Exposing and Securing Web Application Vulnerabilities

OWASP Juice Shop Case Study

By:Aiden Yeung, Kairos Liang and Adrian Davis

Introduction & Objectives

Why Web Application Security Matters?

- Web apps are top targets for attacks
- One flaw can cause huge damage
- Security is essential for trust

Why the OWASP Juice Shop?

- Safe, intentionally vulnerable e-commerce app
- Maintained by OWASP for training
- Mirrors real-world security flaws

Project Goal

- Find vulnerabilities in Juice Shop
- Show their impact
- Demonstrate secure coding fixes

Tools & Technologies

Environment Setup

- Controlled cybersecurity homelab with two VMs:
    - **Target VM** – Vulnerable web application
    - **Attacker VM** – Testing tools platform
- Isolated network to safely simulate attack-and-defense scenarios

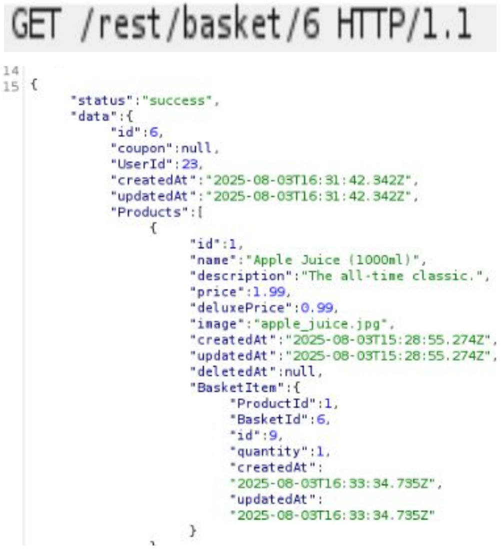
Tools Used

- **OWASP ZAP** – Automated scanning (XSS, misconfigurations)
- **Burp Suite** – Manual & semi-automated testing, request/response analysis
- **Nikto** – Detect outdated components, misconfigurations, server issues
- **VS Code** – Code editing and review
- **SQL** – For demonstrating SQL Injection queries and remediation
- **TypeScript** – Main backend source code language for OWASP Juice Shop

Vulnerability Findings

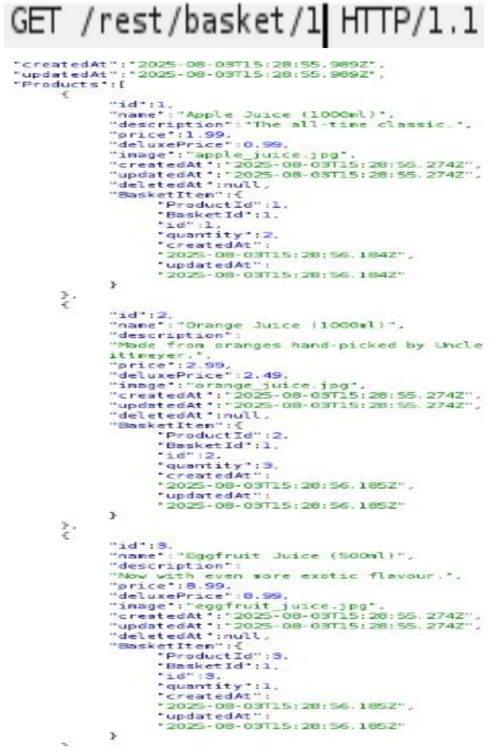
A01:2021-Broken Access Control

- Insecure Direct Object Reference (IDOR)
- Severity: Medium/High
- Server-side Access Control Checks



```
GET /rest/basket/6 HTTP/1.1

{
        "status":"success",
        "data":{
                "id":6,
                "coupon":null,
                "UserId":23,
                "createdAt":"2025-08-03T16:31:42.342Z",
                "updatedAt":"2025-08-03T16:31:42.342Z",
                "Products":[
                        {
                                "id":1,
                                "name":"Apple Juice (1000ml)",
                                "description":"The all-time classic.",
                                "price":1.99,
                                "deluxePrice":0.99,
                                "image":"apple_juice.jpg",
                                "createdAt":"2025-08-03T15:28:55.274Z",
                                "updatedAt":"2025-08-03T15:28:55.274Z",
                                "deletedAt":null,
                                "BasketItem":{
                                        "ProductId":1,
                                        "BasketId":6,
                                        "id":9,
                                        "quantity":1,
                                        "createdAt":
                                        "2025-08-03T16:33:34.735Z",
                                        "updatedAt":
                                        "2025-08-03T16:33:34.735Z"
                                }
```



