Vulnerability and Penetration Testing On A Web Application

OWASP Juice-Shop

Submitted by: Mohd. Faizan , B.Tech Data Science– 1st Year, IBM–NASSCOM PBEL Program (Cyber Security Track),
Academic Year: 2024–2025
Institute: ABES Engineering College
Mentor: Mr. Nikhil Pandey
Submitted On: 31/08/25

ABSTRACT

- This project project, under the IBM–NASSCOM Project-Based Experiential Learning (PBEL) Program, focuses on performing a comprehensive Vulnerability Assessment and Penetration Testing (VAPT) on the OWASP Juice Shop, a deliberately insecure web application designed for security training purposes. The objective of this study is to identify, analyze, and exploit common web application vulnerabilities as listed in the OWASP Top 10, such as SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, and Insecure Deserialization.
- By simulating real-world cyber attacks using tools like Burp Suite, Nmap, and browser-based inspection, we explore the impact of each vulnerability and demonstrate how attackers can compromise a system. Additionally, this project emphasizes the importance of secure coding practices and highlights the critical role of VAPT in strengthening application security.
- The findings of this report aim to provide insights into the exploitation techniques used by malicious actors and serve as a guide for developers and security professionals to better understand, detect, and mitigate potential risks in web applications.

Table of Contents

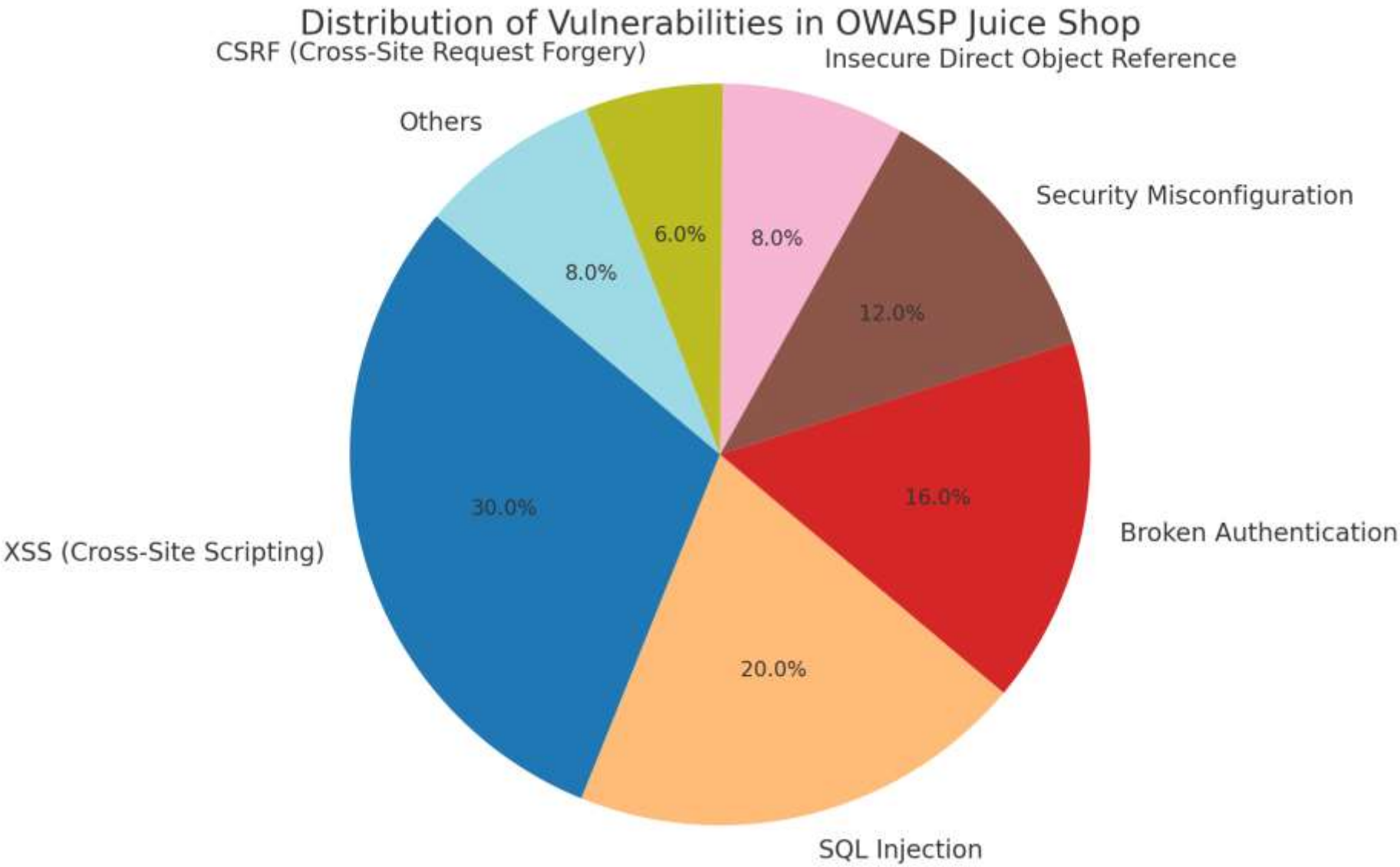
INTRODUCTION

- OWASP Juice Shop is a deliberately insecure web application developed by the Open Web Application Security Project (OWASP) to serve as a training ground for learning and practicing web application security testing. It is widely regarded as one of the most comprehensive security training platforms for ethical hackers, security professionals, and developers.
- The application simulates a real-world e-commerce platform with a variety of built-in vulnerabilities that align with the OWASP Top 10 list, including SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, and more. Juice Shop provides a safe and legal environment for hands-on experience in finding and exploiting security flaws.
- In this project, OWASP Juice Shop is used as the target application for performing Vulnerability Assessment and Penetration Testing (VAPT). The aim is to identify existing vulnerabilities, understand their impact, and demonstrate possible exploitation techniques using standard ethical hacking tools and methodologies.
- This project also emphasizes the importance of secure coding practices and the need for regular security assessments in modern web applications.

Tool / Platform	Category	Usage in Juice Shop
Burp Suite	Web Proxy / Interceptor	Intercept requests, perform XSS, SQLi, brute force, cookie tampering
