Assessment 2
Web Application Security Testing Report (OWASP ZAP + WSTG)
EPT232 Ethical Hacking and Penetration Testing

Prepared by:
Smriti Parajuli (Student ID: MDS3000025)

Submitted to:
Sarada Hettiarachchi
Bachelor of Software Engineering (AI)
Media Design School

Date:
11th May 2025

1. Introduction

Web applications play a critical role in today's digital infrastructure, supporting services across sectors such as e-commerce, healthcare, education, and enterprise systems. However, their exposure to the internet makes them inherently vulnerable to a wide range of cybersecurity threats. To proactively address these risks, web application security testing employs a structured methodology aimed at identifying and mitigating vulnerabilities before they can be exploited by malicious actors. This practical assessment focuses on conducting a hands-on security evaluation of Damn Vulnerable Web Application (DVWA), a deliberately insecure PHP/MySQL-based web application designed for educational and research purposes. The assessment was performed using OWASP ZAP (Zed Attack Proxy), a powerful open-source web application security scanner, within a Kali Linux virtual environment.

The methodology guiding this assessment is based on the OWASP Web Security Testing Guide (WSTG), version 4.2. The WSTG is widely recognized as a premier industry resource for security testing of web applications and services. It provides a comprehensive and community-driven framework of best practices developed by global cybersecurity professionals and penetration testers (OWASP, 2020). The guide outlines techniques for evaluating common web security issues such as input validation flaws, broken authentication, insecure session management, and improper error handling.

The primary objective of this report is to identify, analyze, and recommend mitigations for at least ten security vulnerabilities discovered during the testing process. Each identified issue is mapped to its corresponding WSTG category to ensure structured and standards-based analysis.

2. Methodology

2.1 Tools and Technologies

- **Kali Linux (Custom Build):**

  A specialized Debian-based Linux distribution widely used for digital forensics and penetration testing. It includes a wide range of security tools by default, including OWASP ZAP.

- **OWASP ZAP (Zed Attack Proxy)**:

  An open-source web application security scanner developed by the OWASP Foundation. ZAP is designed for finding security vulnerabilities in web applications through both automated and manual testing methods.

➢ **DVWA (Damn Vulnerable Web Application)**:

A purpose-built, deliberately vulnerable web application created to help security professionals and students learn about common web vulnerabilities and assess detection tools and methodologies.

➢ **OWASP Web Security Testing Guide (WSTG) v4.2**:

A community-driven framework that provides a comprehensive methodology for web application security testing. Each vulnerability identified during the scan is aligned with a specific WSTG category (e.g., WSTG-ATHN-01 for authentication testing).

2.2 Security Assessment Steps

The security assessment was conducted through a structured multi-phase process designed to emulate both user behavior and automated penetration testing techniques. The first phase involved setting up the testing environment by launching the custom Kali Linux virtual machine, which served as the primary platform for executing security tools. OWASP ZAP was verified for correct installation and functionality. The target application, Damn Vulnerable Web Application (DVWA), was hosted locally via Docker and accessed at `http://127.0.0.1:8888`. Before beginning the scans, the DVWA environment was configured by setting the security level to "Low" to expose a broader range of vulnerabilities. Additionally, the DVWA database was reset via the setup panel to ensure consistency in test conditions.

The next step was manual exploration using OWASP ZAP's "Manual Explore" feature. This allowed the tester to log in to DVWA and interact with key modules such as SQL Injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), and File Upload. During this phase, ZAP passively monitored the HTTP requests and responses while building a detailed site tree of the application. Manual exploration ensured that all authenticated and hidden routes often missed by automated tools were captured and made available for further scanning. This enhanced the accuracy and coverage of the overall test.

Following the manual phase, an automated scan was performed. The target URL was entered into ZAP's "Automated Scan" window, which initiated the spidering process to crawl through accessible application links and pages. This was followed by the active scan phase, where ZAP systematically applied built-in attack payloads to detect potential vulnerabilities such as insecure HTTP headers, injection flaws, and weak session configurations. This automated process closely simulated common attack patterns used by threat actors.

The final stage involved alert analysis. After the scan was completed, the results were reviewed in ZAP's "Alerts" tab. A minimum of ten unique vulnerabilities were identified and documented, each with its corresponding risk level, description, and evidence. To ensure the assessment

aligned with best practices, each vulnerability was mapped to a specific test case in the OWASP Web Security Testing Guide (WSTG v4.2). This structured mapping facilitated a clear understanding of the identified threats and provided guidance for proposing targeted mitigations.

