Security Audit Report

Target Application: OWASP Juice Shop

---

1. Author

Bushra Saleem

Cybersecurity Learner | Aspiring SOC Analyst

---

2. Introduction

This project involved performing a **white-box security audit** on an intentionally vulnerable web application called **OWASP Juice Shop**. The aim was to understand how common web application vulnerabilities occur in modern applications and how attackers may exploit them.

OWASP Juice Shop is a training platform designed to demonstrate real-world security issues. It is built using modern technologies such as **Node.js**, **Express**, and **Angular**, which makes it a good example of real production-style web applications.

The application was deployed locally using **Docker** on **Kali Linux**, ensuring that no real or live systems were harmed. Security testing was conducted using standard tools such as **Burp Suite** for intercepting and analyzing HTTP requests, **Nmap** for basic reconnaissance, and **Firefox** as the testing browser.

---

3. Application Setup and Environment

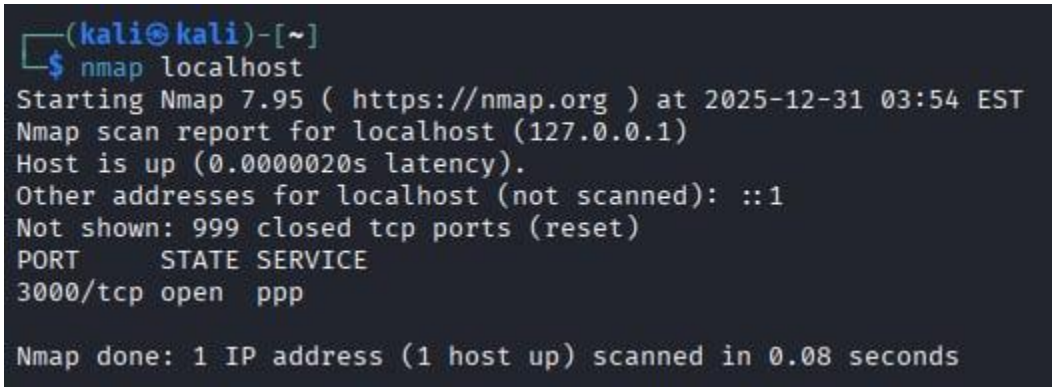
The OWASP Juice Shop application was deployed locally using **Docker** on Kali Linux. Running the application locally ensured that all testing was conducted in a safe and controlled environment without affecting any live systems.

Setup Steps

- A terminal was opened in the Kali Linux environment.
- The OWASP Juice Shop Docker image was downloaded using the following command

Service Verification

- To confirm that the application was running and listening on the correct port, an Nmap scan was performed:



The scan results confirmed that a web service was active on **port 3000**, which matched the Juice Shop application's expected configuration.

---

4. Tools Used and Their Purpose

The following tools were used during the security assessment:

- **Kali Linux**
  Used as the primary penetration testing operating system, providing built-in security tools.

- **Docker**
  Used to run OWASP Juice Shop in an isolated containerized environment.

- **Nmap**
  Used to verify running services and confirm that the application was accessible on the expected port.

- **Burp Suite**
  Used to intercept, inspect, modify, and replay HTTP requests between the browser and the application for vulnerability testing.

- **Firefox Browser**
  Used to interact with the application while routing traffic through Burp Suite.

---

5. Methodology Overview

The testing process followed a simple and structured approach:

1. **Application Reconnaissance**
   The application was explored as a normal user to understand available features such as login, registration, product browsing, and user baskets.
2. **Traffic Analysis**
   Burp Suite was configured as a proxy to observe how the browser communicates with the server.
3. **Vulnerability Testing**
   Requests were intercepted and modified to test how the application handles invalid or malicious input.
4. **Exploitation and Confirmation**
   When unexpected behavior was observed, the issue was confirmed as a vulnerability.
5. **Documentation**
   Each confirmed vulnerability was documented with steps, evidence, and remediation suggestions.

---

6. Vulnerability Findings

A total of **five vulnerabilities** were successfully identified and exploited across multiple OWASP Top 10 categories.

Vulnerability 1: Scoreboard Access Without Authorization

OWASP Category: A01 – Broken Access Control

Description

The Juice Shop Scoreboard page, which shows challenge progress and internal details, was accessible without proper authorization. This page should normally be restricted to authorized users only.