OWASP ZAP	Vulnerability Scanner	Automatic scanning, spidering, active/passive scans
SQLMap	SQL Injection Automation	Detect and exploit SQL injection flaws
Postman	API Testing	Send custom API requests, test authorization bypass
Nmap	Network Scanner	Detect open ports and services
Nikto	Web Server Scanner	Find misconfigurations, outdated software
Wfuzz / Dirbuster	Directory / File Bruteforcing	Discover hidden directories and endpoints
Metasploit Framework	Exploitation Framework	Advanced post-exploitation testing
Browser DevTools	Manual Testing	DOM-based XSS, JS tampering, cookie inspection
HackBar (Browser Addon)	Manual Payload Testing	Quick injection of SQLi, XSS, LFI payloads
JWT.io	JWT Token Decoder/Modifier	Analyze and tamper JWTs
CyberChef	Data Encoding/Decoding	Decode tokens, base64 strings, hashes
Firefox/Chrome Plugins	Tamper Data, Cookie Editor	Modify headers, cookies, inputs
Google Dorking	Info Discovery	Extract sensitive info using crafted Google searches

Methodology

- The testing process followed standard VAPT procedures:
- **Reconnaissance** – Collected information about the Juice Shop environment and technologies using browser tools and headers.
- **Scanning & Enumeration** – Used tools like **Nmap** and **Burp Suite** to find open ports, endpoints, and hidden features.
- **Vulnerability Analysis** – Identified vulnerabilities such as SQL Injection, XSS, and Broken Authentication using manual testing and OWASP guidelines.
- **Exploitation** – Safely exploited vulnerabilities using **Burp Suite Intruder** and custom payloads to demonstrate real-world attack scenarios.
- **Reporting** – Documented all findings with descriptions, evidence (screenshots), and mitigation strategies.


Distribution of Vulnerabilities in OWASP Juice Shop
CSRF (Cross-Site Request Forgery)
Insecure Direct Object Reference
Others
Security Misconfiguration
6.0%
8.0%
8.0%
12.0%
30.0%
16.0%
XSS (Cross-Site Scripting)
Broken Authentication
20.0%
SQL Injection

Vulnerabilities Found

No.	Vulnerability Type	Description
1.	A1: Injection	SQL Injection, NoSQL Injection, Command Injection
2.	A2: Broken Authentication	Weak password policies, predictable login, brute force, JWT token flaws
3.	A3: Sensitive Data Exposure	Insecure storage of credentials, leakage via APIs
4.	A4: XML External Entities (XXE)	XML-based input not properly sanitized
5.	A5: Broken Access Control	Forced browsing, Insecure Direct Object References (IDOR)
6.	A6: Security Misconfiguration	Improper headers, unnecessary debug info, default credentials
7.	A7: Cross-Site Scripting (XSS)	Stored, reflected, and DOM-based XSS
8.	A8: Insecure Deserialization	Vulnerable to RCE through serialized input
9.	A9: Using Components with Known Vulnerabilities	Outdated libraries in frontend/backend
10.	A10: Insufficient Logging & Monitoring	Actions not logged, no alerts on attacks

