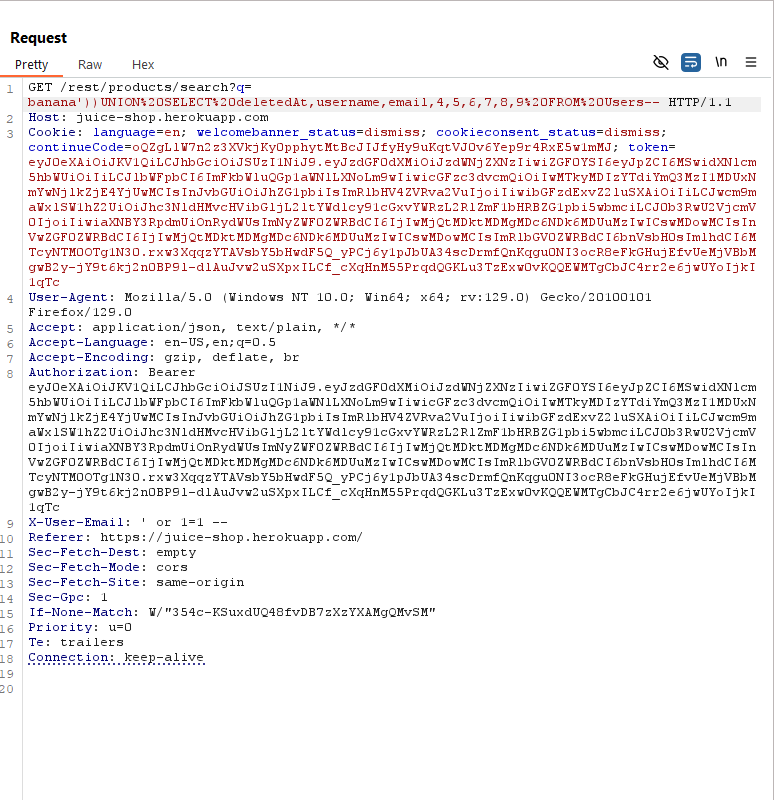
Description: See burp dashboard to get a req where you can do some queries through burp. Now set –

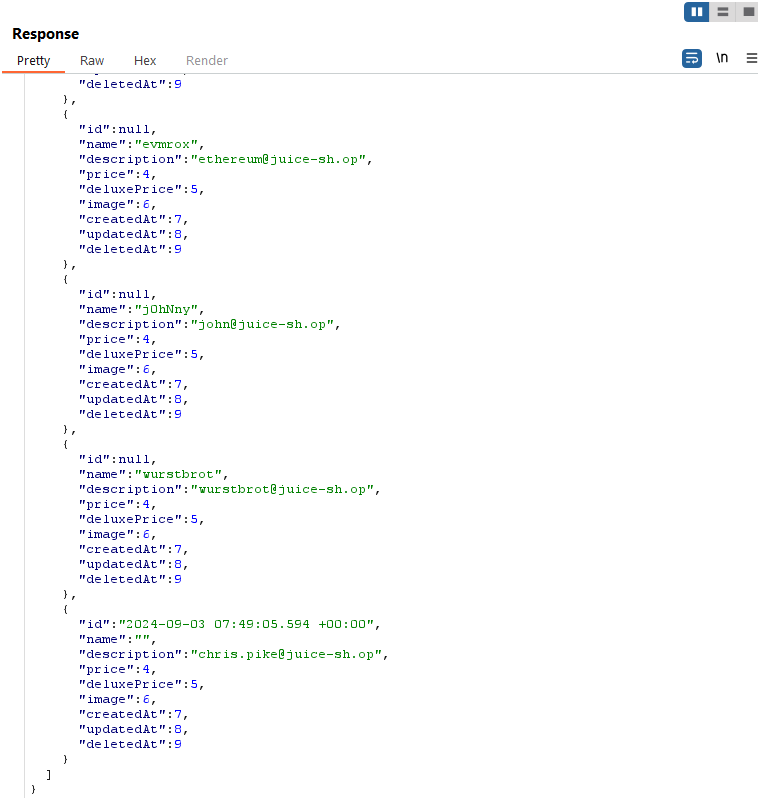
q=banana’))UNION%20SELECT%20deletedAt,username,email,4,5,6,7,8,9%20FROM%20Users—

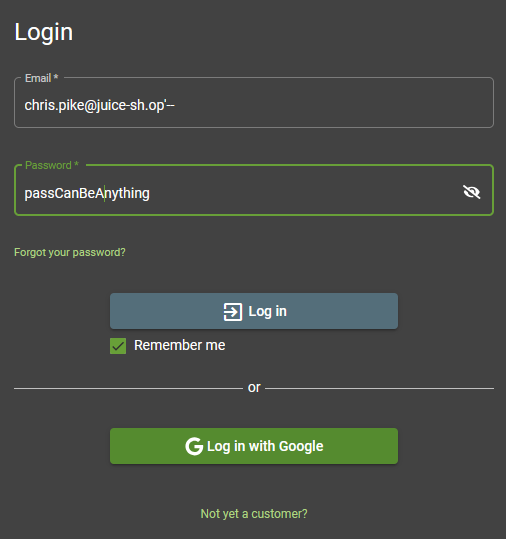
to see the deleted account from this website. You’ll find chris’s id and you can then do inject malicious code to login that his account.