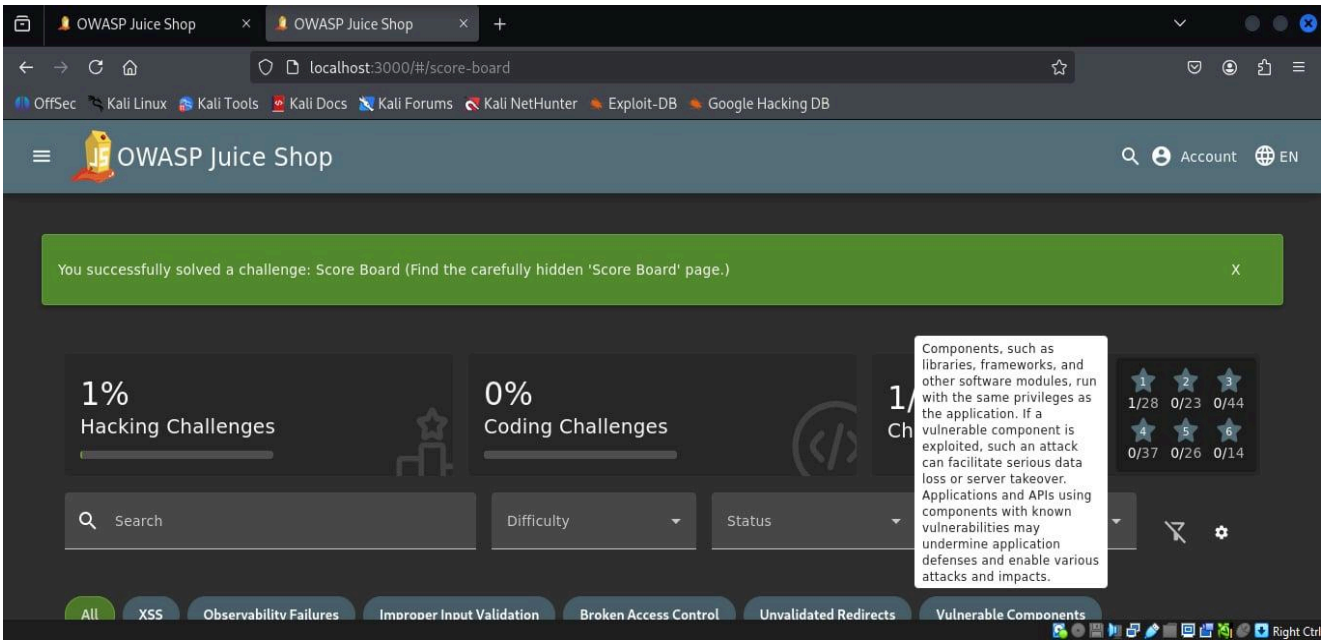
Steps to Reproduce

1. The tester opened the Juice Shop application in a web browser using Firefox.
2. Using the browser's developer tools (**F12 / Inspect**), the tester explored the network activity and page structure to understand application endpoints.
3. While exploring, the tester discovered the Scoreboard endpoint by examining requests and responses in the **Network** tab.

4. Without logging in as an administrator, the tester manually sent a request to the Scoreboard endpoint using the browser.
5. The application responded by displaying the Scoreboard page successfully, confirming that access control was not enforced for this sensitive functionality.

Evidence

- The Scoreboard page was visible and accessible without logging in as an administrator, confirming the absence of proper authorization checks.



Impact

Unauthorized users can gain insight into application structure and challenges, which can assist attackers in identifying additional attack vectors.

Remediation

The application should enforce server-side access control checks for sensitive pages such as the Scoreboard. Only authenticated and authorized users (for example, administrators) should be able to access this functionality. Direct access to protected endpoints should be blocked if proper authorization is not present.

Vulnerability 2: SQL Injection in Login Functionality

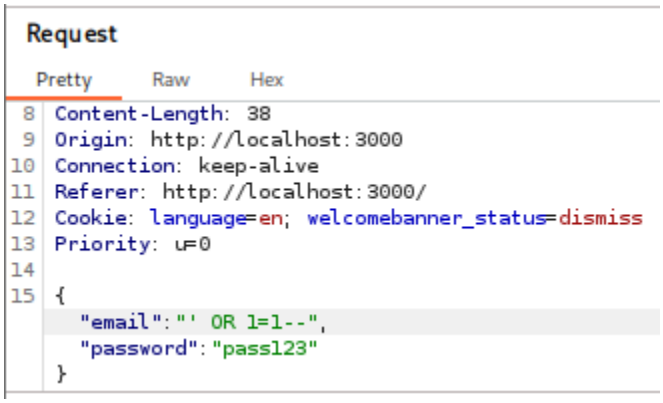
OWASP Category: A03 – Injection

Description

The login functionality did not properly validate user input, allowing SQL Injection to bypass authentication.