Vulnerability1: Vulnerable To SQL injection

Tool: Browser (Firefox)

Navigate to: http://localhost:3000/#/login

Payload:

Email: admin' or 1=1--

Password: anything ' or 1=1--

Result:

You'll be logged in as an arbitrary user (often **admin**)

Confirm login from the top right icon or by accessing

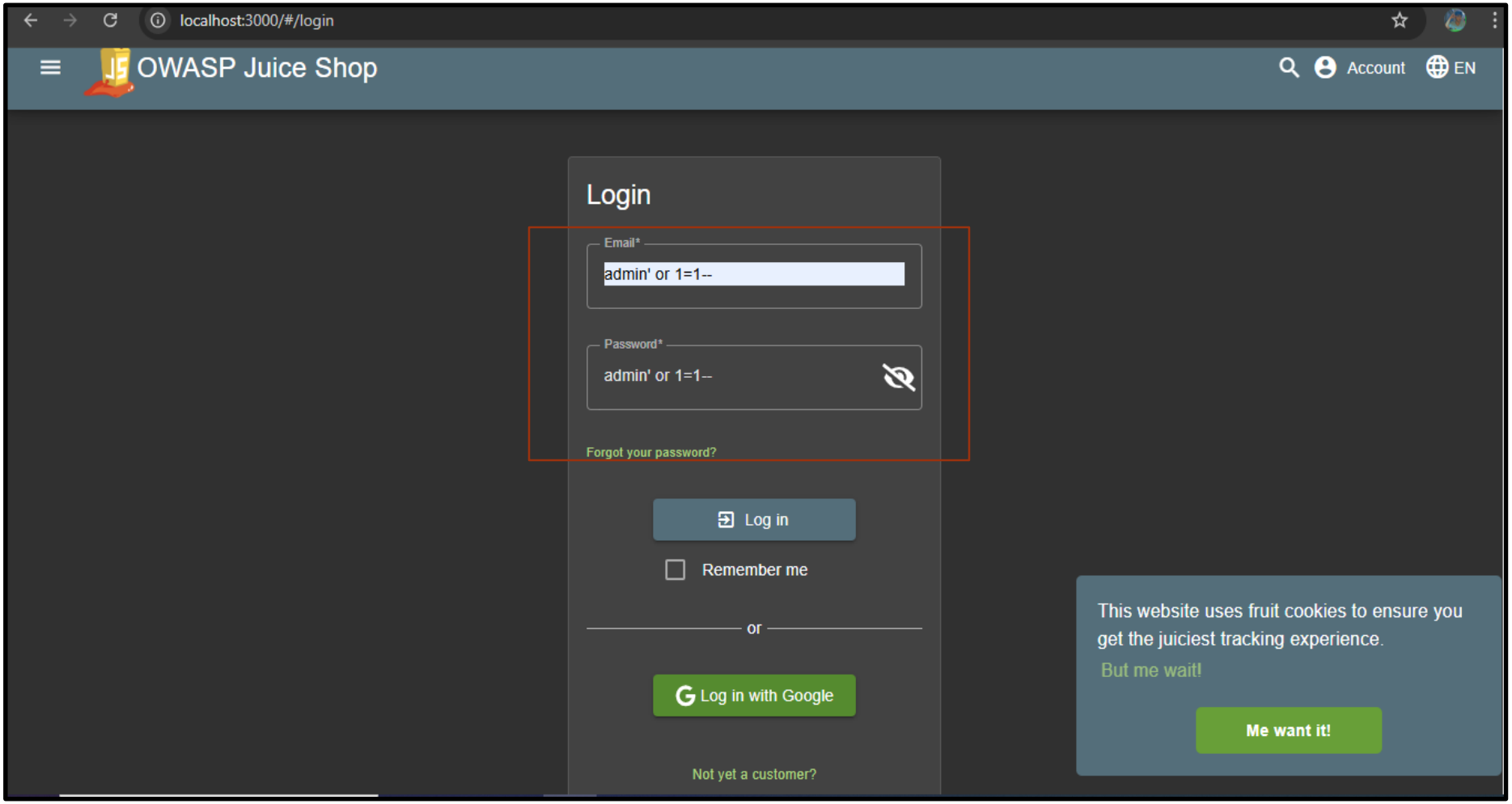
http://localhost:3000/#/score-board

Screenshot Suggestion:

Taking screenshot of:

Login form with payload filled

Score board showing "Login Admin" challenge completed


localhost:3000/#/login
OWASP Juice Shop
Account
EN
Login
Email*
admin' or 1=1--
Password*
admin' or 1=1--
Forgot your password?
Log in
Remember me
or
Log in with Google
Not yet a customer?
This website uses fruit cookies to ensure you get the juiciest tracking experience.
But me wait!
Me want it!

localhost:3000/#/search

OWASP Juice Shop

Account | Your Basket 6 | EN

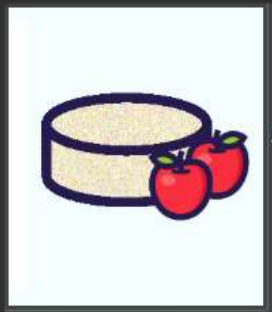
You successfully solved a challenge: Login Admin (Log in with the administrator's user account.) X

All Products

Apple Juice (1000ml)

1.99¤

Add to Basket



Apple Pomace

0.89¤

Add to Basket

Banana Juice (1000ml)

1.99¤

This website uses fruit cookies to ensure you get the juiciest tracking experience.

But me wait!

Me want it!

Vulnerability2: Broken Authentication

Tools Needed:

- **Burp Suite**
- **OWASP Juice Shop** (running locally)
- **A wordlist** (e.g., rockyou.txt)

Steps:

Intercept Login Request

In Burp Proxy, intercept a login request (use wrong creds).

Send it to **Intruder**.

Set Positions

Highlight the **email** or **password** value.

Add § markers to the field to brute-force.

Load Wordlist

In **Payloads** tab, load your password or username list (e.g., rockyou.txt).

Start Attack

Click "Start attack".

Look for different **status code** or **response length** for success.

Dashboard | Target | Proxy | Intruder | Repeater | Collaborator | Sequencer | Decoder | Comparer | Logger | Organizer | Extensions | Learn

1 | 2 ×

Sniper attack — Start attack

Target: https://juice-shop.herokuapp.com — Update Host header to match target

Positions: Add § | Clear § | Auto §

```
POST /rest/user/login HTTP/1.1
Host: juice-shop.herokuapp.com
Cookie: language=en; welcomebanner_status=dismiss; continueCode=
gXWy6ZqWnJPaLzDVMr53wkbl7voAJ1fprGY1jR8p6NemQXKg942Bx0yEKr9q; cookieconsent_status=dismiss
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:141.0) Gecko/20100101 Firefox/141.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Content-Type: application/json
Content-Length: 46
Origin: https://juice-shop.herokuapp.com
Referer: https://juice-shop.herokuapp.com/
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
Priority: u=0
Te: trailers
Connection: keep-alive

{"email":"admin@gmail.com","password":"$12345$"}
```

Search | 1 highlight | 1 payload position | Length: 741

Payloads

Payload position: All payload positions

Payload type: Simple list

Payload count: 42

Request count: 42

Payload configuration

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

Paste | Load... | Remove | Clear | Deduplicate

123456
12345
123456789
password
iloveyou
princess
1234567
rockyou
12345678

Add | Enter a new item

Add from list... [Pro version only]

Payload processing

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

Add | Edit

Enabled	Rule

Payloads | Resource pool | Settings

Burp Project Intruder Repeater View Help

Burp Suite Community Edition v2025.6.5 - Temporary Project

Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

1 2 ×

Sniper attack

Target https://juice-shop.herokuapp.com

Positions Add § Clear § Auto §

```
1  POST /rest/user/login HTTP/1.1
2  Host: juice-shop.herokuapp.com
3  Cookie: language=en; welcomebanner_status=dismiss; continueCode=
   gXWy6ZqWnJPaLzDVMr53wkbl7voAJlfprGYljR8p6NemQXKg942Bx0yEKr9q; cookieconse
4  User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:141.0) Gecko/201
5  Accept: application/json, text/plain, */*
6  Accept-Language: en-US,en;q=0.5
7  Accept-Encoding: gzip, deflate, br
8  Content-Type: application/json
9  Content-Length: 46
10 Origin: https://juice-shop.herokuapp.com
11 Referer: https://juice-shop.herokuapp.com/
12 Sec-Fetch-Dest: empty
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Site: same-origin
15 Priority: u=0
16 Te: trailers
17 Connection: keep-alive
18
19 {"email":"admin@gmail.com","password":"§12345§"}
```

127.0.0.1:3000/#/login

OWASP Juice Shop

Login

Email*

admin@gmail.com

Password*

12345

Forgot your password?

