JRU-Pulse Web Application Security Assessment Report

Author: Timoteo, Juan Carlos Miguel V.

Statement of Scope

The security assessment was conducted on the local development version of the *JRU Pulse* web application. The application was hosted on a Windows 11 workstation using XAMPP (Apache, PHP, and MySQL). The tested instance was accessible at http://localhost/project-2-jru-pulse-main/.

The scope of this assessment included:

- **Dynamic Application Security Testing (DAST):** using **OWASP ZAP** to identify runtime vulnerabilities through active scanning of the running application.
- **Server and Configuration Review:** validation of PHP, Apache, and XAMPP settings related to information disclosure, cookie handling, and header hardening.
- **Source Code Security Practices (limited):** specific adjustments to PHP session handling, .env file placement, and Apache .htaccess directives.

No production systems, public-facing environments, or real user data were included in scope. The database schema was loaded for testing, but no sensitive or real records were used. All testing was performed in a contained local development environment.

Methodology

The security testing was carried out using a structured process that aligns with common application security assessment practices. The following steps were followed:

1. **Environment Setup**
    - Deployed the application locally using XAMPP with Apache, PHP, and MySQL.
    - Configured the database schema to replicate the intended application environment.
    - Ensured OWASP ZAP was installed and configured to intercept and analyze requests to http://localhost/project-2-jru-pulse-main/.

2. **Reconnaissance and Baseline Scan**
    - Performed an **initial automated scan** using OWASP ZAP's "Automated Scan" feature to establish a baseline of detected issues.
    - Collected findings including missing security headers, cookie attributes, exposure of sensitive files, and server information leakage.
3. **Manual Verification and Review**
    - Validated the identified findings by manually inspecting HTTP response headers, cookies, and directory structures using browser developer tools and file system checks.
    - Reviewed application source code to verify configuration issues (e.g., session_start() usage, .env file placement, .htaccess settings).
4. **Remediation and Hardening**
    - Applied security controls in the application code (e.g., PHP session cookie flags) and Apache configuration (e.g., disabling server signature, adding Content Security Policy).
    - Moved sensitive files out of the public webroot and restricted access using .htaccess rules.
    - Disabled directory browsing and reduced information disclosure from the server.
5. **Re-Scan and Validation**
    - Re-ran OWASP ZAP scans after changes to confirm that high- and medium-severity findings were addressed.
    - Compared "before" and "after" results to demonstrate reduced risk.
    - Documented remaining informational alerts and justified them as non-critical or false positives.
6. **Reporting**
    - Consolidated findings, remediation actions, and validation evidence into this report.
    - Screenshots of ZAP alerts, response headers, and configuration changes are included in the appendices as supporting proof.

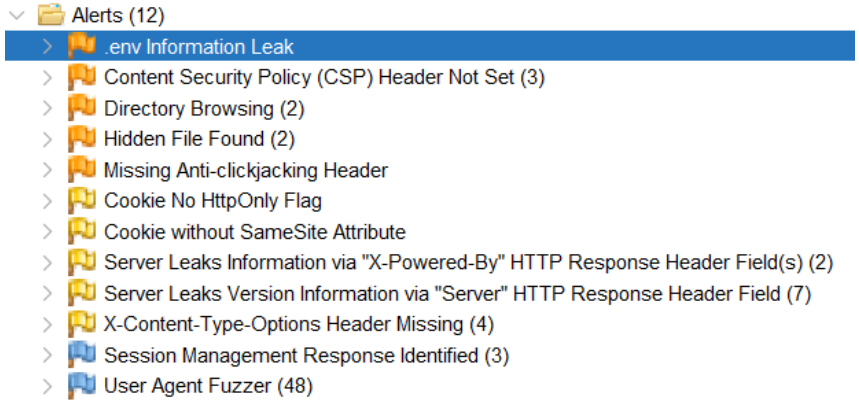
Findings & Remediation Summary

The initial OWASP ZAP scan identified several high- and medium-severity issues, primarily related to configuration and security headers. Each finding was reviewed, and mitigation steps were applied at the application or server level. A re-scan was then performed to validate the effectiveness of these measures.