Steps to Reproduce

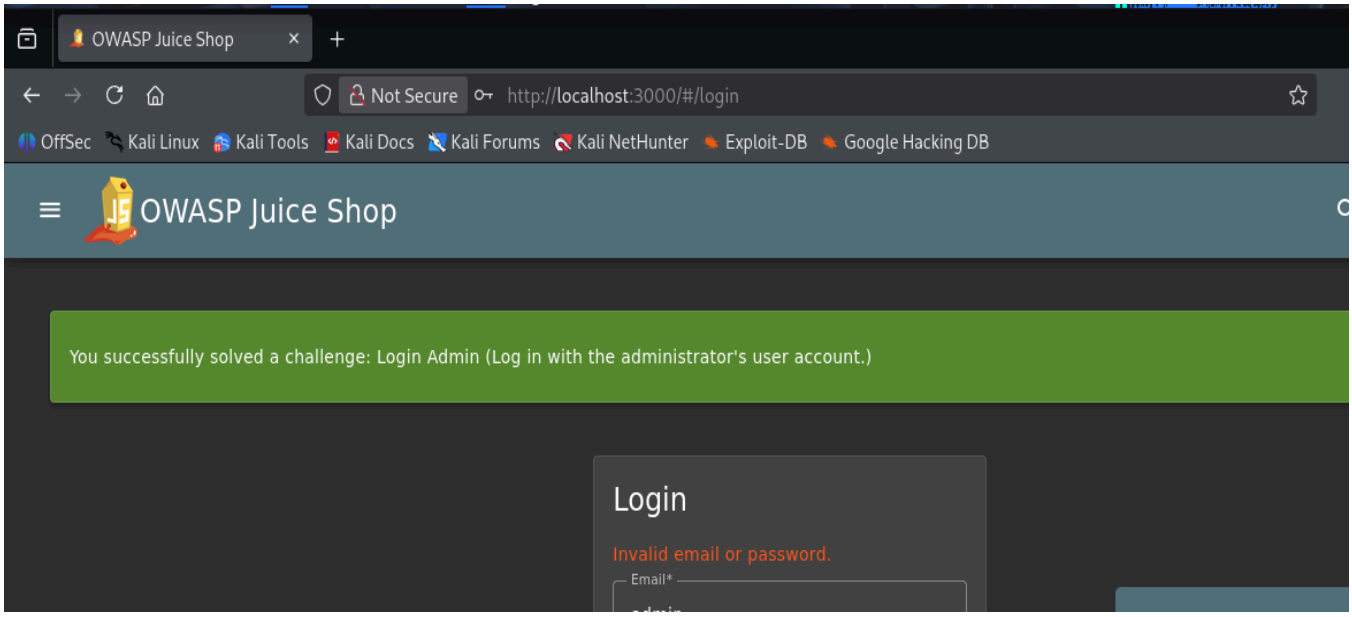
1. The tester navigated to the login page of the application.
2. The login request was intercepted using Burp Suite.
3. The intercepted request was sent to Burp Repeater for testing.
4. In Repeater, the email field was modified using the following payload:
   ' OR 1=1--



5. The modified request was sent to the server.

Evidence

- The application logged in successfully without valid credentials.



Impact

Attackers can gain unauthorized access to user accounts, including privileged accounts.

Remediation

User input should never be directly embedded into database queries. The application should use parameterized queries or prepared statements to safely handle user input. Additionally, proper input validation should be implemented to reject suspicious or malformed input before it reaches the database layer.

Vulnerability 3: Insecure Direct Object Reference (IDOR)

