SECURITY ASSESSMENT

Submitted to: Dojuicer
Pentester Name:
Youssef Fathy Abdul Moez
Mohammed Khaled Abdelraziq
Hadeer Amr Fawzy
Toqa Ayman Gomaa

Date of Testing: 1, Oct 2024
Date of Report Delivery: 20, Oct 2024

[Dojuicer – BlackBox – Web Application Pentest]

Table of Contents

Contents

Table of Contents

Security Engagement Summary

Executive Summary

This web penetration test assessed the security posture of a legacy web application provided by Dojuicer's development team, focusing on identifying vulnerabilities that may compromise the application's confidentiality, integrity, and availability.

Key Findings

Several high-risk vulnerabilities were identified, including:
- Broken Authentication: Weak authentication mechanisms susceptible to brute-force attacks.
- Unrestricted File Upload: Absence of proper validation allowing attackers to upload malicious files.
- Sensitive Data Exposure: Lack of secure storage and transmission protocols for sensitive data.
- SQL Injection: Flawed input validation leading to unauthorized database access.

Overall Risk

The overall risk level is classified as HIGH due to the presence of multiple critical vulnerabilities, posing significant threats to user privacy and organizational data.

Recommendations

To address these vulnerabilities, the following actions are recommended:
- Implement strong access control measures.
- Employ encryption for sensitive data in transit and at rest.
- Secure authentication mechanisms using rate limiting and multi-factor authentication.
- Regularly update and patch the application to prevent exploitation of known vulnerabilities.

This engagement aims to improve the security posture of the application, ensuring compliance with industry standards and minimizing risks.

Scope

In this engagement, the assessment focused on the legacy web application provided as a virtual machine environment, specifically the OWASP Juice Shop. The evaluation aimed to test the application's security posture against modern web security standards by identifying potential vulnerabilities, prioritizing them by risk, and recommending actionable mitigations. This scope includes testing common vulnerabilities such as broken authentication, sensitive data exposure, XSS, and SQL injection, among others.

This vulnerability assessment engagement was requested by the Development Team at Dojuicer, which is responsible for maintaining the organization's web applications. The team seeks a comprehensive security review of a legacy web application to understand the potential risks it poses and to identify possible mitigations that can improve the security posture and reduce risks to the organization.

- **What are the engagement's goals?**

The primary goals of this engagement are:

  - To identify vulnerabilities that could impact the confidentiality, integrity, and availability of the legacy web application.
  - To understand the severity of the identified vulnerabilities and how they could impact the organization.
  - To provide actionable mitigation recommendations to improve the application's security and minimize risks to the organization.

- **Who is completing the engagement?**

The engagement is being conducted by a Security Penetration Testing from Dojuicer's Information Security department, using the OWASP Juice Shop web Application as the target environment for the assessment.

The assessment of the legacy web application is part of a proactive security measure that is undertaken on an ad-hoc basis, as requested by the development team. It is designed to ensure that even legacy systems are evaluated periodically to maintain an acceptable security posture within the organization.

Scope

In high-level terms, the scope of this engagement covers a thorough vulnerability assessment of a legacy web application provided in a virtual machine environment, specifically OWASP Juice Shop. This assessment is designed to evaluate the application against modern web security standards, focusing on identifying vulnerabilities that could be exploited by attackers to compromise the system. Given the critical nature of web applications to the organization's operations, this scope is appropriate to ensure that any underlying security risks, especially in legacy systems, are identified and addressed.

Risk Analysis

The overall risk for the assessed scope is High. This risk level is reported due to the presence of multiple severe vulnerabilities, including critical issues such as Broken Access Control and Unrestricted File Upload Vulnerability, which could allow unauthorized access to sensitive data or system compromise. In addition, Broken Authentication and Sensitive Data Exposure represent high-impact risks that could lead to data breaches or unauthorized system access.

Moderate-risk vulnerabilities like Reflected XSS and DOM-based XSS could be exploited for client-side attacks, posing a risk to users. Lower-risk issues such as improper login configurations (e.g., Login Jim, Login Bender, Login Admin) and inadequate Client-side/Server-side XSS Protection represent weaker control mechanisms but pose less immediate danger. However, when combined, they still contribute to the overall attack surface.

Recommendation

Remediation efforts are warranted, especially for the critical vulnerabilities. The highest-risk vulnerabilities should be prioritized as follows:

- **Broken Access Control** should be addressed first, as it represents the most critical flaw, allowing unauthorized users to gain access to restricted data or functionality. Remediating this will close off access to sensitive areas of the application.
- **Unrestricted File Upload Vulnerability** must be fixed immediately, as it can enable attackers to upload malicious files, potentially leading to complete system compromise. Employing strict file validation checks and restricting allowed file types can mitigate this risk.
- **Broken Authentication** and **Sensitive Data Exposure** should follow next, as they can lead to unauthorized access and data breaches. Implementing strong encryption for sensitive data and securing authentication mechanisms (e.g., multi-factor authentication) will reduce these risks significantly.

After these, medium-risk issues like **XSS vulnerabilities** should be addressed to prevent client-side attacks. Finally, low-risk issues, while not immediately dangerous, should be handled as part of a broader hardening effort to minimize the overall attack surface.

Significant Vulnerability Summary

High Risk Vulnerabilities

- **Broken Authentication:** Weak or missing authentication mechanisms that can result in unauthorized access to sensitive systems or data.
- **Unrestricted File Upload Vulnerability:** Enables attackers to upload malicious files, potentially leading to system compromise.
- **Broken Access Control**: Allows unauthorized users to access restricted data or functionality, posing a critical risk to the system.
- **Sensitive Data Exposure:** Improper handling or lack of encryption for sensitive data, increasing the risk of data breaches
- **Improper Input Validation**: **Improper Input Validation - Poison Null Byte Attack**: A security control bypass vulnerability where the attacker uses a null byte injection to manipulate input and access files that should not be accessible.
- **Security Misconfiguration**: Inadequate settings or defaults leading to vulnerabilities in applications or systems.
- **Cryptographic Issues**: Weak encryption algorithms or improper implementation that jeopardize data confidentiality and integrity.

- **Insecure Key Management**: Poor handling of cryptographic keys, increasing risk of unauthorized access and data breaches.
- **Login Admin-SQL injection**: Administrator-level login vulnerability that, while important, poses lower risk under normal circumstances.
- **Medium Risk Vulnerabilities**
- **DOM-based XSS**: A client-side vulnerability that can allow an attacker to execute arbitrary scripts in the user's browser.
- **Reflected XSS**: Vulnerability enabling attackers to inject malicious scripts into web pages viewed by other users.
- **HTTP-Header XSS:** Like reflected XSS but targeting HTTP headers to inject scripts that could be executed by the client.
- **Server-side XSS Protection:** Inadequate server-side measures to prevent cross-site scripting attacks.
- **Database Schema:** Potential exposure of database structure, which can assist attackers in crafting specific exploits.

Low Risk Vulnerabilities

- **Client-side XSS Protection**: Lack or poor implementation of client-side protection against XSS attacks.
- **Login Jim**: Low-level risk of weak or exposed login credentials.
- **Cross-Origin Resource Sharing (CORS):** is a browser feature that allows controlled access to resources from different domains. It builds on the same-origin policy (SOP) but can lead to cross-domain attacks if not properly configured. However, CORS does not protect against attacks like cross-site request forgery (CSRF).

Significant Vulnerability Detail

Broken Authentication

Risk Level: HIGH

Vulnerability Detail:

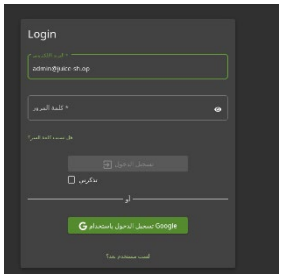
Executive Summary:

During the security assessment of the web application, it was discovered that the admin login page was vulnerable to password guessing attacks due to insufficient rate limiting and account lockout mechanisms. Attackers could leverage this vulnerability to gain unauthorized access to sensitive administrative functions, posing significant risks to the application and its data integrity.

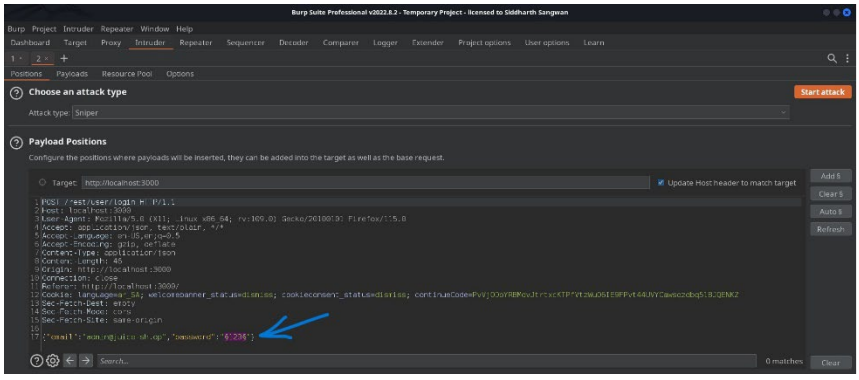
The vulnerability was identified through a combination of manual testing and automated scanning techniques. By utilizing a password dictionary attack against the admin login endpoint, it was demonstrated that the application did not impose restrictions on failed login attempts. The lack of account lockout mechanisms allowed multiple rapid attempts, which facilitated successful login through brute-force methods.

Evidence of Validation:

1. After getting admin email by The Reconnaissance, now we are trying to get his password
2. As we see admin@juice-sh.op



3. Then we will use a Brute Force attack to get the Password

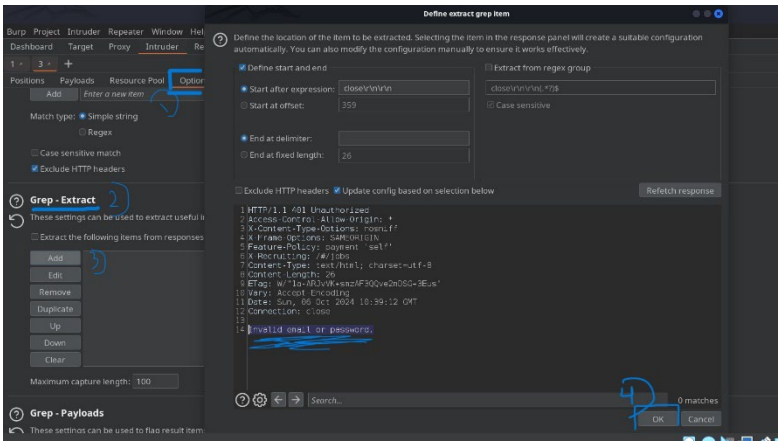


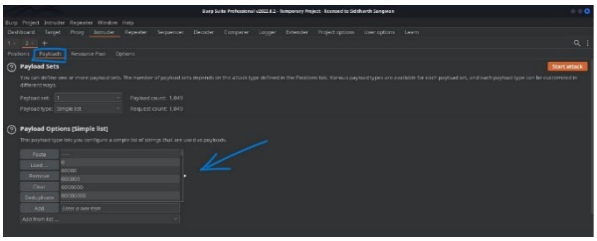
4. Ok... send the request to burp suite intruder

5. Then add the copied SecList password wordlist https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/best1050.txt into payload section in burp suite intruder
6. In order to identify the correct password, will Grep part of the response Then in burp suite intruder Go to option > Grep - Extract >Add >"then chose"
7. Then press on start attack
8. we have a 200 response status code, Then the password is admin123 As we see in below picture
9. Then we have administration email and password

   - Email: admin@juice-sh.op

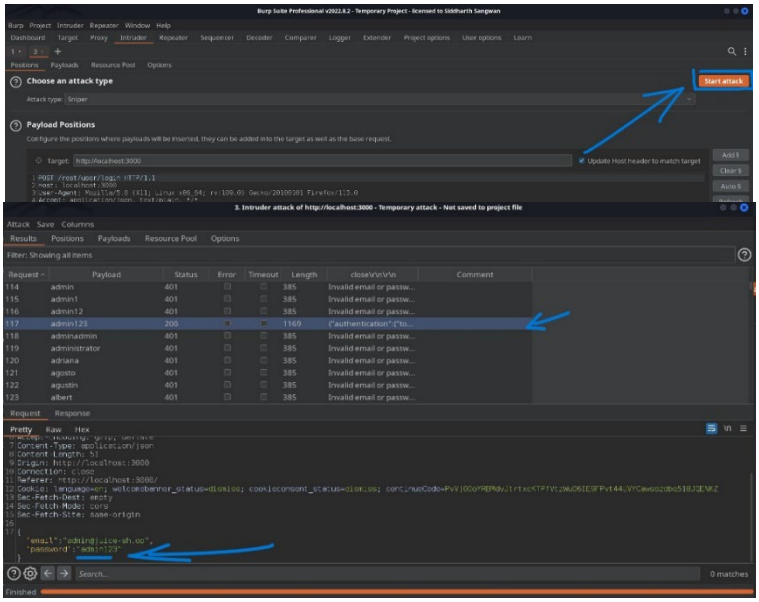
   - Password: admin123





Impact of Exploit:

If an attacker successfully exploits this vulnerability, they could:



- Gain unauthorized access to user accounts, leading to the compromise of sensitive information and data.

- Perform actions on behalf of other users, including accessing personal data, modifying settings, and potentially conducting fraudulent activities.

- Cause significant reputational damage to the organization and expose it to legal ramifications due to data breaches.

Potential Remediation:

To mitigate the risk associated with broken authentication, the following remediation steps are recommended:

Strong Password Policies: Enforce strong password requirements and implement mechanisms to prevent weak passwords during account creation.

Session Management Best Practices: Use secure, unpredictable session tokens, and ensure that tokens are invalidated upon logout or after a specified timeout period.

Rate Limiting and Account Lockout: Implement rate limiting on login attempts and account lockout mechanisms after a specified number of failed attempts to prevent brute-force attacks.

Multi-Factor Authentication (MFA): Encourage or require the use of MFA to add an additional layer of security to user accounts.

User Enumeration Prevention: Implement measures to prevent the enumeration of valid usernames by providing generic error messages for failed login attempts.