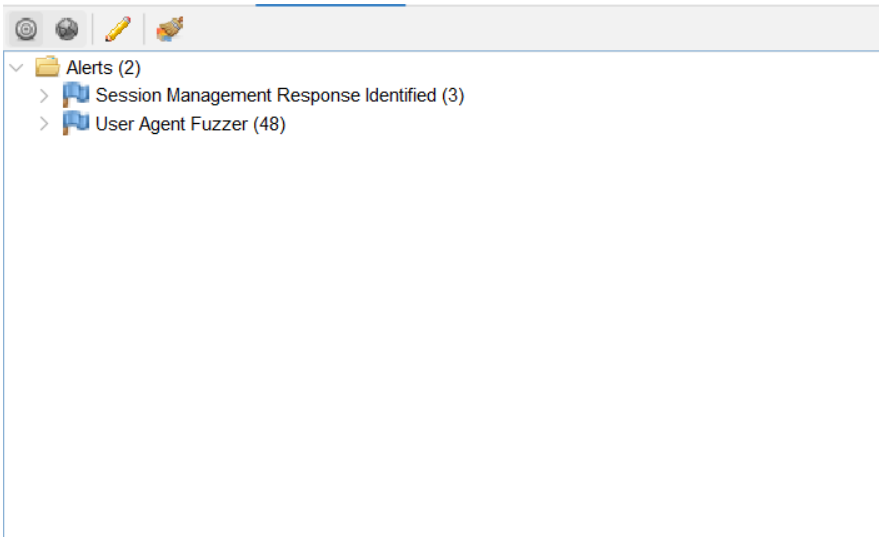
The table below summarizes the main findings, actions taken, and validation outcomes:

Finding	Severity	Description	Remediation Action	Validation Result
.env File Exposure	High	The application's .env file containing credentials was accessible under the webroot.	Relocated .env outside of the public directory; restricted access via .htaccess.	Direct access attempt returned 403 Forbidden. ZAP no longer flagged .env.
Content Security Policy (CSP) Not Set	Medium	No CSP header was present, leaving the app vulnerable to XSS and injection via external resources.	Added CSP in .htaccess: default-src 'self'; frame-ancestors 'none'; object-src 'none'; base-uri 'self';.	Browser Developer Tools confirmed CSP header present. ZAP reported improved CSP compliance.
Directory Browsing Enabled	Medium	Web directories were viewable if index files were missing.	Disabled directory listing with .htaccess directive Options -Indexes.	Attempt to access a directory returned 403 Forbidden.
Insecure Cookies (No HttpOnly / SameSite)	Medium	Session cookies lacked HttpOnly and SameSite attributes.	Updated session handling with session_set_cookie_params() before session_start().	Browser Network tab showed cookies with HttpOnly and SameSite=Strict.
Server Version Disclosure	Low	Apache revealed version number and PHP exposed version info via X-Powered-By.	Updated Apache config: ServerSignature Off, ServerTokens Prod; set expose_php = Off in php.ini.	Response headers showed generic Server: Apache without version strings.
Hidden Files Detected	Medium	Backup or hidden files were accessible under the project root.	Removed unnecessary files; blocked patterns (.git, .bak, .swp, .env) in .htaccess.	Attempt to access flagged files returned 403 Forbidden.
Informational Alerts (User-Agent Fuzzing, Session Management Identified)	Info	ZAP reported variations in responses based on User-Agent and session use.	Reviewed manually; no sensitive data exposed. Marked as informational only.	Documented justification in report.

Before:



After:



Summary of Results

- The **critical issue** of .env exposure was fully mitigated.
- **Header-based protections** (CSP, X-Frame-Options, X-Content-Type-Options) were successfully applied.
- **Cookie handling** was hardened with HttpOnly and SameSite.
- **Server disclosure** was minimized by adjusting Apache and PHP settings.
- **Directory browsing** and access to hidden files were blocked.
- Remaining informational alerts were assessed and documented as non-critical.

Implications for Live Web Hosting Security

While this assessment was conducted in a local development environment, the findings and remediations directly translate to strengthening security in a production (live) hosting scenario. The vulnerabilities identified — such as exposed configuration files, missing security headers, and insecure session handling — are among the most common attack vectors exploited on publicly accessible servers.

Applying these remediations in a live hosting environment provides the following benefits:

- **Protection of Sensitive Credentials:** Relocating .env files and restricting direct access prevents database and API credentials from being exposed to the internet.
- **Improved Session Security:** Setting secure cookie attributes (HttpOnly, SameSite) reduces the risk of session hijacking and cross-site request forgery (CSRF).
- **Resilience Against Common Attacks:** Implementing CSP, X-Frame-Options, and related headers mitigates common exploits such as cross-site scripting (XSS), clickjacking, and MIME sniffing.
- **Minimized Attack Surface:** Disabling directory browsing and blocking hidden/backup files limits the amount of information an attacker can gather about the system.
- **Reduced Fingerprinting:** Removing server and PHP version information makes it more difficult for attackers to target known vulnerabilities.

By addressing these issues before deployment, the application achieves a stronger **baseline security posture**, making it significantly more resistant to automated scans, opportunistic attacks, and targeted exploitation attempts on a live server.

Conclusion

The security testing of *JRU Pulse* successfully demonstrated how structured vulnerability assessment can uncover and remediate weaknesses in a web application environment. Through the use of **OWASP ZAP** for dynamic testing, paired with manual validation and configuration adjustments, the system's security posture was significantly improved.