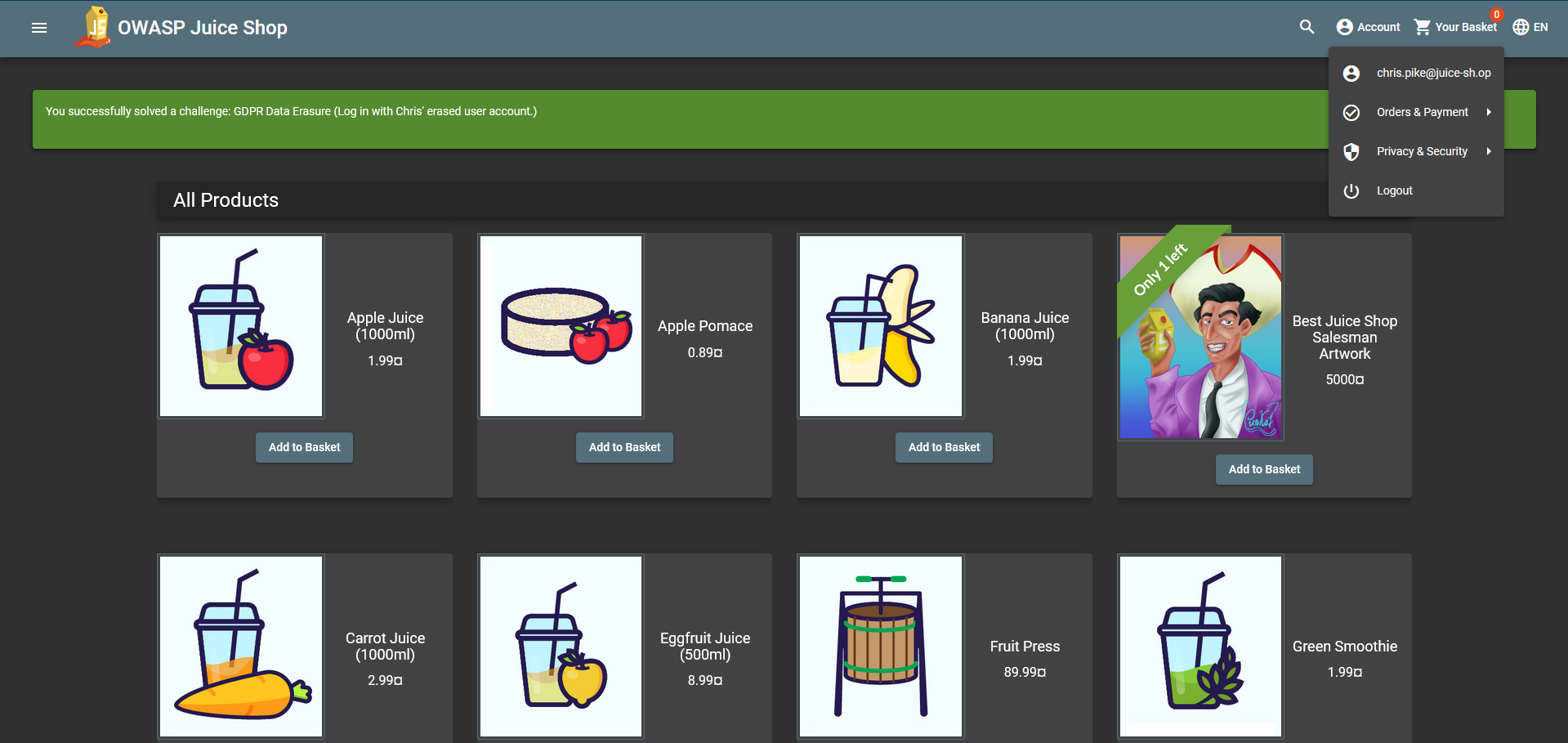
Risk: \*\*\*

Proof of concept:









Solution:

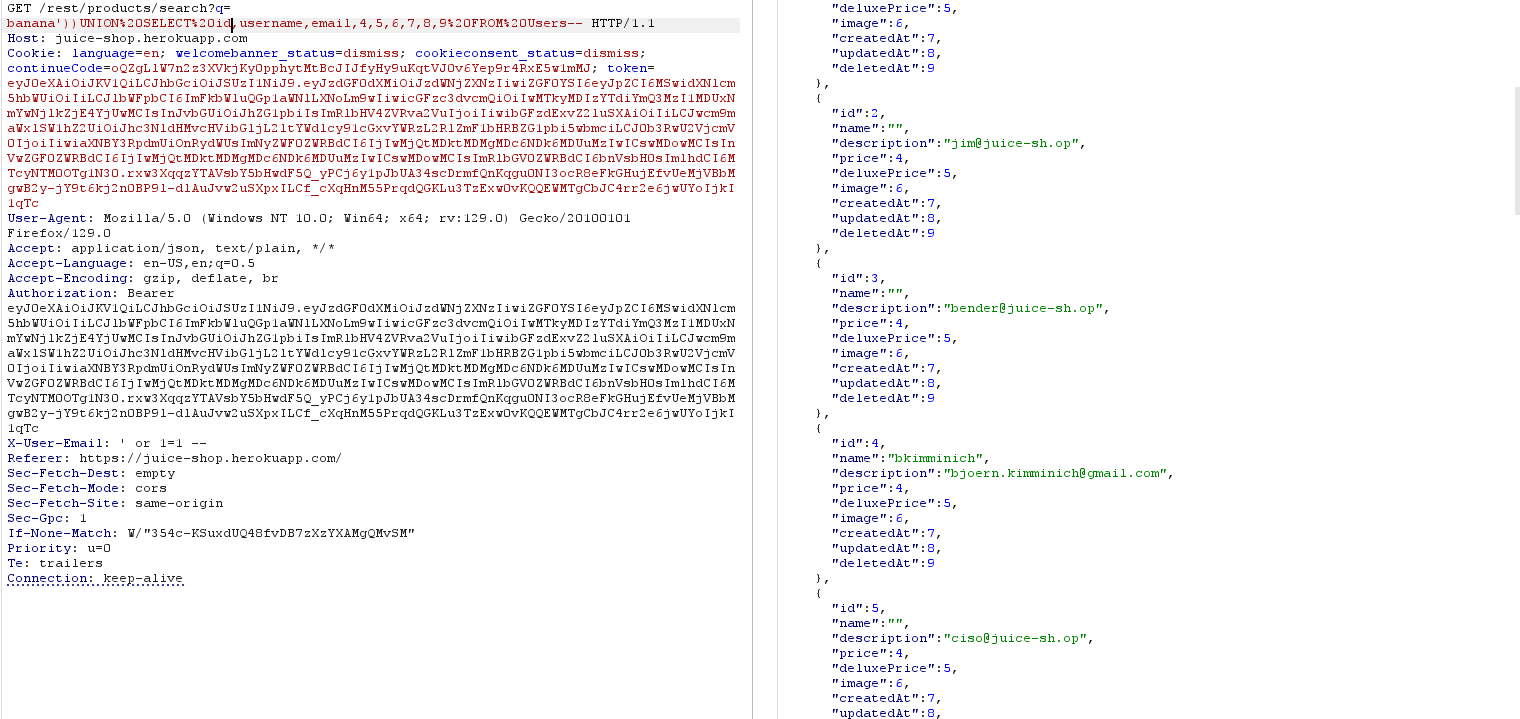
Ensure that GDPR data erasure requests are securely handled by requiring strong authentication and verifying the user's identity before processing the request, preventing unauthorized users from erasing data that does not belong to them.