Unrestricted File Upload Vulnerability

Risk Level: HIGH

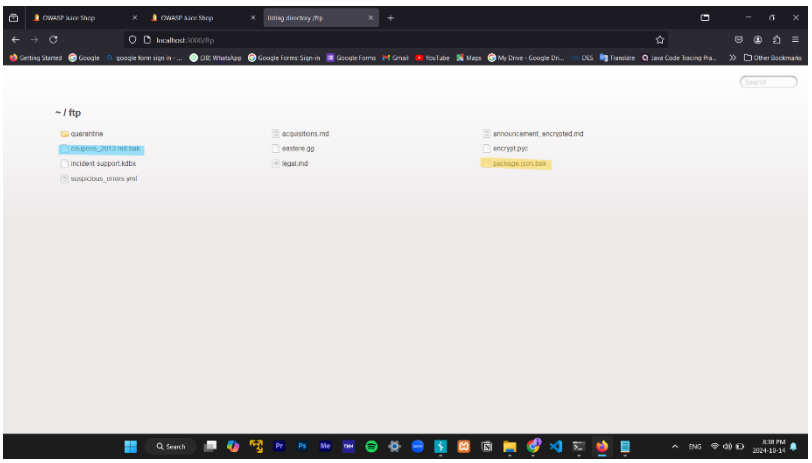
Vulnerability Detail:

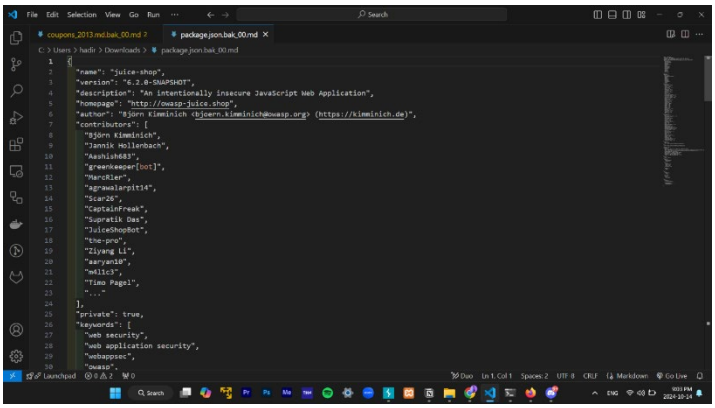
Executive Summary:

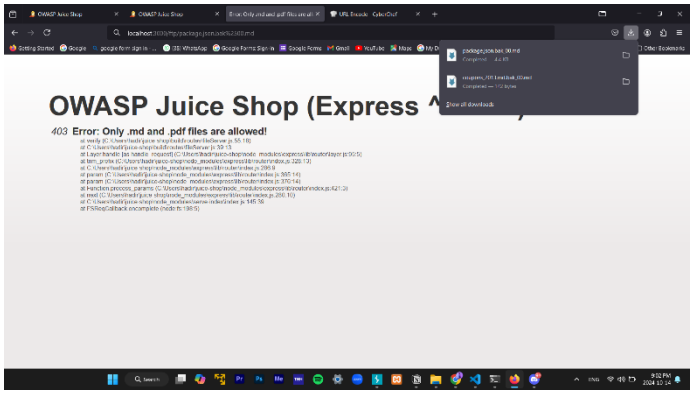
Typosquatting Tier 1: In this challenge, we have to inform the shop about a typosquatting trick that is there within it.

As I got this access at the last challenge this time I tried to download packge .json and then I do the same steps “null byte” then I got the file

Evidence of Validation:

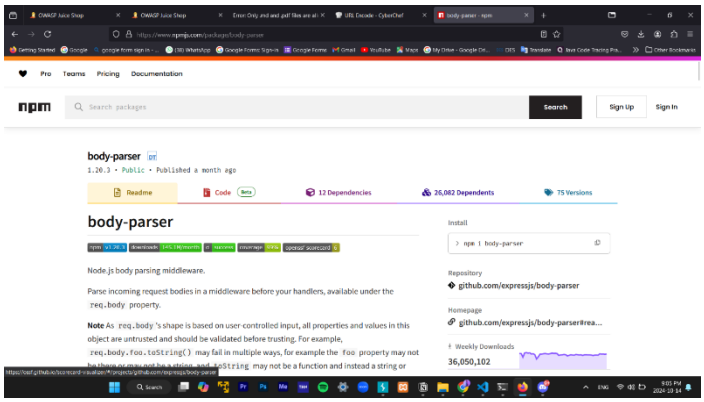


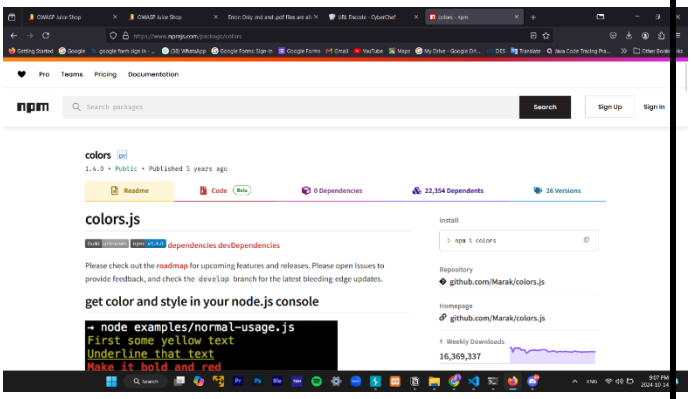




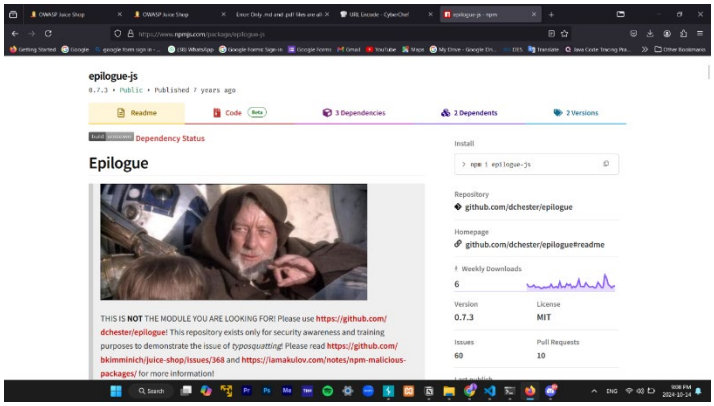
What I gave my attention was dependencies then I try some of the to detect what was the phishing

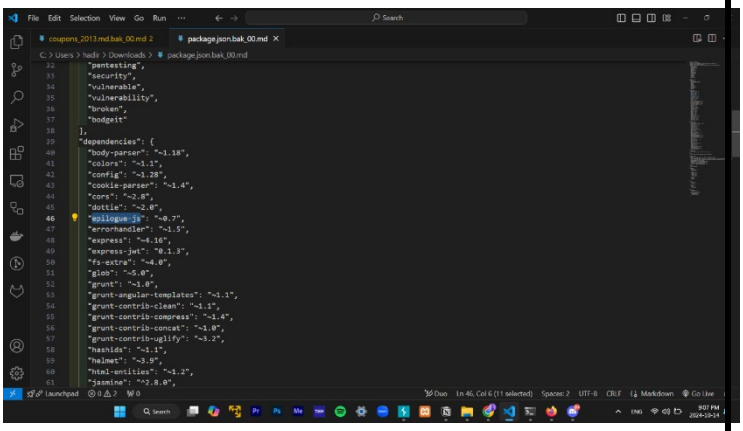
First, I tried body-parser then colors both was not contain abnormal things

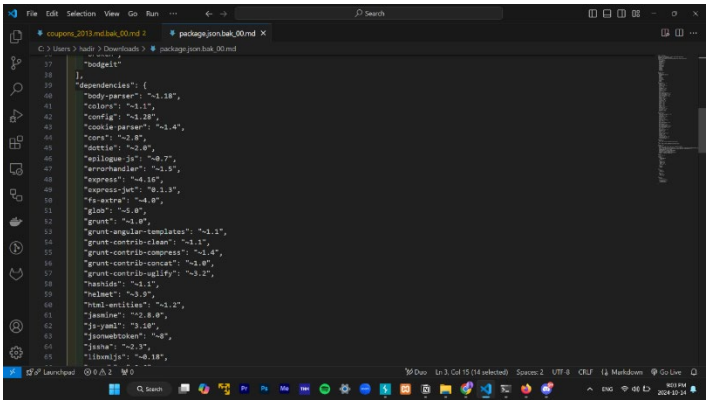


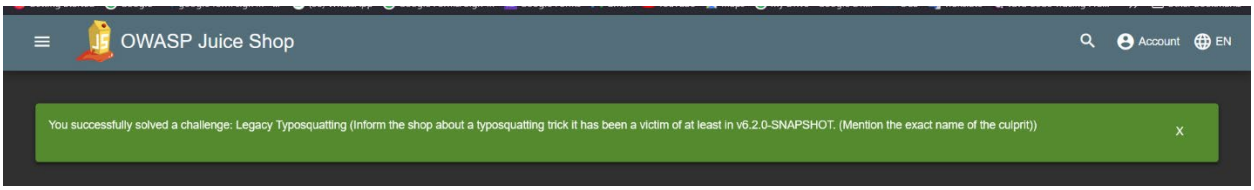


then I decided to visit **https://www.npmjs.com/package/epilogue-js** and finally this what I was trying to reach









Then The challenge has been solved

Impact of Exploit:

If an attacker successfully exploits this vulnerability, they could:

- Cause reputational damage to the organization and result in financial losses from data breaches.

- Use the compromised server to launch further attacks on internal networks or other connected systems.

- Access and manipulate sensitive user data or application databases.

- Upload and execute malicious scripts on the server, potentially gaining control over the server environment.

Potential Remediation:

To mitigate the risk associated with broken authentication, the following remediation steps are recommended:

File Type Validation: Implement strict validation of file types based on both the file extension and the MIME type. Only allow specific file types necessary for application functionality.

Content Inspection: Analyze the contents of uploaded files to ensure they do not contain executable code. Use tools to scan for malicious payloads.

Upload Location: Store uploaded files outside the web root directory to prevent direct access via the web.

File Size Limitations: Set restrictions on the size of uploaded files to prevent denial-of-service (DoS) attacks.

User Permissions: Ensure that only authorized users have access to upload functionality, and apply proper authentication and authorization checks.

Broken Access Control

Risk Level: HIGH

Vulnerability Detail:

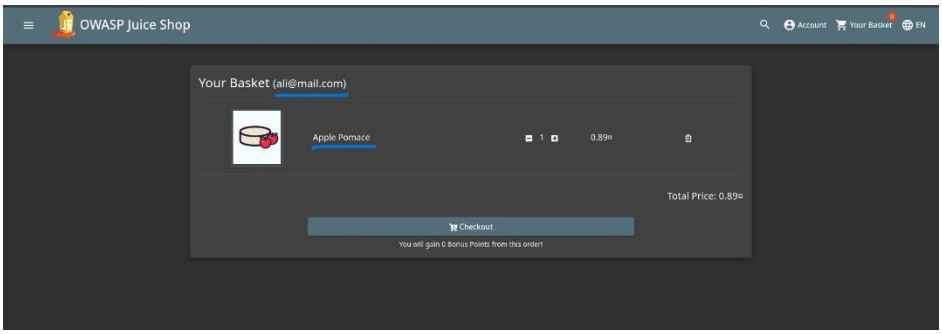
Executive Summary:

During the security assessment of the web application, it was identified that the authentication mechanism was vulnerable to exploitation due to weak session management practices and inadequate security measures. This vulnerability could allow attackers to bypass authentication controls, leading to unauthorized access to user accounts and sensitive application functions.

The vulnerability was identified through a combination of manual testing and automated tools. It was discovered that session tokens were predictable and not properly invalidated upon logout. Additionally, there were no mechanisms in place to prevent enumeration of valid usernames, allowing attackers to easily identify existing accounts.

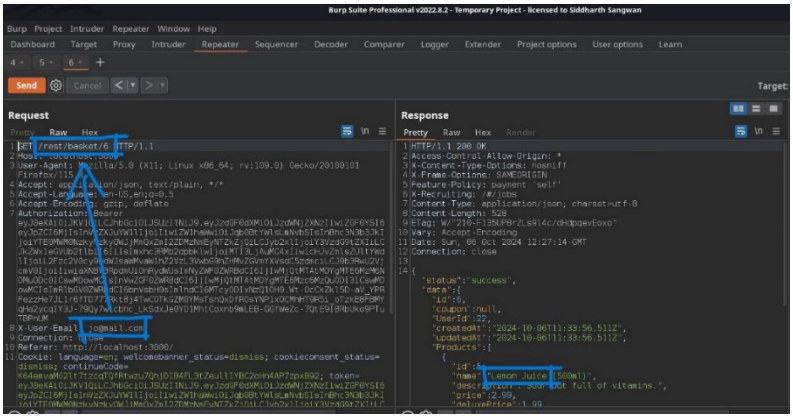
Evidence of Validation:

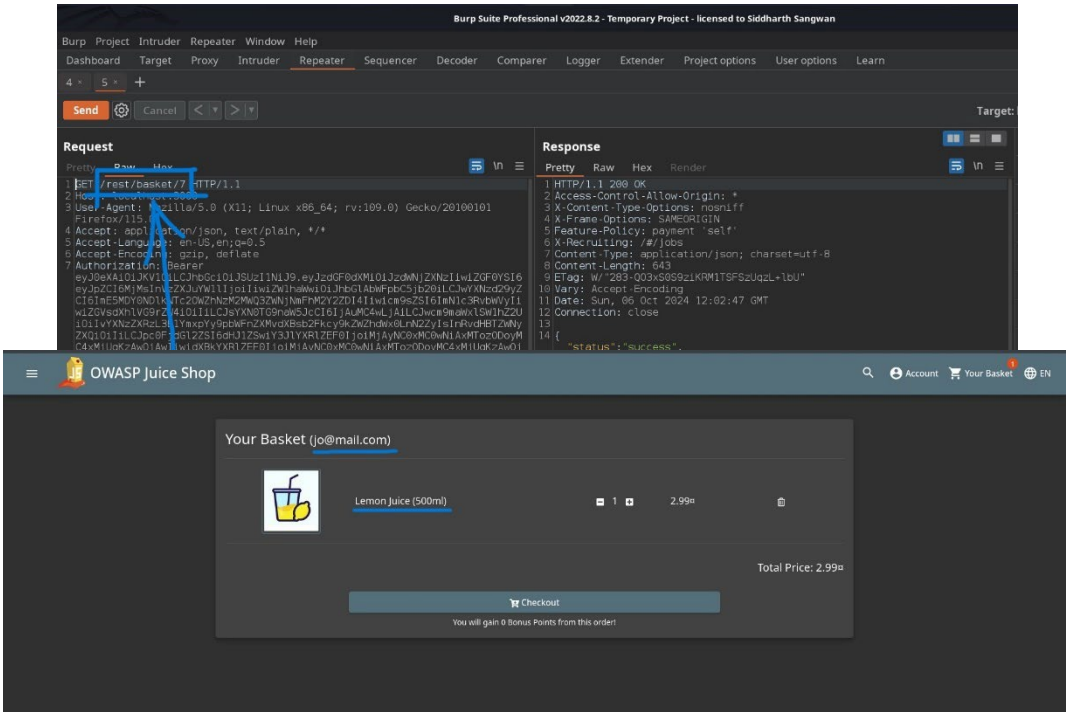
1. We have two account jo@mail.com and ali@mail.com , Each account have his own items Cart



2. For ali@mail.com have an Apple Pomace in his Basket [Cart] as shownWith basket id 7 for ali@mail.com in request after intercept it by burp sutie as shown

3. For jo@mail.com have an Lemon Juice in his Basket [Cart] as shown

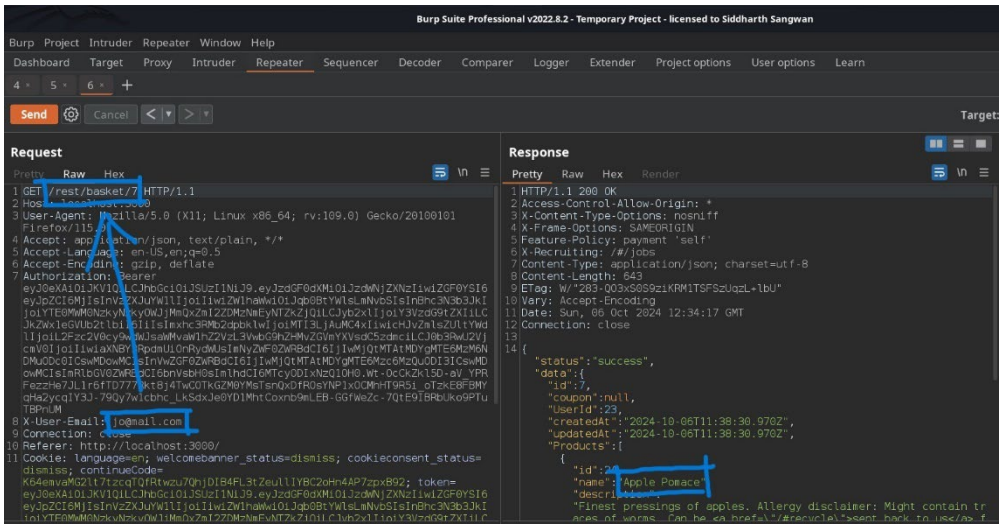




4. With basket id 6 for jo@mail.com in request after intercept it by burp sutie as shown
5. So… we will try to manipulate the basket id 6 to 7
6. as we see in response > it returned Apple Pomace in Basket, that owned by ali@mail.com but jo@mail.com could see it
7. Given that jo@mail.com unauthorized to see this basket > Then it's a Broken Access Control Vulnerabil

Impact of Exploit:

If an attacker successfully exploits this vulnerability, they could:



- Expose the organization to legal liabilities and regulatory penalties due to violations of data protection laws.