OWASP Category: A07 – Identification and Authentication Failures

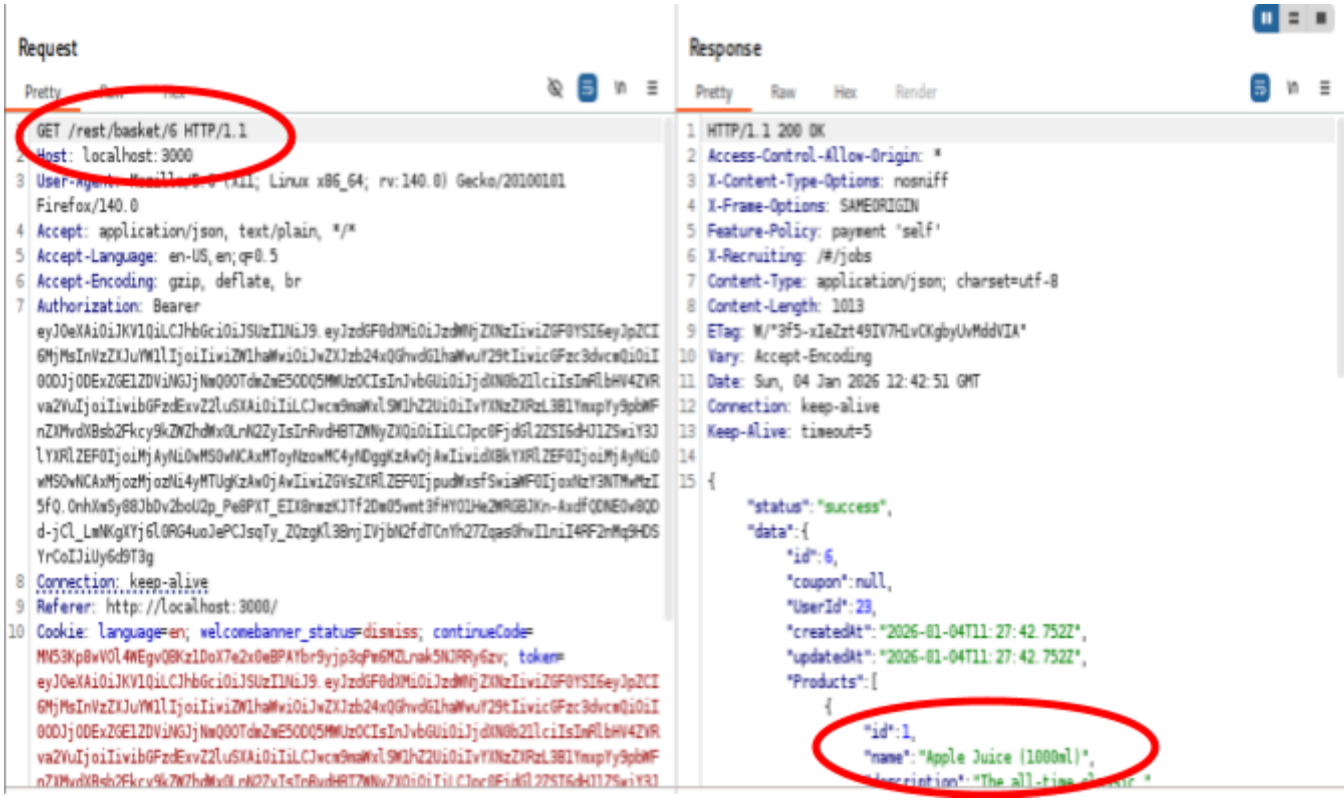
Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the basket functionality of the OWASP Juice Shop

application. The application exposes internal object identifiers in HTTP requests and fails to properly verify user authentication and authorization before returning data. As a result, sensitive resources can be accessed directly by manipulating or replaying requests, even without an active user session.

Steps to Reproduce

1. The tester logged into the application using a normal user account through the Firefox browser.
2. An item (Apple Juice) was viewed, which generated a request related to the user's basket.
3. Using Burp Suite, the request was intercepted and sent to **Burp Repeater** for further testing.
4. The tester then logged out of the application from the browser.
5. After logging out, the previously captured request was sent again from Burp Repeater.
6. The server responded with **HTTP 200 OK**, even though no user was logged in, indicating that authentication was not being properly enforced.

