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

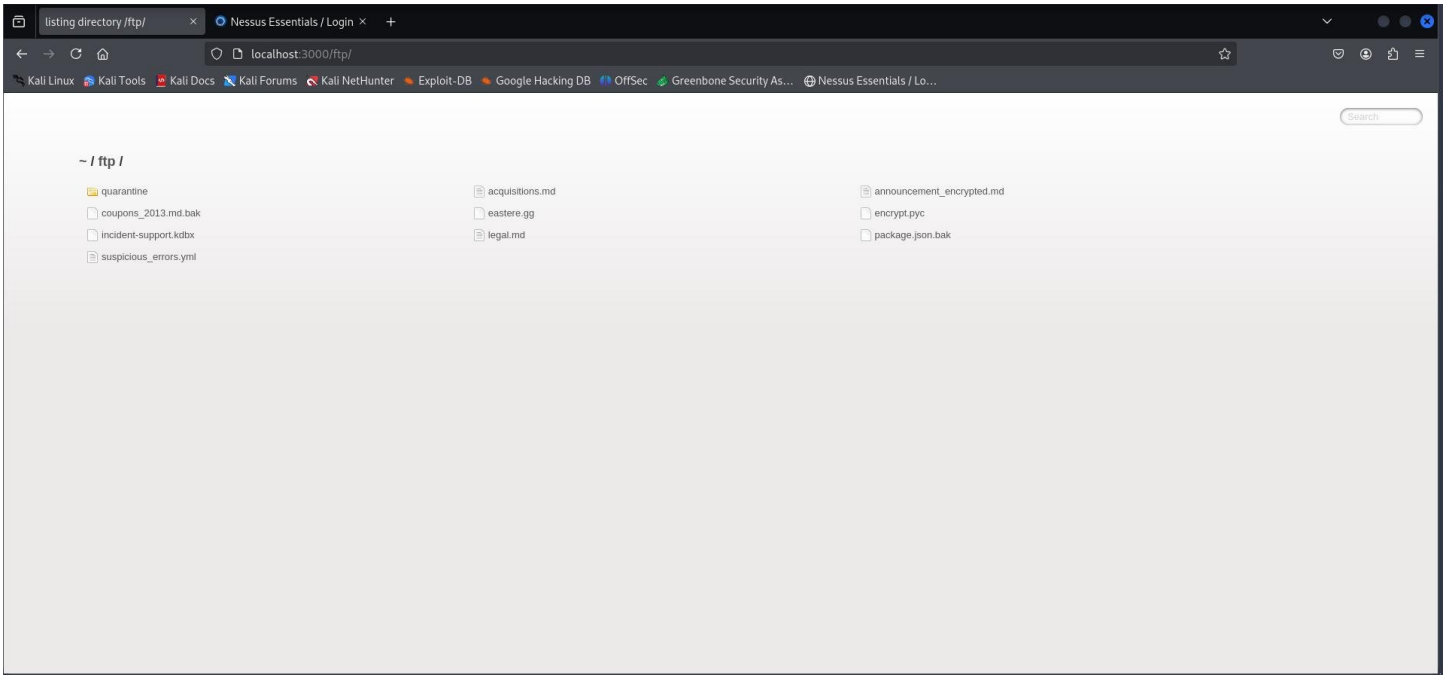
- **Severity:** High
- **Description:** Cookie with session ID was retrievable via script
- **Impact:** Allowed session hijacking
- **Payload:**<iframe src="javascript:alert(document.cookie)"></iframe>
- **Mitigation:** Use Http Only and Secure flags on cookies

6.7 Procedure:

```
rudrakali@kali: ~/juice-shop
File  Actions  Edit  View  Help
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_js.ico: The X-Content-Type-Options header is not set. This could allow the user agent to render the content o
f 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.



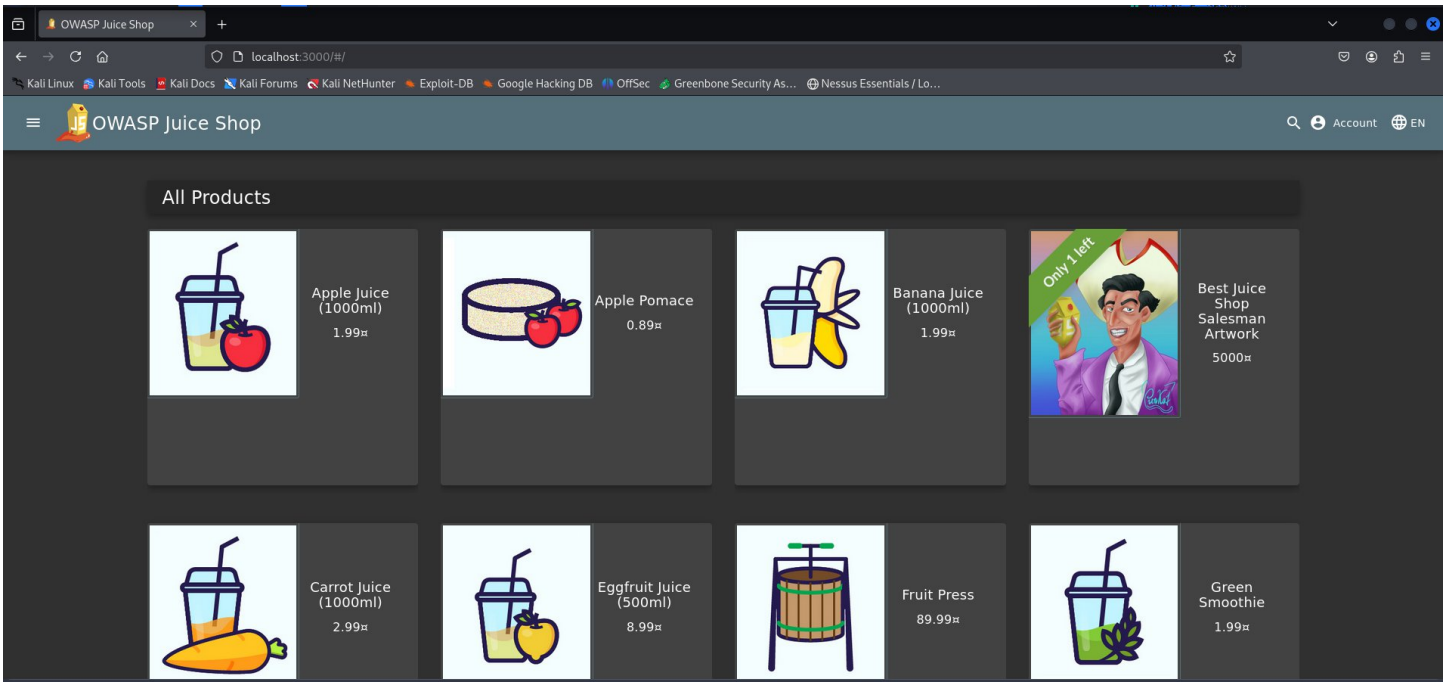
It successfully accessible to the hidden files in the web page. Next we see the nmap scan to see the what ports are open and services

```
┌──(rudrakali㉿kali)-[~/juice-shop]
└─$ sudo nmap -sS -sV -sC -O -T2 127.0.0.1
[sudo] password for rudrakali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-06 13:46 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.000053s latency).
Not shown: 999 closed tcp ports (reset)
PORT     STATE SERVICE VERSION
3000/tcp open  ppp?
| fingerprint-strings:
|   GetRequest:
|     HTTP/1.1 200 OK
|     Access-Control-Allow-Origin: *
|     X-Content-Type-Options: nosniff
|     X-Frame-Options: SAMEORIGIN
|     Feature-Policy: payment 'self'
|     X-Recruiting: /#/jobs
|     Accept-Ranges: bytes
|     Cache-Control: public, max-age=0
|     Last-Modified: Sun, 06 Apr 2025 17:45:28 GMT
|     ETag: W/"11708-1960c351a48"
|     Content-Type: text/html; charset=UTF-8
|     Content-Length: 71432
|     Vary: Accept-Encoding
|     Date: Sun, 06 Apr 2025 17:53:42 GMT
|     Connection: close
|     <!--
|     Copyright (c) 2014-2025 Bjoern Kimminich & the OWASP Juice Shop contributors.
|     SPDX-License-Identifier: MIT
|     <!doctype html>
|     <html lang="en" data-critters-container>
|     <head>
|     <meta charset="utf-8">
|     <title>OWASP Juice Shop</title>
|     <meta name="description" content="Probably the most modern and sophisticated insecure web application"
|     <meta name="viewport" content="width=device-width, initial-scale=1">
|     <link id="favicon" rel="icon"
|   HTTPOptions, RTSPRequest:
|     HTTP/1.1 204 No Content
|     Access-Control-Allow-Origin: *
|     Access-Control-Allow-Methods: GET,HEAD,PUT,PATCH,POST,DELETE
|     Vary: Access-Control-Request-Headers
|     Content-Length: 0
|     Date: Sun, 06 Apr 2025 17:53:42 GMT
|     Connection: close
|   Help, NCP:
|     HTTP/1.1 400 Bad Request
|_    Connection: close
```

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.