- Compromise user privacy and trust, resulting in reputational damage for the organization.

- Gain unauthorized access to sensitive information of other users, such as personal details, email addresses, and phone numbers, potentially leading to identity theft or harassment.

Potential Remediation:

To mitigate the risk associated with broken authentication, the following remediation steps are recommended:

Implement Proper Access Controls: Enforce strict authorization checks for all API endpoints and ensure that users can only access their own data.

Role-Based Access Control (RBAC): Utilize RBAC to define roles and permissions, ensuring that users have the appropriate level of access based on their roles within the application.

Input Validation: Validate user inputs to ensure that requests for user data are properly authenticated and authorized before processing.

Security Testing: Regularly conduct security testing, including penetration testing, to identify and address access control vulnerabilities.

Logging and Monitoring: Implement logging and monitoring of access attempts to detect and respond to unauthorized access patterns promptly.

Sensitive Data Exposure

Risk Level: HIGH

Vulnerability Detail:

Executive Summary:

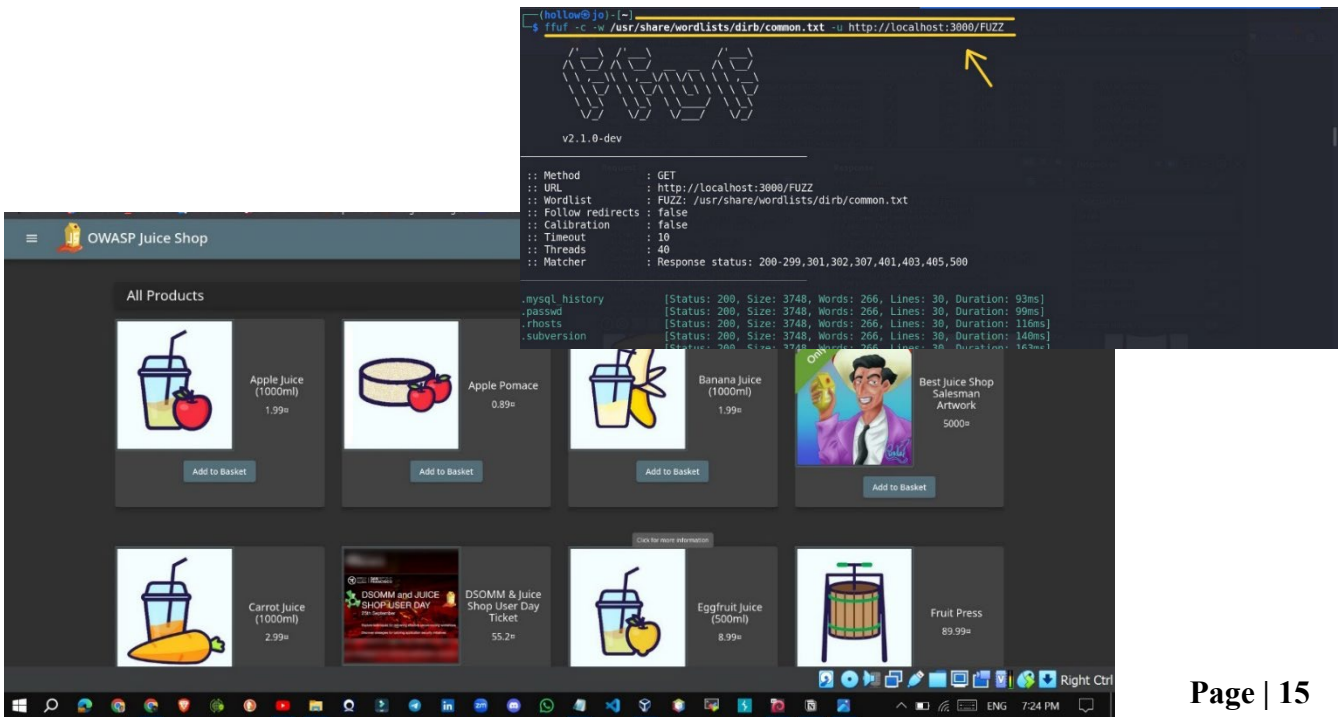
During the security assessment of the web application, it was identified that sensitive data, including sensitive information about access log files, was not adequately protected during storage and transmission. This vulnerability could allow attackers to intercept or access sensitive information, leading to data breaches, identity theft, and significant reputational damage.

The vulnerability was identified through manual testing and directory enumeration techniques. It was discovered that access log files were stored in a publicly accessible directory without any access controls, enabling anyone with knowledge of the file location to view the contents.

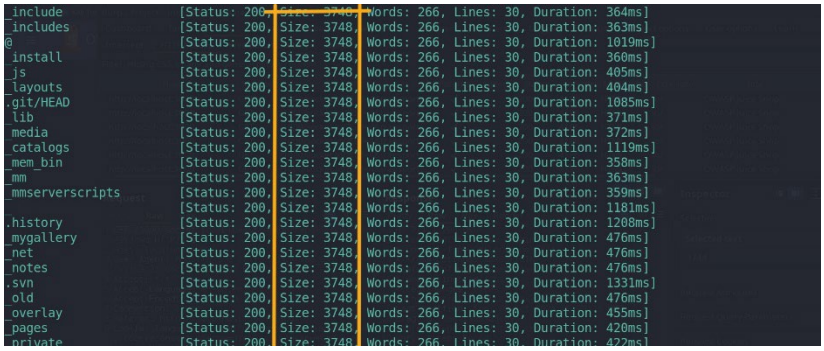
Evidence of Validation:

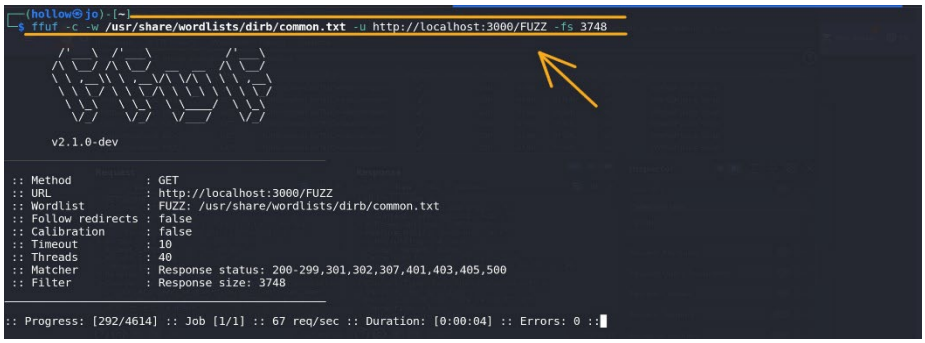
1. We will try to find and sensitive information in OWSP juice shop site

2. First, we will fuzz the directories using FFUF tool with this command ffuf -c -w /usr/share/wordlists/dirb/common.txt -u http://localhost:3000/FUZZ



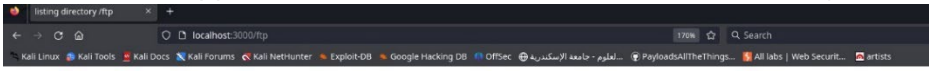
3. As we see there is a lot of directory and we noticed that it have the same size=3748, Because of the

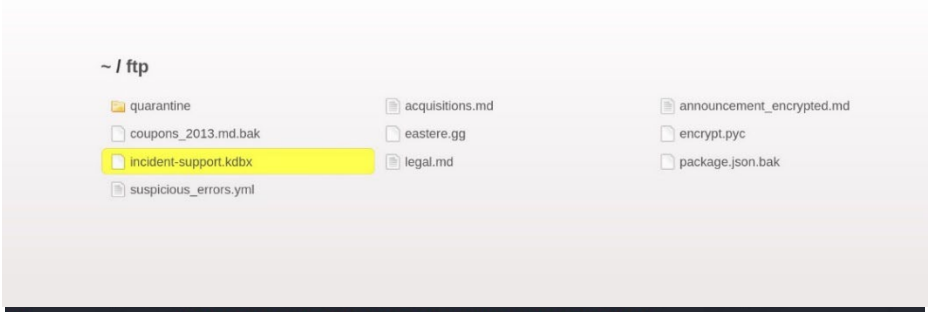


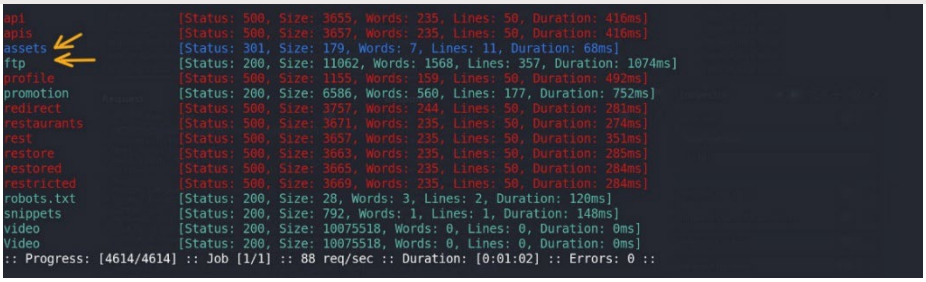


redirection to home page

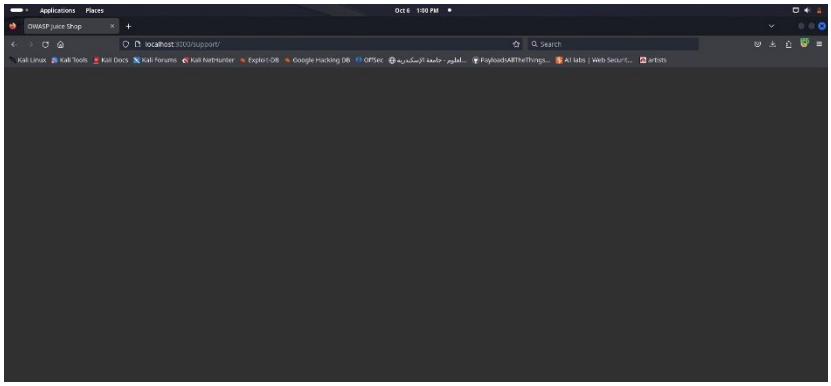
4. So we will Filter HTTP response size by option -fs then add the size need to filter as shown
5. As we see we find a some interested like ftp & asset
6. Let's figure out ftp directory, we found some files maybe important
7. But the file incident-support.kdbx make us wondered if there is a directory named by support



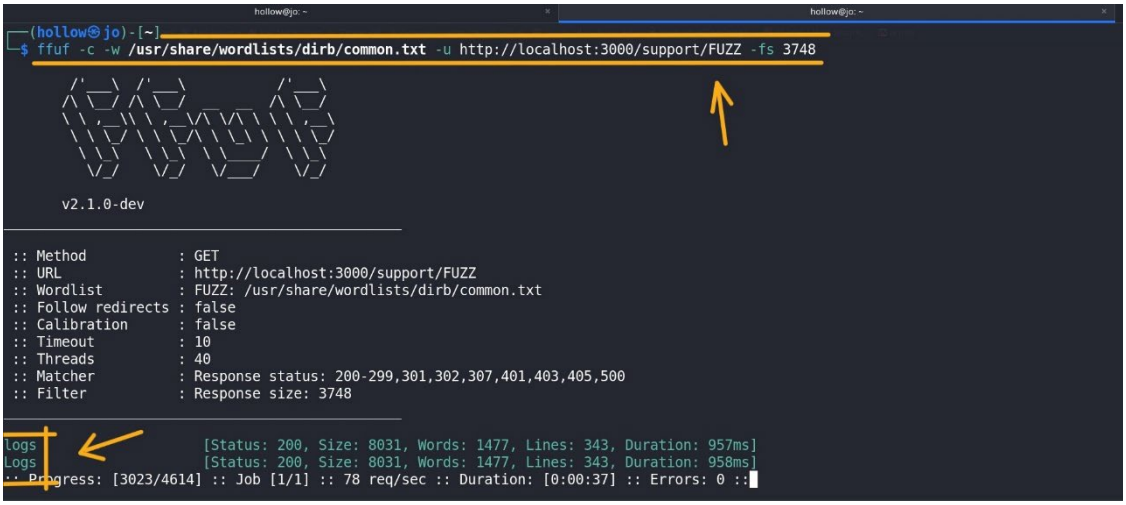


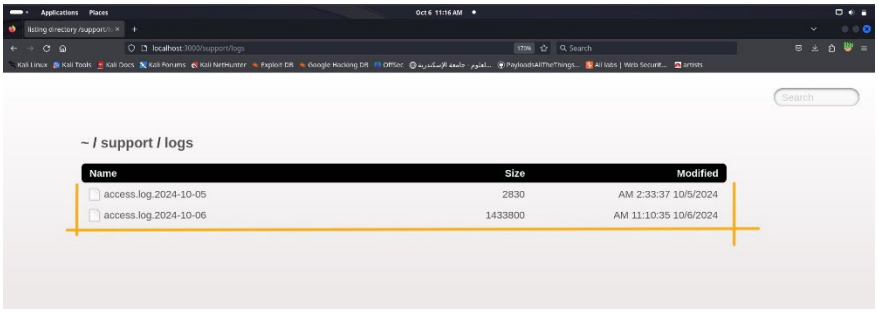


8. As we thought we found a directory named by support, but as we see it's blank page



9. So we think to make directories fuzzing again as shown,... we found a directories named by logs





10. Let's Figure out this directory as shown, then We found the access log files

Impact of Exploit:

If an attacker successfully exploits this vulnerability, they could:

- Gain access to sensitive user credentials and session tokens, leading to unauthorized access to user accounts and applications.

