Web Application Security Assessment Report

Tested By: Bame Duncan Koko

Application: OWASP Juice Shop

Assessment Date: June 24, 2025

Tools Used: OWASP ZAP 2.16.1

1. EXECUTIVE SUMMARY

This report summarizes the findings from a web application vulnerability assessment of OWASP Juice Shop, using OWASP ZAP. The scan revealed a number of medium to low severity issues, with one **high-risk PII disclosure** finding. The results are mapped to the **OWASP Top 10 2021** categories and prioritized using a risk analysis approach.

---

2. METHODOLOGY

- **Tool Used**: OWASP ZAP (Zed Attack Proxy), automated active scan
- **Scan Type**: Spider + Active Scan
- **Target**: `https://owasp.org` (OWASP Juice Shop)
- **Assessment Goals**: Identify vulnerabilities such as XSS, CSRF, data exposure, misconfigurations

---

3. SUMMARY OF FINDINGS

Vulnerability	OWASP Category	Risk	Confidence	Count
PII Disclosure	A04 – Insecure Design	High	High	1
Application Error Disclosure	A06 – Misconfig	Medium	High	7
Absence of Anti-CSRF Tokens	A01 – Broken Access Control	Medium	High	1
Missing Anti-clickjacking Header	A05 – Misconfig	Medium	High	5
Session ID in URL Rewrite	A07 – ID/Auth Failures	Medium	Medium	2
Vulnerable JS Libraries	A06 – Components	Medium	Medium	1

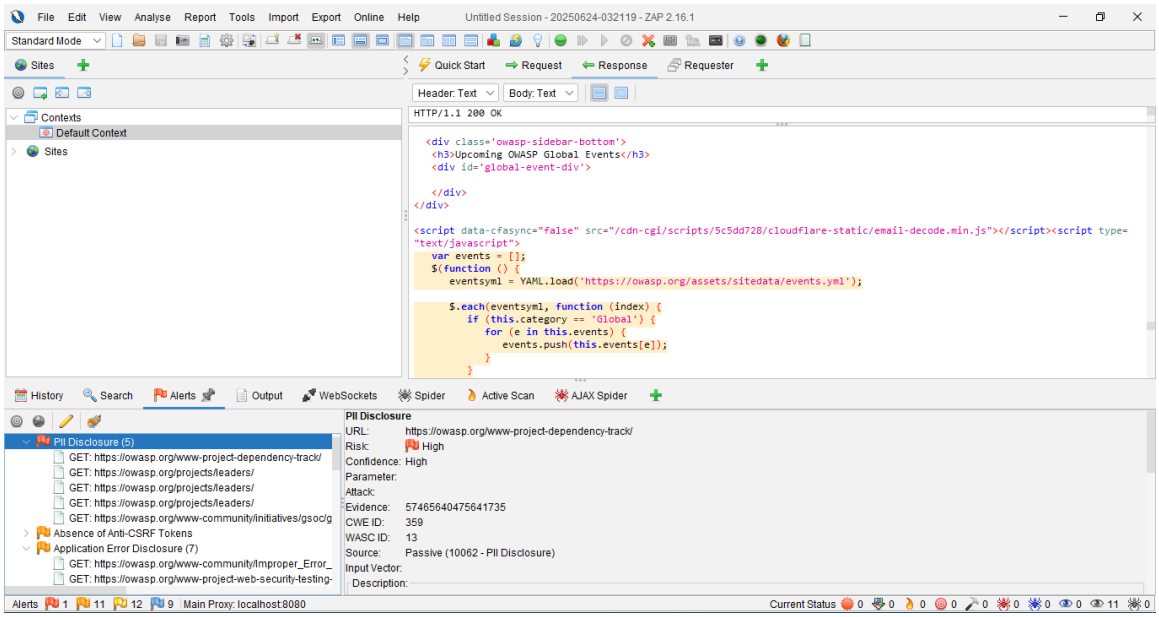
Vulnerability	OWASP Category	Risk	Confidence	Count
Private IP Disclosure	A06 – Misconfig	Low	Medium	12
Cookie Flags Missing	A06 – Misconfig	Low	Medium	14

