Technical Report: : Analyze Web Application Authentication Security Using OWASP ZAP

1. Introduction

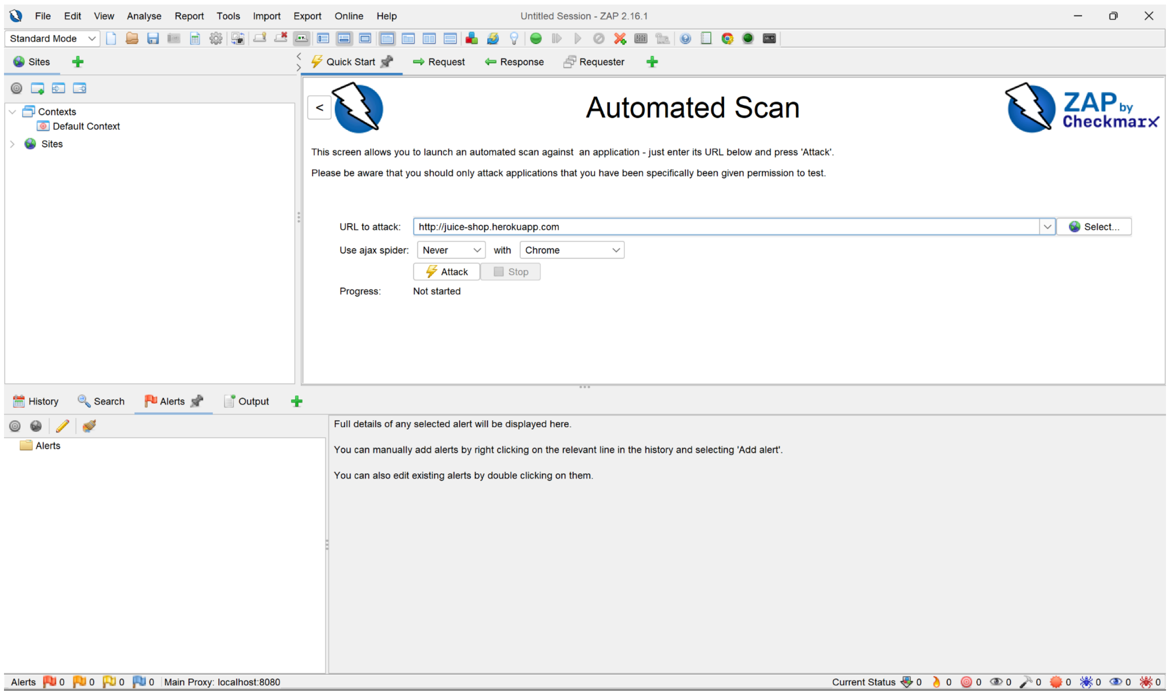
Web application authentication is a critical component in ensuring that only authorized users access sensitive resources. Insecure authentication mechanisms can result in data breaches, account takeovers, and overall system compromise. This mini project focuses on testing and evaluating the authentication security of the OWASP Juice Shop using OWASP ZAP (Zed Attack Proxy). The primary goal is to identify potential vulnerabilities, assess their impact, and recommend mitigation strategies.

2. Tools Used

Tool	Purpose
OWASP ZAP 2.16.1	Used to intercept, analyze, and actively scan the web application for vulnerabilities
Web Browser (Firefox/Chrome)	Used to simulate user interaction and direct traffic through the ZAP proxy
OWASP Juice Shop	A deliberately insecure web application used as the testing target

3. Step-by-Step Execution

- **Step 1: Tool Setup**
  - Installed OWASP ZAP v2.16.1 on the local machine.
  - Configured browser proxy settings to 127.0.0.1:8080 to route HTTP/HTTPS traffic through ZAP.
  - Opened https://juice-shop.herokuapp.com in the browser.



Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.

Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://juice-shop.herokuapp.com

Use ajax spider: Never with Chrome

Attack Stop

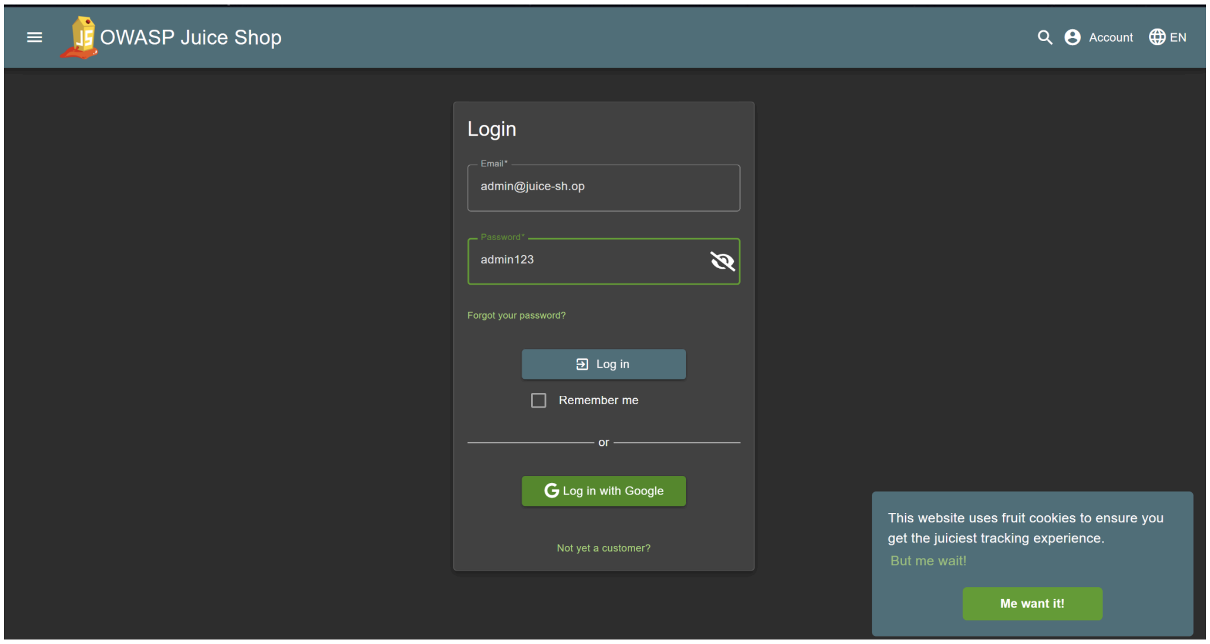
Progress: Not started

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.

- **Step 2: Capturing Login Request**
  - Logged in using credentials: admin@juice-sh.op / admin123.
  - ZAP successfully intercepted the POST login request.