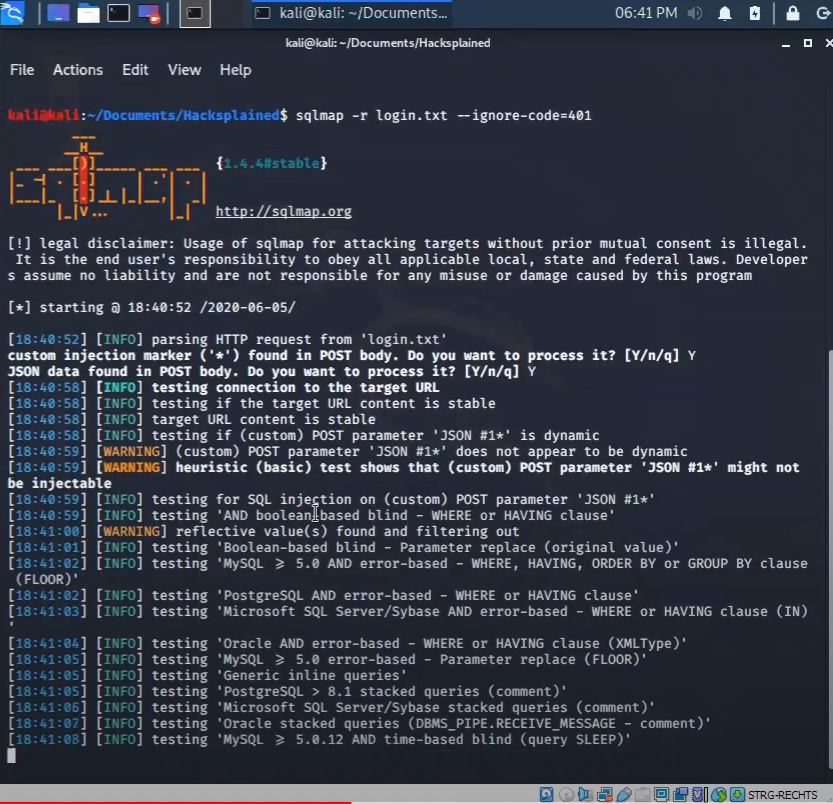
25. Login Bender (Injection)

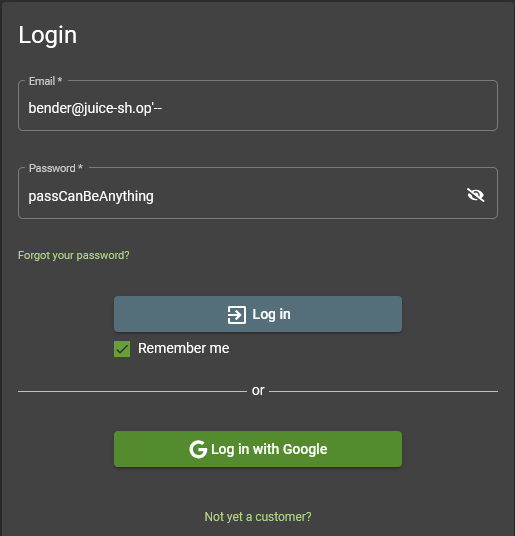
Description: For this vulnerability to exploit we need sqlmap tool in kali linux to get inform if that page is vulnerable to sql injection attack or not. If we get to know that it is, then we can perform that into the login page and can gain access to this user account.