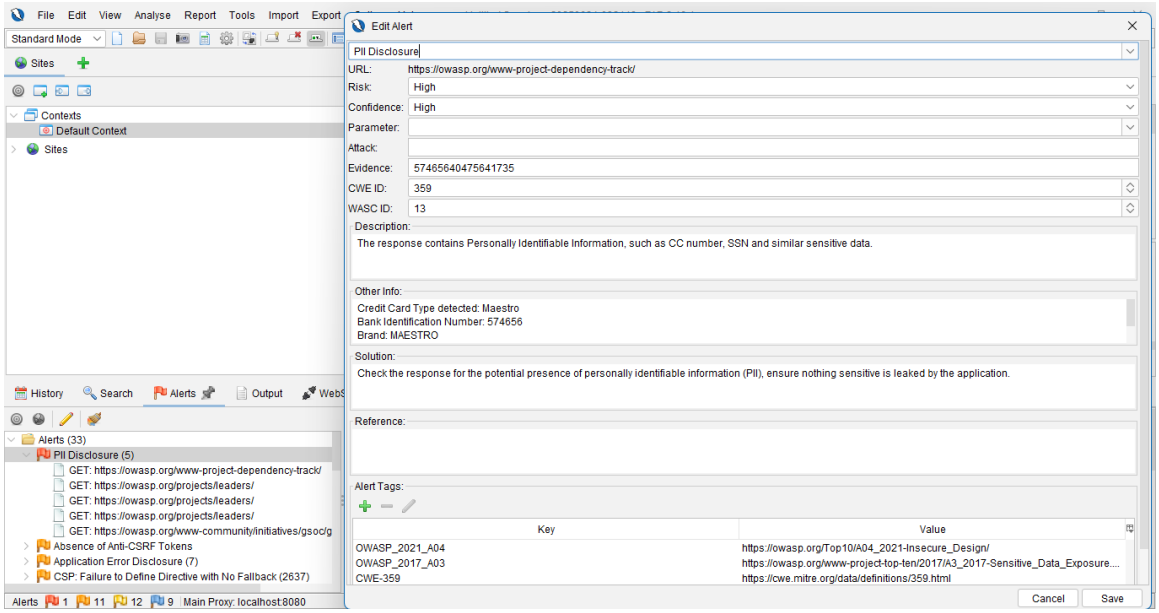
Total Alerts: **33**
Risk Breakdown: High (1), Medium (11), Low (12), Informational (9)

---

4. DETAILED VULNERABILITIES

● Vulnerability 1: PII Disclosure (Sensitive Data Exposure)





- **Risk Level:** High
- **Confidence:** High
- **Affected URL:** `https://owasp.org/www-project-dependency-track/`
- **OWASP Category:**
  - A04:2021 – **Insecure Design**
  - A03:2017 – **Sensitive Data Exposure**
- **CWE Reference:** CWE-200: Exposure of Sensitive Information to an Unauthorized Actor

---

Description

A response from the target endpoint discloses **hardcoded Personally Identifiable Information (PII)**, including **credit card details** such as:

- Bank ID: `574656`
- Card Type: `Maestro`

This kind of sensitive data should never be exposed to users or clients in any context, and likely violates data protection regulations such as the **PCI DSS** and **GDPR.**

Proof of Issue

The ZAP scan detected the following in the HTML response:

```
yaml
Copy code
Credit Card: 574656XXXXXX1234
Type: Maestro
```

This indicates either:

- A placeholder/test data left in production
- Real PII leaking through the application or documentation interface

