

# 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	Ao4 – Insecure Design	High	High	1
Application Error Disclosure	Ao6 – Misconfig	Medium	High	7
Absence of Anti-CSRF Tokens	Ao1 – Broken Access Control	Medium	High	1
Missing Anti-clickjacking Header	Ao5 – Misconfig	Medium	High	5
Session ID in URL Rewrite	Ao7 – ID/Auth Failures	Medium	Medium	2
Vulnerable JS Libraries	Ao6 – Components	Medium	Medium	1

Vulnerability	OWASP Category	Risk	Confidence	Count
Private IP Disclosure	Ao6 – Misconfig	Low	Medium	12
Cookie Flags Missing	Ao6 – Misconfig	Low	Medium	14

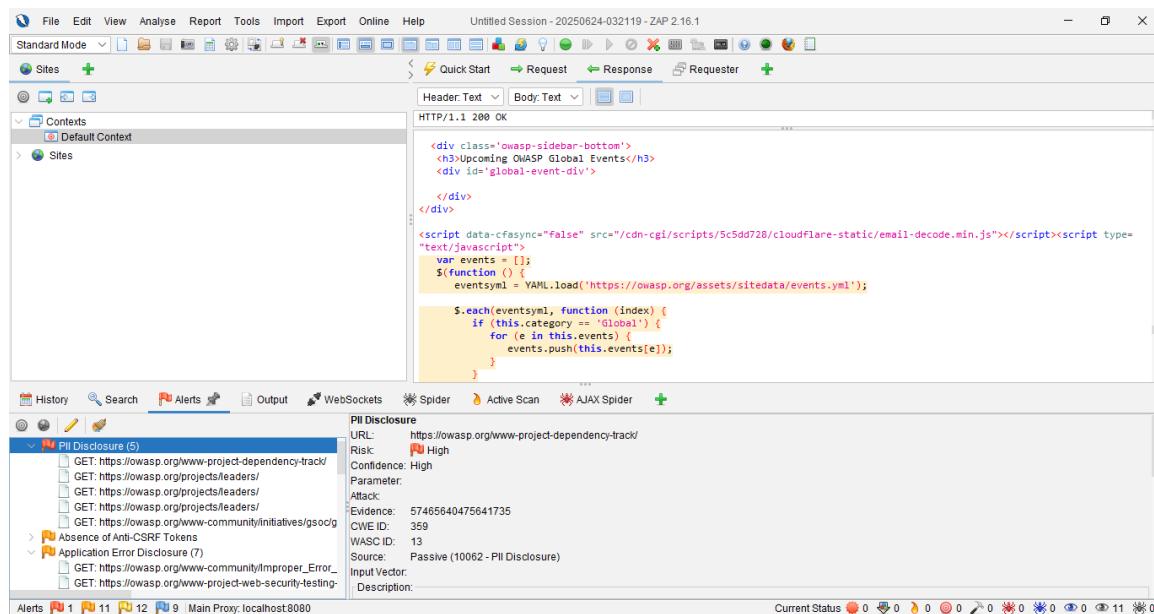
Total Alerts: 33

Risk Breakdown: High (1), Medium (11), Low (12), Informational (9)