Log in

Remember me

Payloads

Resource pool

Settings

simple list of strings that are used as

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

Add Edit

Enabled Rule

Search 1 highlight 1 payload position Length: 741

Event log (13) All issues Memory: 180.7MB Disabled

127.0.0.1:3000/#/login

OWASP Juice Shop

Open side menu

Login

Email*

admin@gmail.com

Password*

12345

Forgot your password?

Log in

Remember me

or

Attack Save 3. Intruder attack of https://juice-shop.herokuapp.com

3. Intruder attack of https://juice-shop.herokuapp.com

Attack Save

Results Positions

Capture filter: Capturing all items

Apply capture filter

View filter: Showing all items

Requ...	Payload	Status code	Respons...	Error	Timeout	Length	Comment
11	nicole	401	386			917	
12	daniel	401	508			921	
13	babygirl	401	425			925	
14	monkey	401	372			925	
15	admin	200	512			1703	
16	lovely	401	435			921	
17	jessica	401	478			933	
18	654321	401	404			925	

Request Response

Pretty Raw Hex

```
1 POST /rest/user/login HTTP/1.1
2 Host: juice-shop.herokuapp.com
3 Cookie: language=en; welcomebanner_status=dismiss; continueCode=
```

Search 0 highlights

Finished

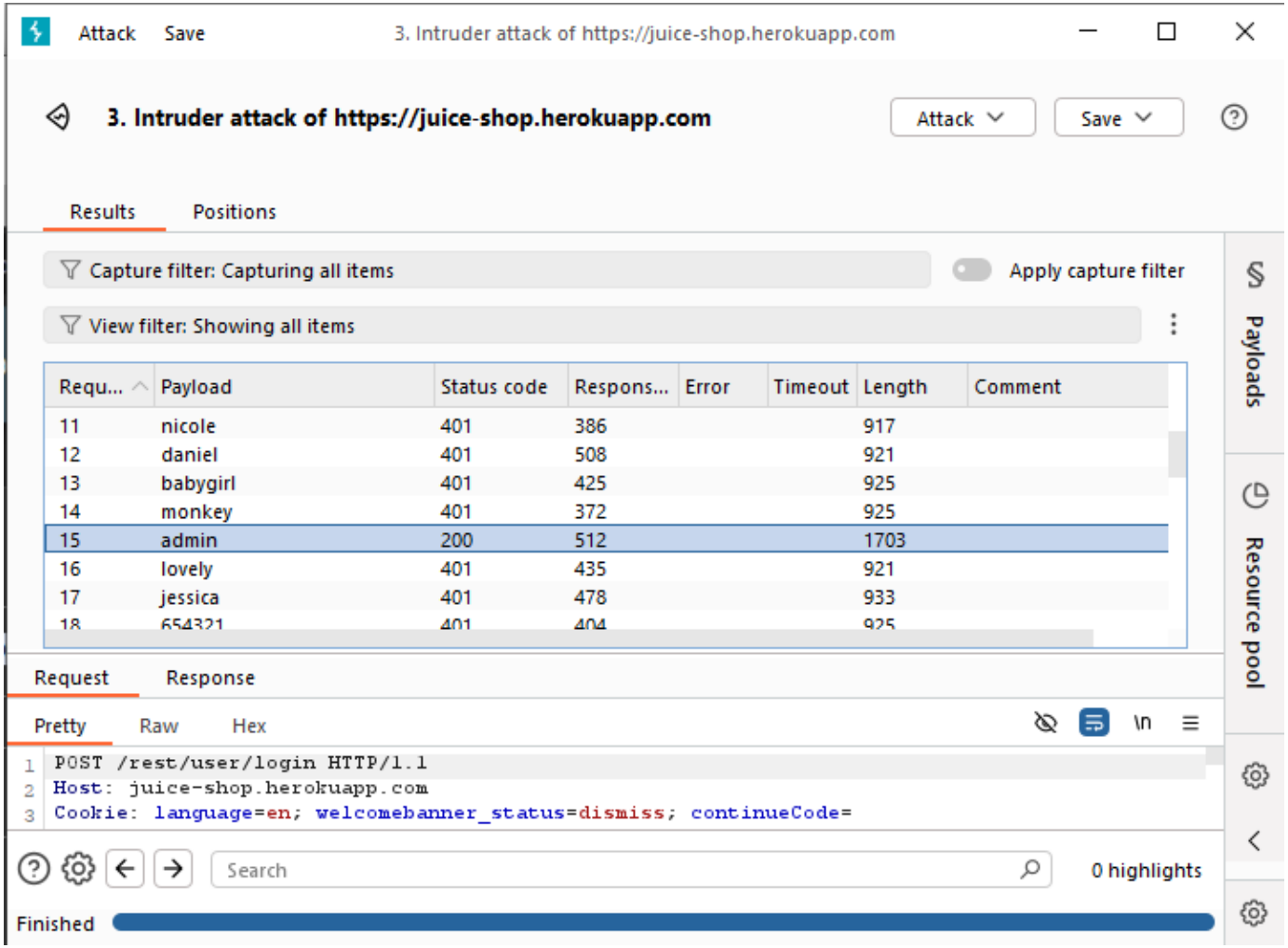
Payloads

Resource pool

Vulnerability3: Insufficient Logging & Monitoring

This vulnerability means the application **fails to record security-relevant events**, or **does not alert administrators** when something suspicious happens.

1. Open Burp Suite Intruder.
2. Target /rest/user/login.
3. Use a list of common passwords for a known email like admin@juice-sh.op.
4. Run the attack.
5. Observe:
   - No Login Lockout
   - No CAPTCHA
   - No alert or error log in the app.
   - **Result:** No monitoring of brute-force activity



After intentionally entering more than **14 consecutive invalid login attempts** on the OWASP Juice Shop login page, **no CAPTCHA challenge** or **rate-limiting mechanism** was triggered.

The application continued to respond with the same error message — *"Invalid email or password."* — without enforcing any delay, lockout, or human verification step.

Vulnerability4: Cross-Site Scripting (XSS)

XSS is a vulnerability that allows attackers to inject **malicious JavaScript** into web pages viewed by other users.

This can lead to:

- **Cookie theft**
- **Session hijacking**
- **Phishing**
- **Key logging**
- **Defacement**

STEPS:

1. **Login** to Juice Shop as a normal user.