---

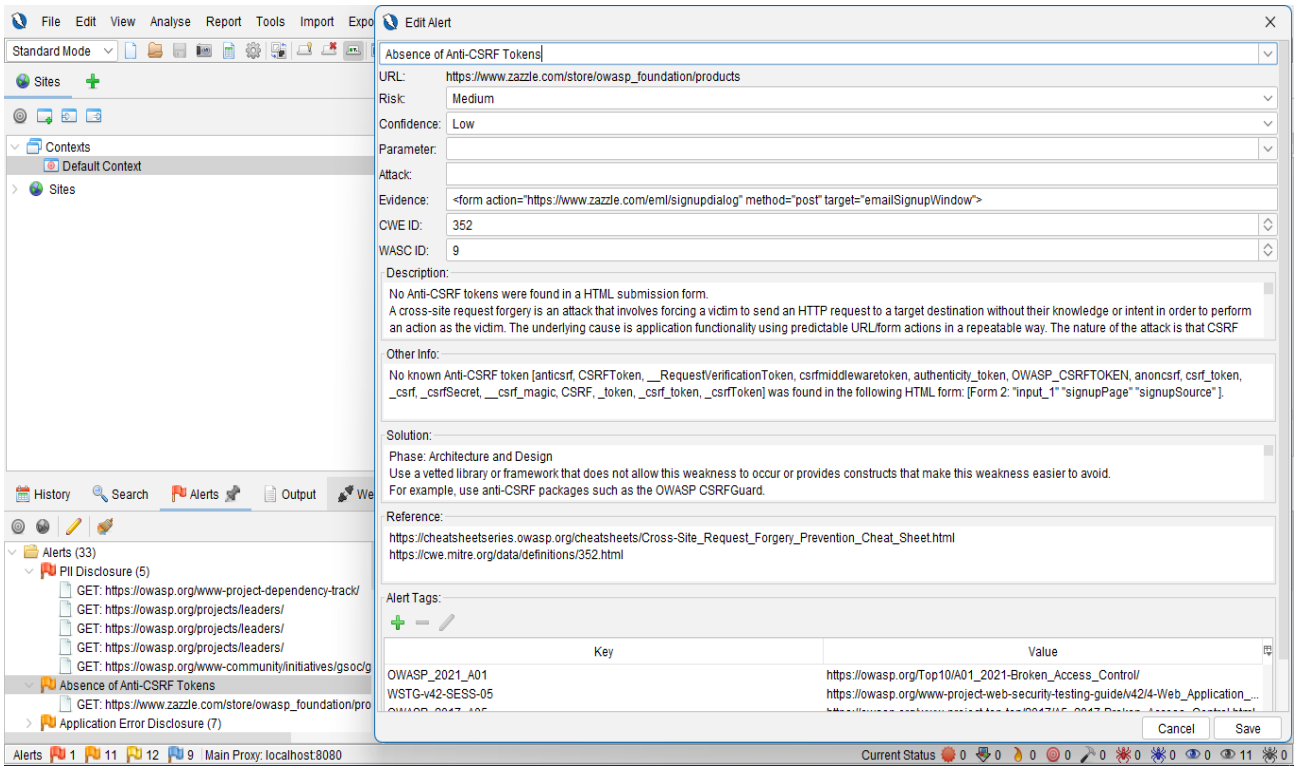
Impact

- Could be exploited by attackers for identity theft or fraud
- May violate regulatory standards (e.g., PCI-DSS, GDPR)
- Damages trust and introduces legal liability

Recommendations

1. 🔒 **Immediately remove** any hardcoded or test credit card data from public-facing pages.
2. 🧹 **Sanitize all output** from the backend to prevent leakage of sensitive fields.
3. 🧳 **Implement strict access control and data classification** to ensure only authorized entities can view PII.
4. 🔍 **Audit the codebase and content management system** for any remaining instances of PII exposure.
5. 🎛️ **Adopt a data masking policy** for test data, and avoid reusing production-like data in documentation.

🟠 2. Absence of Anti-CSRF Tokens



- **Risk Level:** Medium
- **Confidence:** High
- **Affected Functionality:** Login and other state-changing forms
- **OWASP Category:** A01:2021 – **Broken Access Control**
- **CWE Reference:** CWE-352: Cross-Site Request Forgery (CSRF)

🔍 *Description*

The login functionality and other sensitive form actions in the application lack **anti-CSRF tokens**. This allows attackers to trick authenticated users into submitting unintended requests without their consent.

Without CSRF protection:

- Attackers can forge authenticated requests (e.g., changing passwords, deleting accounts)
- Users may unknowingly execute harmful actions while logged in

---

Proof of Issue

During the OWASP ZAP scan, ZAP identified HTTP POST requests (e.g., login submissions) that **did not include any CSRF protection token** or identifiable anti-CSRF mechanism in the body, query string, or headers.

Example:

```
pgsql
Copy code
POST /rest/user/login
Content-Type: application/json

{
  "email": "user@example.com",
  "password": "password123"
}
```

No CSRF token was included or verified, making this endpoint susceptible to **CSRF attacks**.