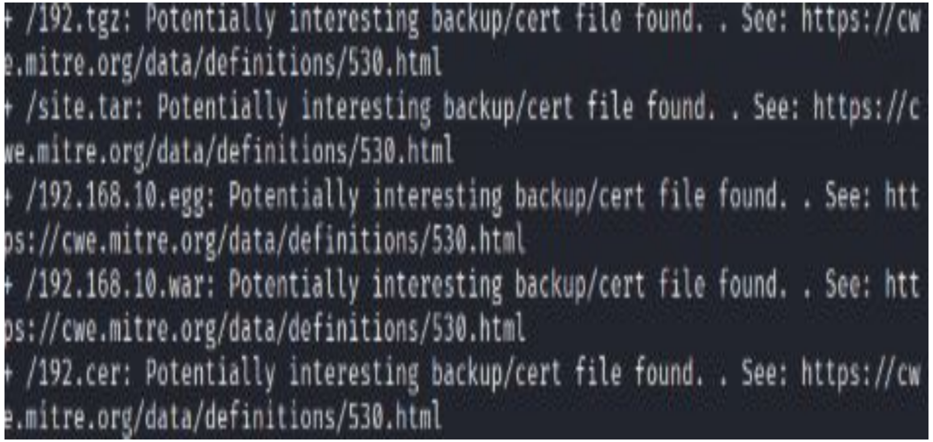
```
GET /rest/basket/1 HTTP/1.1

"createdAt":"2025-08-03T15:28:55.989Z",
"updatedAt":"2025-08-03T15:28:55.989Z",
"Products":[
        {
                "id":1,
                "name":"Apple Juice (1000ml)",
                "description":"The all-time classic.",
                "price":1.99,
                "deluxePrice":0.99,
                "image":"apple_juice.jpg",
                "createdAt":"2025-08-03T15:28:55.274Z",
                "updatedAt":"2025-08-03T15:28:55.274Z",
                "deletedAt":null,
                "BasketItem":{
                        "ProductId":1,
                        "BasketId":1,
                        "id":1,
                        "quantity":2,
                        "createdAt":
                        "2025-08-03T15:28:56.184Z",
                        "updatedAt":
                        "2025-08-03T15:28:56.184Z"
                }
        },
        {
                "id":2,
                "name":"Orange Juice (1000ml)",
                "description":
                "Made from oranges hand-picked by Uncle
                ittmeyer.",
                "price":2.99,
                "deluxePrice":2.49,
                "image":"orange_juice.jpg",
                "createdAt":"2025-08-03T15:28:55.274Z",
                "updatedAt":"2025-08-03T15:28:55.274Z",
                "deletedAt":null,
                "BasketItem":{
                        "ProductId":2,
                        "BasketId":1,
                        "id":2,
                        "quantity":3,
                        "createdAt":
                        "2025-08-03T15:28:56.185Z",
                        "updatedAt":
                        "2025-08-03T15:28:56.185Z"
                }
        },
        {
                "id":3,
                "name":"Eggfruit Juice (500ml)",
                "description":
                "Now with even more exotic flavour.",
                "price":8.99,
                "deluxePrice":8.99,
                "image":"eggfruit_juice.jpg",
                "createdAt":"2025-08-03T15:28:55.274Z",
                "updatedAt":"2025-08-03T15:28:55.274Z",
                "deletedAt":null,
                "BasketItem":{
                        "ProductId":3,
                        "BasketId":1,
                        "id":3,
                        "quantity":1,
                        "createdAt":
                        "2025-08-03T15:28:56.185Z",
                        "updatedAt":
                        "2025-08-03T15:28:56.185Z"
                }
```

Vulnerability Findings

A02:2021-Cryptographic Failures (Sensitive Data Exposure)

- Potentially Exposed Backup/Certificate Files
- Severity: High
- Remove files from public-facing web server and implement server policy