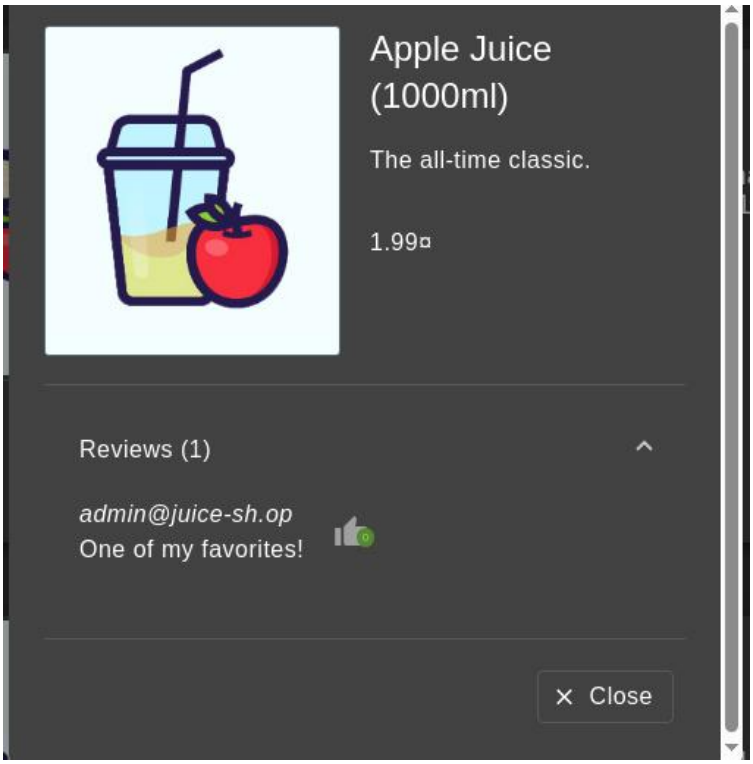
In this penetration testing we are using the web site from our localhost

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



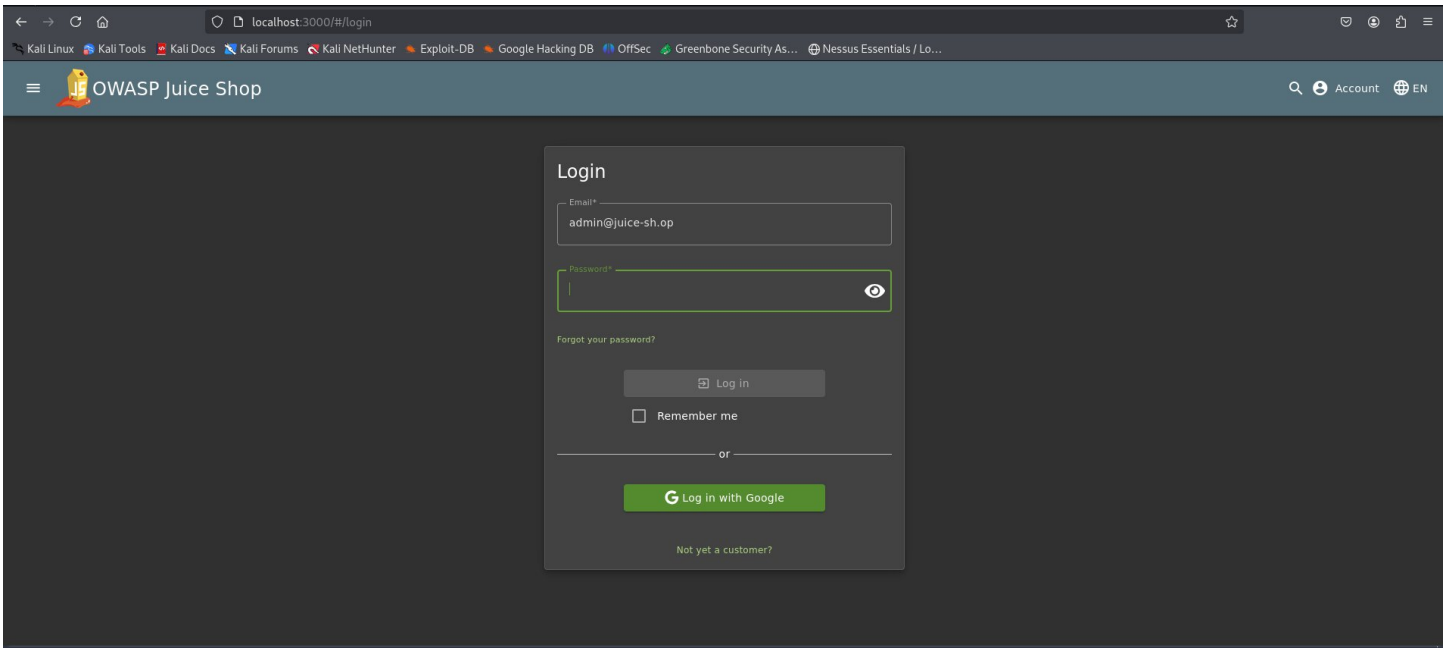
So we can see the web site has some products and a search button and account option for login.

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

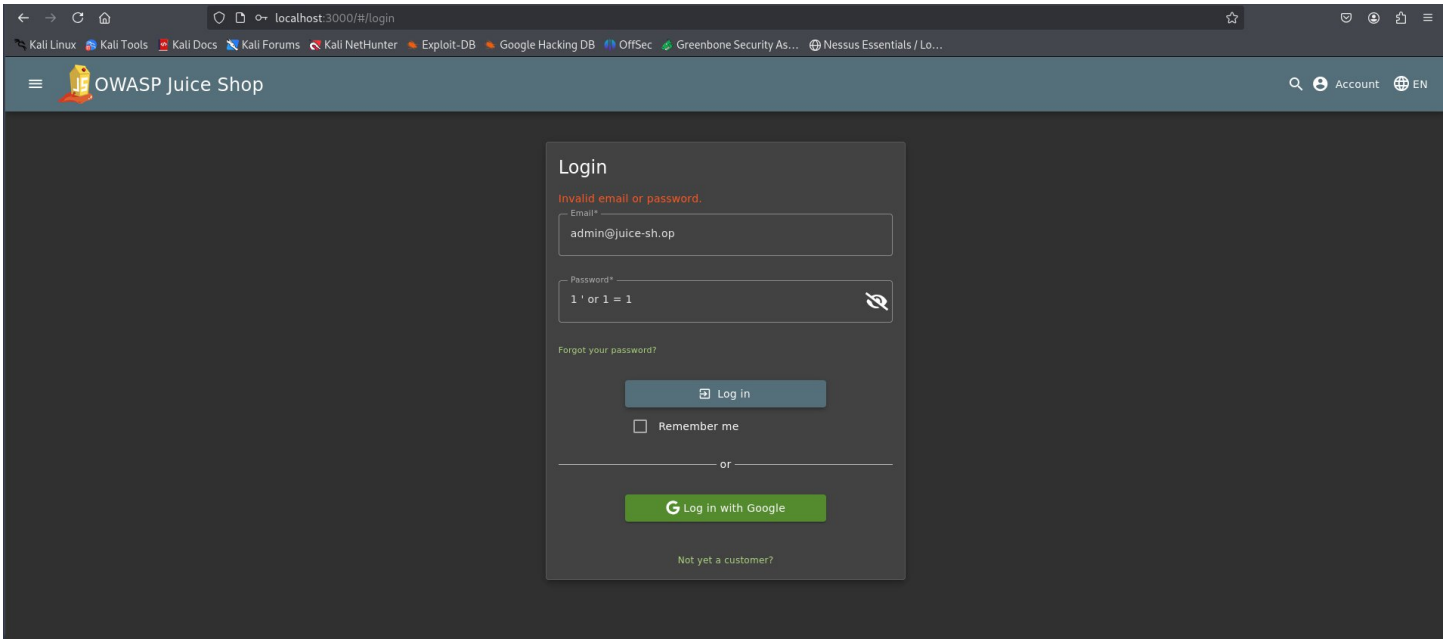


So when we click this product it opens like this and we have some information like reviews and intrestingly there ,they are uploding reviews with emails and its a admin mail : **admin@juice-sh.op**

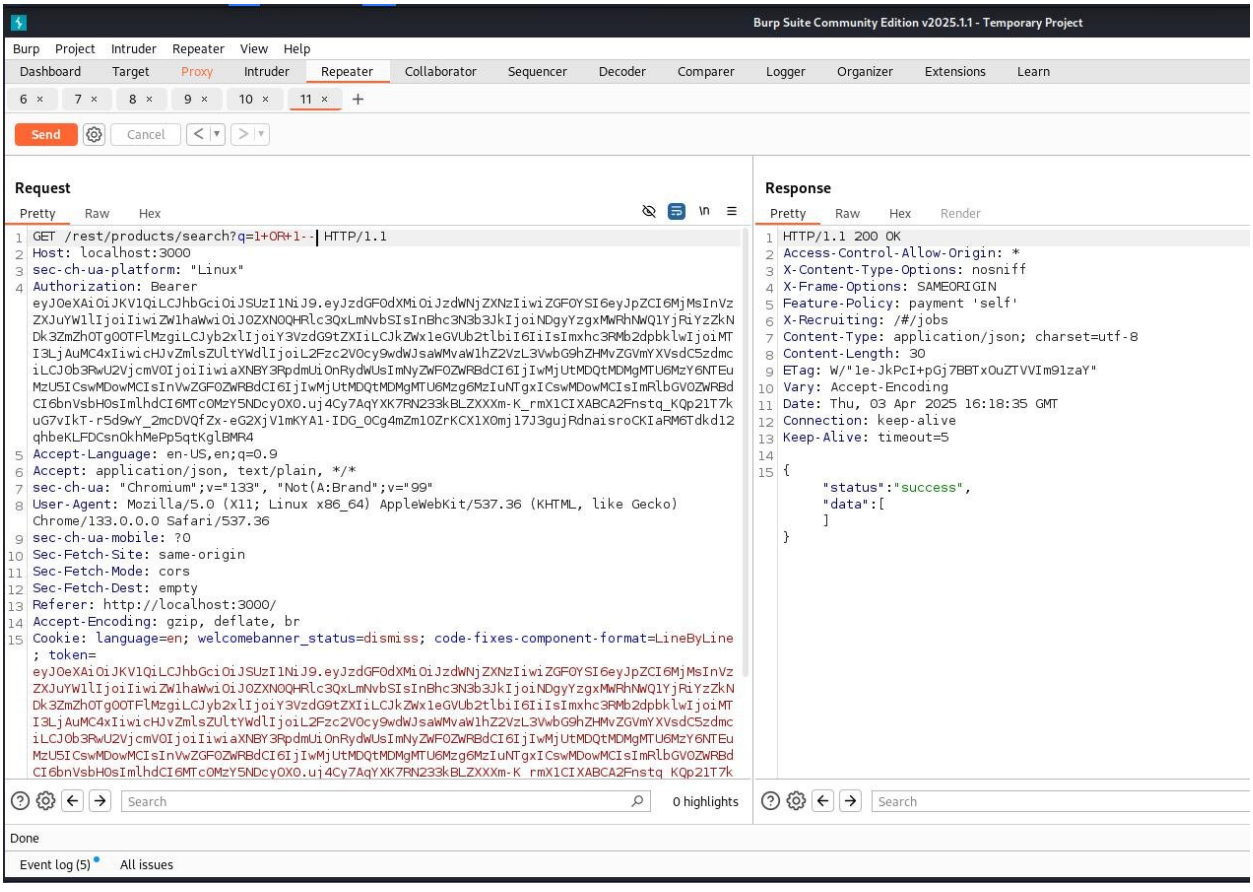
So now lets try to login to admin account, but we don't know the password.