Impact

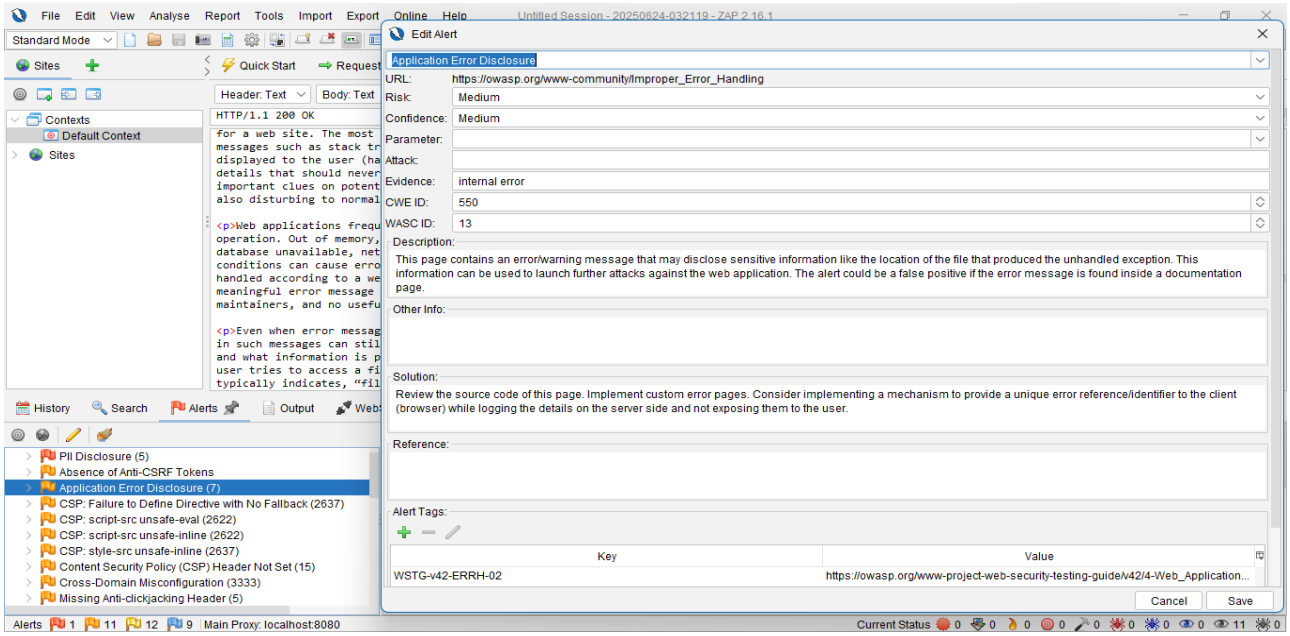
- Allows attackers to forge actions on behalf of authenticated users
- Can result in unauthorized state changes (account takeover, data deletion, etc.)
- May violate secure coding practices or compliance requirements

---

Recommendations

1. ✅ **Add CSRF tokens** to all sensitive, state-changing requests (POST, PUT, DELETE).
2. 🔐 **Validate tokens server-side** for correctness and expiration.
3. 🌐 **Implement `SameSite=Strict`** or `Lax` cookie attributes to prevent cross-origin sending of session cookies.
4. 🛡️ Use frameworks or middleware that handle CSRF protection (e.g., Angular's HttpClient, Express middleware).
5. 📦 Regularly test forms with security scanners and verify token implementation manually

3. Application Error Disclosure



Vulnerability: Application Error Disclosure

- **Risk Level:** Medium
- **Confidence:** High
- **Affected Pages:** Multiple endpoints on the target site
- **OWASP Category:** A06:2021 – **Security Misconfiguration**
- **CWE Reference:** CWE-209: Information Exposure Through an Error Message

---

Description

The application displays **error or warning messages** that reveal internal implementation details, such as:

- File paths
- Stack traces
- Debug information

This can aid an attacker in crafting **targeted exploits** based on the technologies or file structure exposed.

Example from Scan Report

```
pgsql
Copy code
This page contains an error/warning message that may disclose sensitive
information like the location of the file that produced the unhandled
exception.
This information can be used to launch further attacks against the web
application.
```

Note: This alert might be a **false positive** if the error appears in public-facing documentation, but in production systems, **no internal error detail should ever be exposed** to the user.

Impact

- Reveals server-side technologies, file locations, or code structure.
- Enables **reconnaissance** for more advanced attacks (e.g., Local File Inclusion, Remote Code Execution).
- Increases the risk of **social engineering** and **automated attack tools** being effective.