-Compromise user privacy by exposing personal data, potentially resulting in identity theft or financial fraud.

-Lead to regulatory non-compliance and associated penalties due to mishandling of sensitive information, affecting the organization's reputation and customer trust.

Potential Remediation:

To mitigate the risk associated with sensitive data exposure via access log files, the following remediation steps are recommended:

Access Control Implementation: Restrict access to log files to only authorized personnel by using proper authentication and authorization mechanisms.

Log File Redaction: Ensure sensitive information, such as passwords and session tokens, are redacted or not logged in plain text.

Secure Storage: Move log files to a secure directory outside of the web root and apply file permissions to restrict unauthorized access.

Regular Log Review: Conduct regular reviews and audits of log files to ensure that sensitive information is not being captured or stored inappropriately.

Incident Response Planning: Develop and implement an incident response plan to quickly address any breaches involving sensitive data exposure.

Improper Input Validation

Risk Level: HIGH

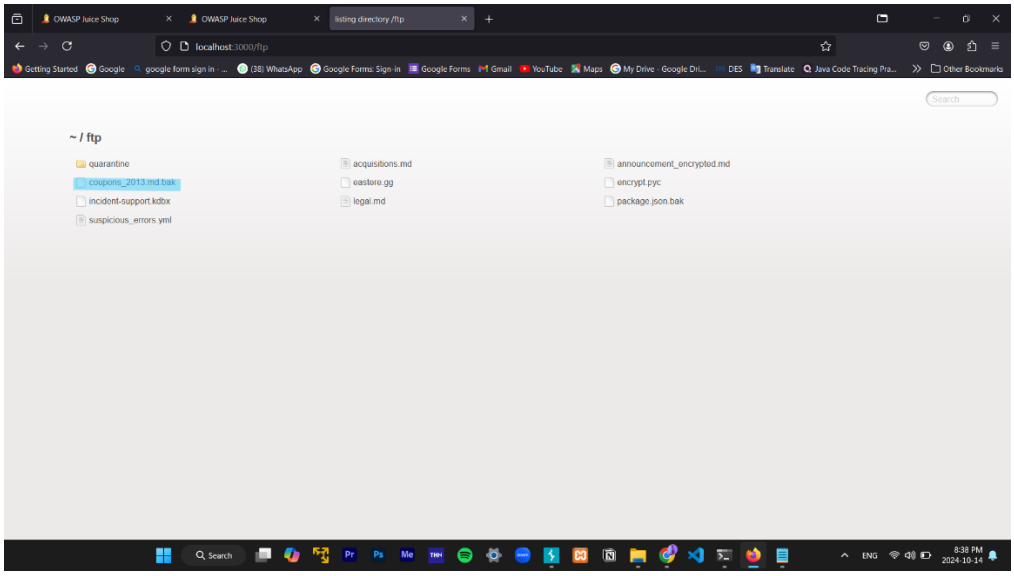
Vulnerability Detail:

Executive Summary:

This vulnerability arises due to insufficient input validation, allowing attackers to inject malicious characters, such as a null byte, to bypass security controls. By exploiting this weakness, attackers can manipulate file paths or requests, potentially gaining unauthorized access to sensitive files or resources that are not intended for public viewing. This bypass technique can lead to critical data exposure or further system compromise. we have to access a developer's forgotten backup file.

Evidence of Validation:

I have tried to access ftp file in url and I have entered



Security misconfiguration

Risk Level: HIGH

Vulnerability Detail:

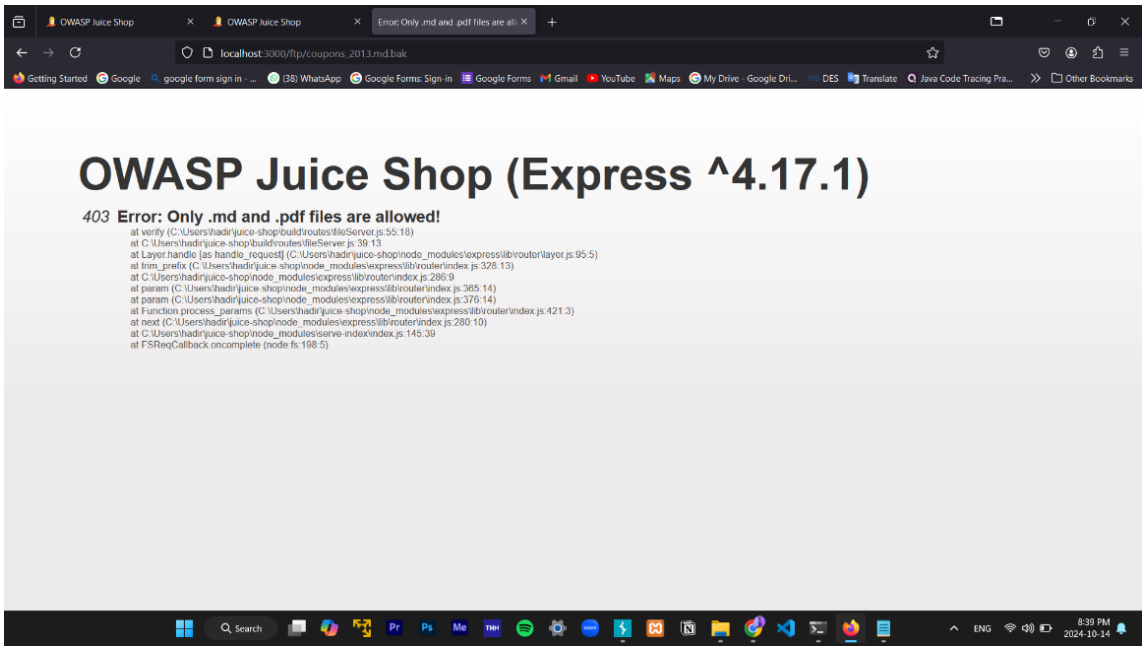
Executive Summary:

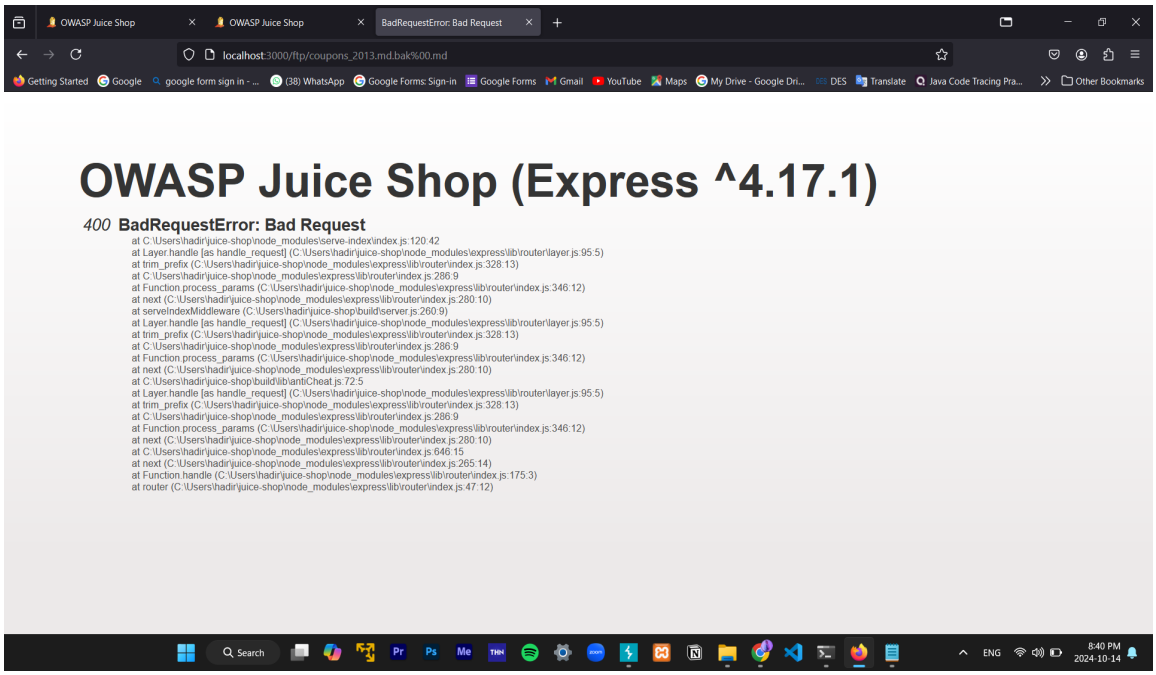
During the security assessment of the Juice Shop application, it was discovered that the discount coupon functionality is vulnerable to reverse engineering. This vulnerability allows an attacker to forge valid discount coupons by decoding the coupon encoding algorithm, leading to potential financial losses for the business.

The vulnerability was identified through the analysis of exposed files and encoded coupon strings. By utilizing publicly available tools, attackers can reverse the encoding process and generate their own discount coupons.

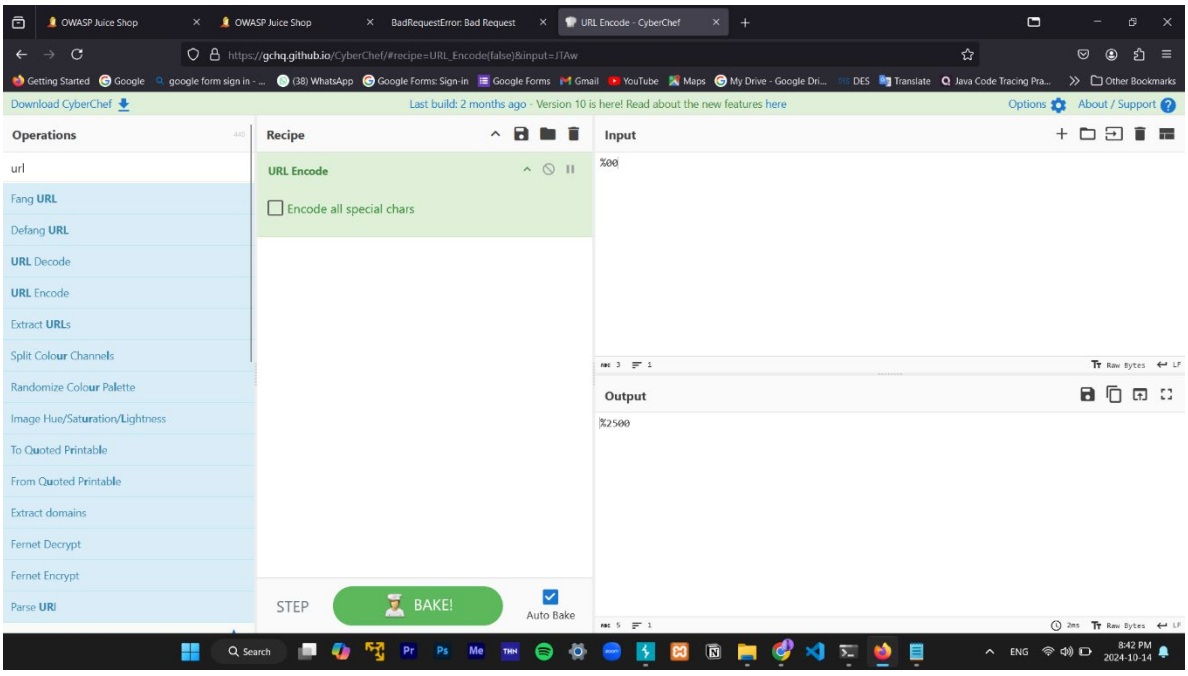
Evidence of Validation:

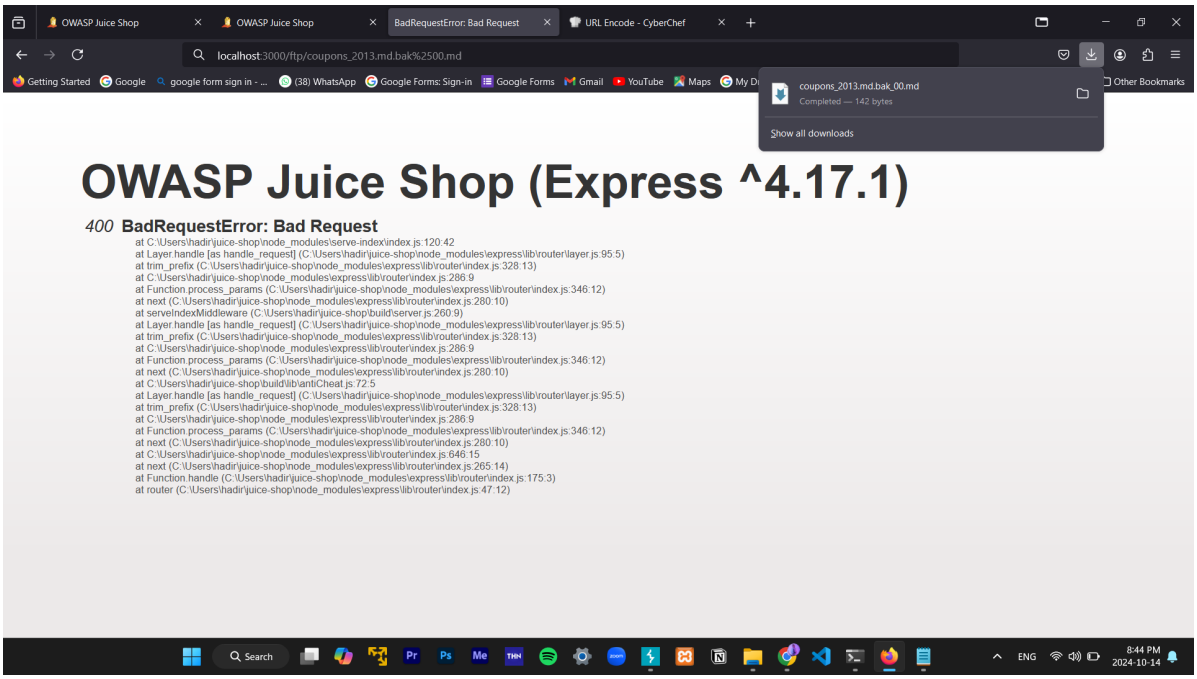
Accessing the Coupon File:
Navigate to the followingURL:http://localhost:3000/ftp/coupons_2013.md.bak%2500.md
Then I have found forgotten Sale's man Backup file which is **coupons_2013.md.bak** but I could not downloaded it