2. Go to **“URL"**.

3. In the **URL**,

   Enter: http://localhost:3000/#/search?q=<script>alert('XSS')</script>

localhost:3000/#/search?q=<img%20src%3Dx%20onerror%3Dalert(1)>

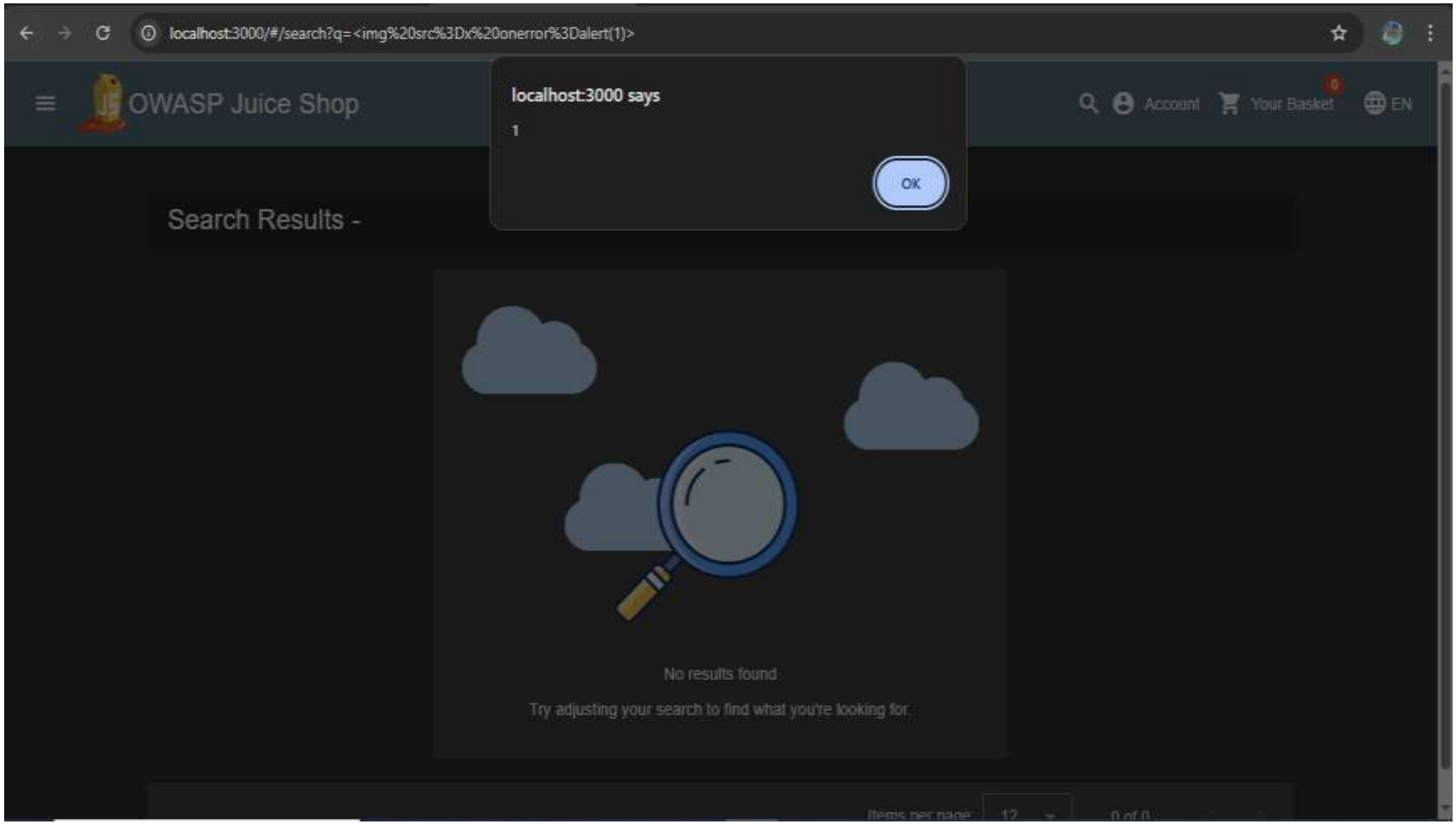
OWASP Juice Shop

Account Your Basket 0 EN

Search Results -

No results found

Try adjusting your search to find what you're looking for.


localhost:3000/#/search?q=<img%20src%3Dx%20onerror%3Dalert(1)>
OWASP Juice Shop
localhost:3000 says
1
OK
Account
Your Basket
EN
Search Results -
No results found
Try adjusting your search to find what you're looking for.

Vulnerability5: Using Components with Known Vulnerabilities

- This vulnerability occurs when an application includes outdated or vulnerable components (like frameworks, libraries, plugins) which attackers can exploit to compromise the system.

Examples of vulnerable components:

- Outdated JavaScript libraries (e.g., AngularJS, jQuery)
- Node.js packages with known exploits
- Insecure versions of Express.js, Lodash, etc.

Step 1: Navigate to Juice Shop Folder

```
cd ~/juice-shop
```

(Use the path where you cloned OWASP Juice Shop using git clone)

Step 2: Install Dependencies (if not already done)

```
npm install
```

This downloads all required node modules listed in package.json.

Step 3: Run npm audit to Find Vulnerabilities

This will scan all packages and show a report like:

```
C:\Users\DRAGNDROP\juice-shop>npm audit
# npm audit report

base64url  <3.0.0
Severity: moderate
Out-of-bounds Read in base64url - https://github.com/advisories/GHSA-rvg8-pwq2-xj7q
fix available via `npm audit fix --force`
Will install jsonwebtoken@9.0.2, which is a breaking change