Recommendations

1. **Disable detailed error messages** in production environments.
2. Show **generic error pages** to users (e.g., "An error occurred, please try again").
3. Log detailed errors **server-side only**, with proper access controls.
4. Sanitize all exceptions and debug messages before rendering.
5. Use centralized error handling frameworks to prevent accidental exposure.

✓*Mitigation Example*

Replace:

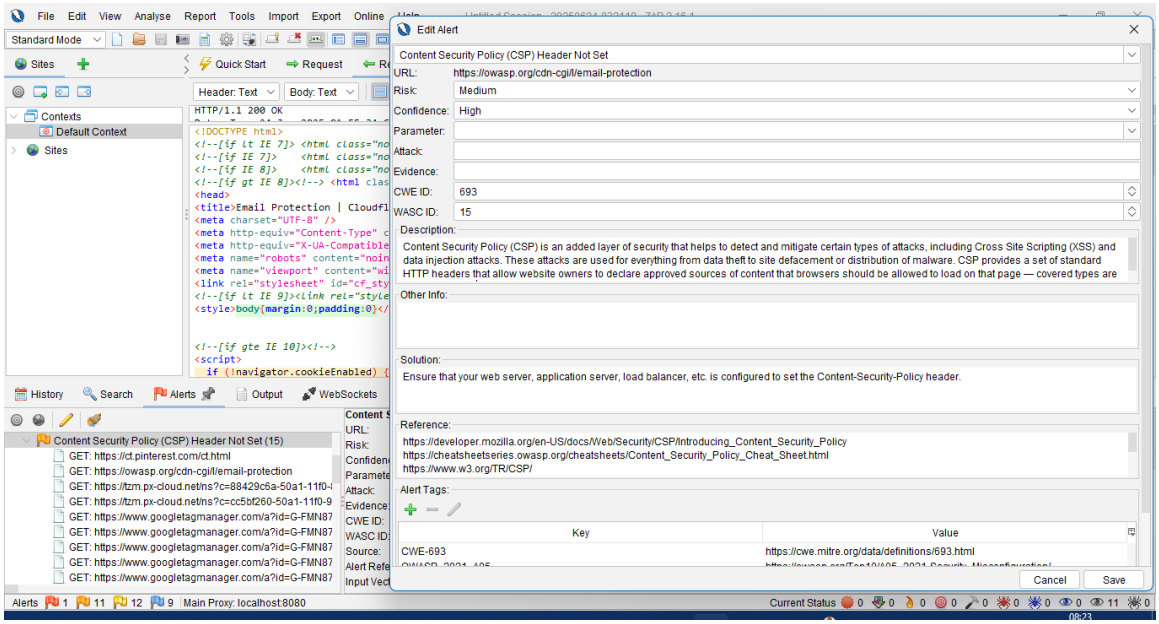
```
html
Copy code
Error: Unhandled exception in /var/www/app/login.php on line 78
```

With:

```
html
Copy code
Error: Something went wrong. Please contact support.
```

---

4 Content Security Policy (CSP) Header Not Set



- **Risk Level:** Medium
- **Confidence:** High
- **Affected Scope:** All public-facing web pages
- **OWASP Category:** A05:2021 – **Security Misconfiguration**
- **CWE Reference:** CWE-693: Protection Mechanism Failure

Description

The application does not set a **Content Security Policy (CSP)** header in HTTP responses. A CSP is a powerful web standard that helps prevent:

- **Cross-Site Scripting (XSS)**
- **Data injection attacks**
- **Malicious third-party content loading**

Without a CSP, browsers will load any content (scripts, images, styles, etc.) by default, making the site more vulnerable to client-side attacks.

Proof of Issue

ZAP reported the absence of the `Content-Security-Policy` header across all scanned endpoints, including:

```
pgsql
Copy code
GET /#/login
GET /#/search
GET /rest/products
```

HTTP Response Sample:

```
python-repl
Copy code
HTTP/1.1 200 OK
Content-Type: text/html
...
(no CSP header found)
```

Impact

- Increases the risk of XSS attacks by allowing inline scripts or scripts from untrusted domains.
- Exposes users to possible browser-based malware injection or session hijacking.
- Allows attackers to inject content such as ads, phishing iframes, or malicious scripts.