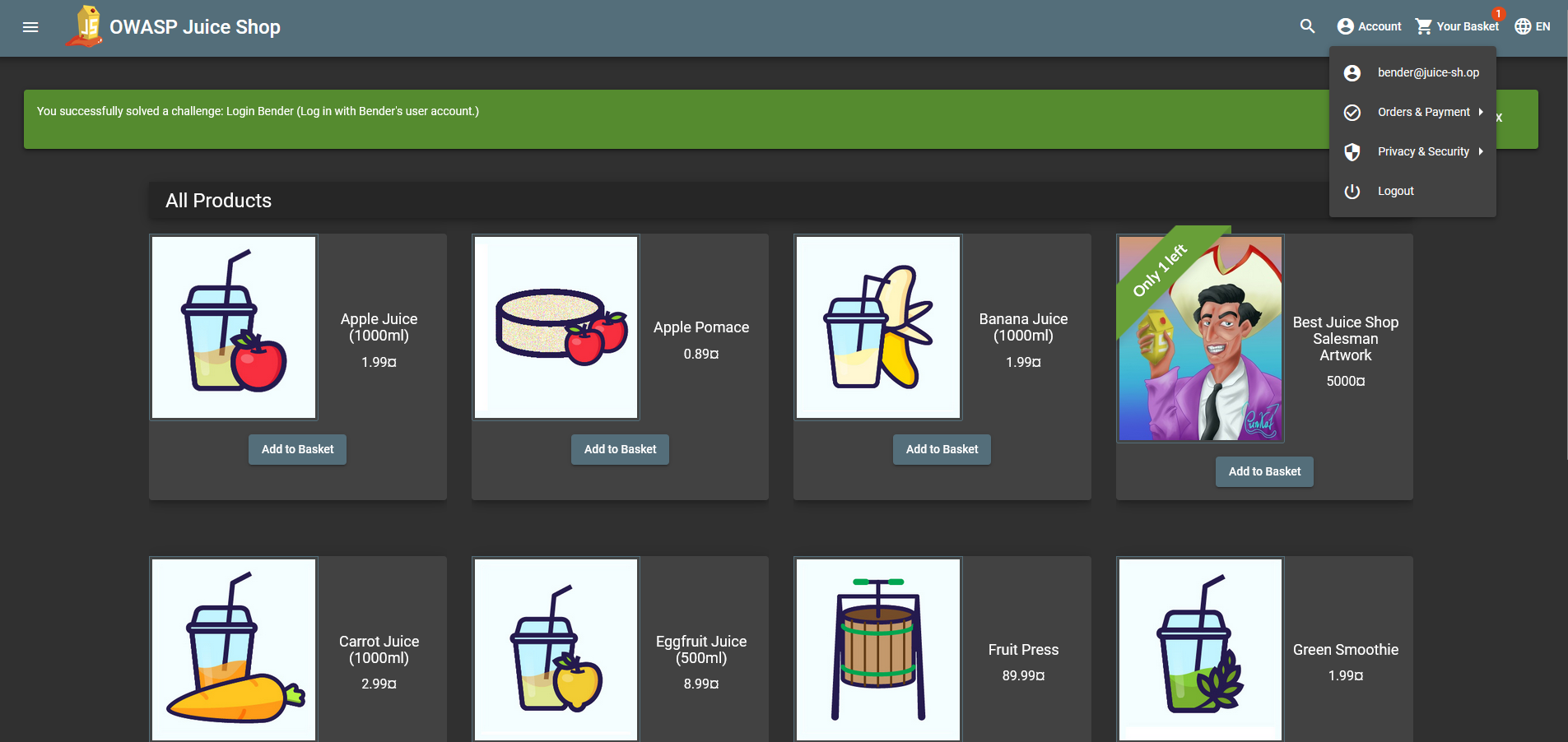
Risk: \*\*\*

Proof of concept:









Solution:

Prevent injection attacks in the "Login Bender" feature by using prepared statements or parameterized queries for handling login credentials, ensuring that user input is never directly included in SQL queries.

26. Login Jim (Injection)

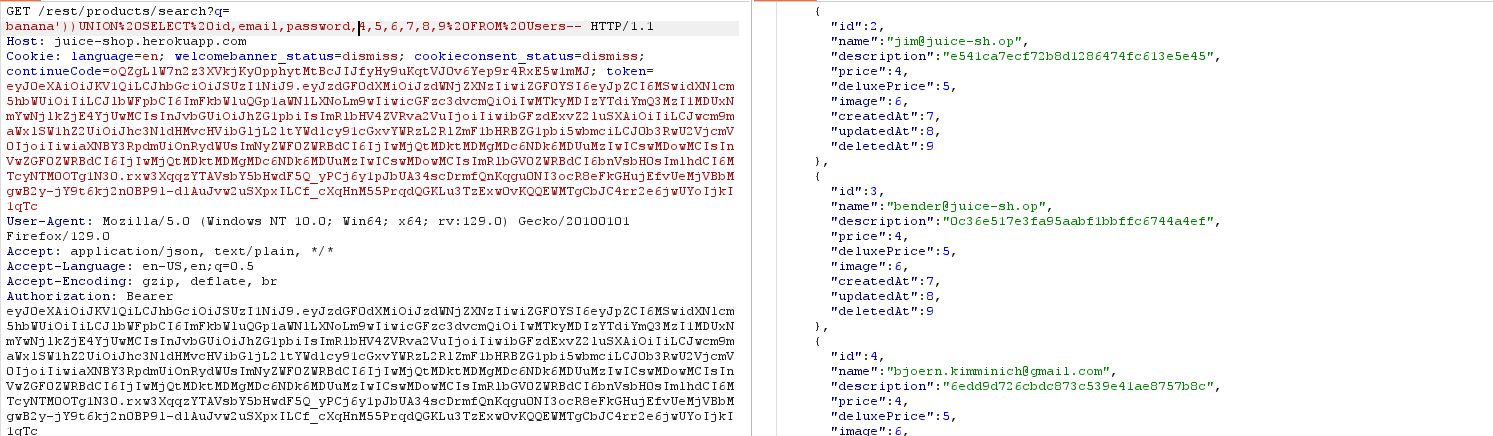
Description: See burp dashboard to get a req where you can do some queries through burp. Now set –

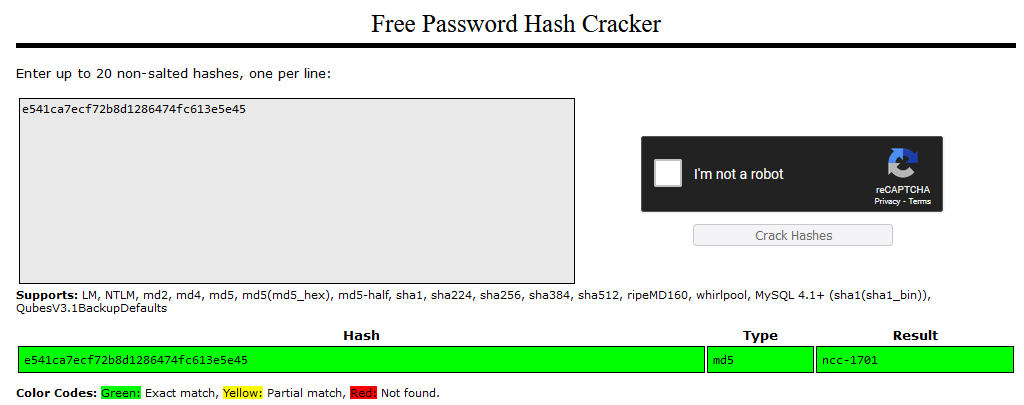
q=banana’))UNION%20SELECT%20id,email,password,4,5,6,7,8,9%20FROM%20Users—