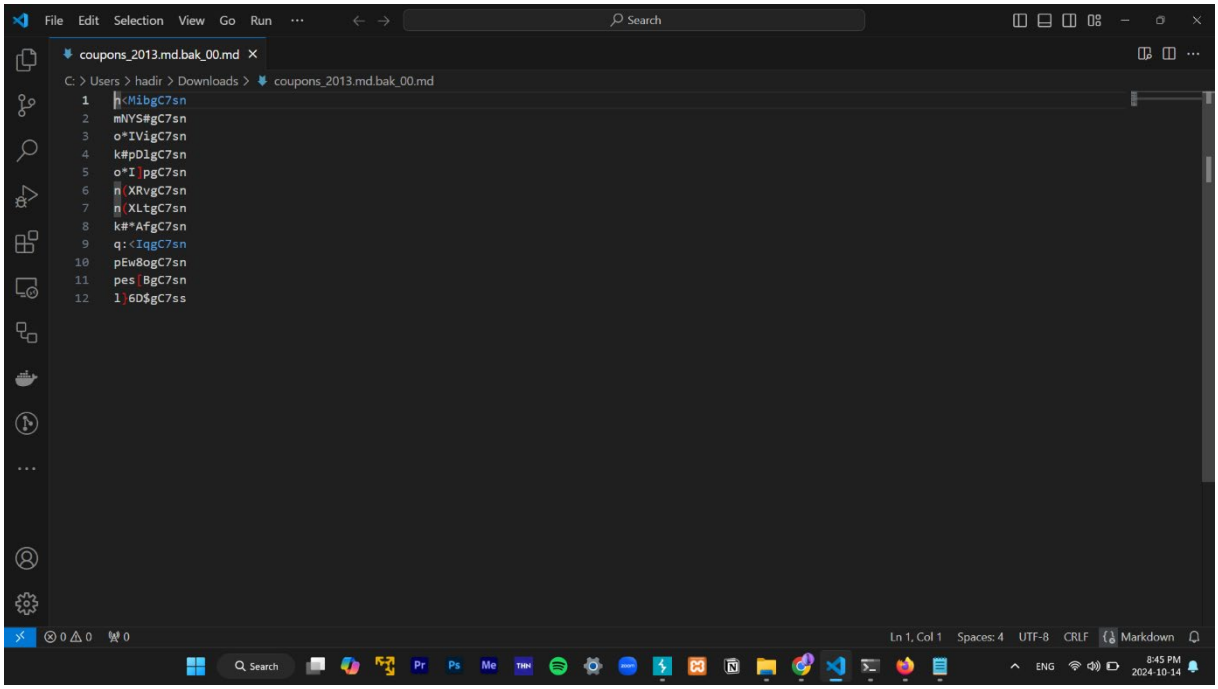


Then I have tried null byte %00 in the url but also does not work so I went for cyber chef and tried url encode for the null byte





Then it works and I have downloaded and accessed the file content



Impact:

Vulnerability: Replace z85 with amore secure encoding or encryption algorithm to prevent reverse engineering.

Exploitability: Publicly available tools like CyberChef can easily decode Base85-encoded values, making this vulnerability.

Potential Remediation:

Stronger Encoding: Replace Z85 with more secure encoding or encryption algorithm to prevent reverse engineering.

Access Control: Sensitive data such aspackage.jsonshould not be publicly accessible, proper access measures should be implemented.

Cryptographic Issues

Risk Level: HIGH

Vulnerability Detail:

Executive Summary:

During the security assessment of the Juice Shop application, it was discovered that the functionality related to ContinueCodesis vulnerable due to the use of default values in cryptographic tools. This vulnerability allows an attacker to predict and potentially generate valid ContinueCodesby leveraging the default configuration of the hashidlibrary, compromising challenge completion mechanisms.

The vulnerability was identified through code analysis of the package.jsonfile and understanding the dependency on the hashidlibrary. Attackers can exploit this weakness to generate their own ContinueCodesand bypass the intended challenge validation.

Evidence of Validation:

1. Accessing the package.jsonFile:
    a. Navigate to:http://localhost:3000/ftp/package.json.bak%2500.md
    b. This file reveals the use of the hashidlibrary in the Juice Shop dependencies.
2. **Hashid Vulnerability**:

The hashidlibrary uses default salts to generate encoded values. This makes it easy for attackers to generate valid ContinueCodesby encoding numbers (e.g., **999**) using hashid.

3. **Applying the ContinueCode**: After generating a valid ContinueCode, an attacker can make a PUT request to apply the code through:http://localhost:3000/rest/continuecode/apply/{ContinueCode}

4. **Potential Attack Example**:

Use default encoding for the number **999**with hashidto generate a valid ContinueCode.

Apply this code in the application, bypassing the validation mechanism.

Impact:

· **Vulnerability**: The use of default cryptographic settings allows attackers to generate predictable codes, leading to a compromise in challenge validation.

· **Exploitability**: Attackers can easily exploit this by using public tools to generate valid codes and bypass the system's challenge mechanisms.

Potential Remediation:

· **Avoid Default Cryptographic Values**: Use custom salts or other cryptographic techniques to secure challenge mechanisms and prevent predictable code generation.

· **Access Control**: Sensitive configuration files like package.jsonshould not be publicly accessible, and proper access controls should be implemented to protect them.

Insecure Key Management

Risk Level: HIGH

Vulnerability Detail:

Executive Summary:

During the security assessment of the Juice Shop application, it was discovered that premium content, meant to be accessible only through payments, could be unlocked by retrieving an exposed encryption key. This allowed the attacker to bypass the paywall and access restricted content without any form of payment.

The vulnerability was identified through directory brute-forcing and exploiting weak encryption key management practices. Using the found key, the attacker was able to decrypt an encoded URL leading to the premium content.

Evidence of Validation:

1. **Locating the Encryption Key**:A directory called encryptionkeyswas discovered at:http://127.0.0.1:3000/encryptionkeys.Inside, the file premium.keywas found, containing a hexadecimal string:
    a. 1337133713371337.EA99A61D92D2955B1E9285B55BF2AD42

2. **Decrypting the Content**:

The hexadecimal key was used to decrypt an encrypted comment found in the page's source code. The comment contained a Base64-encoded string

3. **Decrypting the URL**:

Using the premium.keywith OpenSSL, the Base64-encoded string was decrypted, revealing the hidden premium content URL:<!--
IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeqYfxq+OJVVa0gNbqg
YkUNvn//UbE7e95C+6e+7GtdpqJ8mqm4WcPvUGIUxmGLTTAC2+G9UuFCD1DUjg==-->

http://localhost:3000/this/page/is/hidden/behind/an/incredibly/high/paywall/that/could/only/be/unlocked/by/sending/1btc/to/us

4. **Accessing Premium Content**:

Navigating to the decrypted URL bypassed the paywall, granting full access to the premium content.

Impact:

Vulnerability: Poor encryption key management led to exposure of sensitive premium content that could easily be unlocked using the found key.

Exploitability: Publicly accessible directories containing encryption keys pose a severe risk, as attackers can retrieve and use the key to decrypt protected content.

Potential Remediation:

Secure Key Management: Store encryption keys in secure, inaccessible locations outside of public directories. Use environment variables or secured key vaults to protect sensitive keys.

Access Control: Ensure premium content is only accessible after strict authentication and authorization checks. Avoid exposing. sensitive information or encryption keys client-side.

Login Admin-SQL injection

Risk Level: HIGH

Vulnerability Detail:

Executive Summary:

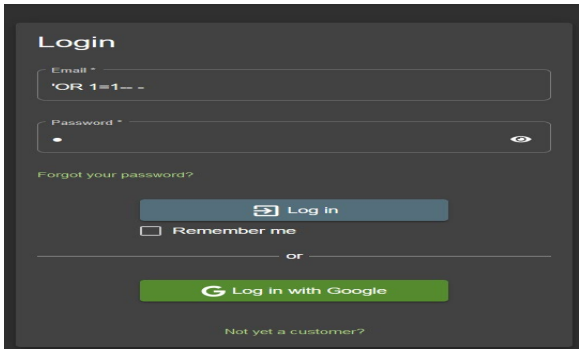
Log in with the administrator's user account During the security assessment , it was discovered that the admin login functionality is vulnerable to SQL injection attacks. This vulnerability allows an attacker to manipulate SQL queries, potentially bypassing authentication and gaining unauthorized access to the admin panel, which could lead to data breaches, data manipulation, and other serious security incidents.

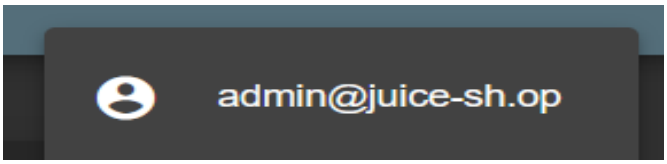
The vulnerability was identified through manual testing, where input parameters for the admin login form were tested with SQL injection payloads. It was confirmed that the application did not properly sanitize user inputs, allowing an attacker to craft a malicious input to manipulate SQL queries.

Evidence of Validation:

Navigate to the login page by clicking on the "log in" In the username field, enter " ’ OR 1 = 1 -- " and in the password field, enter any password.

This vulnerability can be exploited by an attacker to gain unauthorized access to the administrator account and potentially gain access to sensitive information or perform malicious actions within the application.





As we show we could able to sign in by admin account by SQL injection attack

DOM-based XSS

Risk Level: MEDIUM

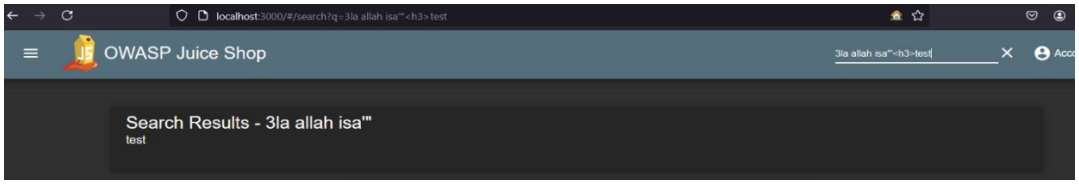
Vulnerability Detail:

Executive Summary:

This vulnerability allows an attacker to send malicious payloads in the search-bar of the website. This can lead to a website defacement.

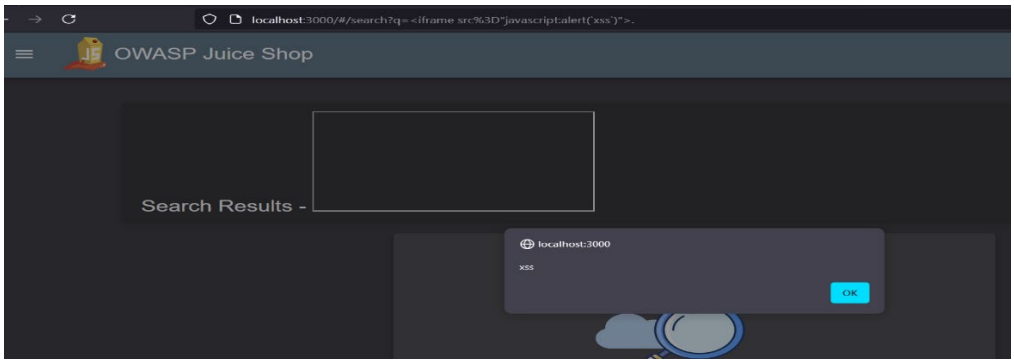
going to use the search input field. When we search for something, we can see the search term appears at top of the page as search results. Now we are going to input below HTML code as a search input to see if we can inject HTML into the web page.

3la allah isa’”< h3>test



Next, let's try the below code

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



Impact of Exploit:

If an attacker successfully exploits this vulnerability, they could:

- steal sensitive information like passwords or alert(document.cookie) to steal client's session cookie .

- Attackers can make unauthorized changes, like purchases or account settings.

- Session Hijacking Attackers could impersonate users by stealing their session data.

Potential Remediation:

To mitigate the risk associated with XSS, the following remediation steps are recommended:

- Validate and Sanitize User Input Always check and clean user input to prevent harmful code from being injected.

- Use Content Security Policies (CSP) Implement CSP to control which sources can load resources on your webpage, helping to block XSS attacks.

- Escape Special Characters When displaying user input, escape characters like <, >, and & to prevent code injection.

- Use a Library or Framework Choose a library or framework that automatically escapes user input to protect against XSS.

- Regularly Test for Vulnerabilities Frequently test your application for security issues, including XSS, to keep it safe.

Reflected XSS

Risk Level: MEDIUM

Vulnerability Detail:

Executive Summary:

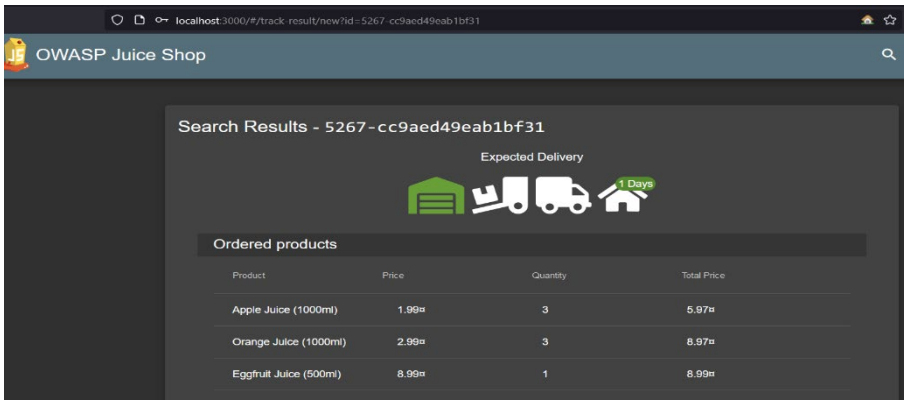
occurs when untrusted data is immediately reflected back to the user without proper validation or sanitization. The user-provided input is included in the URL, and when executed, it allows attackers to inject malicious scripts into the browser.

I started by testing a basic XSS payload in the search bar of the application:

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

While this payload executed successfully, it did not meet the challenge requirements as it wasn't reflected on the page.

To find a place where the input is reflected directly on the page, I navigated through the application and identified that the order tracking page uses an id parameter in the URL to display the order details.

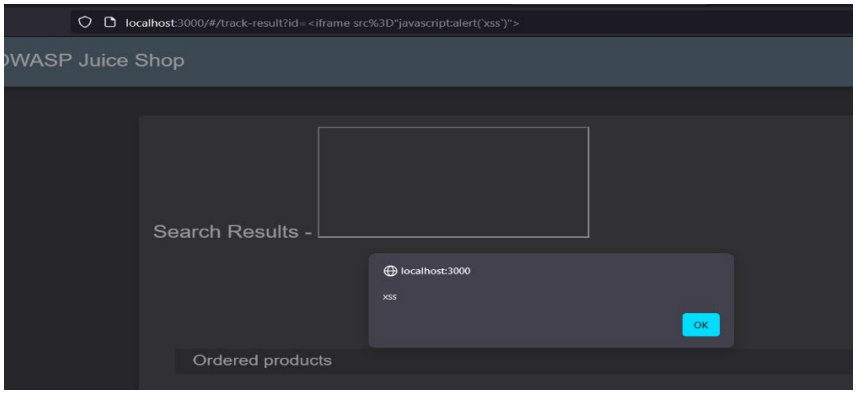


Next, let's try the below code in URL

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

Final payload in URL

localhost:3000/#/track-result?id=<iframe src="javascript:alert('xss')">



You successfully solved a challenge: Reflected XSS (Perform a reflected XSS attack with <iframe src="javascript:alert(`xss`)">.)