So lets try with an SQL injection

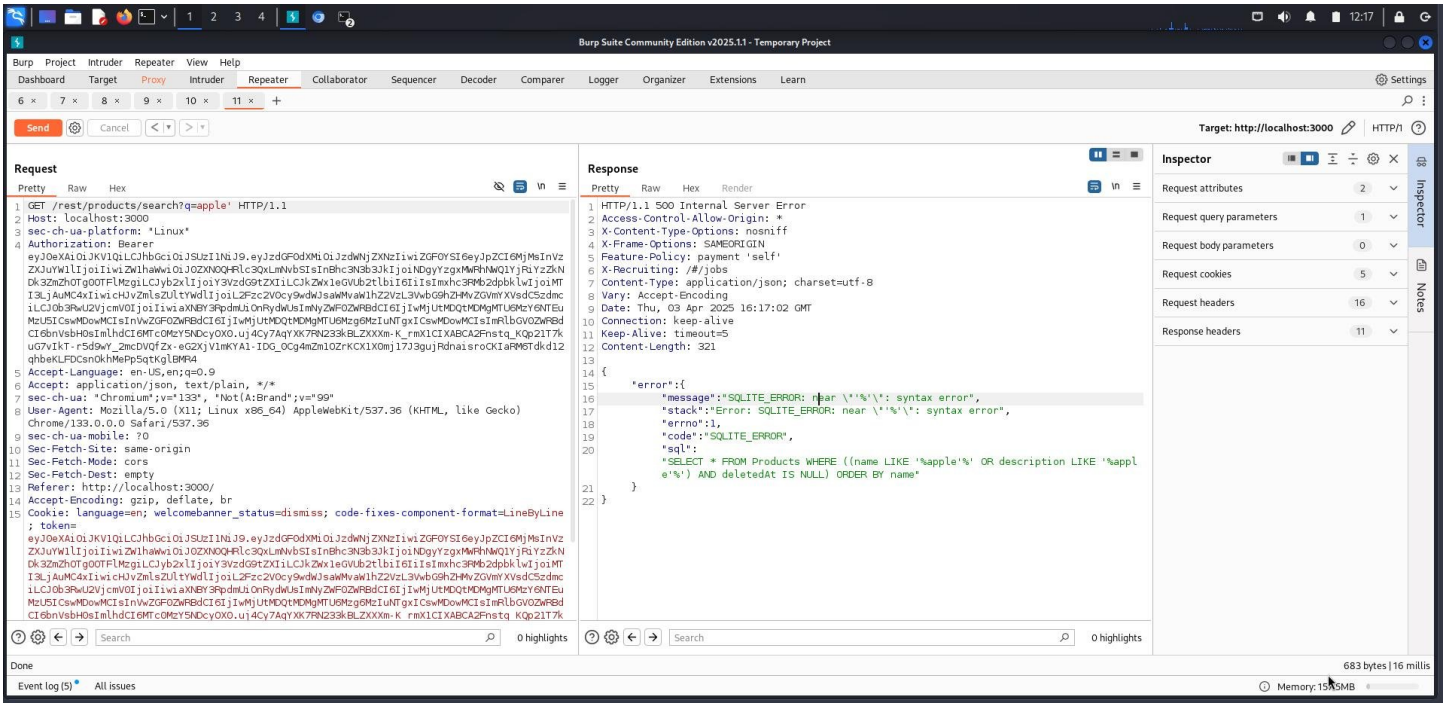


But well it was failed now lets check that how does it sending or taking the data using the burpsuite



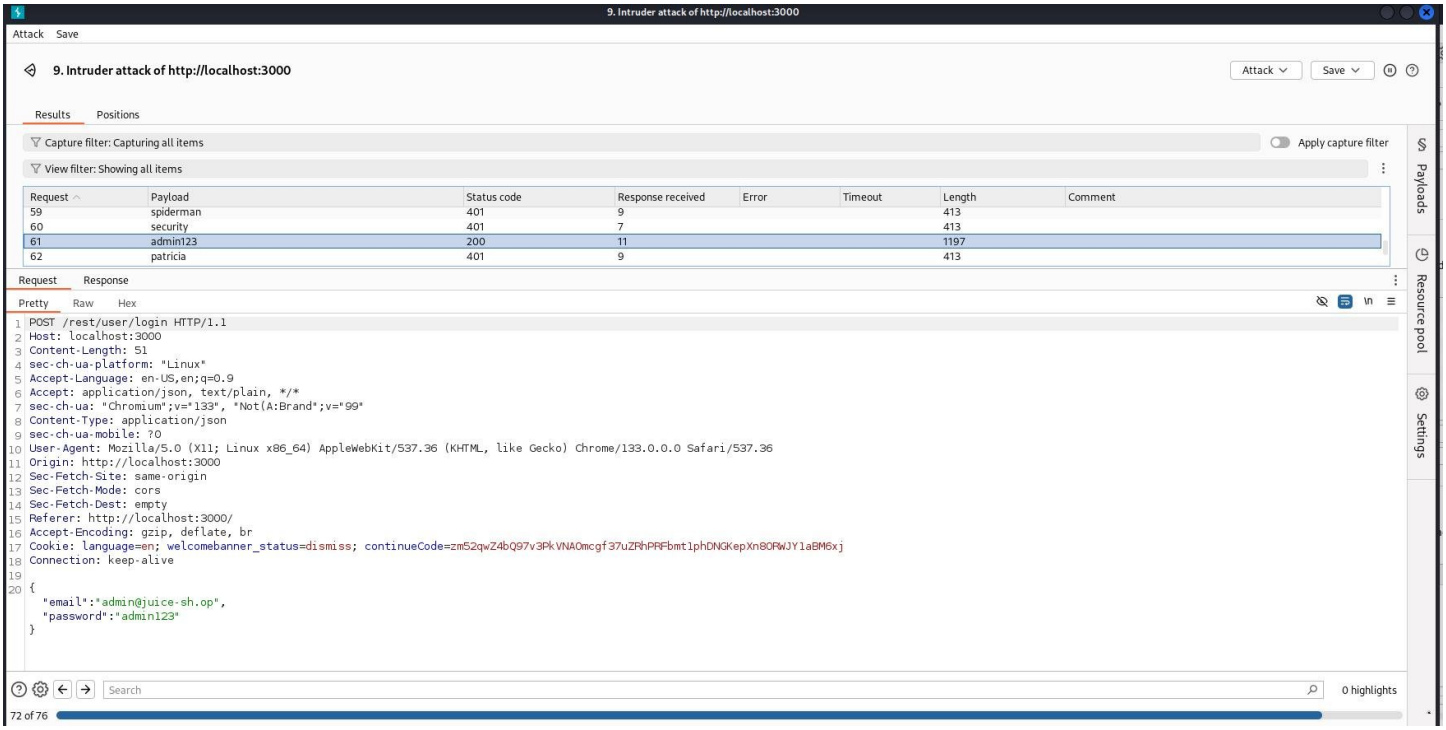
See here when we try for the SQL injection at the search field ,at the response the request is getting accepted and status is showing success. That mean the SQL Injection is working but need to specify or modify the payload.