---

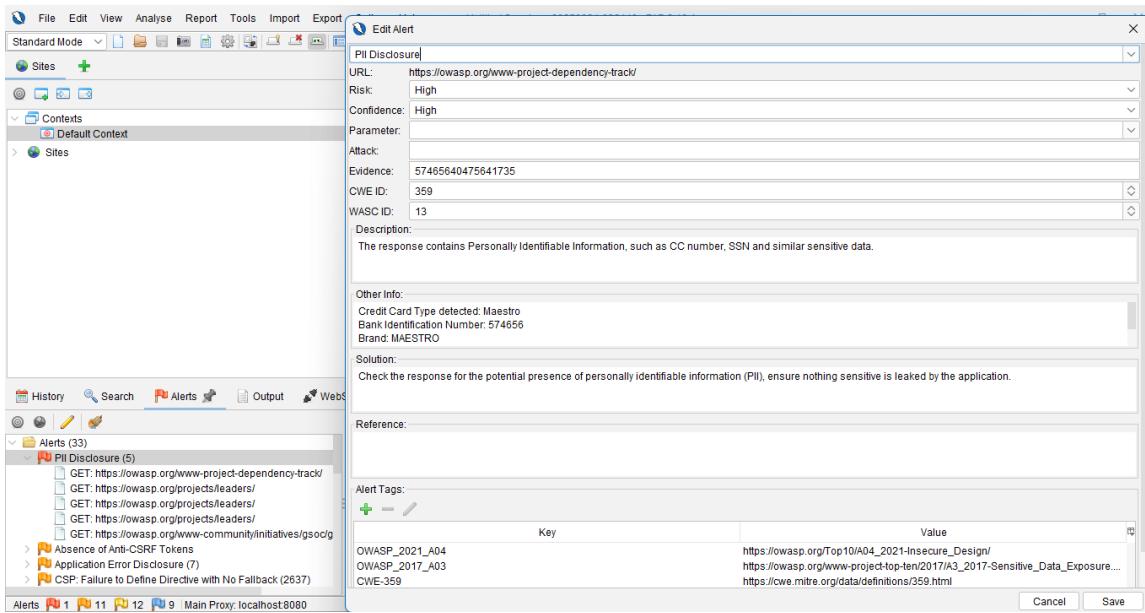
## 4. DETAILED VULNERABILITIES

### ● Vulnerability 1: PII Disclosure (Sensitive Data Exposure)



The screenshot shows the ZAP 2.16.1 interface with the following details:

- Header/Body Text:** Shows an HTTP request and response. The response body contains a script that loads a YAML file from <https://owasp.org/assets/sitedata/events.yml>.
- Alerts:** A summary table shows 1 PII Disclosure, 11 Absence of Anti-CSRF Tokens, 12 Application Error Disclosure, and 9 Main Proxy:localhost:8080.
- Panels:**
  - Sites:** Shows contexts like 'Default Context' and 'Sites'.
  - Spider:** Shows a tree view of vulnerabilities:
    - PII Disclosure (5):** URL: <https://owasp.org/www-project-dependency-track/>, Risk: High, Confidence: High, Parameter: None, Attack: None, Evidence: 57465640475641735, CWE ID: 359, WASC ID: 13, Source: Passive (10062 - PII Disclosure), Input Vector: None, Description: None.
    - Absence of Anti-CSRF Tokens (11):** URLs include <https://owasp.org/projects/leaders/>, <https://owasp.org/projects/leaders/>, <https://owasp.org/projects/leaders/>, <https://owasp.org/projects/leaders/>, <https://owasp.org/www-community/initiatives/gsoc/>, and others.
    - Application Error Disclosure (7):** URLs include [https://owasp.org/www-community/improper\\_Error-handling/](https://owasp.org/www-community/improper_Error-handling/) and <https://owasp.org/www-project-web-security-testing-and-analysis/WAPT/>.
  - Active Scan:** Shows various scan results and metrics at the bottom.



- **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

The screenshot shows the OWASP ZAP interface. On the left, there's a navigation sidebar with 'Sites', 'Contexts' (containing 'Default Context'), and 'Alerts' (33). Under 'Alerts', there are several items, including 'Absence of Anti-CSRF Tokens'. A detailed alert dialog box is open over the interface, titled 'Edit Alert' for 'Absence of Anti-CSRF Tokens'. The alert details are as follows:

- URL:** https://www.zazzle.com/store/owasp\_foundation/products
- Risk:** Medium
- Confidence:** Low
- Parameter:** (empty)
- Attack:** (empty)
- Evidence:** <form action="https://www.zazzle.com/email/signupdialog" method="post" target="emailSignupWindow">
- CWE ID:** 352
- WASC ID:** 9

**Description:**  
No Anti-CSRF tokens were found in a HTML submission form.  
A cross-site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL/form actions in a repeatable way. The nature of the attack is that CSRF

**Other Info:**  
No known Anti-CSRF token [anticsrf, CSRFToken, \_\_RequestVerificationToken, csrfmiddlewaretoken, authenticity\_token, OWASP\_CSRFTOKEN, anoncsrf, csrf\_token, \_csrf, \_csrfSecret, \_\_csrf\_magic, CSRF, \_token, \_csrf\_token, \_csrfToken] was found in the following HTML form: [Form 2: "input\_1" "signupPage" "signupSource"].