2.3 Ethical Considerations

This security assessment was conducted entirely within a safe environment using a virtual machine setup and a deliberately vulnerable application Damn Vulnerable Web Application (DVWA). DVWA is designed specifically for educational and research purposes, making it an ideal platform for students and cybersecurity practitioners to practice vulnerability detection without risk to real-world systems. At no point were any live websites, third-party systems, or unauthorized networks accessed during the course of this assessment.

All scanning activities, including manual exploration, automated spidering, and active vulnerability scanning using OWASP ZAP, were confined to the local instance of DVWA running on http://127.0.0.1:8888. This ensures that the integrity, confidentiality, and availability of external systems and data are fully preserved. The assessment strictly followed ethical hacking principles, including informed usage, respect for system boundaries, and adherence to responsible disclosure standards. Since no sensitive or personal data was involved, the risk of harm was completely mitigated.

The intent of this assessment is purely academic and skill-building in nature. It serves to enhance understanding of web application vulnerabilities, ethical penetration testing workflows, and structured vulnerability reporting aligned with the OWASP Web Security Testing Guide (WSTG). By simulating real-world scanning techniques in a controlled environment, this exercise contributes meaningfully to the development of professional security testing practices.

3. Findings and Analysis

The OWASP ZAP scan conducted against the DVWA environment identified a variety of common web application vulnerabilities. These findings were discovered using a combination of passive and active scanning techniques, and each vulnerability was mapped to a corresponding section in the OWASP Web Security Testing Guide (WSTG v4.2) for structured reporting.
Vulnerability

Vulnerability 1: Application Error Disclosure

OWASP ZAP identified multiple instances of application error disclosure within the DVWA environment. These error messages reveal sensitive implementation details such as directory structure or file paths (e.g., Parent Directory), which could aid attackers in crafting targeted attacks. Such disclosures are typically caused by unhandled exceptions and improper error

handling mechanisms. This vulnerability is rated as medium risk and aligns with WSTG-ERRH-01. Proper custom error pages and logging practices should be implemented to avoid exposing such information to end users.