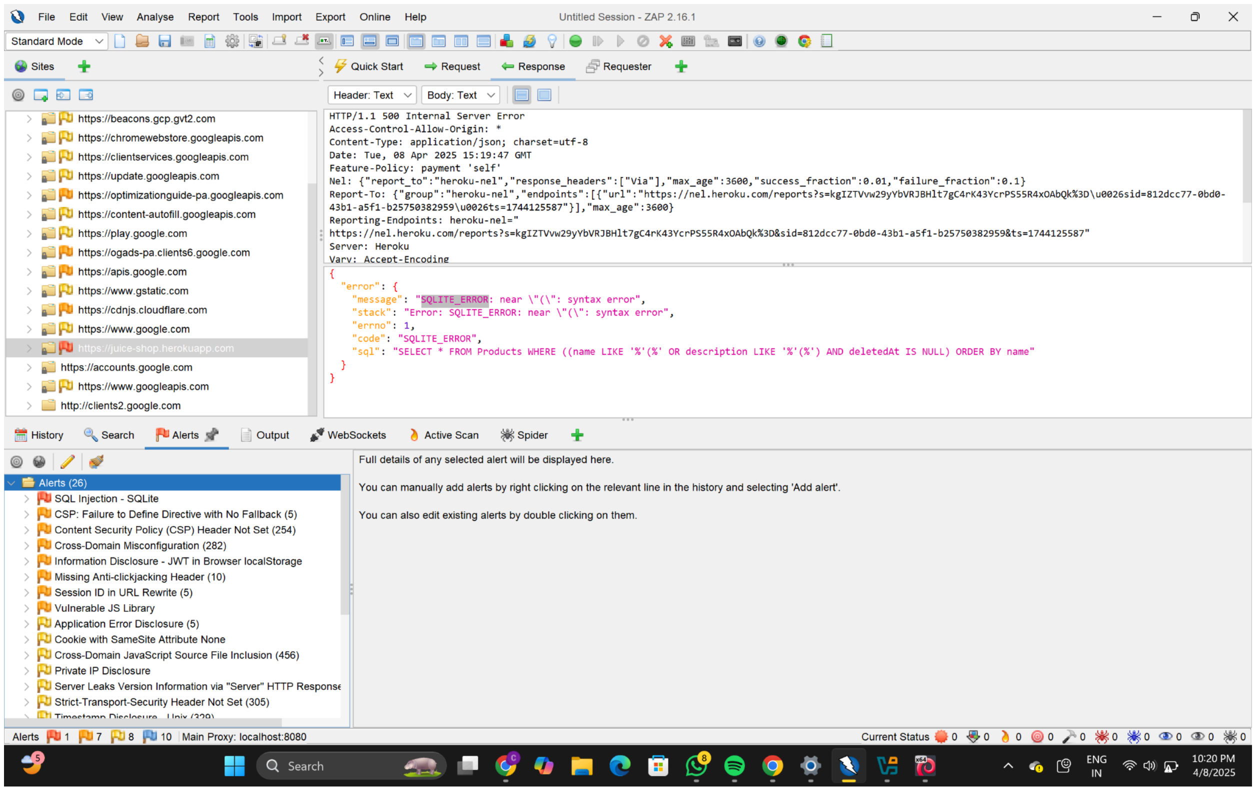
Login

Email*: admin@juice-sh.op

Password*: admin123

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

- **Step 3: Scanning the Application**
  - Enabled Spider and Active Scan on the target URL.
  - Focused especially on /rest/user/login and product search endpoints.
  - Used SQL injection payloads to test backend logic.



```
HTTP/1.1 500 Internal Server Error
Access-Control-Allow-Origin: *
Content-Type: application/json; charset=utf-8
Date: Tue, 08 Apr 2025 15:19:47 GMT
Feature-Policy: payment 'self'
Nel: {"report_to":"heroku-nel","response_headers":["Via"],"max_age":3600,"success_fraction":0.01,"failure_fraction":0.1}
Report-To: {"group":"heroku-nel","endpoints":[{"url":"https://nel.heroku.com/reports?s=kgIZTVvw29yYbVRJBHlt7gC4rK43YcrPS55R4xOAbQk%3D\u0026sid=812dcc77-0bd0-
43b1-a5f1-b25750382959\u0026ts=1744125587"}],"max_age":3600}
Reporting-Endpoints: heroku-nel="
https://nel.heroku.com/reports?s=kgIZTVvw29yYbVRJBHlt7gC4rK43YcrPS55R4xOAbQk%3D&sid=812dcc77-0bd0-43b1-a5f1-b25750382959&ts=1744125587"
Server: Heroku
Vary: Accept-Encoding
```

```
{
  "error": {
    "message": "SQLITE_ERROR: near \"(\": syntax error",
    "stack": "Error: SQLITE_ERROR: near \"(\": syntax error",
    "errno": 1,
    "code": "SQLITE_ERROR",
    "sql": "SELECT * FROM Products WHERE ((name LIKE '%'(%' OR description LIKE '%'(%') AND deletedAt IS NULL) ORDER BY name"
  }
}
```