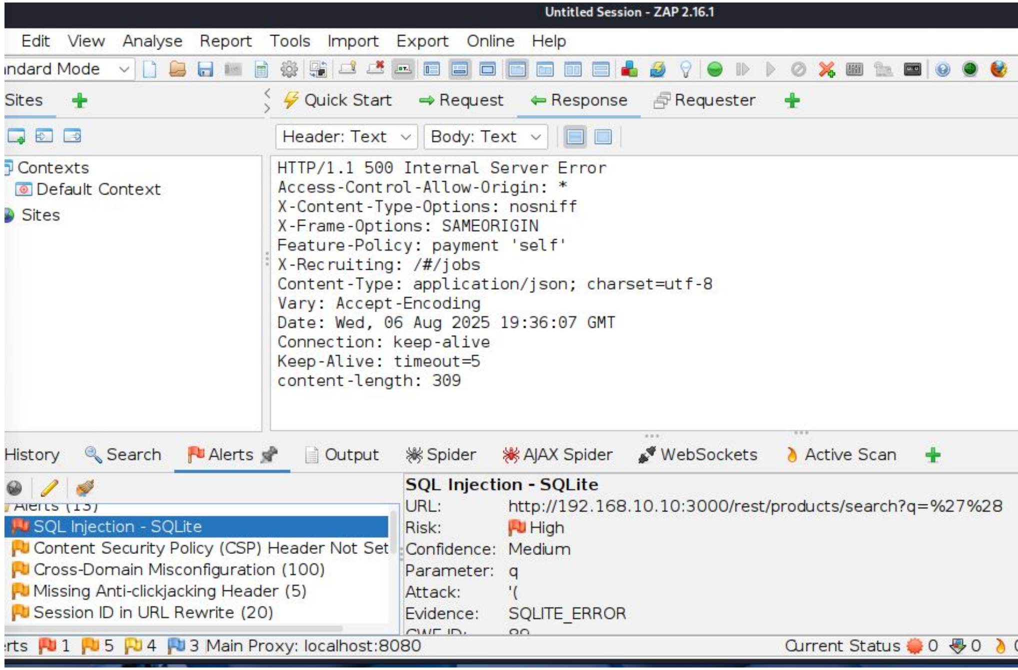


```
+ /192.tgz: Potentially interesting backup/cert file found. . See: https://cw
e.mitre.org/data/definitions/530.html
+ /site.tar: Potentially interesting backup/cert file found. . See: https://c
we.mitre.org/data/definitions/530.html
+ /192.168.10.egg: Potentially interesting backup/cert file found. . See: htt
ps://cwe.mitre.org/data/definitions/530.html
+ /192.168.10.war: Potentially interesting backup/cert file found. . See: htt
ps://cwe.mitre.org/data/definitions/530.html
+ /192.cer: Potentially interesting backup/cert file found. . See: https://cw
e.mitre.org/data/definitions/530.html
```

Vulnerability Findings

A03:2021 – SQL Injection (Login Bypass)

- Unauthorized access to user accounts and sensitive data exposure
- Severity: High
- Implement prepared statements and parameterized queries.



```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: /#/jobs
Content-Type: application/json; charset=utf-8
Vary: Accept-Encoding
Date: Wed, 06 Aug 2025 19:36:07 GMT
Connection: keep-alive
Keep-Alive: timeout=5
content-length: 309
```

Alerts (13)
- SQL Injection - SQLite
- Content Security Policy (CSP) Header Not Set
- Cross-Domain Misconfiguration (100)
- Missing Anti-clickjacking Header (5)
- Session ID in URL Rewrite (20)