Impact of exploit:

If an attacker successfully exploits this vulnerability, they could:

- Steal sensitive information like passwords or alert(document.cookie) to steal client's session cookie .

- Users may be tricked into entering personal information on fake forms.

- Scripts can redirect users to harmful sites or initiate downloads.

- Session Hijacking Attackers could impersonate users by stealing their session data.

Potential Remediation:

To mitigate the risk associated with XSS, the following remediation steps are recommended:

- Validate and Sanitize User Input Always check and clean user input to prevent harmful code from being injected.

- Use Content Security Policies (CSP) Implement CSP to control which sources can load resources on your webpage, helping to block XSS attacks.

- Escape Special Characters When displaying user input, escape characters like <, >, and & to prevent code injection.

- Use a Library or Framework Choose a library or framework that automatically escapes user input to protect against XSS.

- Regularly Test for Vulnerabilities Frequently test your application for security issues, including XSS, to keep it safe.

HTTP-Header XSS

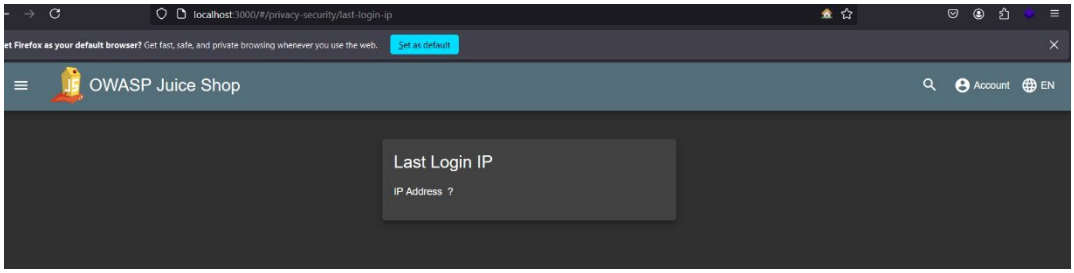
Risk Level: MEDIUM

Vulnerability Detail:

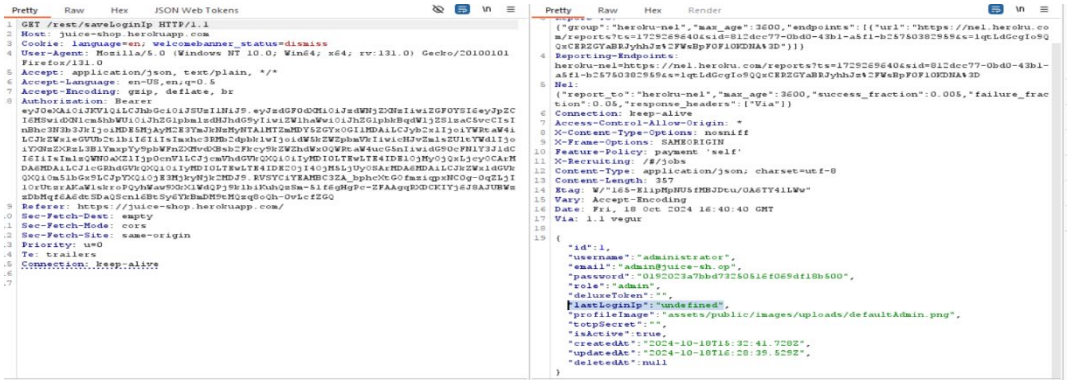
Executive Summary:

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

The challenge involves injecting an XSS payload through an HTTP header. We believe that the IP address logging feature in the admin panel could be a potential target, as it's one of the few inputs we can send via a header and is clearly displayed somewhere.



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

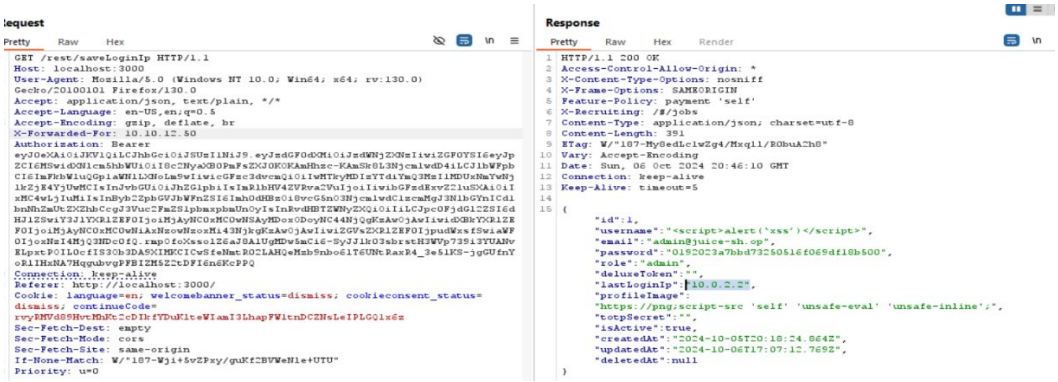


go to search about spoofing client Ip

Remediation: Spoofable client IP address

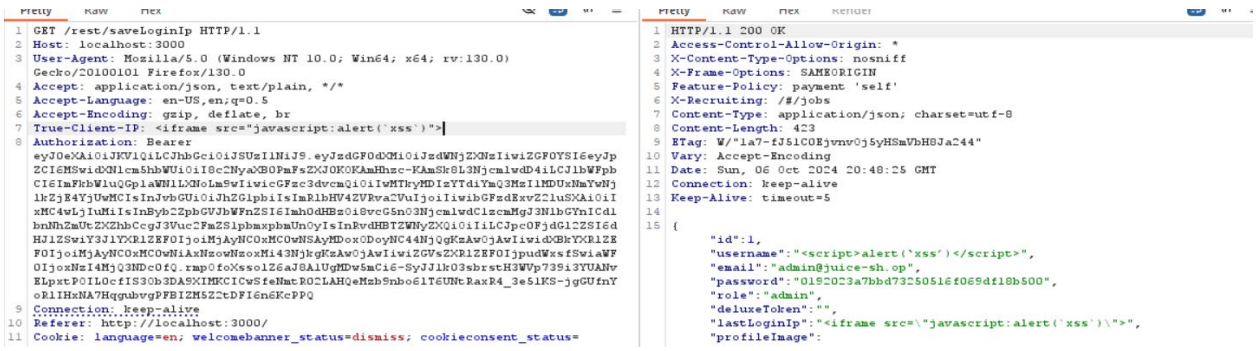
HTTP request headers such as X-Forwarded-For, True-Client-IP, and X-Real-IP are not a robust foundation on which

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



The next let’s Use the True-Client-IP header

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



After sending the modified request, the challenge will be marked as completed.

Impact of exploit:

If an attacker successfully exploits this vulnerability, they could:

- Steal sensitive information like passwords or alert(document.cookie) to steal client's session cookie .

- Users may be tricked into entering personal information on fake forms.

- Scripts can redirect users to harmful sites or initiate downloads.

- Session Hijacking Attackers could impersonate users by stealing their session data.

Potential Remediation:

To mitigate the risk associated with XSS, the following remediation steps are recommended:

- Validate and Sanitize User Input Always check and clean user input to prevent harmful code from being injected.

- Use Content Security Policies (CSP) Implement CSP to control which sources can load resources on your webpage, helping to block XSS attacks.

- Escape Special Characters When displaying user input, escape characters like <, >, and & to prevent code injection.

- Use a Library or Framework Choose a library or framework that automatically escapes user input to protect against XSS.

- **Regularly Test for Vulnerabilities** Frequently test your application for security issues, including XSS, to keep it safe.

Server-side XSS Protection

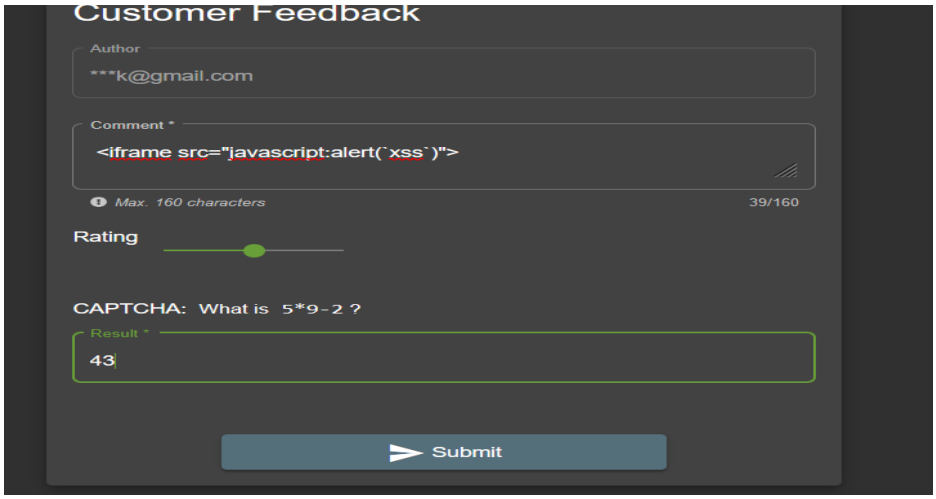
Risk Level: MEDIUM

Vulnerability Detail:

Executive Summary:

In this challenge, you will encounter a scenario where the Juice Shop application has implemented server-side protections to prevent XSS attacks. Your objective is to identify and understand these protective measures while attempting to bypass them.

The challenge involves finding an input field that reflects user input on the server-side.so let's go to the **Customer Feedback** and inject the payload in comment



Submit the feedback and login with admin account and access administrative functionalities,when I capture the request it's not work.

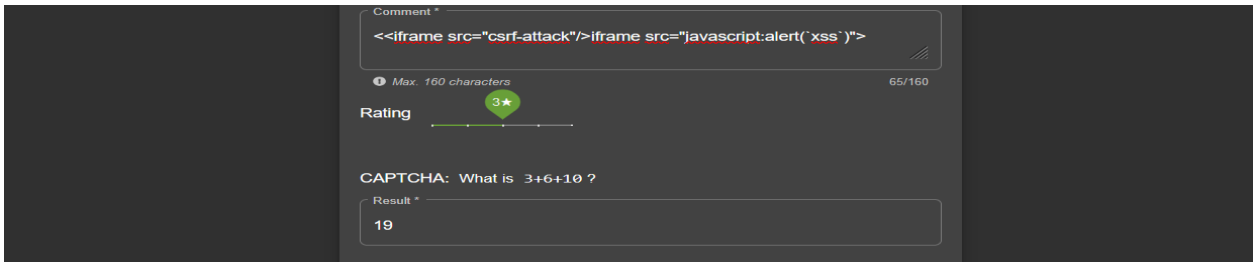
Understanding Server-side Sanitization

The package.json file on the server, accessible at http://localhost :3000/ftp/package.json, reveals the use of the “sanitize-html”library version 1.4.2

```
┌──(kali㉿kali)-[~/Downloads/juice-shop_17.1.1]
└─$ cat package.json|grep -i sanitize
    "sanitize-filename": "^1.6.3",
    "sanitize-html": "1.4.2",
    "@types/sanitize-html": "^1.27.0",
```

Researching the "sanitize-html" library, we found a vulnerability in GitHub
https://github.com/apostrophecms/sanitize-html/issues/29

<<iframe src="csrf-attack"/>iframe src="javascript:alert(`xss`)">



Logged into the application as an admin to access administrative functionalities.

The challenge has been solved

Impact of exploit:

If an attacker successfully exploits this vulnerability, they could:

- Steal sensitive information like passwords or alert(document.cookie) to steal client's session cookie .

- Users may be tricked into entering personal information on fake forms.

- Scripts can redirect users to harmful sites or initiate downloads.

- Session Hijacking Attackers could impersonate users by stealing their session data.

Potential Remediation:

To mitigate the risk associated with XSS, the following remediation steps are recommended:

- Validate and Sanitize User Input Always check and clean user input to prevent harmful code from being injected.

- Use Content Security Policies (CSP) Implement CSP to control which sources can load resources on your webpage, helping to block XSS attacks.

- Escape Special Characters When displaying user input, escape characters like <, >, and & to prevent code injection.

- Use a Library or Framework Choose a library or framework that automatically escapes user input to protect against XSS

- **Regularly Test for Vulnerabilities** Frequently test your application for security issues, including XSS, to keep it safe.

Database Schema

Risk Level: MEDIUM

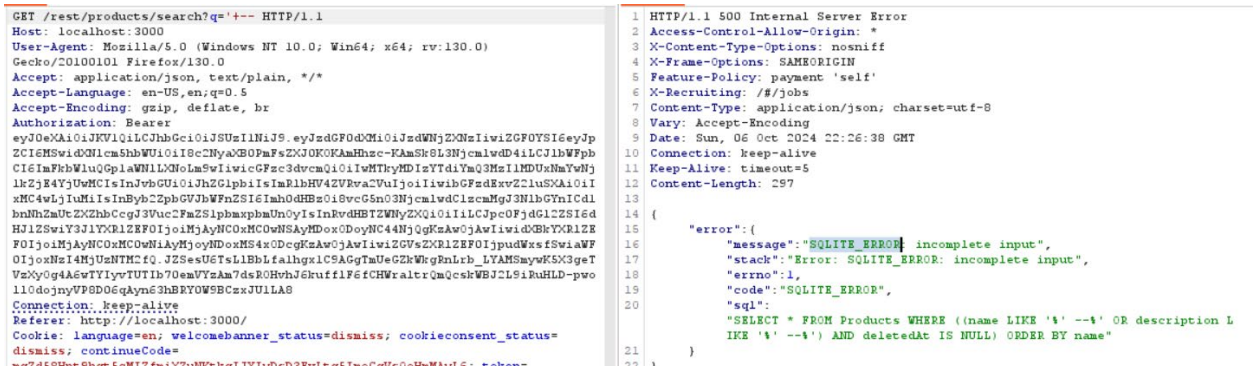
Vulnerability Detail:

Executive Summary:

The Database Schema challenge reveals a SQL Injection vulnerability that allows attackers to obtain details about the database structure. By altering SQL queries, they can list tables, underscoring the importance of secure coding practices and strong input validation to safeguard against unauthorized access to sensitive information.

From previous challenges (**DOM XSS**) I knew that there was no input sanitization on the “Search” field, so I decided to start there.

The provided query is '+--, which results in the following error:



The error message reveals that the database name is **SQLite.**

Utilize the SQL UNION operator to inject a query that will reveal the database schema. Because the product search results are displayed in a structured format, and it is known that product entries typically

have multiple attributes (columns), the SQL injection payload must account for the correct number of columns.

run the query: test%'))+union+all+select+1,2,3,4+--

increment the number of columns in the query until we receive a proper response.