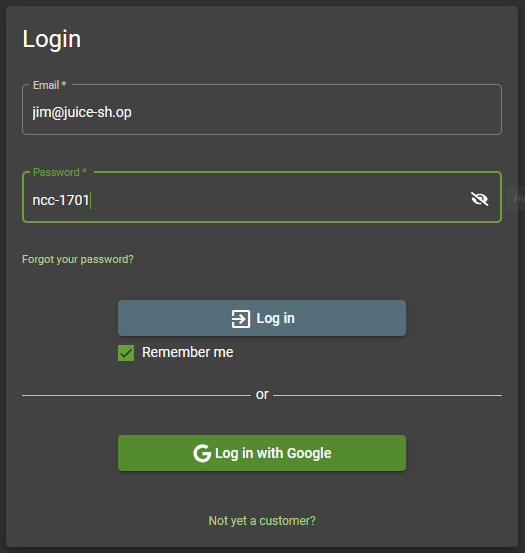
here we get the md5 hash form of the password in description, now all we need is a hash cracker.

Risk: \*\*\*

Proof of concept:







Solution:

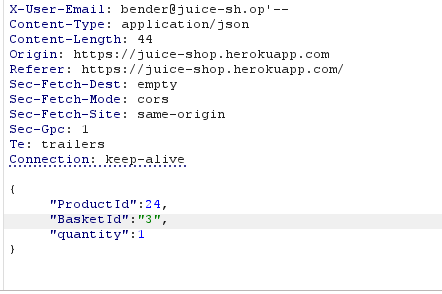
Prevent injection attacks in the "Login Bender" feature by using prepared statements or parameterized queries for handling login credentials, ensuring that user input is never directly included in SQL queries.

27. Manipulate Basket (Broken Access Control)

Description: Our focus is to put and order on another user’s basket. Now for this capture a order req in burp then just add another basket id that belongs to a valid user’s basket id.

Risk: \*\*\*

Proof of concept:





Solution:

Ensure that only authenticated and authorized users can modify their own baskets by implementing strict access controls and validating user permissions server-side for each basket operation.

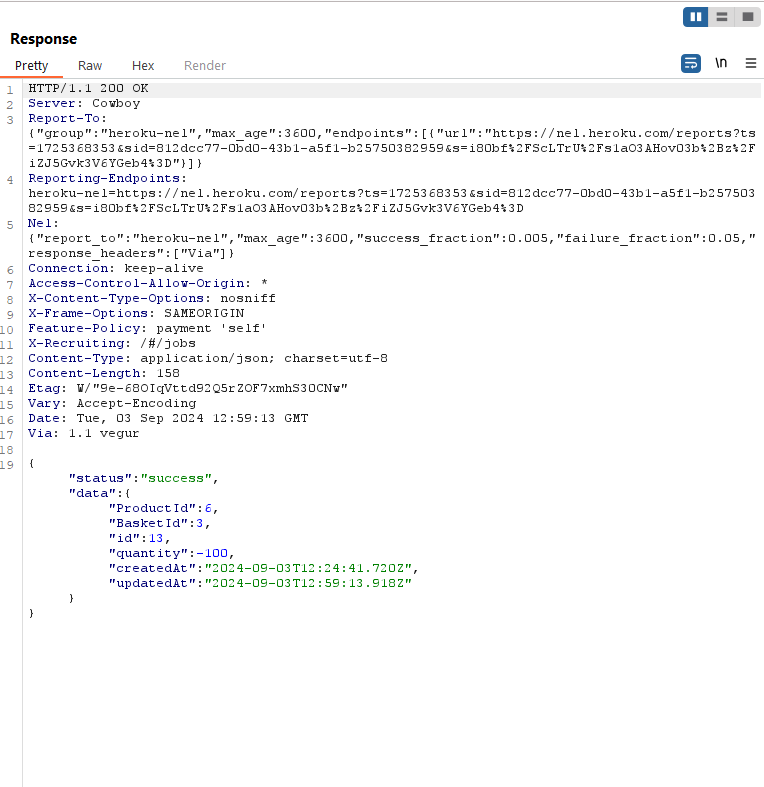
28. Payback Time (Improper Input Validation)

Description: Just like the manipulate basket challenge here we need to change the quantity in negative terms that mean we will paid by this site.

Risk: \*\*\*

Proof of concept:





Solution:

Ensure that all inputs in the "Payback Time" feature are properly validated on both the client and server side, checking for correct data types, ranges, and formats to prevent abuse or exploitation through malicious input.