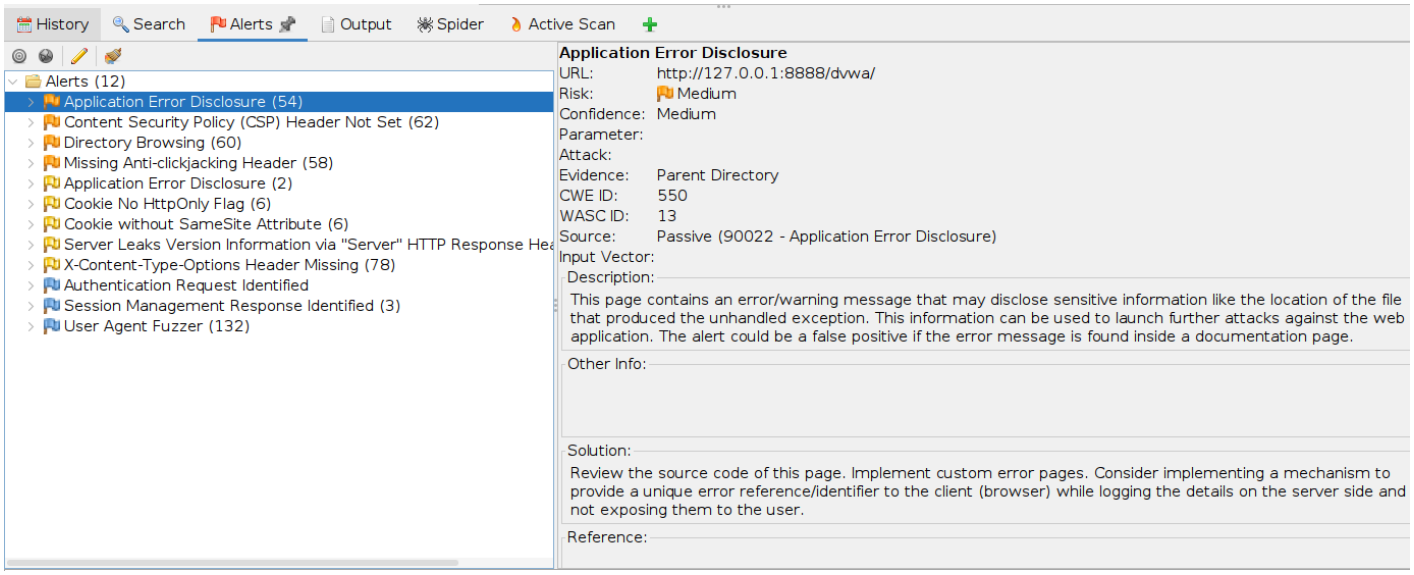

Figure1: Application Error Disclosure Alert Identified by OWASP ZAP

Vulnerability 2: Content Security Policy (CSP) Header Not Set

ZAP detected that the application does not include a Content-Security-Policy (CSP) header in its responses. According to the OWASP Foundation (2020), the CSP header plays a crucial role in mitigating cross-site scripting (XSS) attacks by restricting which content sources can be executed in the browser. Without this header, browsers cannot enforce content restrictions, leaving the application vulnerable to client-side attacks. This vulnerability is classified as medium-risk and aligns with WSTG-CLNT-01.

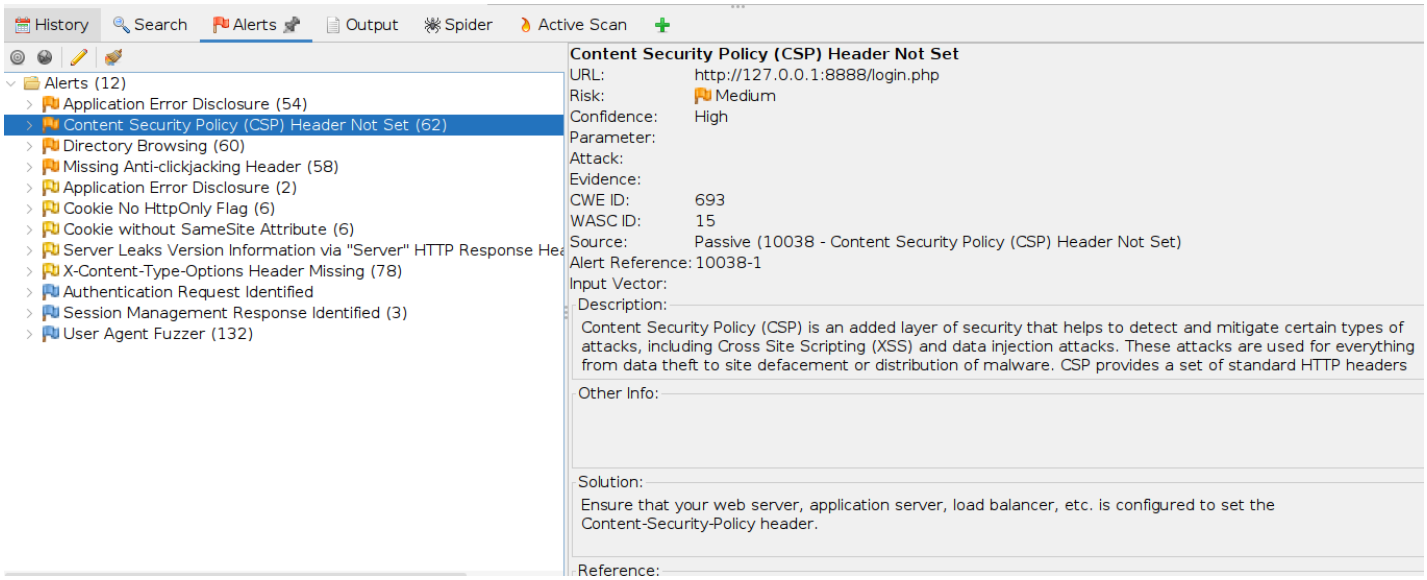

Figure 2: CSP Header Missing Alert Identified by OWASP ZAP

Vulnerability 3: Directory Browsing Enabled

ZAP detected that directory browsing is enabled on the DVWA server. This configuration allows users to view a full listing of the directory contents, including files and scripts that may not be meant for public access. This could expose sensitive files, backup scripts, or configuration data that attackers could exploit to gather further intelligence on the system. The vulnerability is rated as medium risk and aligns with WSTG-CONF-06, which addresses insecure server configurations and information leakage.