**Solution:**  
Phase: Architecture and Design  
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.  
For example, use anti-CSRF packages such as the OWASP CSRFGuard.

**Reference:**  
[https://cheatsheetsseries.owasp.org/cheatsheets/Cross-Site\\_Request\\_Forgery\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetsseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html)  
<https://cwe.mitre.org/data/definitions/352.html>

**Alert Tags:**

Key	Value
OWASP_2021_A01	<a href="https://owasp.org/Top10/A01_2021-Broken_Access_Control/">https://owasp.org/Top10/A01_2021-Broken_Access_Control/</a>
WSTG-V42-SESS-05	<a href="https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/Session_Management/Test_0427_V42-Session_Management">https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/Session_Management/Test_0427_V42-Session_Management</a>
OWASP_2021_A05	<a href="https://owasp.org/Top10/A05_2021-Broken_Access_Control/">https://owasp.org/Top10/A05_2021-Broken_Access_Control/</a>

Buttons at the bottom right of the dialog: Cancel and Save.

- **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)

### Q 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

The screenshot shows the ZAP interface in Standard Mode. The left sidebar shows 'Contexts' with 'Default Context' selected. The main pane displays an alert titled 'Application Error Disclosure' for the URL [https://owasp.org/www-community/improper\\_Error\\_Handling](https://owasp.org/www-community/improper_Error_Handling). The alert details are as follows:

- URL:** https://owasp.org/www-community/improper\_Error\_Handling
- Risk:** Medium
- Confidence:** Medium
- Parameter:**
- Attack:**
- Evidence:** internal error
- CWE ID:** 550
- WASC ID:** 13

**Description:** 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. The alert could be a false positive if the error message is found inside a documentation page.

**Other Info:**

**Solution:** Review the source code of this page. Implement custom error pages. Consider implementing a mechanism to provide a unique error reference/identifier to the client (browser) while logging the details on the server side and not exposing them to the user.

**Reference:**

**Alert Tags:**

Key	Value
WSTG-v42-ERRH-02	<a href="https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/4.2-Testing_for_Security_Misconfiguration/4.2.1-Information_Exposure_Through_Error_Messages">https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/4.2-Testing_for_Security_Misconfiguration/4.2.1-Information_Exposure_Through_Error_Messages</a>

**Buttons:** Cancel, Save

### 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

#### Q 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

The screenshot shows the OWASP ZAP interface with the 'Edit Alert' dialog open. The alert details are as follows:

- Title:** Content Security Policy (CSP) Header Not Set
- URL:** https://owasp.org/cdn-cgi/email-protection
- Risk:** Medium
- Confidence:** High
- Parameter:** [empty]
- Attack:** [empty]
- Evidence:** [empty]
- CWE ID:** 693
- WASC ID:** 15
- Description:** Content Security Policy (CSP) is an added layer of security that helps to detect and mitigate certain types of attacks, including Cross Site Scripting (XSS) and data injection attacks. These attacks are used for everything from data theft to site defacement or distribution of malware. CSP provides a set of standard HTTP headers that allow website owners to declare approved sources of content that browsers should be allowed to load on that page — covered types are
- Solution:** Ensure that your web server, application server, load balancer, etc. is configured to set the Content-Security-Policy header.
- Reference:** [https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing\\_Content\\_Security\\_Policy](https://developer.mozilla.org/en-US/docs/Web/Security/CSP/Introducing_Content_Security_Policy), [https://cheatsheetsseries.owasp.org/cheatsheets/Content\\_Security\\_Policy\\_Cheat\\_Sheet.html](https://cheatsheetsseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html), <https://www.w3.org/TR/CSP/>
- Alert Tags:** [empty]
- Input View:** A table showing the CWE reference and its URL.