So, lets try some other ways



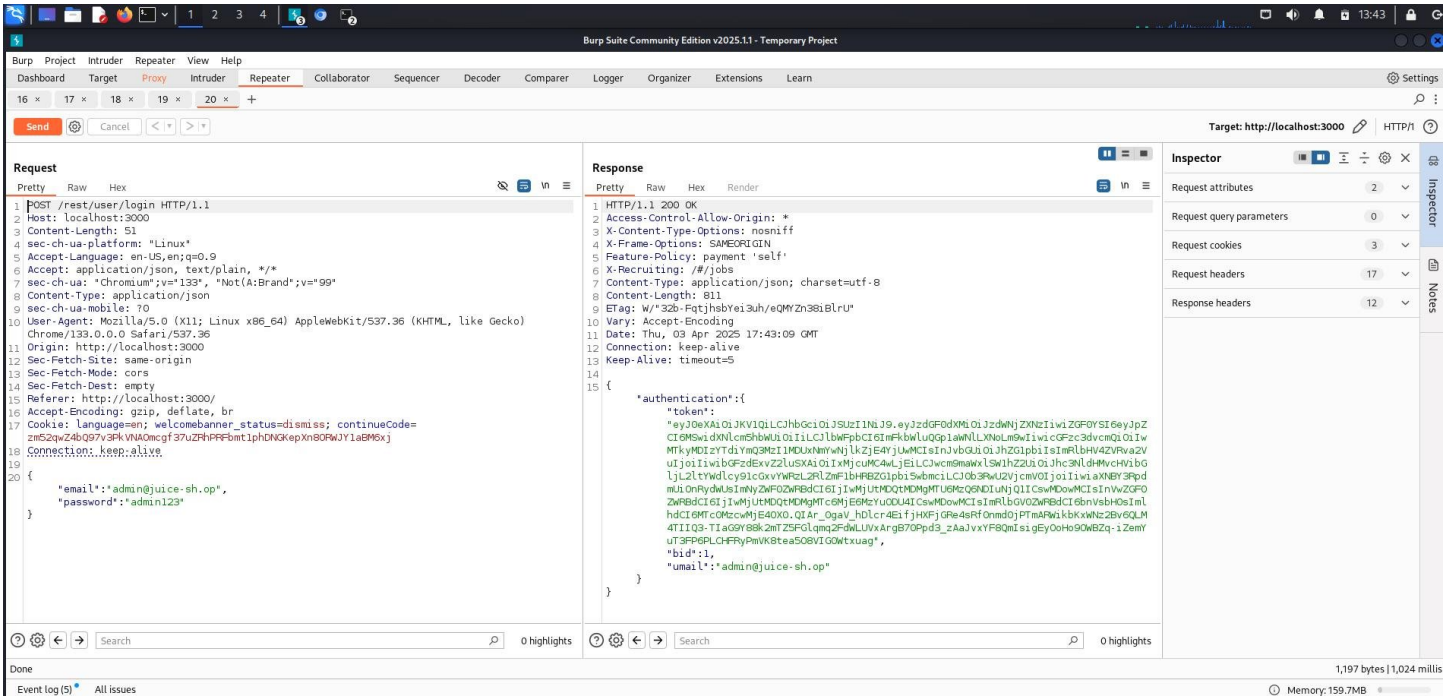
This time we at search field we entered **apple'** and then it returns the error: of SQLITE_ERROR, so here we can say that the web application is using a SQLITE database.

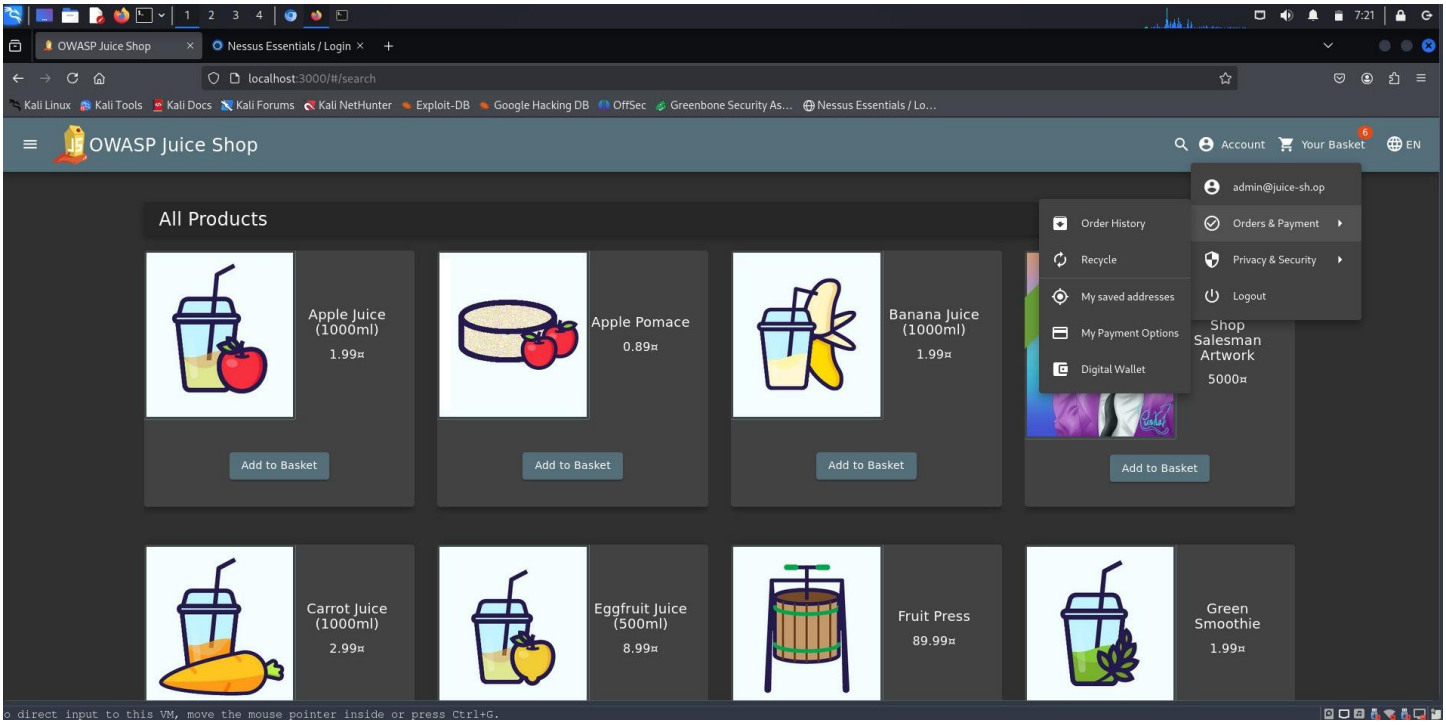
Wait ,we have the email of the admin so lets try to bruteforce using a list of known passwords ,lets do this in burpsuite intruder



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

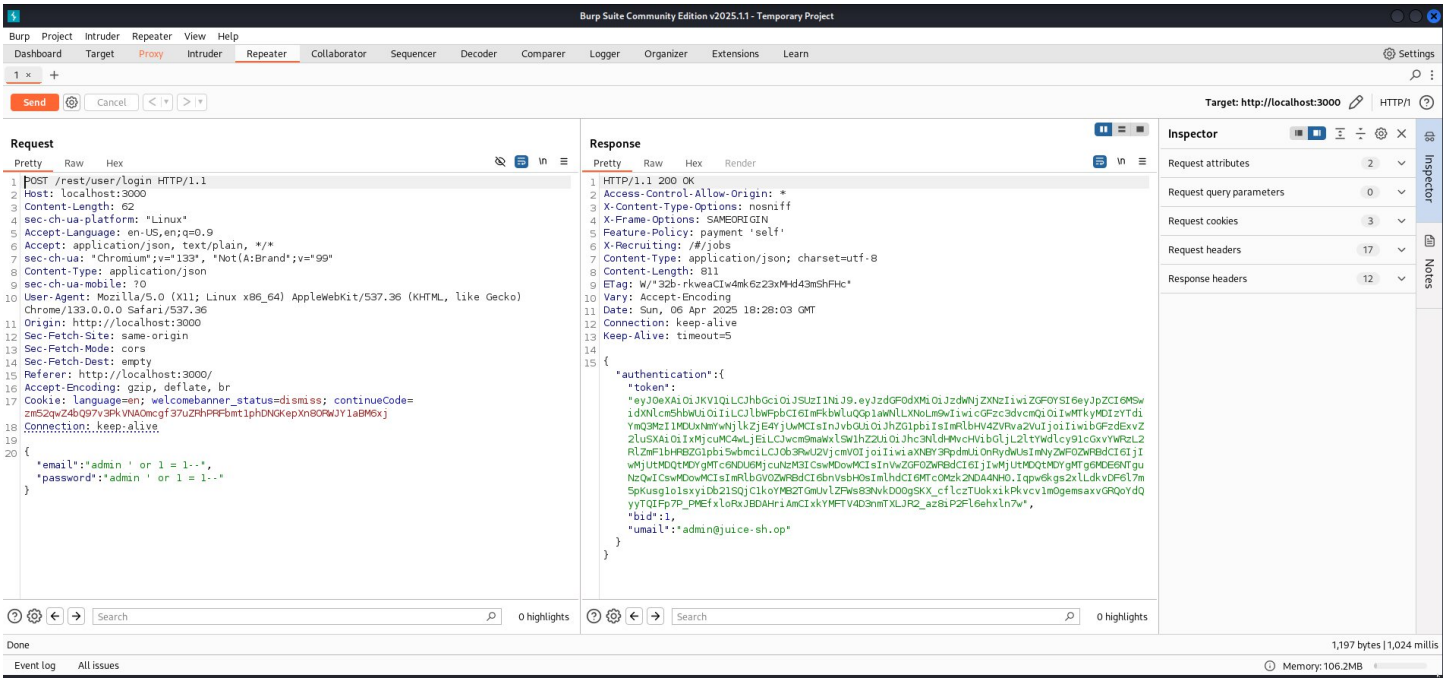




And before we saw that there is a possibility of SQL injection to login so lets check this also

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.