. test%'))+union+all+select+1,2,3,4+--

. test%'))+union+all+select+1,2,3,4,5+--

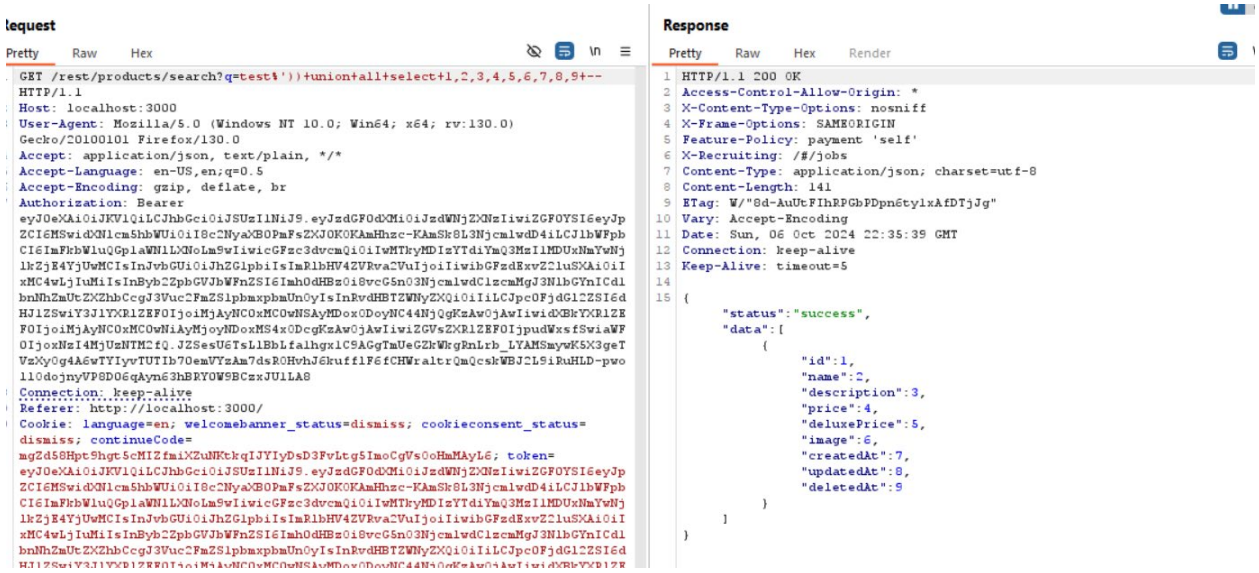
. test%'))+union+all+select+1,2,3,4,5,6+--

. test%'))+union+all+select+1,2,3,4,5,6,7+--

. test%'))+union+all+select+1,2,3,4,5,6,7,8+--

the correct number of columns that nine columns are being used because no error occurs with the query:

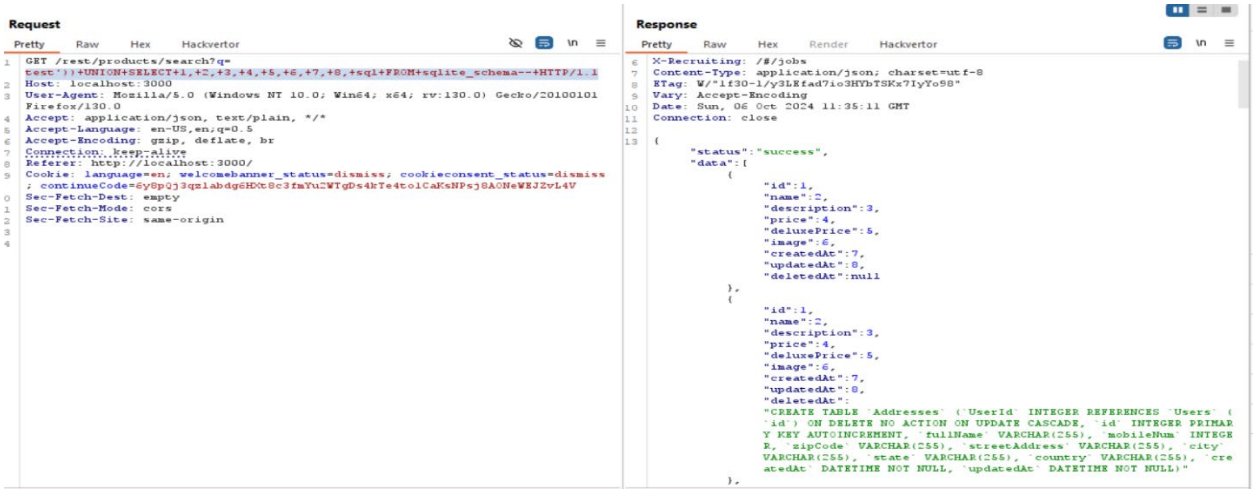
test%'))+union+all+select+1,2,3,4,5,6,7,8,9+--



We will combine the following query to create our final payload:
SELECT sql FROM sqlite_master WHERE type='table'

The final payload will be:

test%'))+union+all+select+1,+2,+3,+4,+5,+6,+7,+8,+sql+FROM+sqlite_sechma -- -



You successfully solved a challenge: Database Schema (Exfiltrate the entire DB schema definition via SQL Injection.) X

Impact **of Exploit:**

If an attacker successfully exploits this vulnerability, they could:

- Attackers can retrieve sensitive information, including user credentials, personal data, or confidential business information.

- attackers can modify or delete records, leading to data integrity issues.

- Gaining access to the database may allow attackers to execute commands on the underlying server, potentially leading to a complete system compromise.

- attackers may establish a persistent backdoor into an organization's systems, enabling long-term compromises that can remain undetected for extended periods.

Potential Remediation:

To mitigate the risk associated with SQL Injection, the following remediation steps are recommended:

- Use Prepared Statements: Avoid SQL injection by using parameterized queries or prepared statements.

- V**alidate and Sanitize Inputs**: Ensure that all user inputs are validated and sanitized to prevent malicious data from affecting SQL queries.

- **Error Handling**: Implement proper error handling to avoid revealing detailed error messages that could help attackers understand the database structure.

Client-side XSS Protection

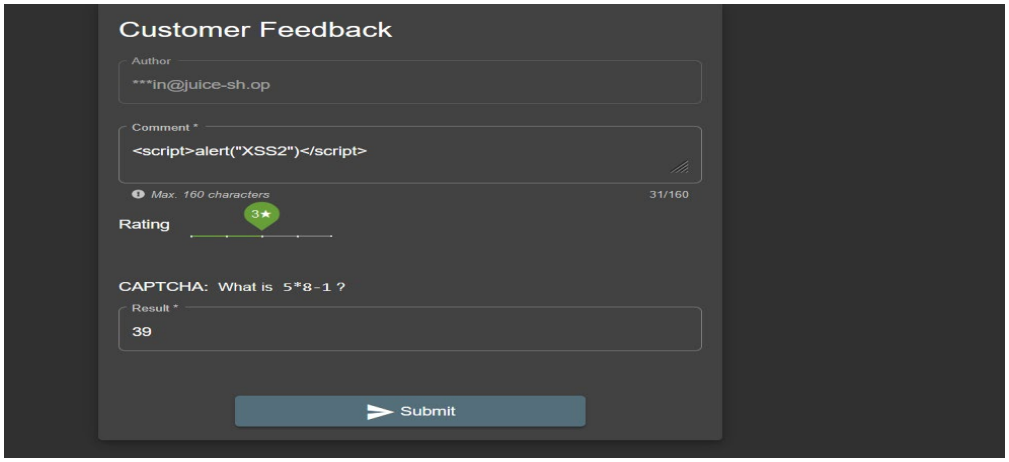
Risk Level: LOW

Vulnerability Detail:

Executive Summary:

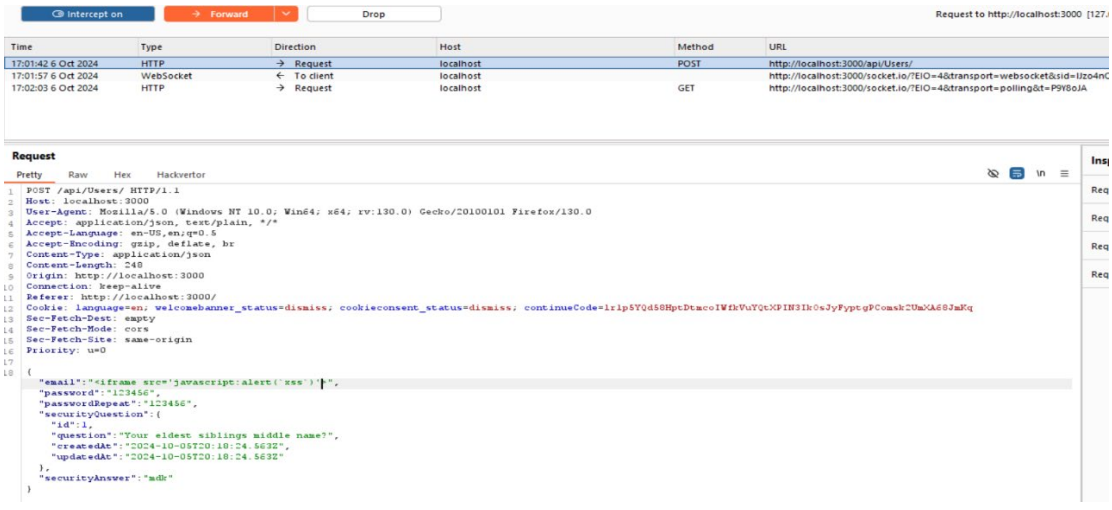
Client-side XSS happens when a web application processes user input in the browser without proper validation. This allows attackers to inject malicious scripts, which are executed in the user's browser.

We need to put the code somewhere and get it executed, so that it doesn't get filtered out by some JavaScript security mechanism on the client side. Simply pasting it in "Contact Us" doesn't work.

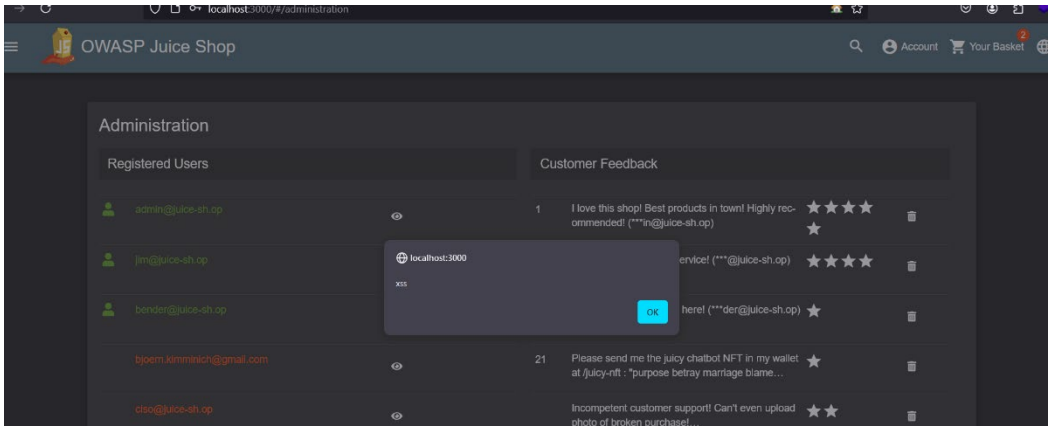


Another try could be to register a user who has this script in his credentials somewhere and thus *storing* it on the server. Putting it directly into the "Email" field does not work (Error:"Email address is not valid.").

But what if we intercept the request (again with Burp) after it was checked by the client-side JavaScript and inject the script there?



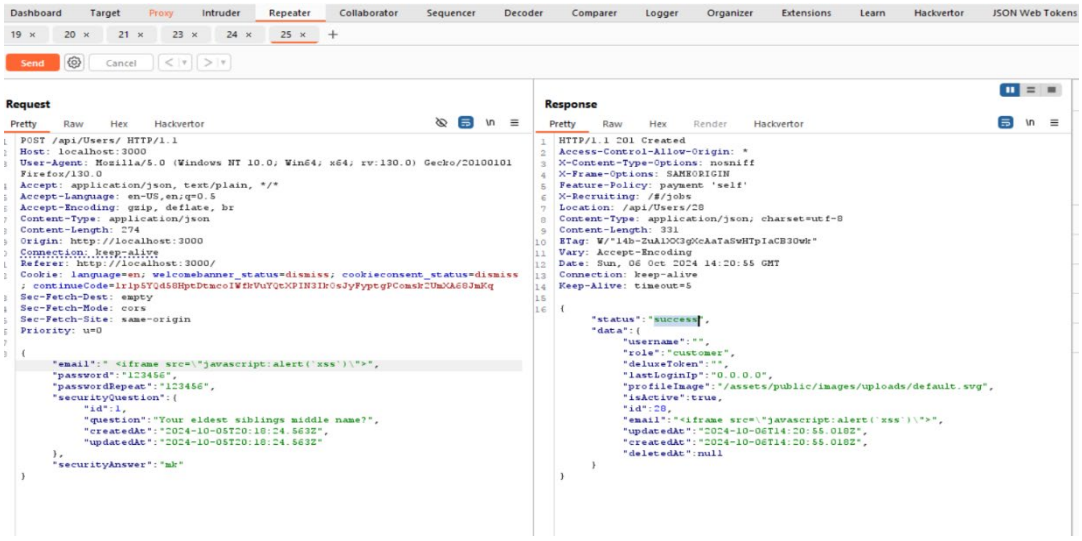
Logged into the application as an admin to access administrative functionalities.



But when I went to score board I found that the challenge not solved because I changed the payload to

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

To solve it send the request to repeater and use the skip char ('\') after double quote(")



Impact of exploit:

If an attacker successfully exploits this vulnerability, they could:

- Steal sensitive information like passwords or alert(document.cookie) to steal client's session cookie .

- Users may be tricked into entering personal information on fake forms.

- Scripts can redirect users to harmful sites or initiate downloads.

- Session Hijacking Attackers could impersonate users by stealing their session data.

Potential Remediation:

To mitigate the risk associated with XSS, the following remediation steps are recommended:

- Validate and Sanitize User Input Always check and clean user input to prevent harmful code from being injected.

- Use Content Security Policies (CSP) Implement CSP to control which sources can load resources on your webpage, helping to block XSS attacks.

- Escape Special Characters When displaying user input, escape characters like <, >, and & to prevent code injection.

- Use a Library or Framework Choose a library or framework that automatically escapes user input to protect against XSS

- **Regularly Test for Vulnerabilities** Frequently test your application for security issues, including XSS, to keep it safe.

Login Jim

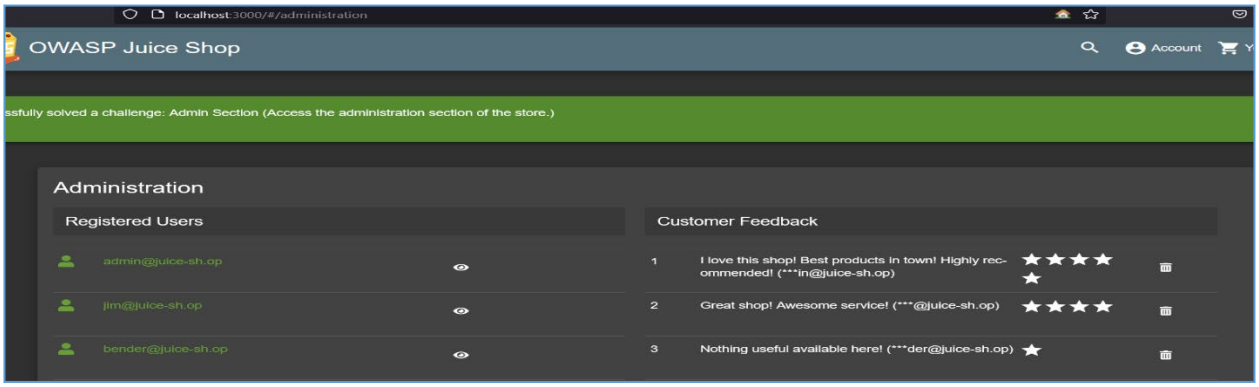
Risk Level: LOW

Vulnerability Detail:

Executive Summary:

The Login Jim challenge demonstrates a SQL Injection vulnerability that allows unauthorized access to a user's account. By entering a specific payload into the email field, an attacker can bypass authentication. This emphasizes the importance of proper input validation to prevent such attacks.