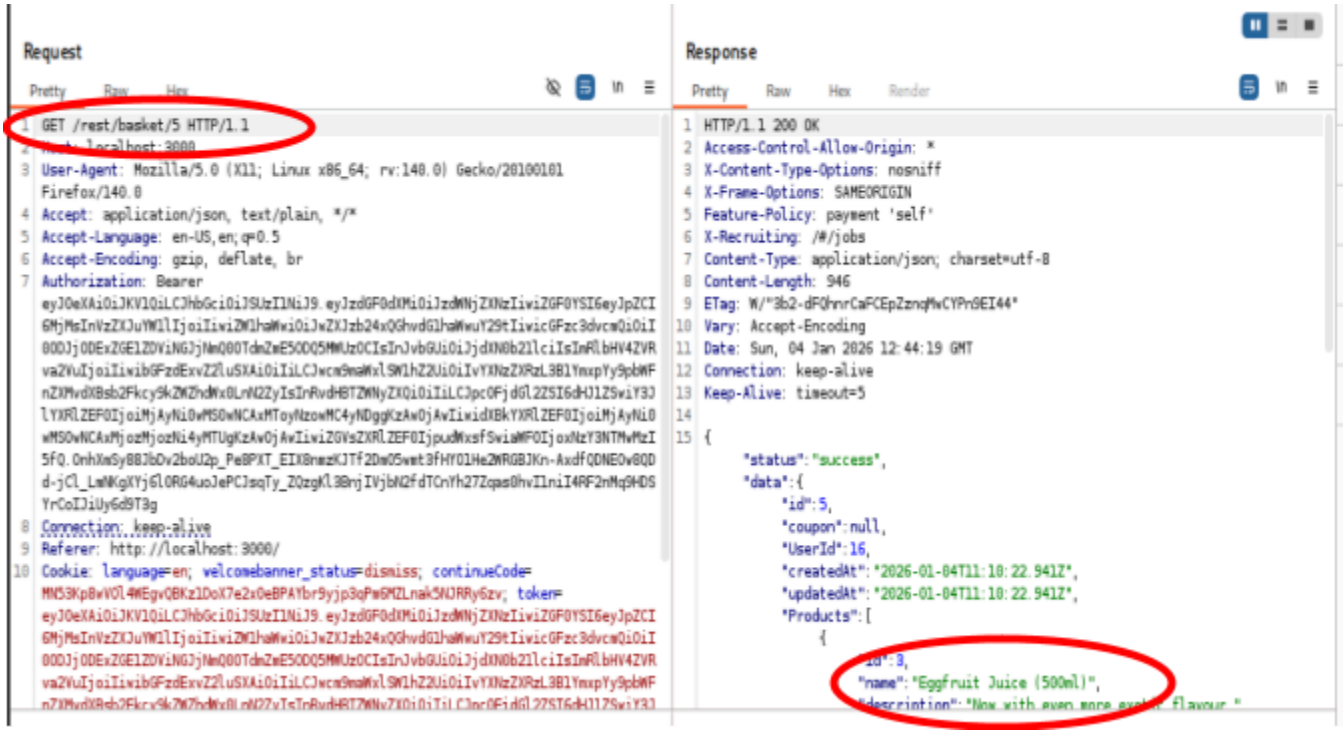


7. The object identifier in the request was then modified in Repeater:

   GET /rest/basket/6 HTTP/1.1 was changed to

   GET /rest/basket/5 HTTP/1.1

8. The modified request was sent, and the server returned details for a different item (Eggfruit Juice), even though this item had not been viewed or accessed through the website interface.



Evidence

- A successful HTTP 200 OK response was received after replaying the request while logged out.
- Changing the basket ID in the request returned data for a different item.
- The response confirmed that the application trusted user-supplied object identifiers without verifying session state or ownership.

Impact

This vulnerability allows unauthorized access to application data by simply replaying or modifying HTTP requests. An attacker could potentially view or manipulate other users' basket data without authentication. In a real-world application, this type of flaw could

lead to data leakage, privacy violations, or unauthorized actions on behalf of other users.

Remediation

The application should enforce authentication and authorization checks on every request that accesses user-specific resources. Backend logic must verify that the requester is logged in and is authorized to access the requested object. Internal object identifiers should not be trusted directly, and access decisions should never rely solely on user-supplied values.

Vulnerability 4: Security Misconfiguration

OWASP Category: A05 – Security Misconfiguration

Description

A Security Misconfiguration vulnerability was identified in the OWASP Juice Shop application due to an insecure Cross-Origin Resource Sharing (CORS) configuration. The application includes the response header:

Access-Control-Allow-Origin: *

This configuration allows any external website to make requests to the application and read responses, which is unsafe for applications that handle user data or authentication.

CORS is a browser security mechanism designed to restrict how resources on a web application can be accessed by other websites. When CORS is configured with a wildcard (*), it disables these protections.

Steps to Reproduce

1. The tester accessed the OWASP Juice Shop application through a web browser.

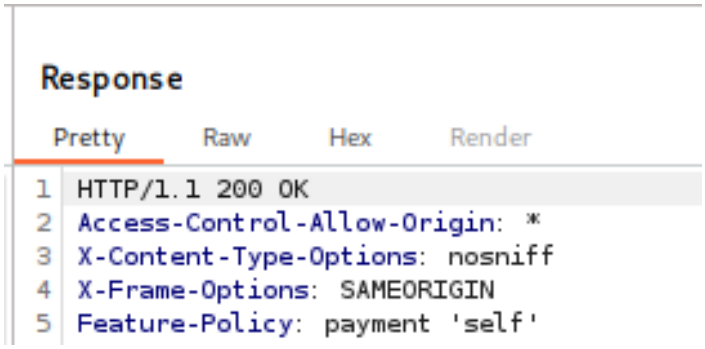
2. Using Burp Suite, HTTP requests and responses were intercepted while interacting with the application.
3. The response headers were inspected in Burp Suite.
4. The following header was consistently observed in server responses:

   `Access-Control-Allow-Origin: *`

5. This confirmed that the application allows requests from **any origin**, without restricting trusted domains.

Evidence

- Response headers showed `Access-Control-Allow-Origin:*`.
- No origin-based restrictions were applied.
- The configuration was visible directly in Burp Suite without requiring exploitation.



Impact

An insecure CORS configuration can allow malicious websites to interact with the application on behalf of users. In real-world scenarios, this could enable attackers to steal sensitive data, perform unauthorized actions, or abuse authenticated sessions if other protections are weak. While CORS alone is not always exploitable, it significantly increases risk when combined with other vulnerabilities.

Remediation

The application should restrict CORS access to trusted and known domains only. Instead of using a wildcard (*), specific allowed origins should be defined. Sensitive endpoints should not be accessible through cross-origin requests unless explicitly required. CORS policies should be reviewed and configured according to the principle of least privilege.