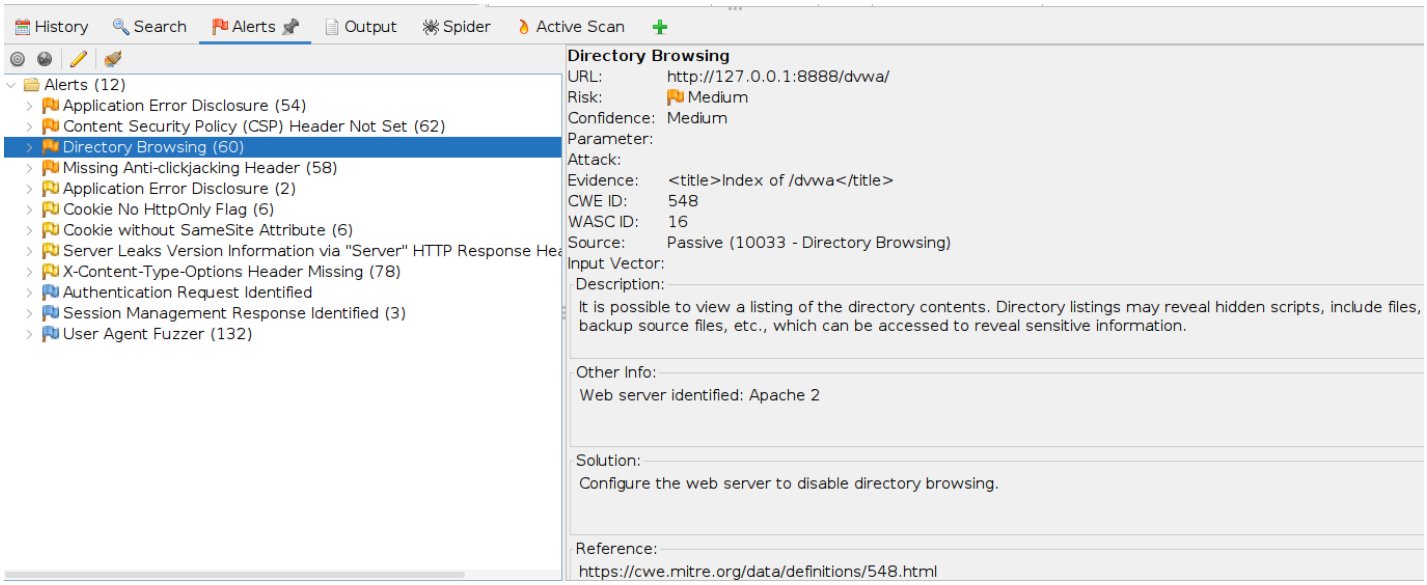

Figure 3: Directory Browsing Alert Detected by OWASP ZAP

Vulnerability 4: Missing Anti-clickjacking Header

OWASP ZAP identified that the application does not implement anti-clickjacking protection headers such as X-Frame-Options or Content-Security-Policy with a frame-ancestors directive. This omission means the application is potentially vulnerable to clickjacking attacks, where malicious websites can embed the application's pages within iframes and trick users into performing unintended actions. The vulnerability is considered medium risk due to the impact on user trust and potential for session manipulation. This finding corresponds with WSTG-CLNT-09, which deals with testing browser-based security controls.