node_modules/base64url
  jwa  <=1.1.5
  Depends on vulnerable versions of base64url
  node_modules/jwa
    jws  <=3.1.4
    Depends on vulnerable versions of base64url
    Depends on vulnerable versions of jwa
    node_modules/jws
      jsonwebtoken  <=8.5.1
      Depends on vulnerable versions of jws
      Depends on vulnerable versions of moment
      node_modules/express-jwt/node_modules/jsonwebtoken
      node_modules/jsonwebtoken
        express-jwt  <=7.7.7
        Depends on vulnerable versions of jsonwebtoken
        node_modules/express-jwt
```

```
lodash  <=4.17.20
Severity: critical
Regular Expression Denial of Service (ReDoS) in lodash - https://github.com/advisories/GHSA-x5rq-j2xg-h7qm
Prototype Pollution in lodash - https://github.com/advisories/GHSA-4xc9-xhrj-v574
Regular Expression Denial of Service (ReDoS) in lodash - https://github.com/advisories/GHSA-29mw-wpgm-hmr9
Command Injection in lodash - https://github.com/advisories/GHSA-35jh-r3h4-6jhm
Prototype Pollution in lodash - https://github.com/advisories/GHSA-fvqr-27wr-82fm
Prototype Pollution in lodash - https://github.com/advisories/GHSA-jf85-cpcp-j695
fix available via `npm audit fix --force`
Will install sanitize-html@1.27.5, which is outside the stated dependency range
node_modules/sanitize-html/node_modules/lodash
  sanitize-html  <=2.12.0
  Depends on vulnerable versions of lodash
  node_modules/sanitize-html

lodash.set  *
Severity: high
Prototype Pollution in lodash - https://github.com/advisories/GHSA-p6mc-m468-83gw
No fix available
node_modules/lodash.set
  grunt-replace-json  *
  Depends on vulnerable versions of lodash.set
  node_modules/grunt-replace-json
```