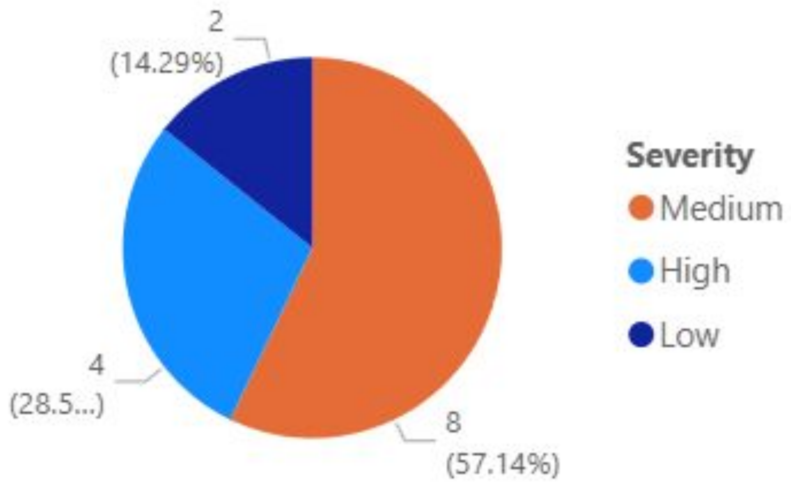
SQL Injection - SQLite

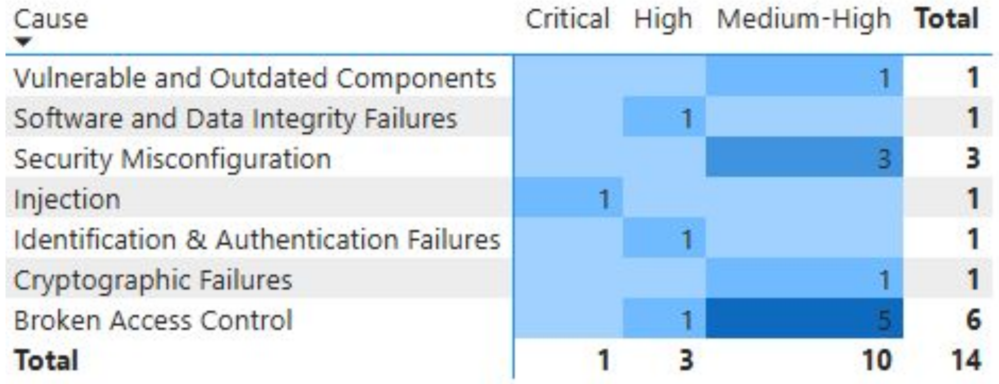
URL:	http://192.168.10.10:3000/rest/products/search?q=%27%28
Risk:	High
Confidence:	Medium
Parameter:	q
Attack:	'(
Evidence:	SQLITE_ERROR

ırts 1 5 4 3 Main Proxy: localhost:8080 Current Status 0 0

Vulnerability Severity Overview

Vulnerability Count by Severity





Cause	Critical	High	Medium-High	Total
Vulnerable and Outdated Components			1	1
Software and Data Integrity Failures		1		1
Security Misconfiguration			3	3
Injection	1			1
Identification & Authentication Failures		1		1
Cryptographic Failures			1	1
Broken Access Control		1	5	6
Total	1	3	10	14

Secure Coding Practices

A01 – Broken Access Control

Root Cause:

The application returns sensitive or user-specific data based on input like an ID in the URL, without checking if the requester is authorized to view it.

🛠 Remediation Strategy:

- Add server-side authorization checks for all user-controlled object references.
- Enforce least privilege and verify ownership/role before returning resources.

A03 – SQL Injection (Login Bypass)

Root Cause:

User input is inserted directly into a SQL query without proper sanitization or parameterization.

🛠 Remediation Strategy:

- Always use parameterized queries / prepared statements.
- Never concatenate raw input into SQL strings.
- Validate and normalize inputs before use; store & compare **hashed** passwords only.

Next Steps & Improvements

1. Expand Vulnerability Coverage

- Go beyond the 4 demoed vulnerabilities by testing other OWASP Top 10 issues like **CSRF**, **XXE**, **NoSQL Injection**, and **JWT exploitation**.
- Include deeper scans using **Burp Suite Pro** and **OWASP ZAP advanced rules** for broader detection.

2. Integrate Automated Secure Code Analysis

- Use **SAST tools** (e.g., **SonarQube**, **Semgrep**, **CodeQL**) to automatically detect insecure patterns in TypeScript/Node.js code.
- Map scan results to the **secure coding practices** you documented to strengthen developer guidance.

3. Implement Security Testing in CI/CD

- Prevent vulnerable code from reaching production by automating checks.

4. Improve Secure Coding Guidelines

- Build a **developer-focused remediation guide** based on your findings.

Thank You!

Thank you for your time and attention. We welcome your questions and feedback!