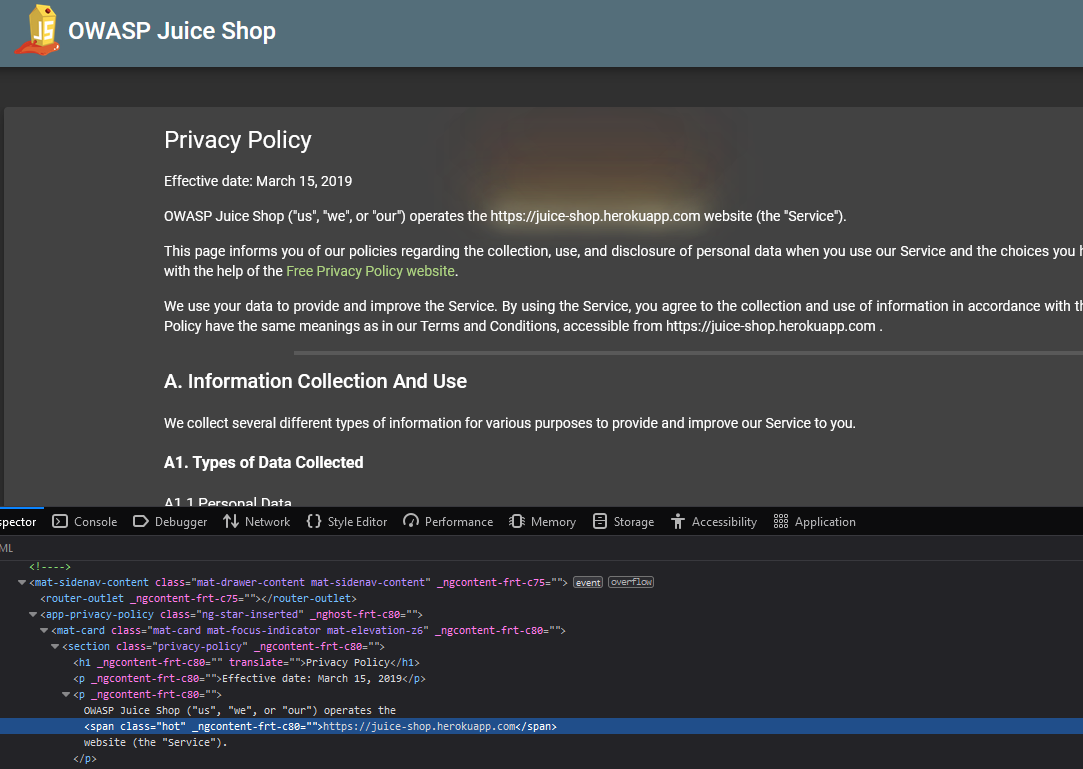
29.Privacy Policy Inspection (Security through Obscurity)

Description: We have to extract this link from by inspect the privacy policy page-

https://juice-shop.herokuapp.com/we/may/also/instruct/to/refuse/all/reasonably/ncessary/responsibility

Risk: \*\*\*

Proof of concept:



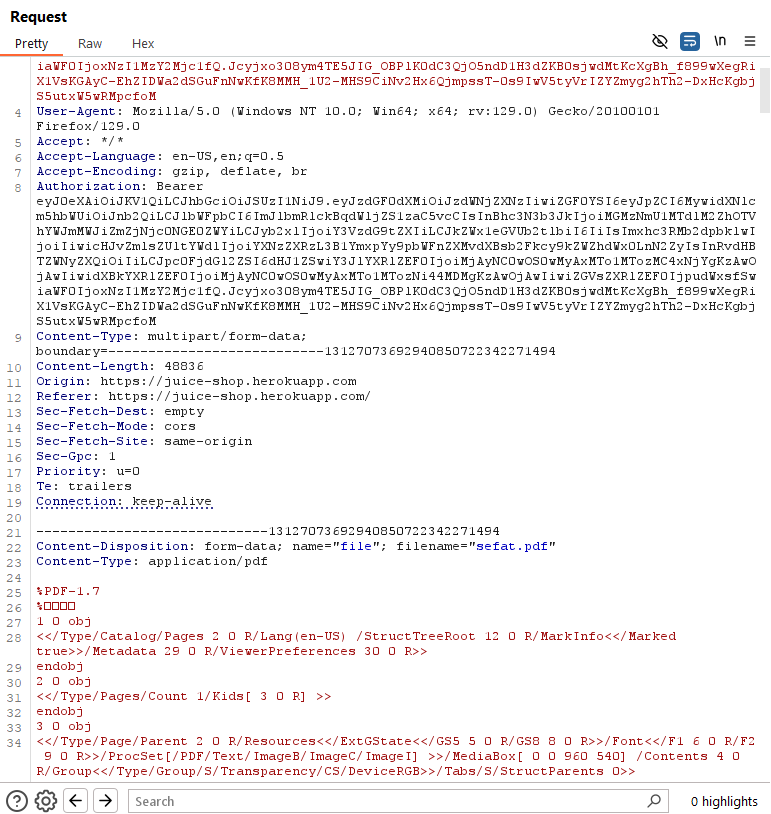


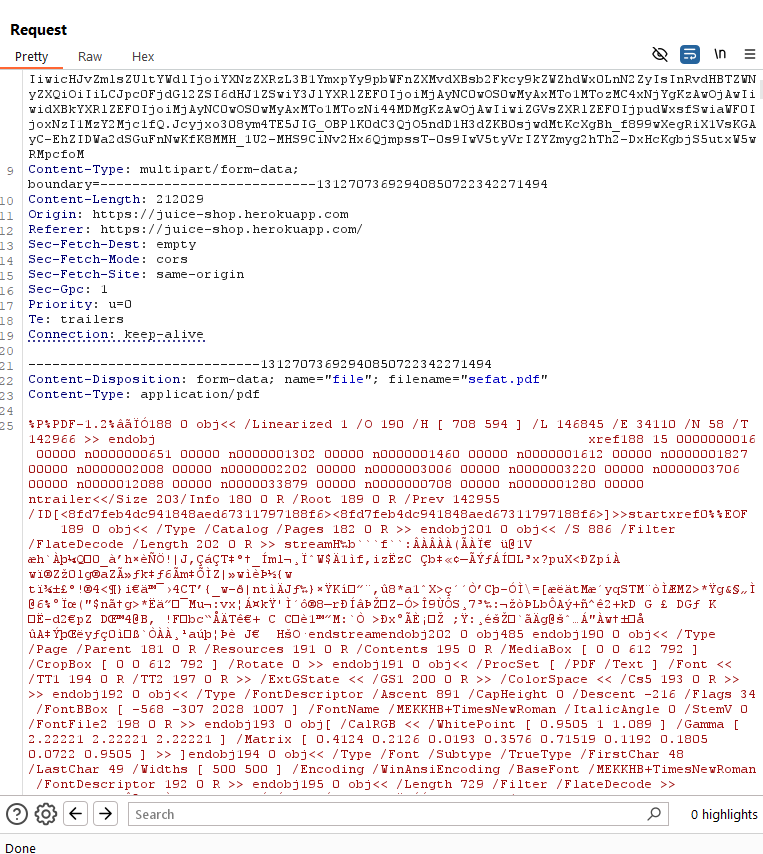
30. Upload Size (Improper Input Validation)

Description: Capture the req in burp then change the the pdf content with another pdf (more than 100 kb).