Vulnerability 5: Sensitive Data Exposure (Exposed Authentication Token)

OWASP Category: A02 – Cryptographic Failures

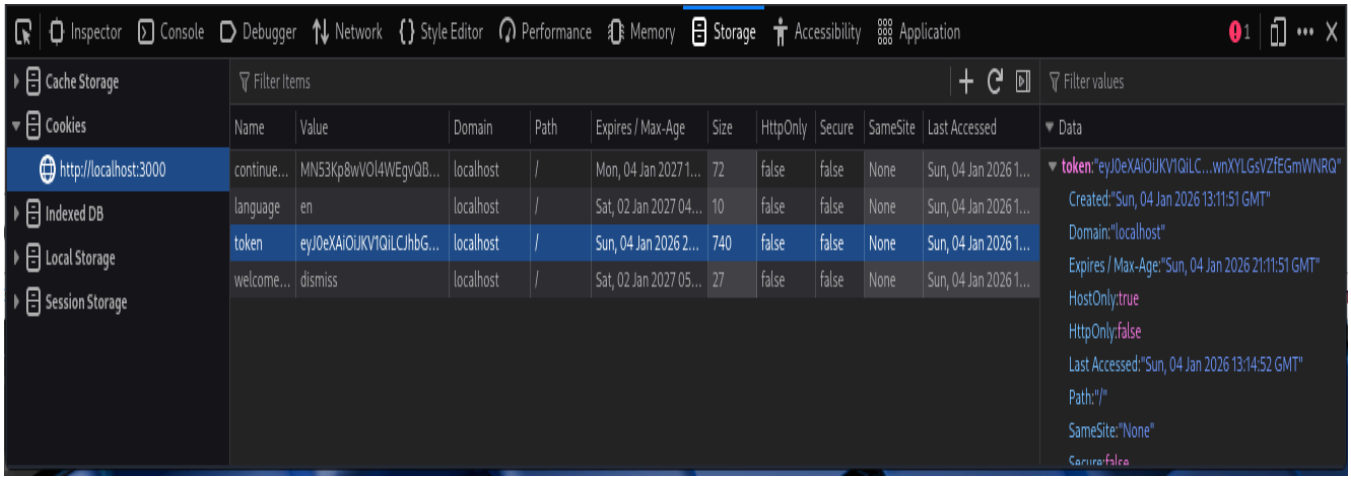
Description

A Sensitive Data Exposure vulnerability was identified in the OWASP Juice Shop application due to the insecure exposure of a JSON Web Token (JWT) used for user authentication. After logging into the application, the JWT token was found stored in the browser and was easily accessible using browser developer tools.

JWTs are sensitive authentication tokens that contain encoded user information and are used by the server to identify authenticated users. Exposure of such tokens can allow attackers to impersonate users if the token is stolen.

Steps to Reproduce

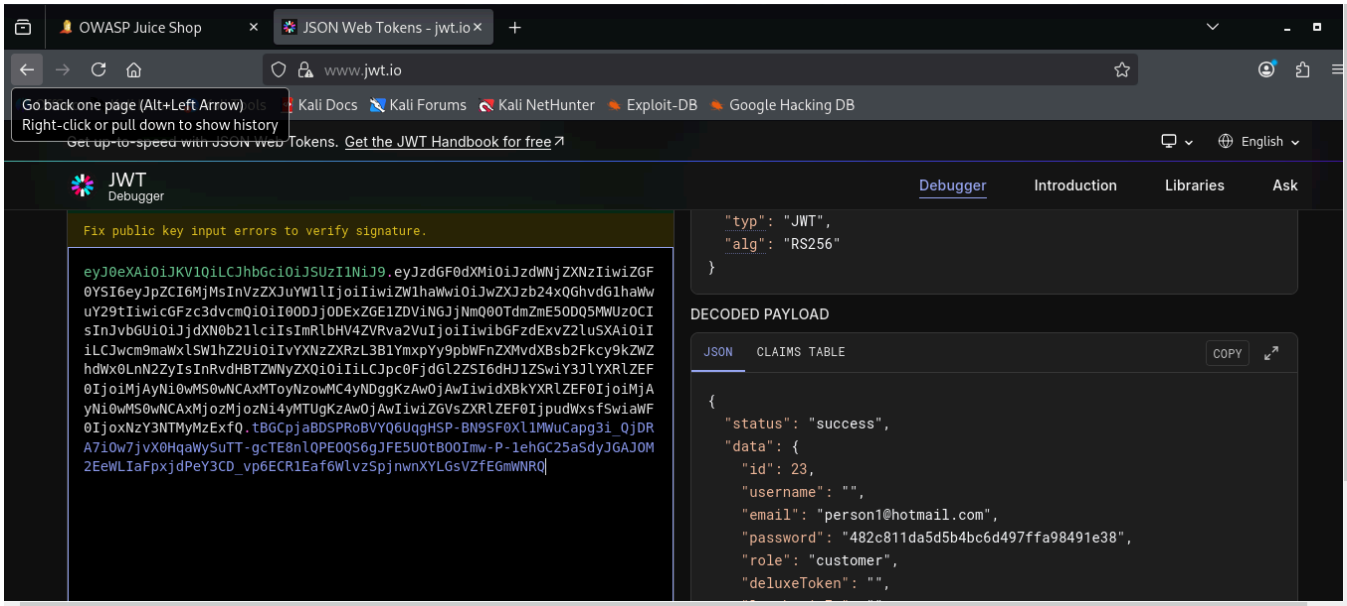
1. The tester logged into the OWASP Juice Shop application using a normal user account.
2. While logged in, the browser's developer tools were opened using **F12 (Inspect)**.
3. The **Application / Storage** section was selected.
4. Under **Cookies**, the authentication token related to the Juice Shop application was located.