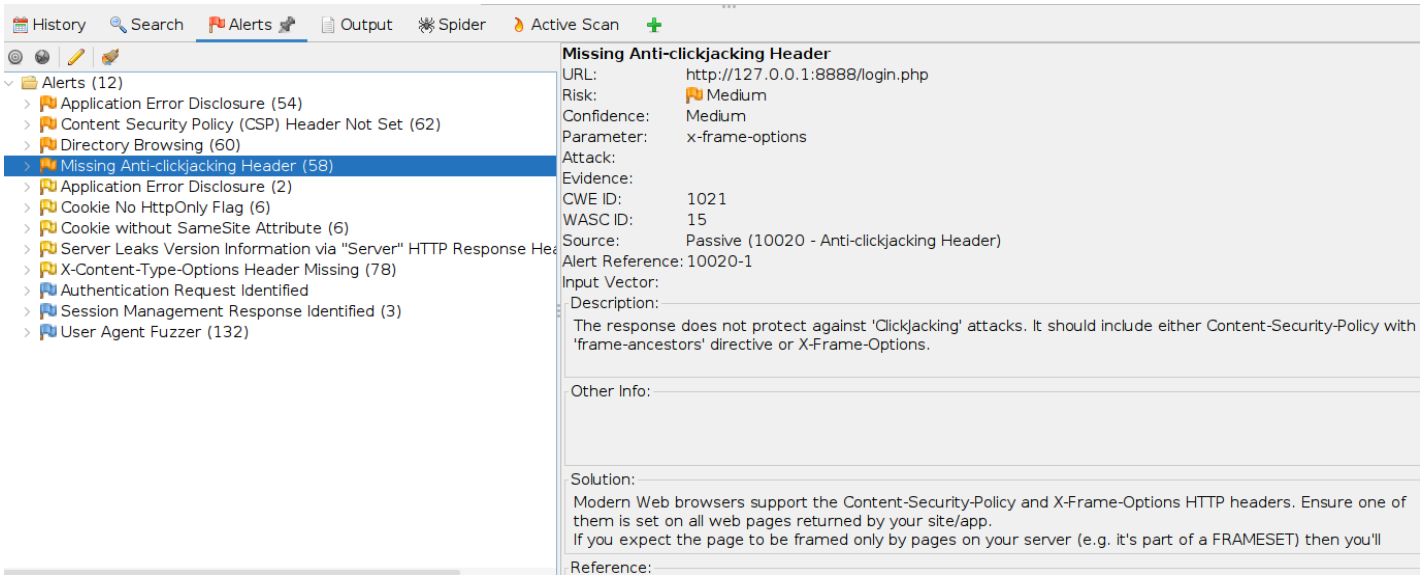

Figure 4: Missing Anti-clickjacking Header Alert Detected in OWASP ZAP

Vulnerability 5: Application Error Disclosure

During the scan, OWASP ZAP detected that the application discloses detailed error messages to the client, particularly through HTTP 500 Internal Server Error responses. These error messages can include sensitive implementation details, such as file paths or database references, which could aid attackers in crafting targeted attacks (e.g., SQL injection or local file inclusion). In this instance, the disclosure occurred in DBMS/MySQL.php, potentially revealing backend structure. This vulnerability, while rated low, still represents a significant risk in insecure environments and aligns with WSTG-ERRH-01, which focuses on error handling testing