Paylod: admin' 1=1 - -



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.

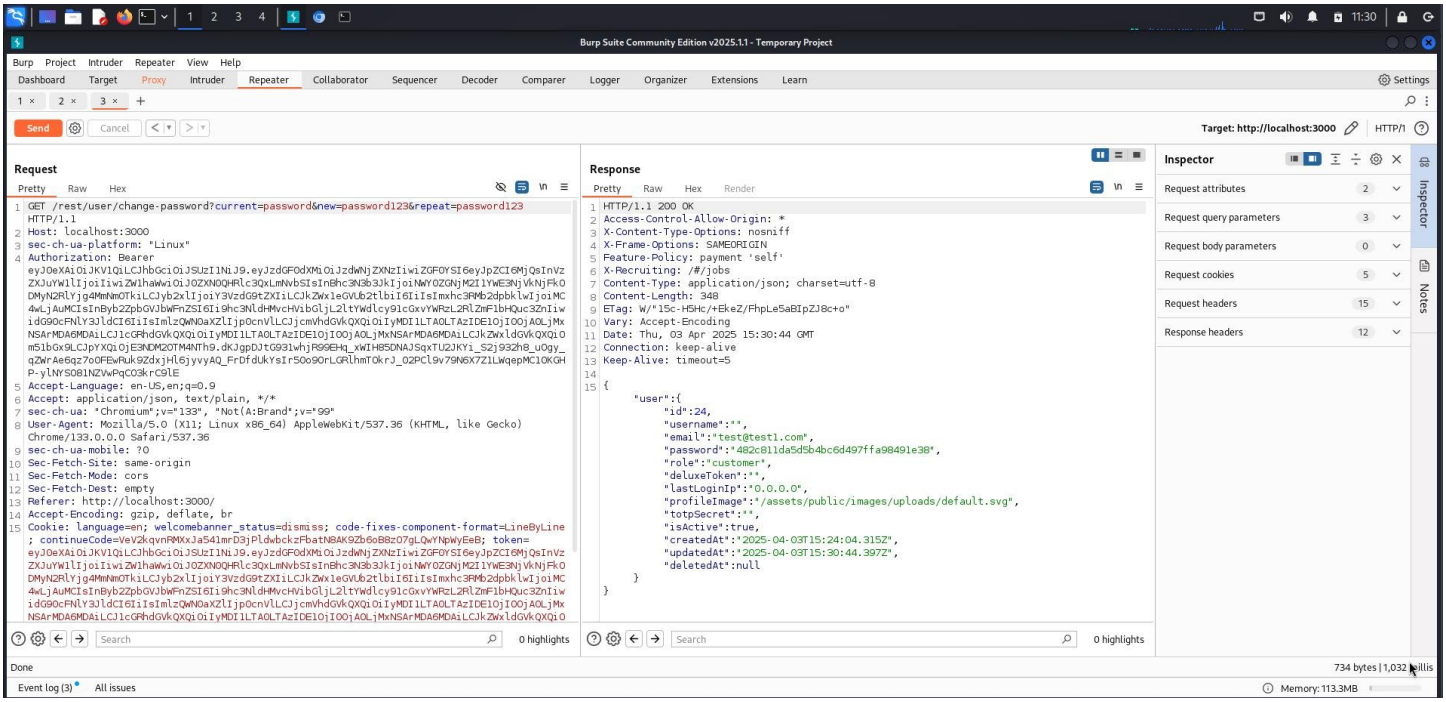
Credentials we are using:

testing email = test@test1.com

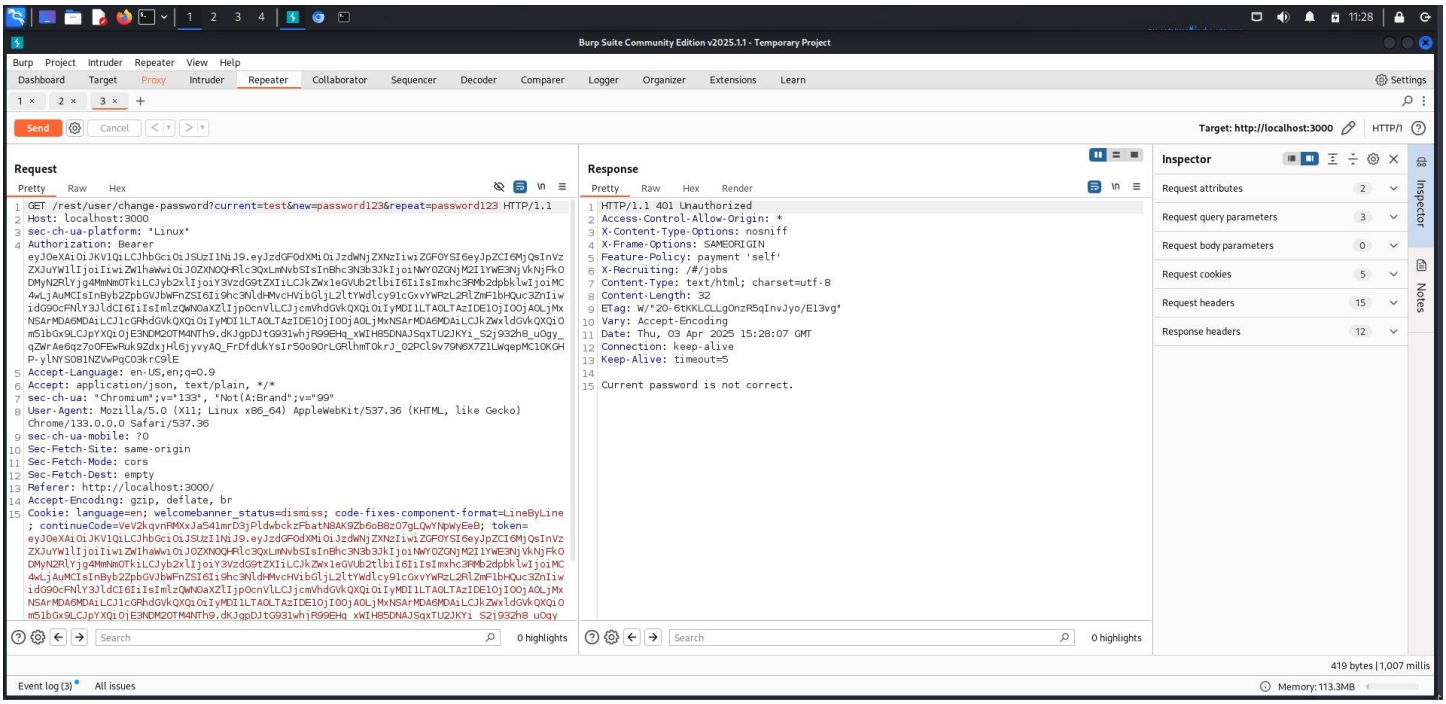
pass = password

security answer = dog

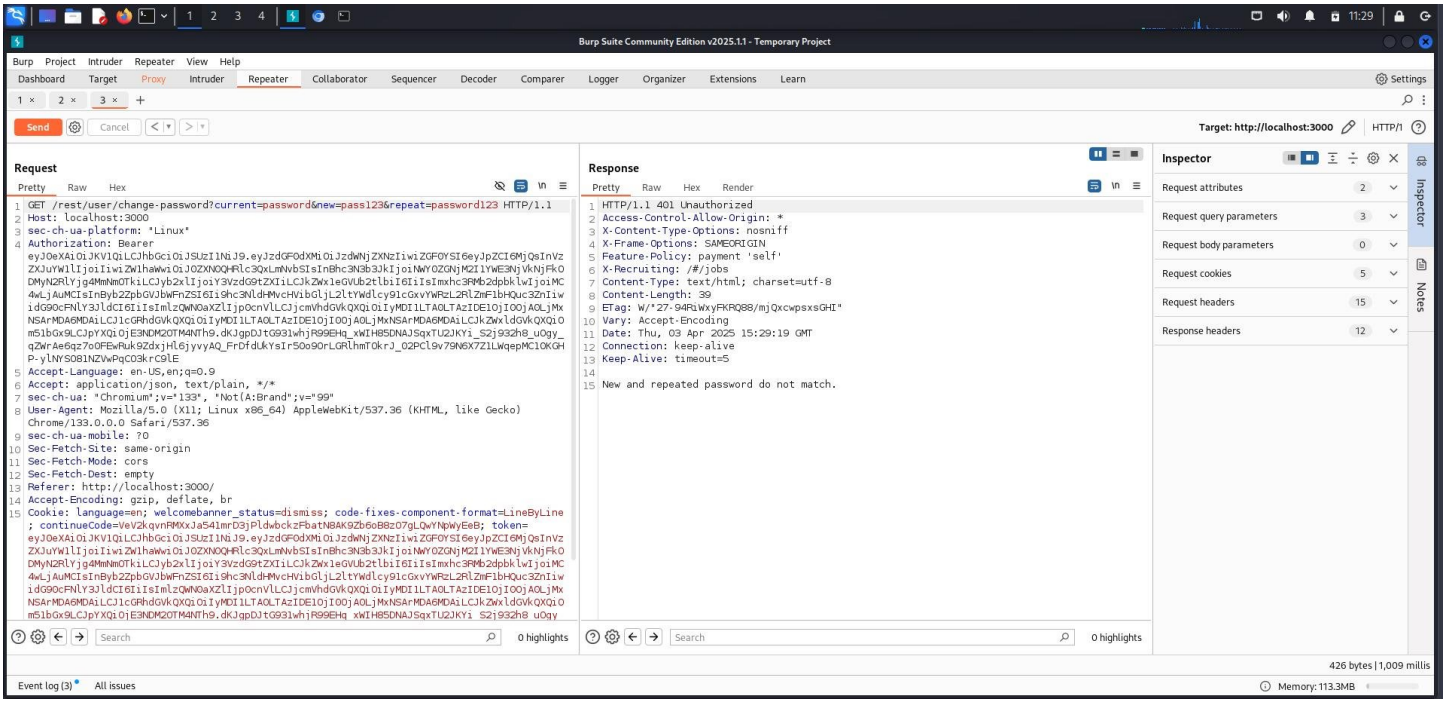
While logged in this account can we change the user password lets check it.