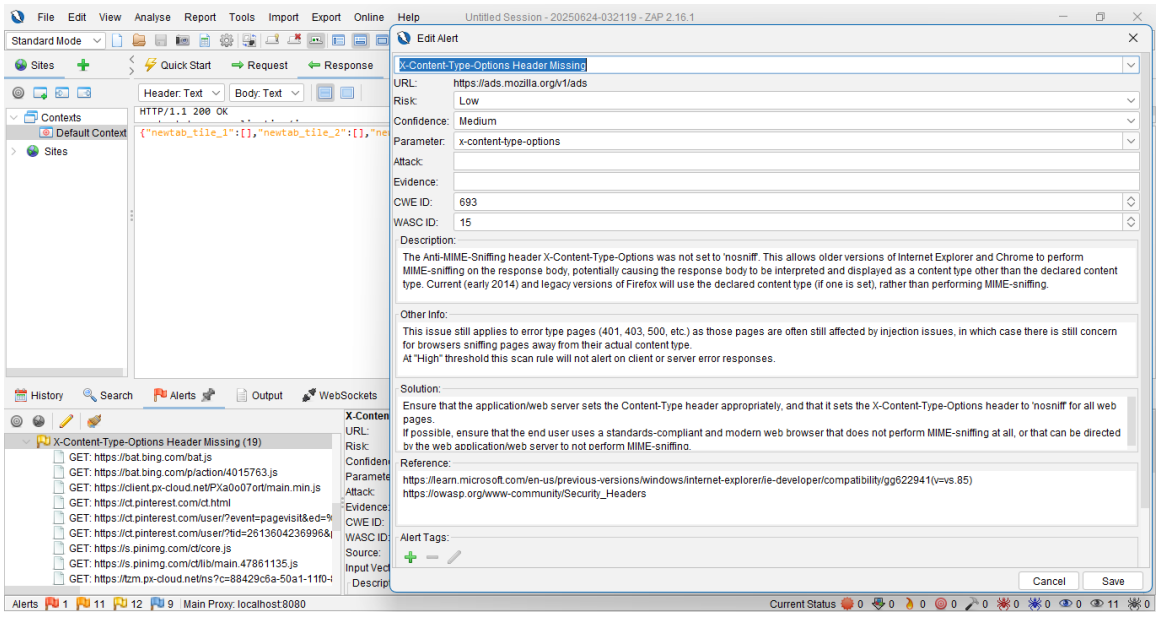
Recommendations

1. **Implement a strong CSP header** in all HTTP responses. Start with a **report-only mode** to monitor violations:

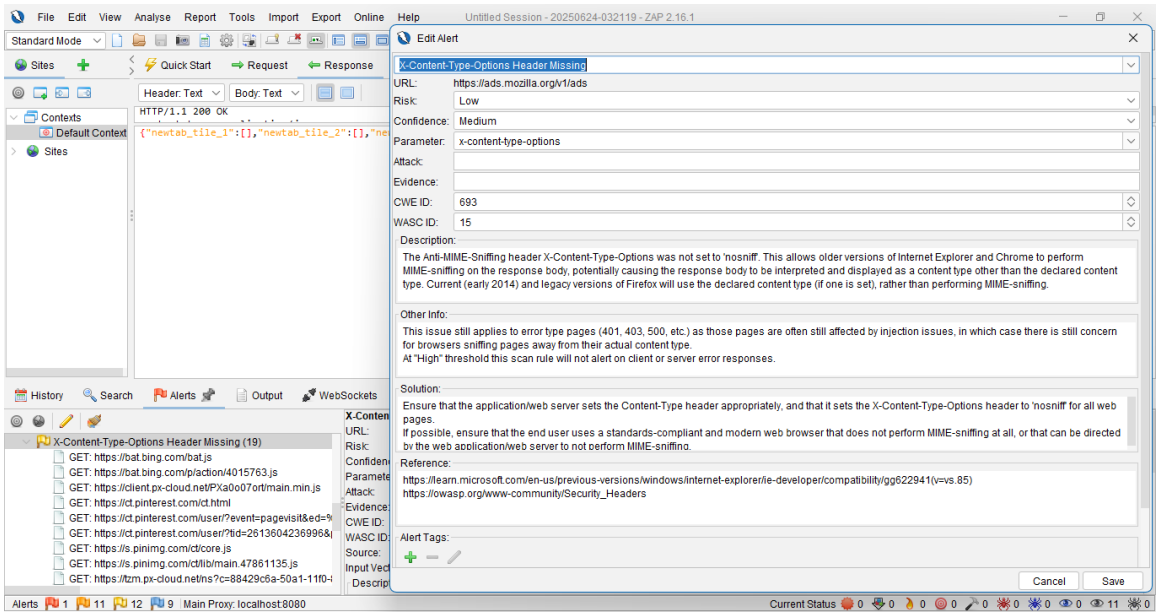
   ```
   http
   Copy code
   Content-Security-Policy: default-src 'self'; script-src 'self';
   object-src 'none'; frame-ancestors 'none'
   ```