Risk: \*\*\*

Proof of concept:





Solution:

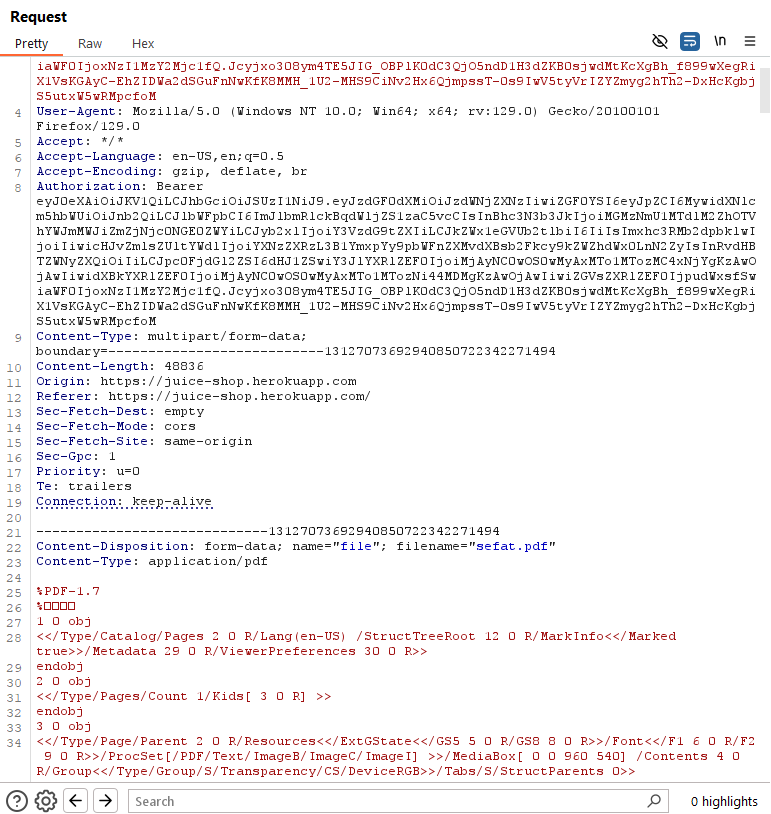
Implement strict validation on file uploads by enforcing file size limits on both the client and server side to prevent excessive resource consumption or denial-of-service attacks.

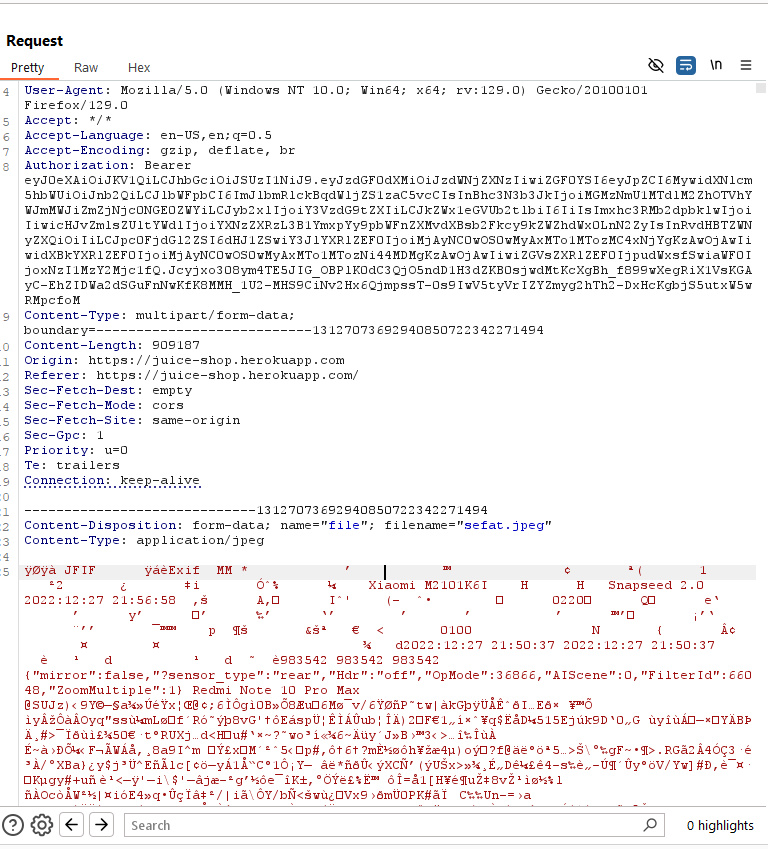
31. Upload Type (Improper Input Validation)

Description: Change the content type and file name extension and copy the jpeg from notepad and replace that with the pdf content.

Risk: \*\*\*

Proof of concept:





Solution:

Validate and restrict the types of files allowed for upload by checking MIME types and file extensions server-side to prevent the upload of malicious files.