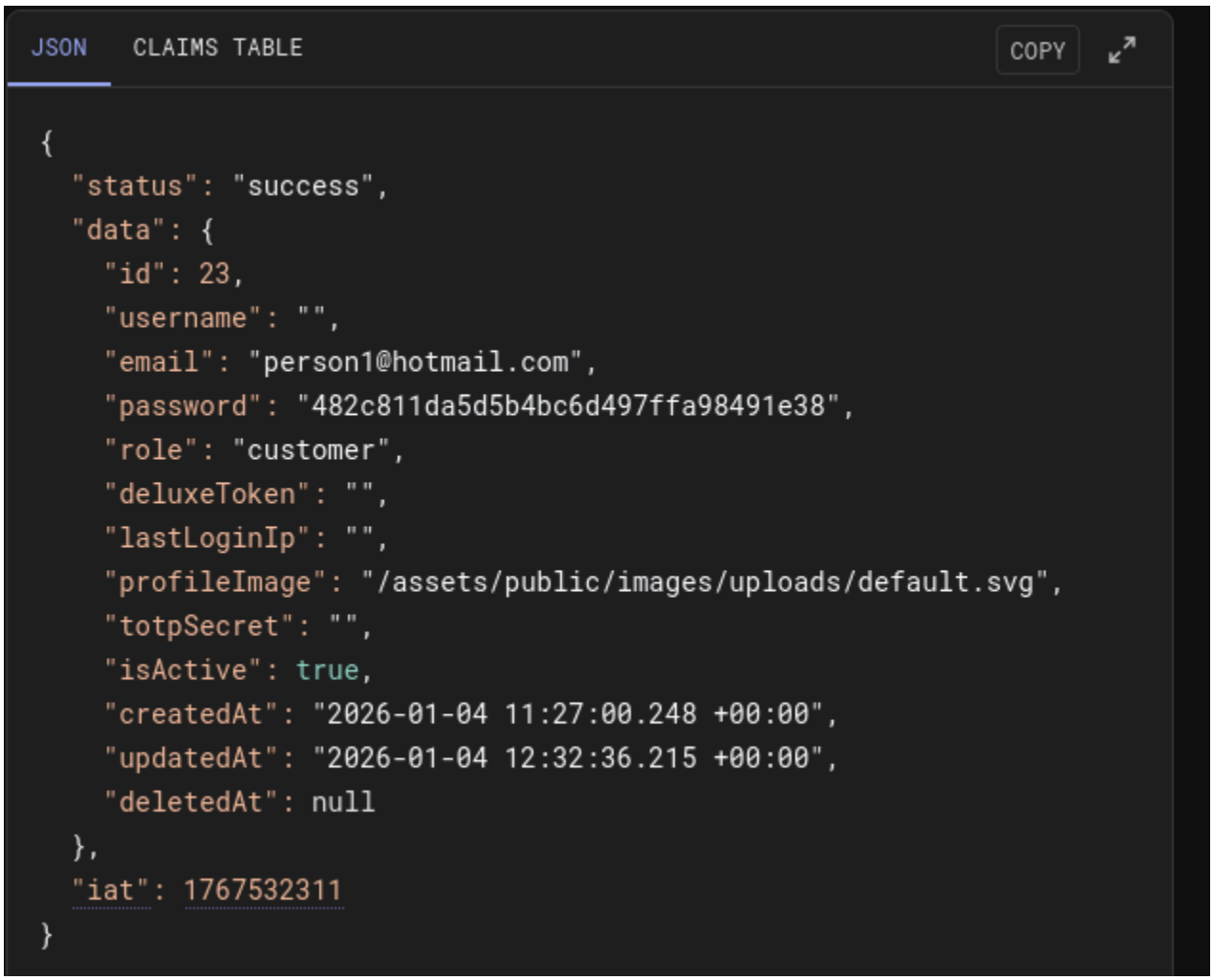


5. The token value was copied from the cookie storage.
6. The copied token was pasted into the online tool https://www.jwt.io/ to analyze its structure.
7. The website decoded the token and displayed the token's contents, showing readable user-related information without requiring the secret key.