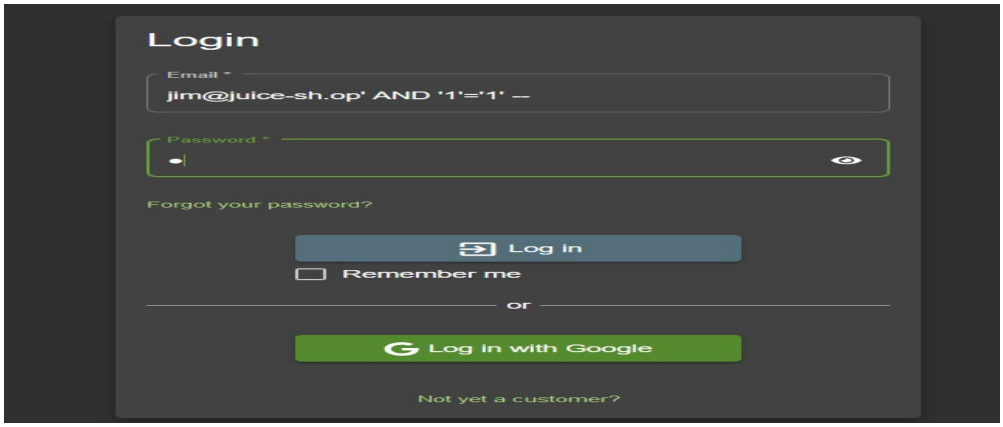
Logged into the application as an admin to access administrative functionalities. Located jim's email (jim@juice-sh.op) within the admin panel.



Go to the login page of the application and Entered jim's email in the email field.

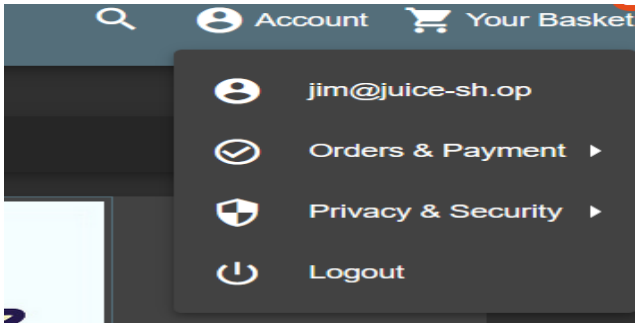
the SQL Injection:

Modified the entry in the email input field to include an SQL injection payload: **jim@juice-sh.op' AND '1'='1'--**



With the payload submitted, the SQL query altered by the injection would incorrectly authenticate the session as jim without requiring a password.

Successfully logged in as jim, demonstrating the exploitation of the SQL injection vulnerability



Impact **of Exploit:**

If an attacker successfully exploits this vulnerability, they could:

- Attackers can retrieve sensitive information, including user credentials, personal data, or confidential business information.

- attackers can modify or delete records, leading to data integrity issues.

- Gaining access to the database may allow attackers to execute commands on the underlying server, potentially leading to a complete system compromise.

- attackers may establish a persistent backdoor into an organization's systems, enabling long-term compromises that can remain undetected for extended periods.

Potential Remediation:

To mitigate the risk associated with SQL Injection, the following remediation steps are recommended:

- Use Prepared Statements: Avoid SQL injection by using parameterized queries or prepared statements.

- V**alidate and Sanitize Inputs**: Ensure that all user inputs are validated and sanitized to prevent malicious data from affecting SQL queries.

- **Error Handling**: Implement proper error handling to avoid revealing detailed error messages that could help attackers understand

the database structure.

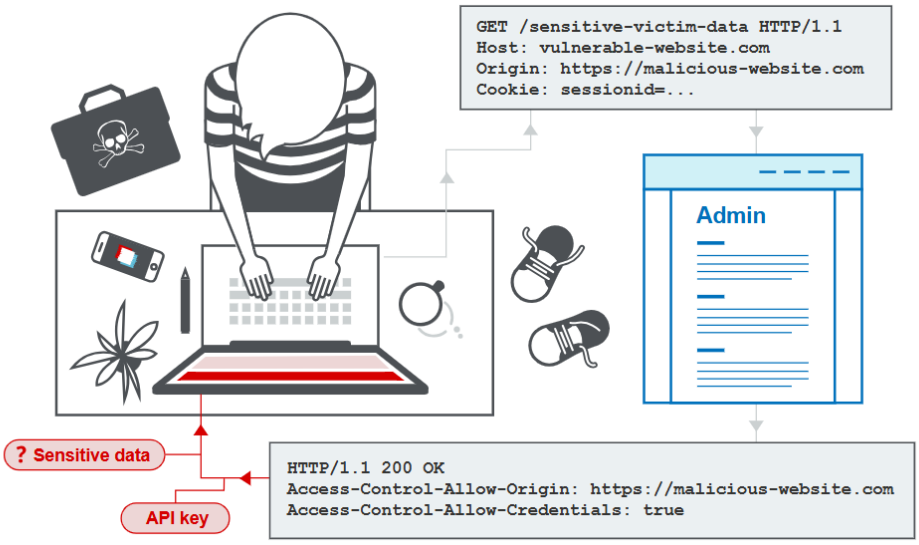
CORS

Risk Level: LOW

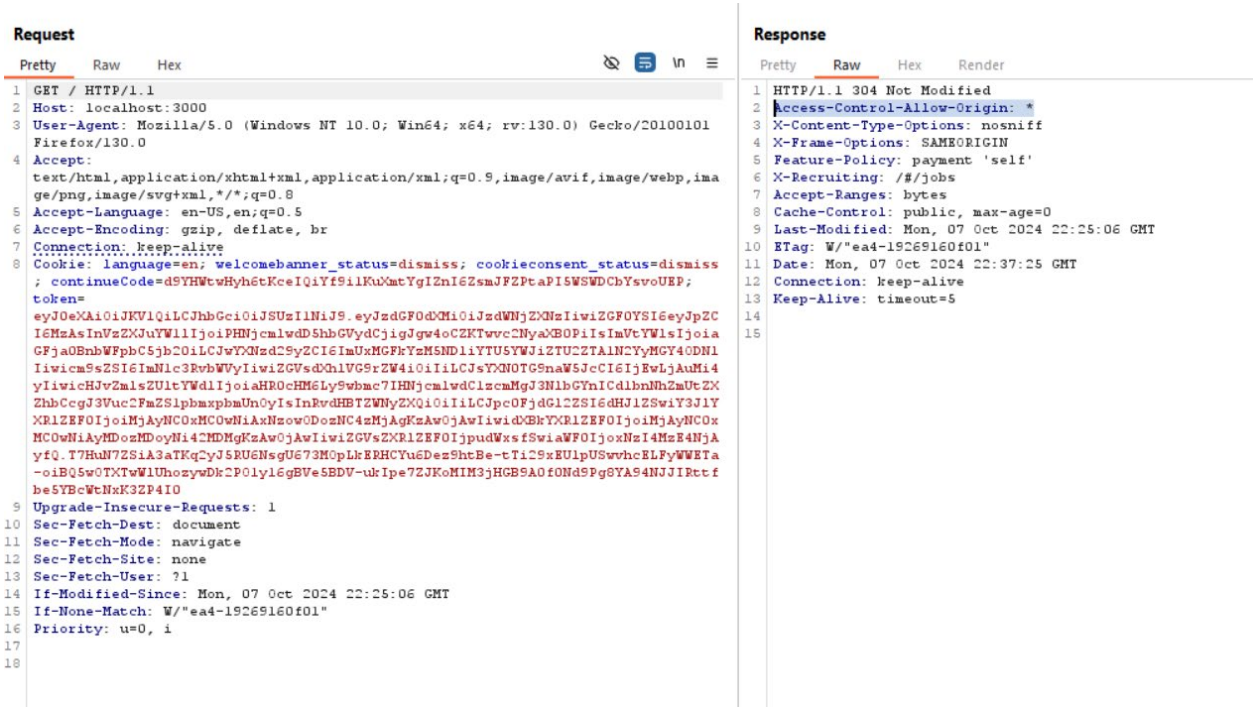
Vulnerability Detail:

Executive Summary:

Cross-origin resource sharing (CORS) is a browser mechanism which enables controlled access to resources located outside of a given domain. It extends and adds flexibility to the same-origin policy (SOP). However, it also provides potential for cross-domain attacks, if a website's CORS policy is poorly configured and implemented. CORS is not a protection against cross-origin attacks such as cross-site request forgery (CSRF).



The application uses wild card for CORS



By using the * wildcard for the Access-Control-Allow-Origins header we allow any origin to access the resource. If this is just a public API that doesn't need anything from the user apart from a simple bearer token, then it's okay. But it's not a good idea if we're dealing with confidential information.

Impact of exploit:

If an attacker successfully exploits this vulnerability, they could:

- Data Exposure: Sensitive information from a user's session can be accessed by unauthorized domains, leading to data leaks**.**

- Account Takeover: Attackers may be able to perform actions on behalf of users, potentially gaining control of their accounts without their consent.

- Malicious Actions: An exploited CORS policy can allow attackers to send requests that alter user data or settings, such as changing passwords or making unauthorized transactions**.**

Potential Remediation:

To mitigate the risk associated with CORS, the following remediation steps are recommended:

- Specify Trusted Origins: For sensitive resources, clearly define which origins are allowed in the Access-Control-Allow-Origin header.

- Limit to Trusted Sites: Only include trusted sites in the Access-Control-Allow-Origin header. Avoid reflecting origins from requests without validation, as this can be exploited.

- Avoid Whitelisting Null: Don't use Access-Control-Allow-Origin: null, as this can allow internal documents and sandboxed requests to bypass security. Instead, specify trusted origins for both private and public servers.

- No Wildcards in Internal Networks: Avoid using wildcards in internal networks, since relying solely on network configuration can be risky when internal browsers access untrusted external domains.

- CORS Is Not a Complete Solution: CORS controls browser behavior but isn't a substitute for server-side security. Always implement strong protections for sensitive data, such as authentication and session management, alongside properly configured CORS

Methodology

This section outlines the penetration testing approach used during the security assessment of the OWASP Juice Shop web application. It is intended for technical practitioners such as security analysts, engineers, developers, and system administrators. Technical terminology will be employed throughout.

The methodology including phases such as information gathering, configuration management testing, authentication, session management, input validation, and error handling. The process is described chronologically, starting with the selection of tools, followed by the execution, analysis of output, and manual validation of any significant vulnerabilities.

Any significant vulnerabilities identified are referenced in the "Significant Vulnerabilities Summary and Detail" section and are further discussed in the "Risk Analysis" section, with recommendations included where applicable.

Assessment Toolset Selection

The following tools were used for various purposes during the vulnerability assessment and validation of the OWASP Juice Shop web application:

Burp Suite: For intercepting traffic, active scanning, and testing injection vulnerabilities (SQLi, XSS).

Nmap: For network discovery and port scanning to detect open services and potential exposure.

Wireshark: Used for packet capturing and analysis during the network layer assessment.

Dirbuster: For directory brute-forcing and uncovering hidden files or paths.

FFUF: For directory fuzzing and uncovering hidden files or paths.

SQLMap: To automate the detection and exploitation of SQL injection vulnerabilities.

Hydra: For brute-force testing on authentication endpoints (e.g., login forms).

JWT Cracker: For testing the security of JSON Web Tokens used in the application.

Postman: For manual API testing, especially useful for verifying API-related issues such as broken access control.

Assessment Methodology Detail

1. Information Gathering (Reconnaissance)

- Gathering information by:
    - **Passive Reconnaissance**: Use search engines to identify publicly available information about Juice Shop.
    - **Active Reconnaissance**: Use tools to map the application and discover input fields, hidden endpoints, API calls, etc.
    - Look for directories, parameters, and hidden resources that might reveal sensitive information.
    - **Tools**: Burp Suite (spider), OWASP ZAP, Dirbuster/FFUF.
    - **Focus Areas**:

      Analyze URLs, hidden parameters, input fields AND Identify the technologies used (Node.js, AngularJS).

2. Vulnerability Identification (Scanning)

- By Useing automated tools to identify vulnerabilities specific to Juice Shop (based on OWASP Top 10):
    - Run an automated scan to identify potential vulnerabilities like SQL Injection, Cross-Site Scripting, and sensitive data exposure.
    - Analyze the scan results and correlate findings with Juice Shop's known vulnerable areas.
    - **Tools**: Burp Suite (active scanning) (automated scanner).
    - **Focus Areas**:

      SQL Injection points.

      Input fields vulnerable to XSS.

      Authentication issues like weak session management.

      Sensitive information leakage via error messages.

3. Exploitation

Exploit identified vulnerabilities to demonstrate real security risks.

- **Input Validation Testing (based on WSTG-INPV-01)**:
    - **SQL Injection**: Use SQLMap to exploit any SQLi vulnerabilities found.

- **Cross-Site Scripting (XSS)**: Craft XSS payloads and inject them into vulnerable input fields to test data session hijacking.
- **Broken Authentication**: Exploit weak login mechanisms, session tokens, or privilege escalation.
- **Tools**: Burp Suite (manual exploitation), SQLMap (SQL injection), Toxssin(automated XSS exploit), Hydra (for brute-forcing login forms).
- **Focus Areas**:

  Exploit SQL injection in product search fields.

  Test XSS in forms and URL parameters.

  Check for improper authentication and authorization mechanisms.

- **Authentication Testing (based on WSTG-AUTHN-01)**

  Authentication mechanisms were tested to ensure they were properly implemented:

  - Hydra was used for brute-force attacks on login pages to test for weak passwords or improper account lockout policies.
  - Burp Suite and Postman were used to inspect and manipulate login requests and authentication tokens, including testing for insecure implementations of JWT.

- **API Testing (WSTG-API-01)**

  APIs were tested to ensure secure implementation:

  - As we saw we performed a XSS into Api.

4. Validation of Significant Vulnerabilities

Significant vulnerabilities discovered through automated tools were manually validated to confirm their existence and impact. This manual testing involved:

- Reproducing the issues by using various attack vectors and manual exploitation techniques.
- Documenting the manual validation process with screenshots, commands, by POC's and any related evidence to demonstrate the exploitation steps.

The methodology and evidence provided ensure that a technical practitioner can replicate the assessment and arrive at the same conclusions regarding the identified vulnerabilities and associated risks.

This concluded the vulnerability assessment methodology portion of this report.