Critical risks, such as the exposure of environment files and insecure session handling, were resolved, while additional protections were introduced through security headers, cookie

hardening, and server configuration changes. As a result, high-severity vulnerabilities were eliminated, and the overall risk profile of the application was reduced to a manageable level.

This exercise highlights the importance of proactive security measures in the software development lifecycle. Even in a controlled local environment, insecure defaults and overlooked configurations can introduce serious risks if not addressed. By implementing these remediations now, the project is better prepared for deployment in a live hosting scenario, where threats are more aggressive and constant.

Ultimately, this assessment confirms that integrating **regular vulnerability scanning, secure coding practices, and server hardening** is essential to maintaining a resilient and trustworthy application.

Alignment with ISO/IEC 27001 Principles

The activities performed during this security assessment align with the objectives and controls defined in the **ISO/IEC 27001 Information Security Management System (ISMS)** framework. While this project was conducted in a development environment, the processes followed demonstrate compliance with key ISO 27001 domains:

- **A.12.6 – Technical Vulnerability Management**
  The use of OWASP ZAP to identify vulnerabilities and the subsequent remediation process aligns with the requirement to establish procedures for identifying, evaluating, and addressing technical vulnerabilities in information systems.

- **A.9 – Access Control**
  Securing sensitive files (e.g., .env) and implementing session cookie hardening demonstrate control over unauthorized access to credentials and session data, consistent with access control requirements.

- **A.13.1 – Network Security Management**
  Restricting information disclosure (server version, directory listings, and hidden files) reduces the attack surface and aligns with the principle of safeguarding network and system boundaries.

- **A.14 – System Acquisition, Development, and Maintenance**
  The remediation process reflects secure coding and system hardening practices, showing integration of security into the development lifecycle as required by ISO 27001.

- **A.18 – Compliance**
  Documenting the assessment scope, methodology, findings, and mitigations

demonstrates due diligence and accountability, supporting compliance requirements under ISO 27001.

Summary of Alignment

This assessment followed the **risk-based approach** emphasized in ISO 27001: identifying risks, treating them with appropriate controls, and validating their effectiveness. This structured methodology ensures that even in a development context, the project demonstrates adherence to international information security standards.

Security Changes Implemented

The following changes were made to improve the security posture of the *JRU Pulse* web application and its hosting environment:

1. File and Directory Security

- **Moved .env file** outside of the project's public webroot.
- Added .htaccess rules to block access to hidden and backup files (.env, .git, *.bak, *.swp).
- Disabled **directory browsing** using Options -Indexes.

2. Session and Cookie Hardening

- Applied session_set_cookie_params() with:
    - HttpOnly = true
    - Secure = true (if HTTPS is used in deployment)
    - SameSite = Strict
- Verified cookie attributes via browser developer tools.
- Ensured session_start() is called after cookie parameters are set.

```php
session_set_cookie_params([
    'httponly' => true,
    'samesite' => 'Strict',
    'secure'   => false // Will set true if once we enable HTTPS during deployment
]);
```

3. Security Headers

Added the following headers in Apache .htaccess:

```
# Security Headers
<IfModule mod_headers.c>
    # Prevent MIME sniffing
    Header set X-Content-Type-Options "nosniff"

    # Clickjacking protection
    Header always set X-Frame-Options "DENY"

    # Basic Content Security Policy (safe default)
    Header set Content-Security-Policy "default-src 'self'; frame-ancestors 'none'; object-src 'none'; base-uri
    'self';"

    # Hide X-Powered-By
    Header unset X-Powered-By
</IfModule>
```

4. Server Configuration Hardening

- Disabled server signature and version disclosure:
    - In Apache: ServerSignature Off, ServerTokens Prod
    - In php.ini: expose_php = Off
- Restarted Apache to apply changes.

```
; Decides whether PHP may expose the fact that it is installed on the server
; (e.g. by adding its signature to the Web server header).  It is no security
; threat in any way, but it makes it possible to determine whether you use PHP
; on your server or not.
; https://php.net/expose-php
expose_php=Off
```

5. Cookie Adjustments

- Updated PHP session handling for cookie security.
- Ensured database credentials are read from .env outside the public directory.
- Removed redundant or test files that could expose information.

```
setcookie("mycookie", "value", [
    'httponly' => true,
    'samesite' => 'Strict',
    'secure'   => false // Will set true if once we enable HTTPS during deployment
]);
```

6. Validation Steps

- Re-ran OWASP ZAP scan and confirmed that:

    - High-severity .env exposure issue was eliminated.
    - CSP and cookie protections were detected as active.
    - Directory browsing attempts returned 403 Forbidden.
    - Server response headers no longer revealed version information.
- Documented remaining **informational alerts** as non-critical after review.