Evidence

- The JWT authentication token was visible in the browser cookies.
- The token could be copied directly from developer tools.
- The token contents were readable when decoded using jwt.io.
- No special permissions were required to view or decode the token.


OWASP Juice Shop
JSON Web Tokens - jwt.io
www.jwt.io
Go back one page (Alt+Left Arrow)
Right-click or pull down to show history
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
Get the JWT Handbook for free
English
JWT Debugger
Debugger
Introduction
Libraries
Ask
Fix public key input errors to verify signature.
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGF0dXMiOiJzdWNjZXNzIiwiZGF0YSI6eyJpZCI6MjMsInVzZXJuYW1lIjoiIiwiZW1haWwiOiJwZXJzb24xQGhvdG1haWwuY29tIiwicGFzc3dvcmQiOiI0ODJjODExZGE1ZDViNGJjNmQ0OTdmZmE5ODQ5MWUzOCIsInJvbGUiOiJjdXN0b21lciIsImRlbHV4ZVRva2VuIjoiIiwibGFzdExvZ2luSXAiOiIiLCJwcm9maWxlSW1hZ2UiOiIvYXNzZXRzL3B1YmxpYy9pbWFnZXMvdXBsb2Fkcy9kZWZhdWx0LnN2ZyIsInRvdHBTZWNyZXQiOiIiLCJpc0FjdGl2ZSI6dHJ1ZSwiY3JlYXRlZEF0IjoiMjAyNi0wMS0wNCAxMToyNzowMC4yNDggKzAwOjAwIiwidXBkYXRlZEF0IjoiMjAyNi0wMS0wNCAxMjozMjozNi4yMTUgKzAwOjAwIiwiZGVsZXRlZEF0IjpudWxsfSwiaWF0IjoxNzY3NTMyMzExfQ.tBGCpjaBDSPRoBVYQ6UqgHSP-BN9SF0Xl1MWuCapg3i_QjDRA7iOw7jvX0HqaWySuTT-gcTE8nlQPEOQS6gJFE5UOtBOOImw-P-1ehGC25aSdyJGAJOM2EeWLIaFpxjdPeY3CD_vp6ECR1Eaf6WlvzSpjnwnXYLGsVZfEGmWNRQ
"typ": "JWT",
"alg": "RS256"
DECODED PAYLOAD
JSON
CLAIMS TABLE
COPY
"status": "success",
"data": {
"id": 23,
"username": "",
"email": "person1@hotmail.com",
"password": "482c811da5d5b4bc6d497ffa98491e38",
"role": "customer",
"deluxeToken": "",


JSON
CLAIMS TABLE
COPY
{
"status": "success",
"data": {
"id": 23,
"username": "",
"email": "person1@hotmail.com",
"password": "482c811da5d5b4bc6d497ffa98491e38",
"role": "customer",
"deluxeToken": "",
"lastLoginIp": "",
"profileImage": "/assets/public/images/uploads/default.svg",
"totpSecret": "",
"isActive": true,
"createdAt": "2026-01-04 11:27:00.248 +00:00",
"updatedAt": "2026-01-04 12:32:36.215 +00:00",
"deletedAt": null
},
"iat": 1767532311
}

Impact

Exposed JWT tokens increase the risk of session hijacking. If an attacker obtains a valid token, they may be able to impersonate a user without knowing their password. Additionally, readable token contents can leak sensitive information such as user identifiers or roles. In real-world applications, this could result in unauthorized access and data breaches.

Remediation

Authentication tokens should be securely stored and protected. Sensitive cookies should use security flags such as `HttpOnly` and `Secure` to prevent access through browser scripts and reduce exposure. JWTs should be designed to minimize sensitive information in their payloads, use strong signing algorithms, and be protected against theft through secure handling practices.

---

7. Conclusion

This project demonstrated how common vulnerabilities from the **OWASP Top 10** can exist in modern web applications. By using standard security testing tools and a structured approach, five serious vulnerabilities were identified and exploited in a controlled environment.

The exercise highlighted the importance of secure coding practices, proper input validation, access control, and secure configuration. Understanding these issues helps developers and security professionals build safer applications and reduce real-world security risks.

---

8. Ethical Considerations

All testing was performed on a **local, intentionally vulnerable application** created for educational purposes. No real systems or user data were targeted. This project strictly followed ethical guidelines for learning and research.