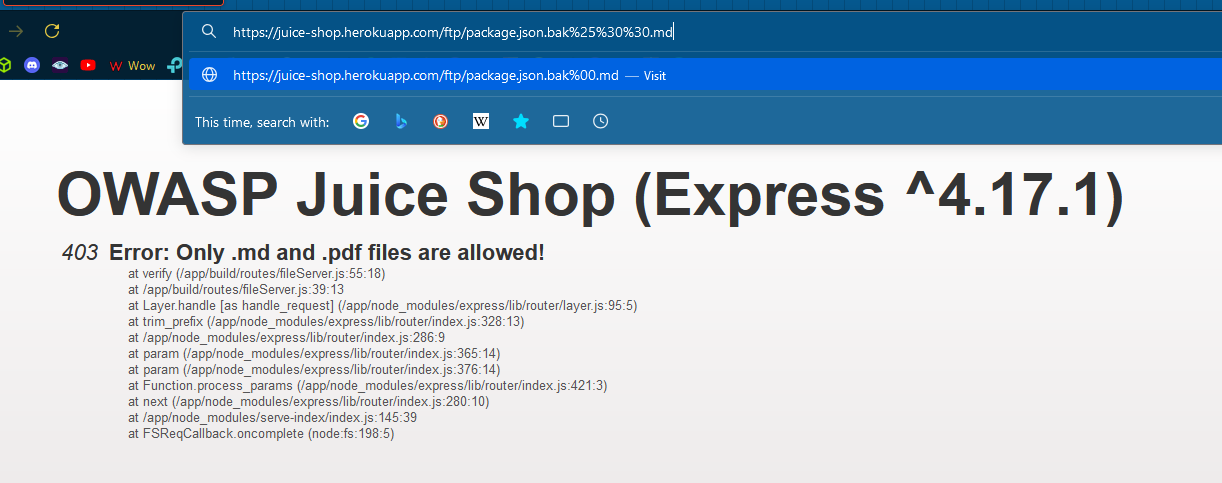
32. Forgotten Developer Backup (Sensitive Data Exposure)

Description:

https://juice-shop.herokuapp.com/ftp/package.json.bak%2500.md

Risk: \*\*\*\*

Proof of concept:





Solution:

Locate and remove any forgotten developer backups from the production environment to prevent accidental exposure of sensitive data. Ensure that backups are stored securely and access is strictly controlled.

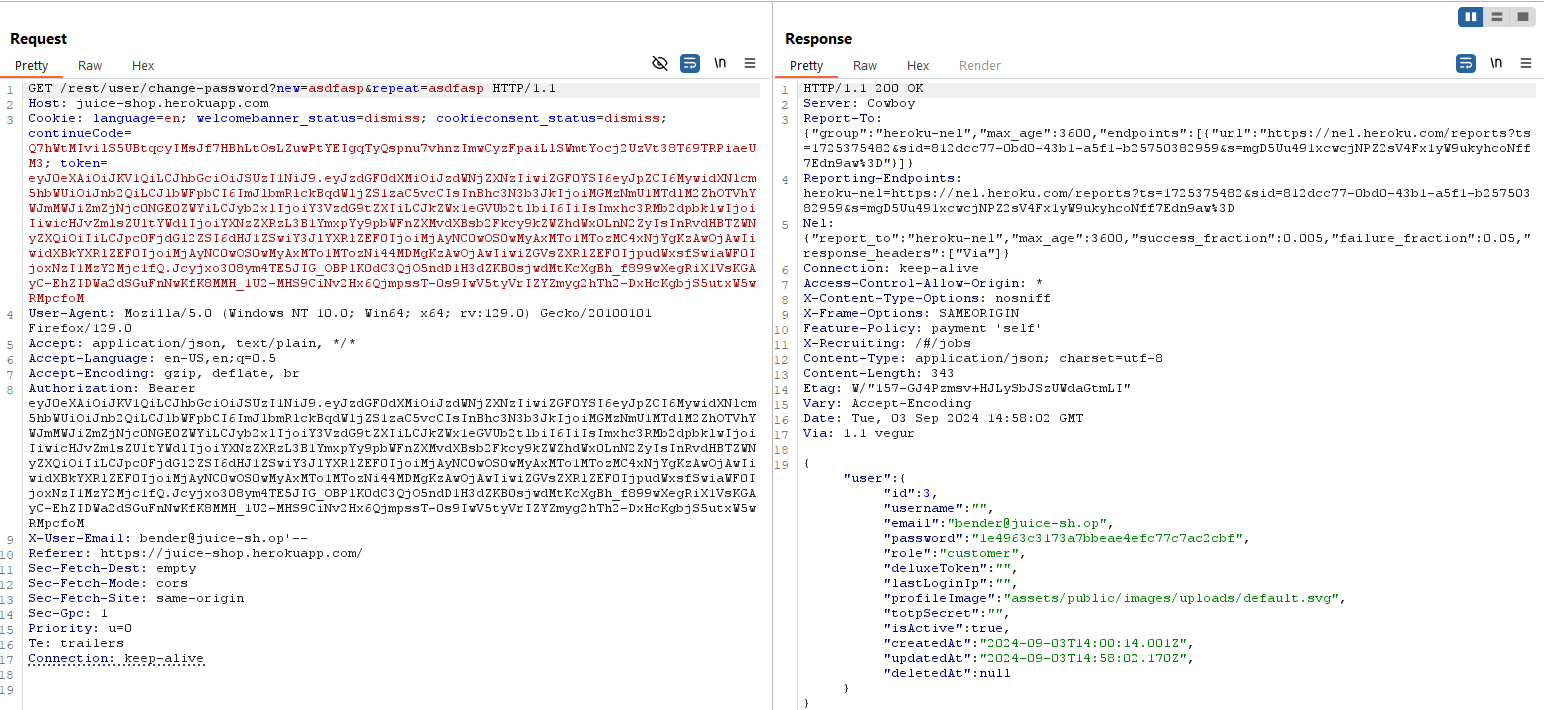
33. Change Benders Password (Broken Authentication)

Description:

Risk:

Proof of concept:





Solution:

Implement strong authentication checks, like requiring the current password or multi-factor authentication, before allowing a password change to prevent unauthorized access.

Thank you[😊](https://emojipedia.org/smiling-face-with-smiling-eyes)