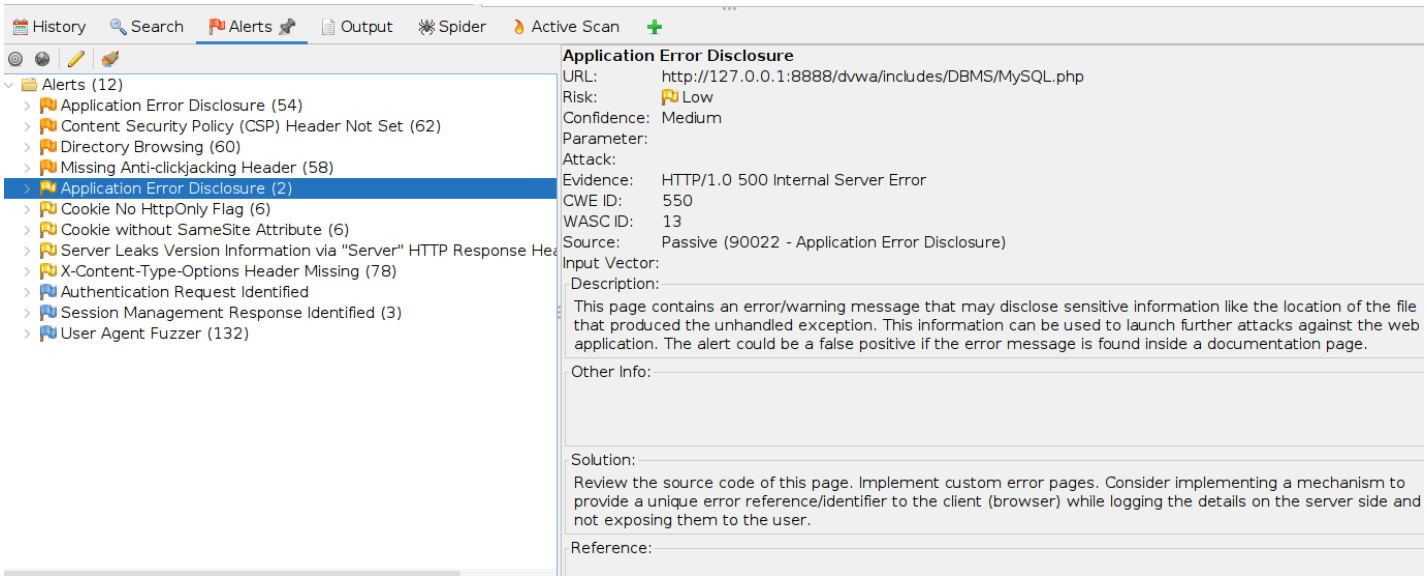

Figure 5: Application Error Disclosure Alert in OWASP ZAP

Vulnerability 6: Cookie Without HttpOnly Flag

OWASP ZAP detected that the application sets cookies (specifically PHPSESSID) without the HttpOnly attribute. The HttpOnly flag is a critical security control that prevents client-side scripts (e.g., JavaScript) from accessing session cookies. Without this flag, an attacker who successfully injects a script into the application via an XSS vulnerability, for instance can steal the session cookie and impersonate the user, potentially leading to session hijacking. Although this issue is categorized as low risk in ZAP, it poses a significant threat when combined with client-side injection flaws. This vulnerability maps to WSTG-SESS-02 and is a common oversight in cookie handling practices.