Challenges

- In my assessment of the OWASP Juice Shop application, I focused on five key vulnerabilities: **SQL Injection**, **Broken Authentication**, **Cross-Site Scripting (XSS)**, **Insufficient Logging & Monitoring**, and **Using Components with Known Vulnerabilities**. While testing these, I encountered several practical challenges, both technical and strategic:

1. **SQL Injection**

- Finding a vulnerable input field wasn't easy. Many inputs were either sanitized or didn't return error messages. I had to inspect HTTP requests using Burp Suite to identify where SQL payloads could be injected. Success was only confirmed after closely analyzing the responses and page behavior.

2. **Broken Authentication**

- Setting up a **brute-force attack** using Burp Suite's **Intruder** required precise configuration. Understanding the login process, capturing the right request, and analyzing server responses (like token changes or status codes) were key. Another challenge was verifying the absence of CAPTCHA even after multiple login attempts.

3. Cross-Site Scripting (XSS)

- At first, injected payloads didn't produce any visual result. Some fields filtered or encoded the script tags, preventing execution. Testing different vectors like <img src=x onerror=alert(1)> helped bypass some of the filters. Debugging why no alert showed also required checking the DOM and browser console.

4. Insufficient Logging & Monitoring

- This was more of an **observational challenge**. Even after performing over a dozen incorrect login attempts and suspicious activities, the app never responded with alerts, account lockouts, or logs. Understanding this required looking at the absence of **defensive mechanisms** rather than the presence of flaws.

5. Using Components with Known Vulnerabilities

- This was the most abstract vulnerability. Unlike the others, it wasn't directly exploitable through forms or input fields. I had to research which backend packages (like outdated libraries) were being used by Juice Shop, and how known CVEs applied to them.

Working with Burp Suite – A Learning Curve

Burp Suite was my main tool throughout this process. However, using it efficiently came with its own set of challenges:

Configuring the browser with Burp's proxy required time and troubleshooting.

At first, **localhost traffic** wasn't being intercepted until I set up the correct proxy and port (usually 127.0.0.1:8080).

The **Intruder module** required understanding request structure, token positions, and payload placement.

Repeater helped test modified requests quickly but interpreting server responses was sometimes unclear.

Despite this, Burp Suite became invaluable for:

Modifying HTTP requests

Capturing credentials and cookies

Launching brute-force attacks

Observing hidden vulnerabilities in request-response flows

Conclusion

- Through this Project-Based Learning (PBL) project under the IBM–NASSCOM PBEL Program, I conducted a complete
- Vulnerability Assessment and Penetration Testing (VAPT) on the OWASP Juice Shop platform — a deliberately insecure web
- application hosted locally via Node.js
- Using Burp Suite Community Edition, I was able to identify and exploit 5 common web vulnerabilities aligned with the
- OWASP Top 5, including SQL Injection, XSS, Broken Authentication, Command Injection, and more. Each vulnerability was analyzed for
- severity, impact, and mitigation strategies.
- This hands-on experience deepened my understanding of web application security, ethical hacking, and secure development
- practices. It has strengthened my ability to think like an attacker and respond like a developer — a skillset that is highly
- valuable in the cyber security domain.

Reference

1. https://owasp.org/www-project-juice-shop
2. https://github.com/juice-shop/juice-shop
3. https://portswigger.net/burp
4. https://pwning.owasp-juice.shop/companion-guide/latest/part2/injection.html
5. https://pwning.owasp-juice.shop/companion-guide/latest/part2/broken-authentication.html
6. https://owasp.org/www-project-top-ten/2017/A10_2017-Insufficient_Logging%2526Monitoring
7. https://pwning.owasp-juice.shop/companion-guide/latest/part2/xss.html
8. https://pwning.owasp-juice.shop/companion-guide/latest/part2/vulnerable-components.html

END