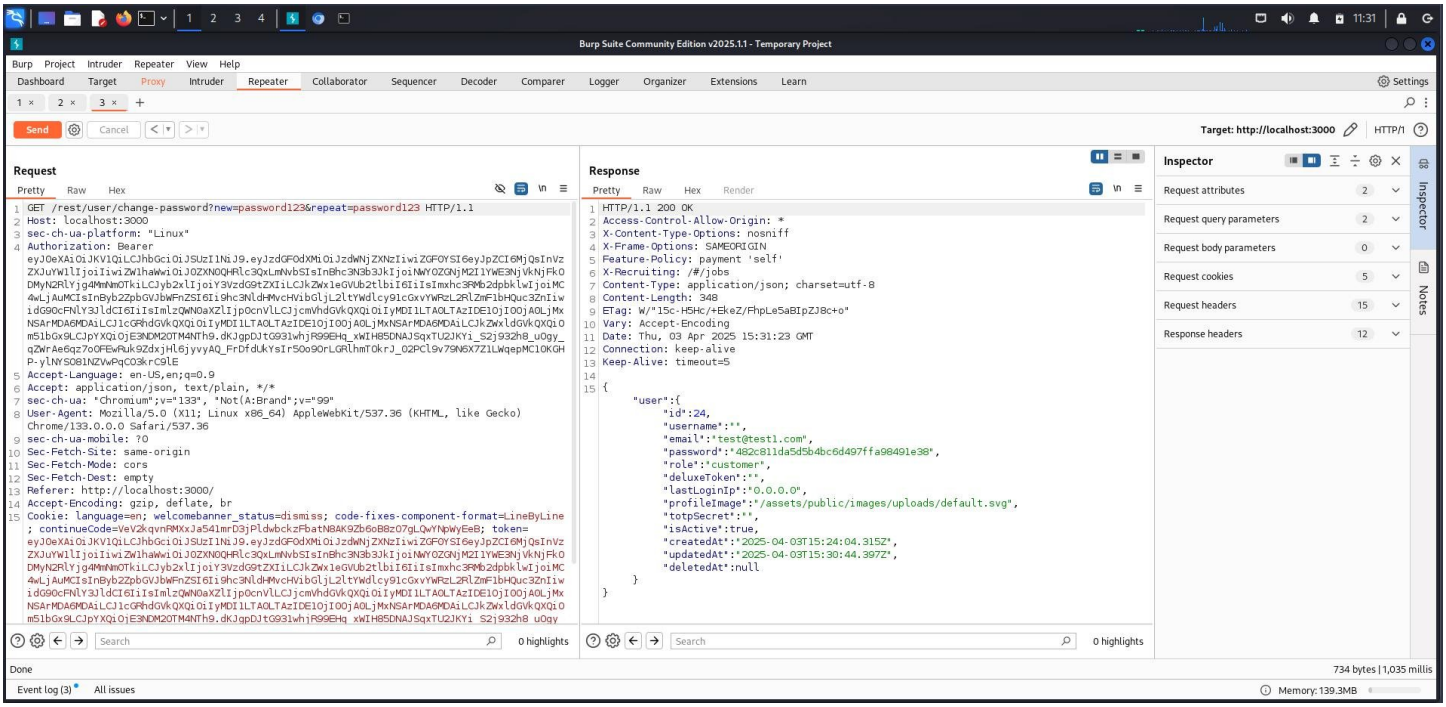
With the http response 200 so we can change the user password with the right current password and double entry of the new password. Now lets check weather if the current password is wrong does it accept.



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

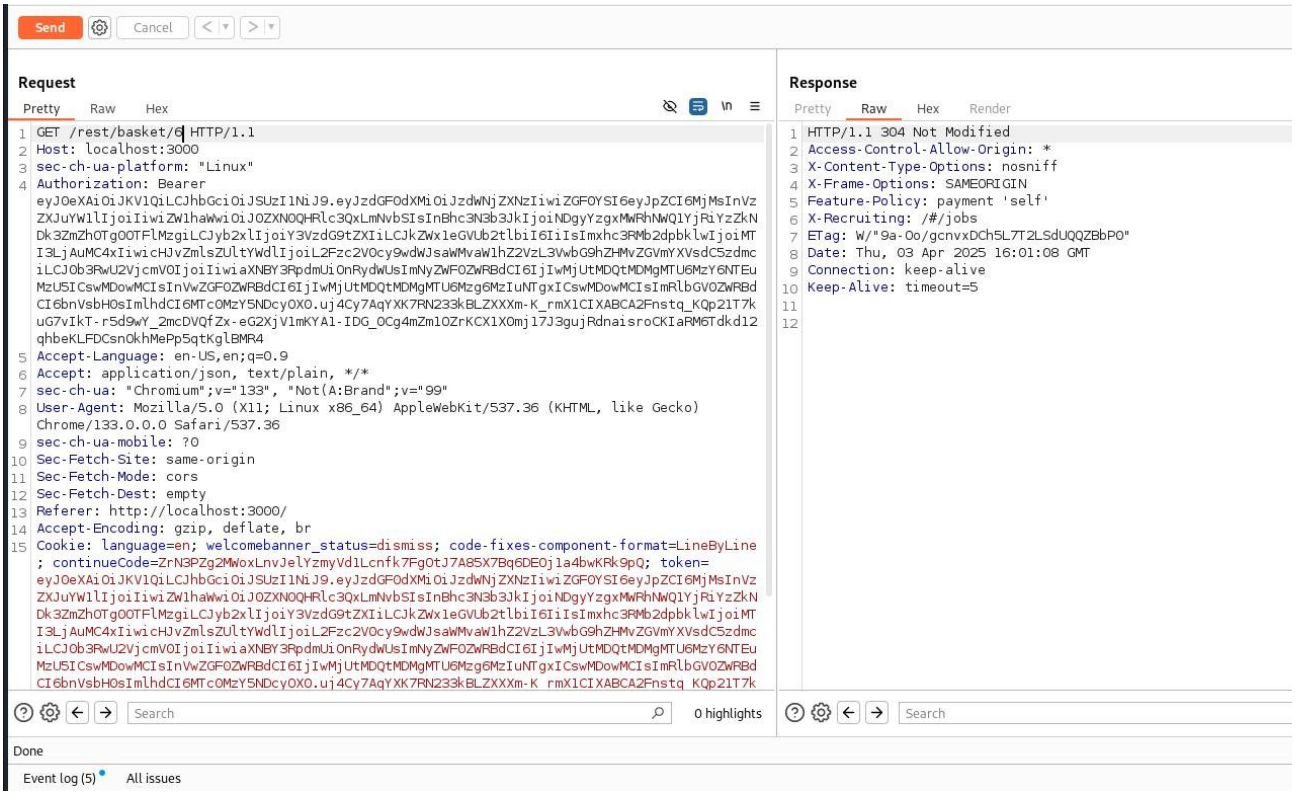


Good even it is also blocked. Now i had a doubt , if the current password field is removed and request is sent with new and retype of the new password then does it work's , because without verifying current password any other person with access the logged in system they can easyly change the password , the vlaue 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.