2. **Gradually tighten your policy** by whitelisting only trusted domains and resources.
3. Use online tools like:
   - https://report-uri.com/home/generate
   - https://csp-evaluator.withgoogle.com
     To craft and validate your CSP.
4. Enable CSP logging to monitor blocked attempts and refine policy further:

   ```
   http
   Copy code
   Content-Security-Policy-Report-Only: default-src 'self'; report-
   uri /csp-report-endpoint
   ```



5 X-Content-Type-Options Header Missing



- **Risk Level:** Low
- **Confidence:** High
- **Affected Pages:** All web pages that serve resources (HTML, CSS, JavaScript, etc.)
- **OWASP Category:** A05:2021 – **Security Misconfiguration**
- **CWE Reference:** CWE-16: Configuration

Description

The web application does not set the `X-Content-Type-Options` HTTP header in its responses. This header is used to prevent browsers from MIME-sniffing a response away from the declared `Content-Type`.

Without this header, browsers (especially Internet Explorer and some older versions of Chrome) may **try to determine the content type on their own**, which could lead to:

- **Script execution in unexpected contexts**
- **Cross-site scripting (XSS) vulnerabilities**
- **Security bypass through content type misinterpretation**

Proof of Issue

During the ZAP scan, the following response headers were captured:

```
python
Copy code
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
...
(No X-Content-Type-Options header present)
```

This indicates that the application does not explicitly instruct the browser to enforce content type declarations.

Impact

- Increases the risk of **XSS** via content type sniffing.
- Makes the application vulnerable to attacks involving unintended content execution.
- Undermines browser-side security mechanisms that rely on strict MIME type handling.

Recommendations

1. **Set the `X-Content-Type-Options` header to `nosniff`** for all responses:

   ```
   http
   Copy code
   X-Content-Type-Options: nosniff
   ```

2. **Ensure proper MIME types** are declared via the `Content-Type` header for all served resources.
3. **Configure your web server or application framework** to apply the header globally:
   - **Apache**:

     ```
     apache
     Copy code
     Header set X-Content-Type-Options "nosniff"
     ```

- **Nginx**:

  ```
  nginx
  Copy code
  add_header X-Content-Type-Options "nosniff";
  ```

- **Express.js (Node)**:

  ```
  js
  Copy code
  app.use((req, res, next) => {
    res.setHeader("X-Content-Type-Options", "nosniff");
    next();
  });
  ```

---

5. RISK ANALYSIS

Risk Level	Count	Priority
High	1	Immediate mitigation required
Medium	11	Fix during next sprint
Low	12	Monitor and fix when feasible
Info	9	No immediate action required

---

6. RECOMMENDATIONS

- Conduct a full code review for insecure design patterns
- Regularly update JavaScript libraries to avoid known vulnerabilities
- Enable CSP, HSTS, and X-Frame-Options headers for hardening
- Enforce token-based CSRF protection and session security policies

---

7. CONCLUSION

The OWASP Juice Shop assessment uncovered several significant issues that, while mostly expected in an intentionally vulnerable app, highlight real-world risks. Using OWASP ZAP, I demonstrated the effectiveness of automated scanning in uncovering vulnerabilities that align with the OWASP Top 10.

---

8. APPENDICES

- 🔧 **Tool Used**: OWASP ZAP v2.16.1
- 💉 **Scan Type**: Active Scan, Spider
- 📎 **Report Format**: HTML export attached (`ZAP_Report.html`)

In **OWASP ZAP**, I used Spider **Mode** (also called the **Spider Scan**) is a **crawling process** used to discover URLs, forms, and other resources on a target web application. It helps ZAP map out the structure of the site **before launching any attacks or scans**.