Alerts (26)
- SQL Injection - SQLite
- CSP: Failure to Define Directive with No Fallback (5)
- Content Security Policy (CSP) Header Not Set (254)
- Cross-Domain Misconfiguration (282)
- Information Disclosure - JWT in Browser localStorage
- Missing Anti-clickjacking Header (10)
- Session ID in URL Rewrite (5)
- Vulnerable JS Library
- Application Error Disclosure (5)
- Cookie with SameSite Attribute None
- Cross-Domain JavaScript Source File Inclusion (456)
- Private IP Disclosure
- Server Leaks Version Information via "Server" HTTP Response
- Strict-Transport-Security Header Not Set (305)
- Timestamp Disclosure - Unix (329)

Full details of any selected alert will be displayed here.

You can manually add alerts by right clicking on the relevant line in the history and selecting 'Add alert'.

You can also edit existing alerts by double clicking on them.

Alerts 1 7 8 10 Main Proxy: localhost:8080

Untitled Session - ZAP 2.16.1
File Edit View Analyse Report Tools Import Export Online Help
Standard Mode
Sites
Quick Start
Request
Response
Requester
Header: Text
Body: Text
https://beacons.gcp.gvt2.com
https://chromewebstore.googleapis.com
https://clientservices.googleapis.com
https://update.googleapis.com
https://optimizationguide-pa.googleapis.com
https://content-autofill.googleapis.com
https://play.google.com
https://ogads-pa.clients6.google.com
https://apis.google.com
https://www.gstatic.com
https://cdnjs.cloudflare.com
https://www.google.com
https://juice-shop.herokuapp.com
https://accounts.google.com
https://www.googleapis.com
http://clients2.google.com
History
Search
Alerts
Output
WebSockets
Active Scan
Spider
Current Status
Search
ENG IN
10:20 PM
4/8/2025

- **Step 4: Analyzing ZAP Results**
  - ZAP flagged several issues with varying risk levels. These included authentication weaknesses, insecure cookie settings, and SQL injection. Screenshots and payloads validated these findings.

4. Findings & Analysis

Vulnerability	**Risk**	**Description**
SQL Injection in Product Search	High	User input directly injected into SQL query, vulnerable to data extraction

No Multi-Factor Authentication (MFA)	High	Login relies solely on password authentication
Weak Password Policy	Medium	Accepts simple passwords like admin123 without validation
Session ID in URL	Medium	Session tokens visible in URLs, exposing them to logging and theft
JWT stored in localStorage	Medium	Tokens vulnerable to XSS-based theft
Missing Security Headers	Low	Absence of headers like Strict-Transport-Security, X-Frame-Options, etc.

5. Recommendations

- **Authentication Improvements**
  - **Enforce Strong Password Policy**: Minimum 8 characters, uppercase, lowercase, number, and symbols.
  - **Implement MFA**: Add one-time passwords or authenticator apps for login.
- **Input & Session Security**
  - **Use Parameterized Queries**: Prevent SQL injection by avoiding direct string concatenation.
  - **Sanitize User Input**: Validate inputs both on client and server side.
- **Cookie & Token Handling**
  - **Secure Session Cookies**: Enable HttpOnly, Secure, and SameSite=Strict.
  - **Avoid JWT in localStorage**: Store tokens in secure cookies to prevent XSS theft.
- **Security Headers**
  - Implement headers like:

    http
    CopyEdit
    ```
    Strict-Transport-Security: max-age=31536000; includeSubDomains
    X-Frame-Options: DENY
    ```

```
X-Content-Type-Options: nosniff
Content-Security-Policy: default-src 'self'
```

- **Monitoring**
  - Set up alerting for brute-force login attempts.
  - Monitor authentication logs and respond to anomalies.

6. Conclusion

The analysis of the OWASP Juice Shop using OWASP ZAP revealed critical vulnerabilities in its authentication mechanisms, such as SQL injection and lack of MFA. These issues can be exploited to gain unauthorized access and compromise sensitive data. Implementing strong password enforcement, multi-factor authentication, secure session handling, and proper input validation can significantly strengthen the security of the application.