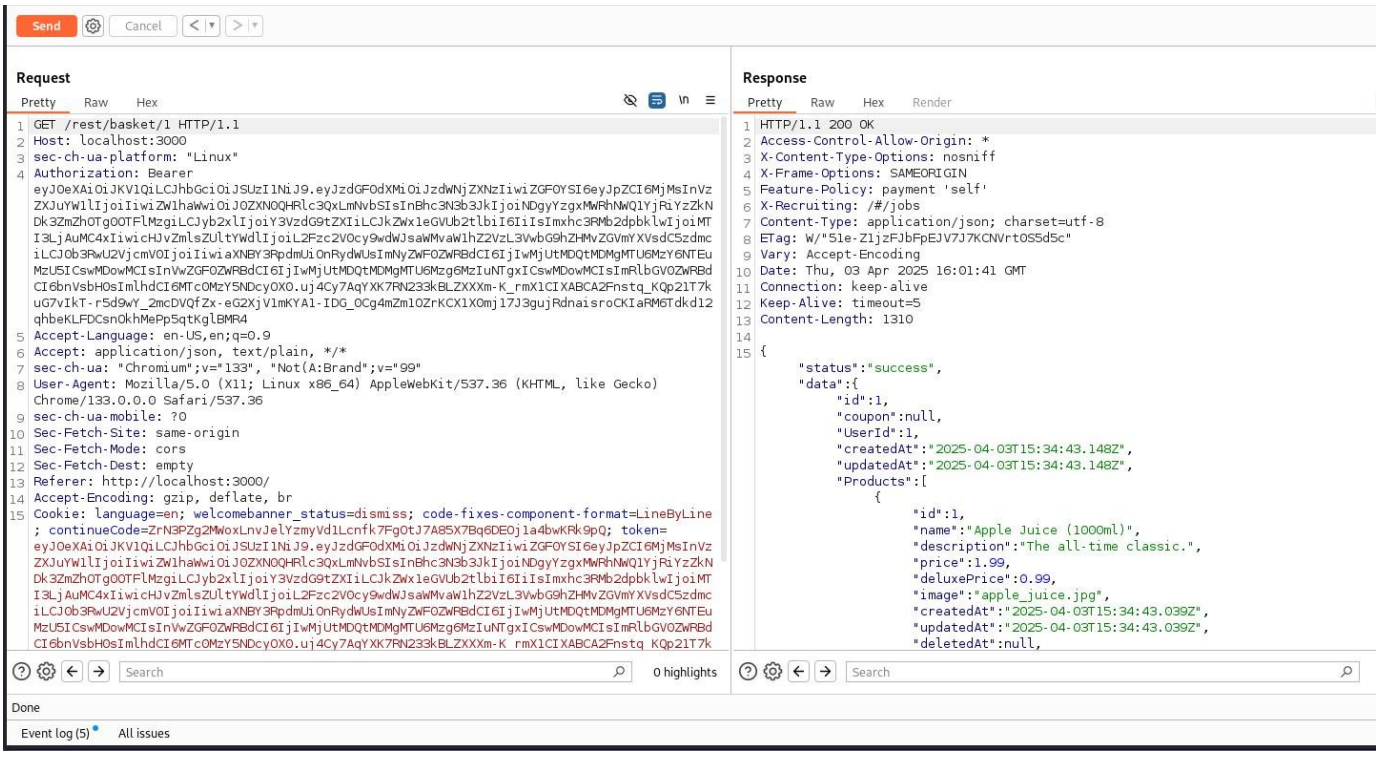
When you are logged in there is a new option that basket, or cart which contain your need to buy products. Lets intercept it and check the security



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

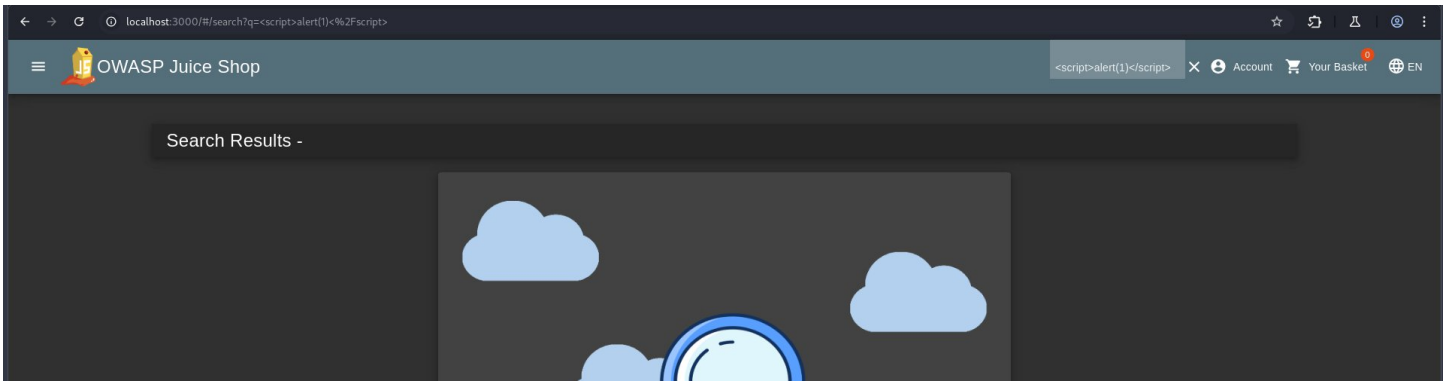
Now lets change the id = 1 from 6



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

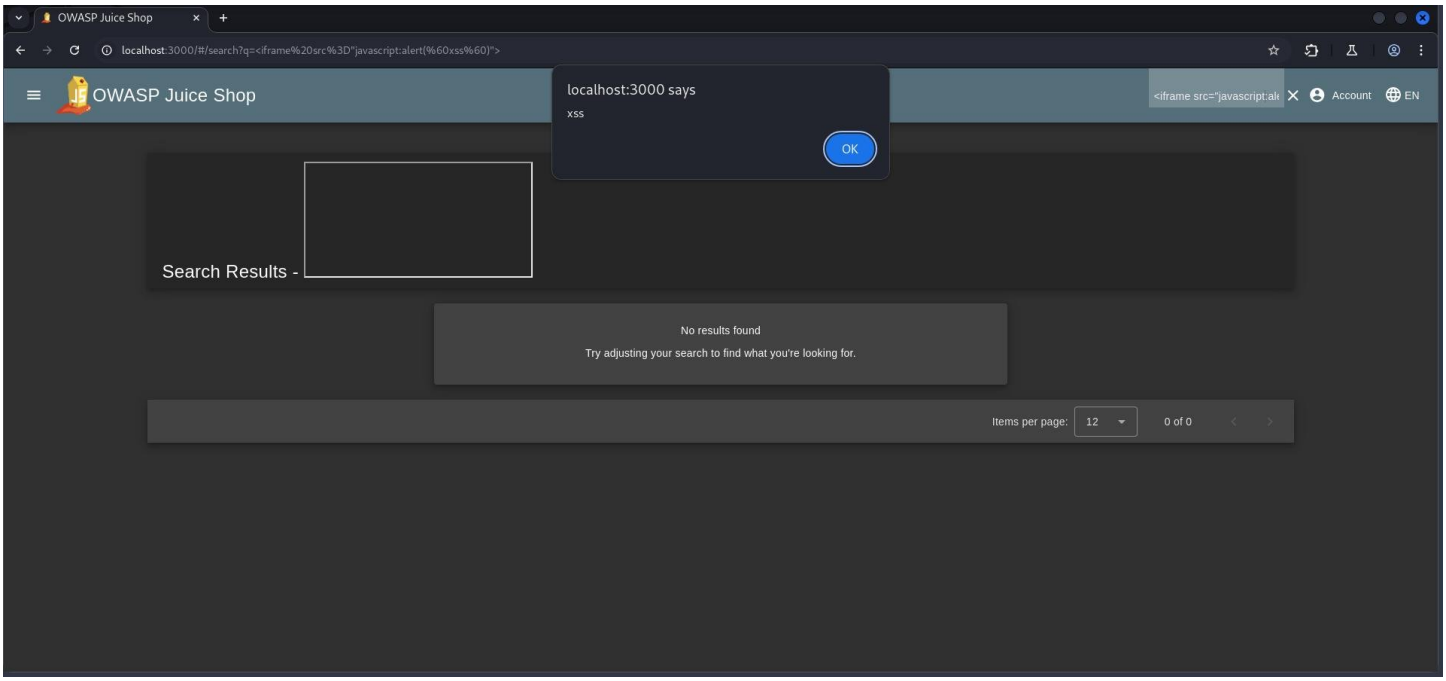
Lets try some famous payload : <script>alert(1)</script>



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>

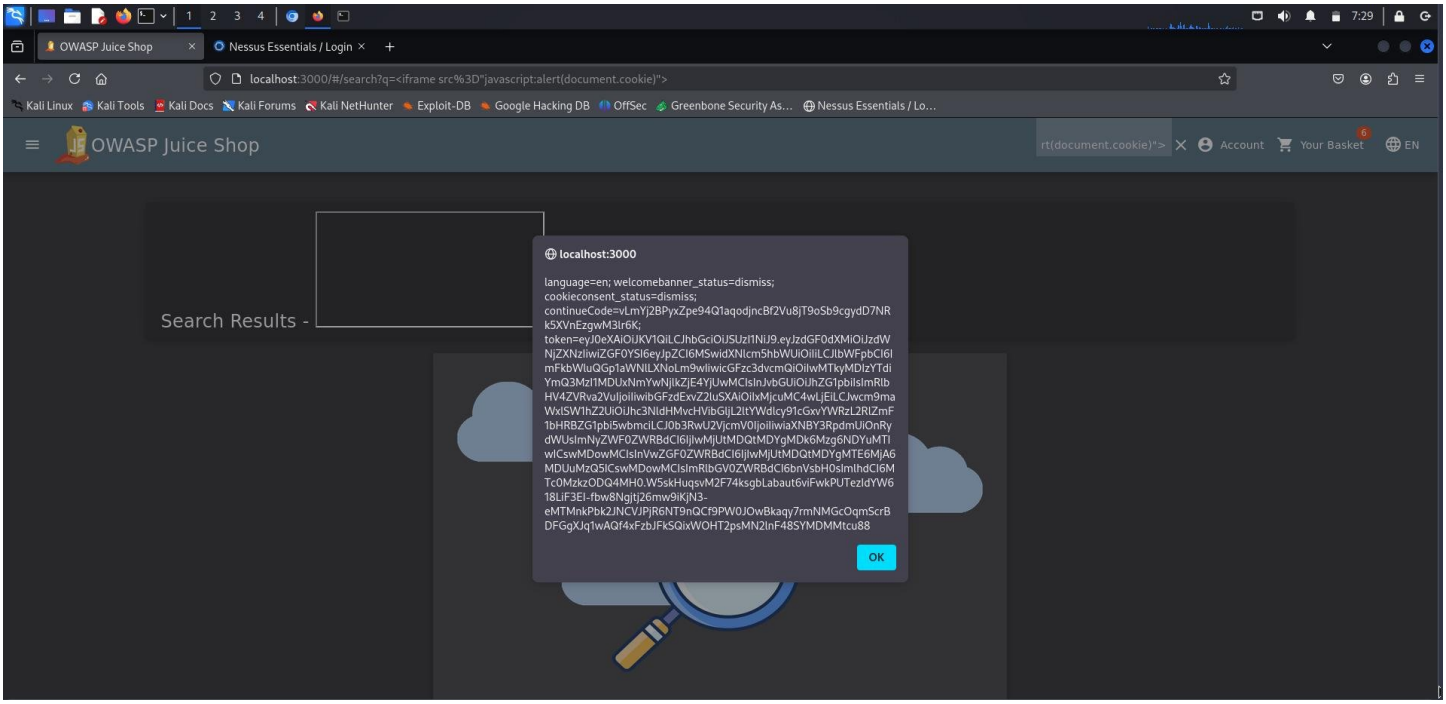


Yes the payload works the payload is **<iframe src="javascript:alert(`xss`)">** lets breakdown what it does

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 javasricpt 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.