- **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.
-

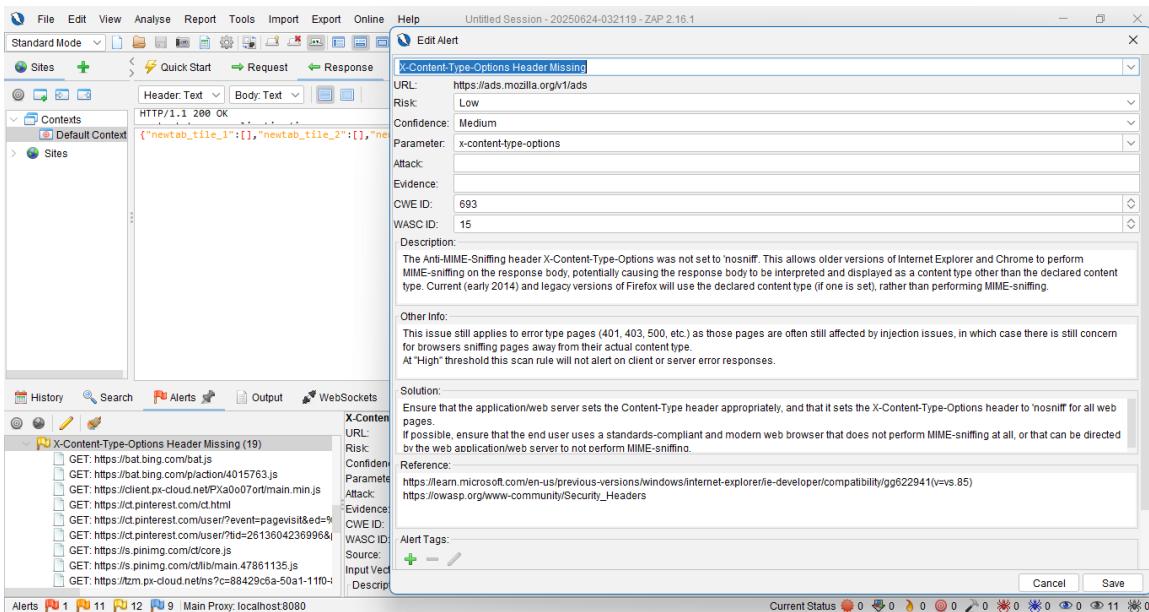
## Recommendations

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:
  - o <https://report-uri.com/home/generate>
  - o <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

The screenshot shows the ZAP interface with the following details:

- Alert Details:** X-Content-Type-Options Header Missing
- URL:** https://ads.mozilla.org/v1/ads
- Risk:** Low
- Confidence:** Medium
- Parameter:** x-content-type-options
- Attack:** None
- Evidence:** None
- CWE ID:** 693
- WASC ID:** 15
- Description:** The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing.
- Other Info:** This issue still applies to error type pages (401, 403, 500, etc.) as those pages are often still affected by injection issues, in which case there is still concern for browsers sniffing pages away from their actual content type. At 'High' threshold this scan rule will not alert on client or server error responses.
- Solution:** Ensure that the application/web server sets the Content-Type header appropriately, and that it sets the X-Content-Type-Options header to 'nosniff' for all web pages. If possible, ensure that the end user uses a standards-compliant and modern web browser that does not perform MIME-sniffing at all, or that can be directed by the web application/web server to not perform MIME-sniffing.
- Reference:** [https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/compatibility/gg622941(v=vs.85)), [https://owasp.org/www-community/security\\_headers](https://owasp.org/www-community/security_headers)
- Alert Tags:** None

Below the alert details, a list of 19 affected URLs is shown, including various GET requests to sites like bing.com, pinterest.com, and pinimg.com.

- **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:
  - o **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**.

The screenshot shows the OWASP ZAP interface in Standard Mode. The main window is titled "Automated Scan". In the center, there's a lightning bolt icon and the text: "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 given permission to test." Below this, the "URL to attack:" field contains "https://owasp.org/www-project-juice-shop/" and has a "Select..." button. There are checkboxes for "Use traditional spider:" (checked) and "Use ajax spider:" (with dropdowns for "If Modern" and "Firefox"). Buttons for "Attack" and "Stop" are present. The "Progress:" status is "Using traditional spider to discover the content". At the bottom, a progress bar shows 10% completion. The bottom navigation bar includes tabs for History, Search, Alerts, Output, WebSockets, Spider, Active Scan, and a New Scan button. The "Spider" tab is selected. The main content area shows a table with columns: Processed, Method, URI, and Flags. The table lists various URLs with their corresponding methods (mostly GET) and flags (mostly Out of Scope). The bottom status bar shows "Current Status" with various icons and counts (e.g., 0 alerts, 6 errors, 2 warnings, 4 info, 0 critical, etc.).