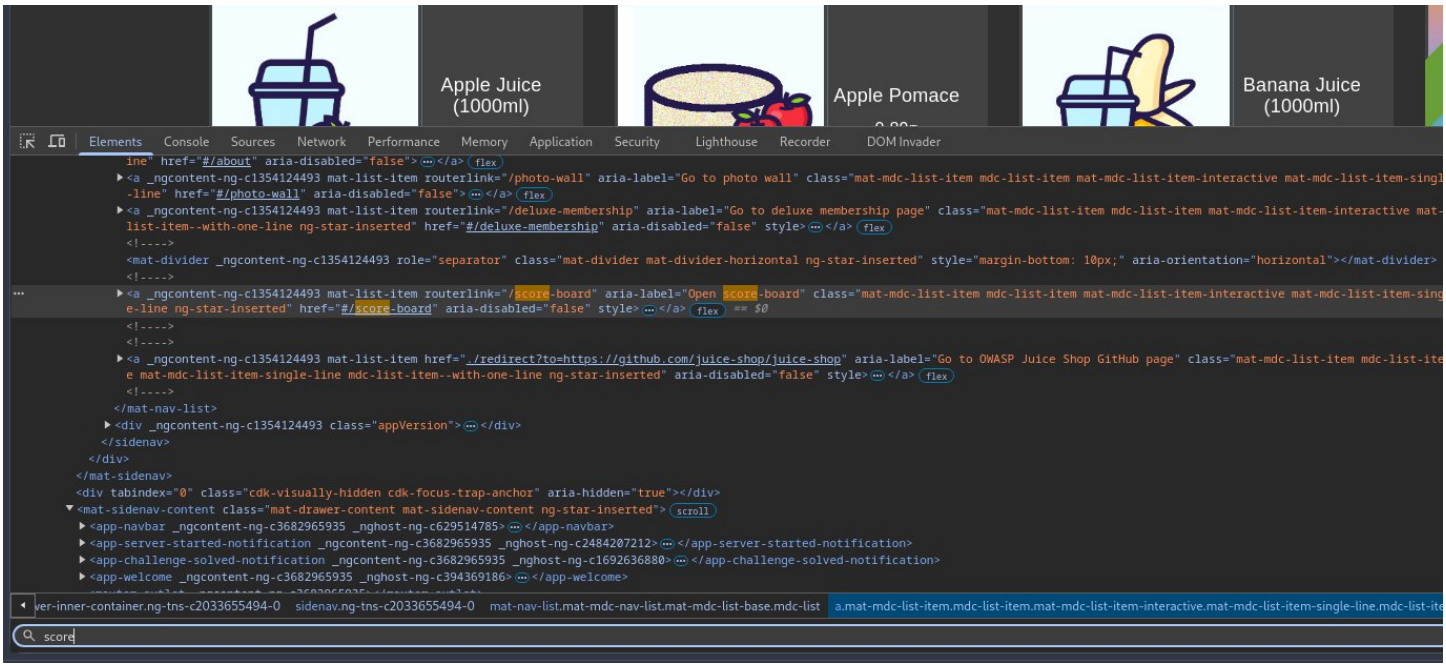
So the payload is this

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

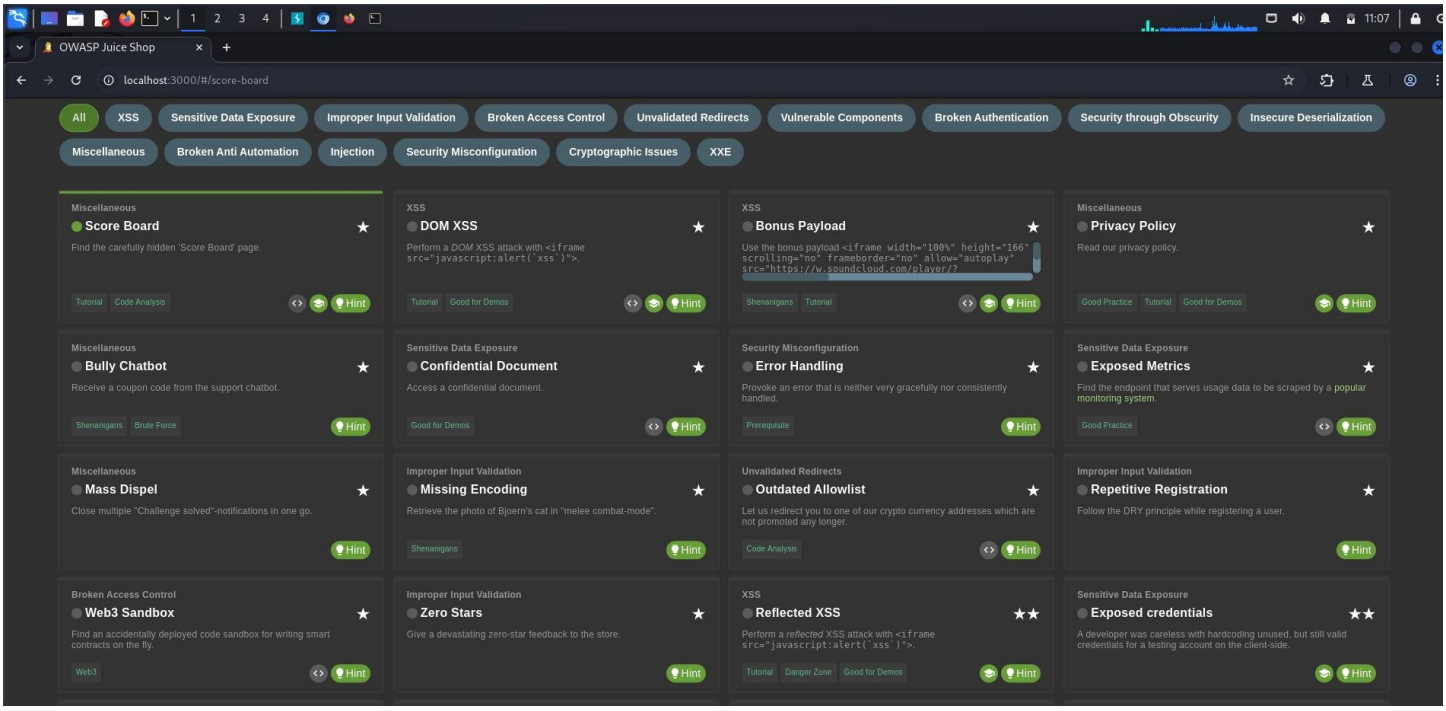
This <iframe> is using a JavaScript URI as its src attribute. javascript:alert(\xss`)` is an instruction to execute JavaScript.

So, when the browser loads this iframe, it will try to run this javasricpt 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



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



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.