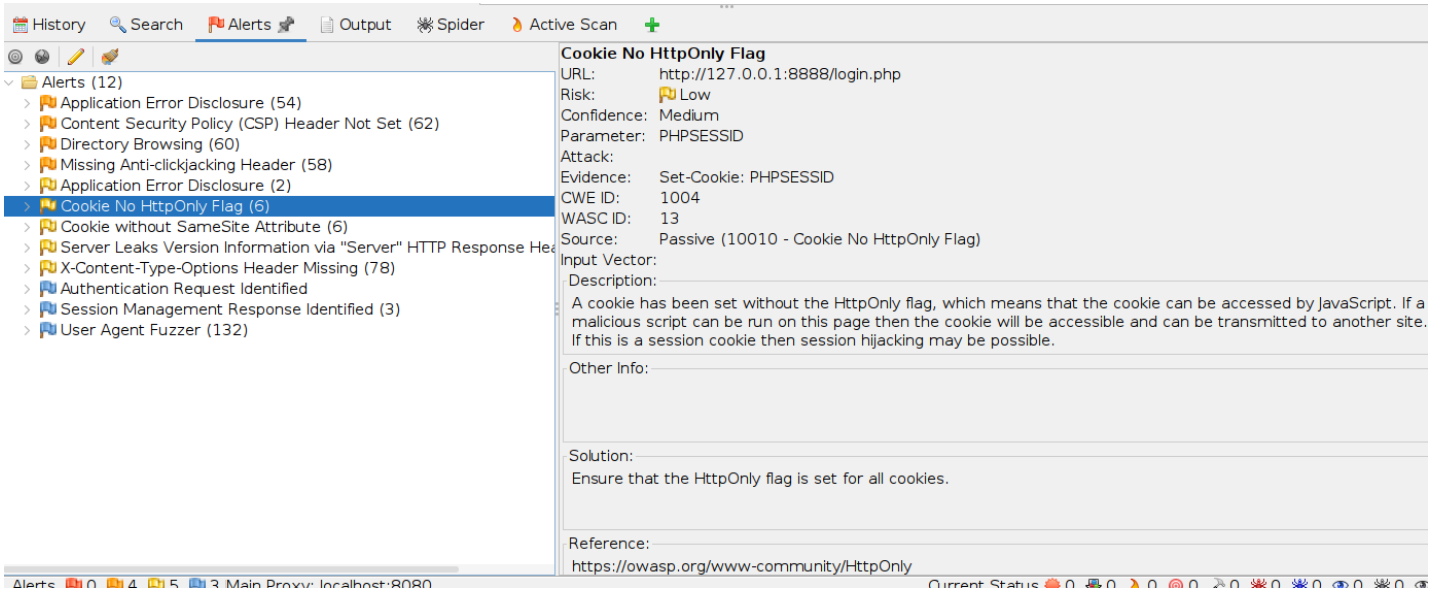

Figure 6: Cookie Without HttpOnly Flag Alert in OWASP ZAP

Vulnerability 7: Cookie without SameSite Attribute

The OWASP ZAP scan identified that session cookies were set without the SameSite attribute, specifically the PHPSESSID cookie. This attribute is a key defense against Cross-Site Request Forgery (CSRF) attacks, as it restricts how cookies are sent with cross-site requests. Without this control, cookies can be transmitted when the user interacts with malicious websites, leading to unauthorized actions on the web application. Although categorized as a low-risk issue, it significantly increases exposure to CSRF and timing attacks. Ensuring the SameSite attribute is set to Lax or ideally Strict is recommended to mitigate this vulnerability. This issue aligns with WSTG-SESS-05.

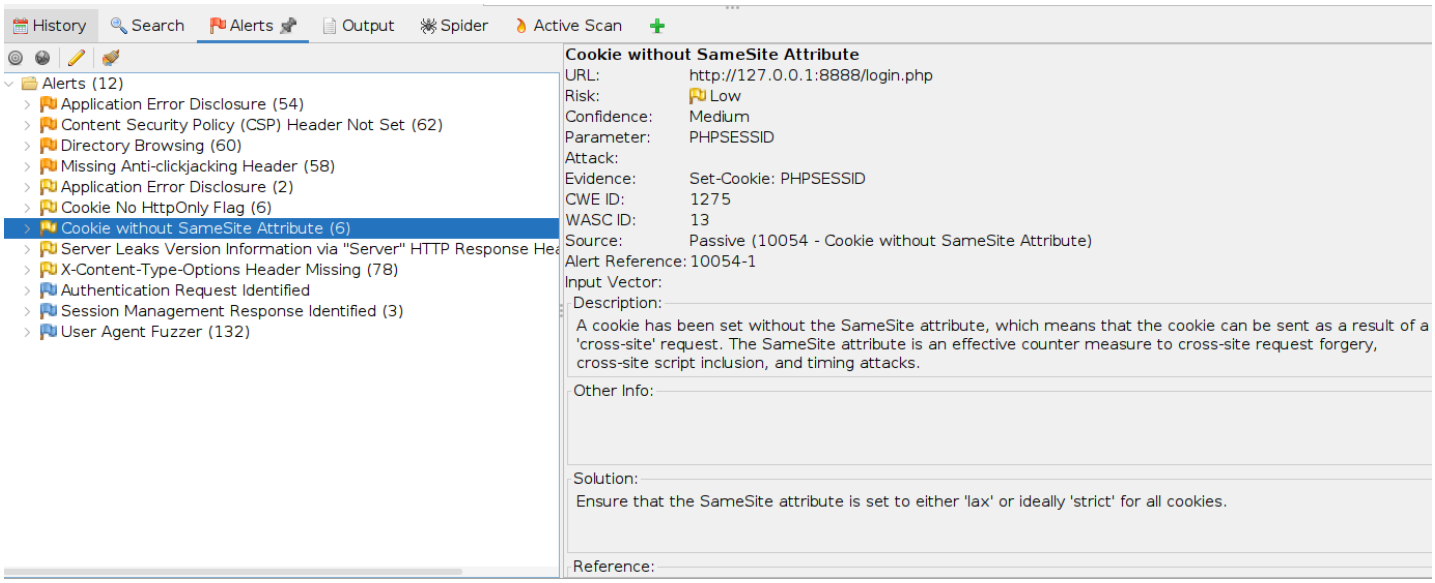

Figure 7: Cookie without SameSite Attribute Alert in OWASP ZAP.

Vulnerability 8: Server Leaks Version Information via "Server" HTTP Response Header

During the OWASP ZAP scan, it was discovered that the HTTP response header discloses the server type and version in this case, Apache/2.4.25 (Debian). This information is included in the "Server" response header field. While this may seem harmless, such disclosures can provide attackers with critical insights about the underlying infrastructure. If an attacker knows the exact server version, they can tailor attacks using known exploits for that version. This type of information leakage is commonly categorized under information disclosure vulnerabilities. It is rated as low risk but poses a significant reconnaissance advantage to attackers. Mitigation involves configuring the server to either suppress or generalize the "Server" header. This vulnerability aligns with WSTG-INFO-01

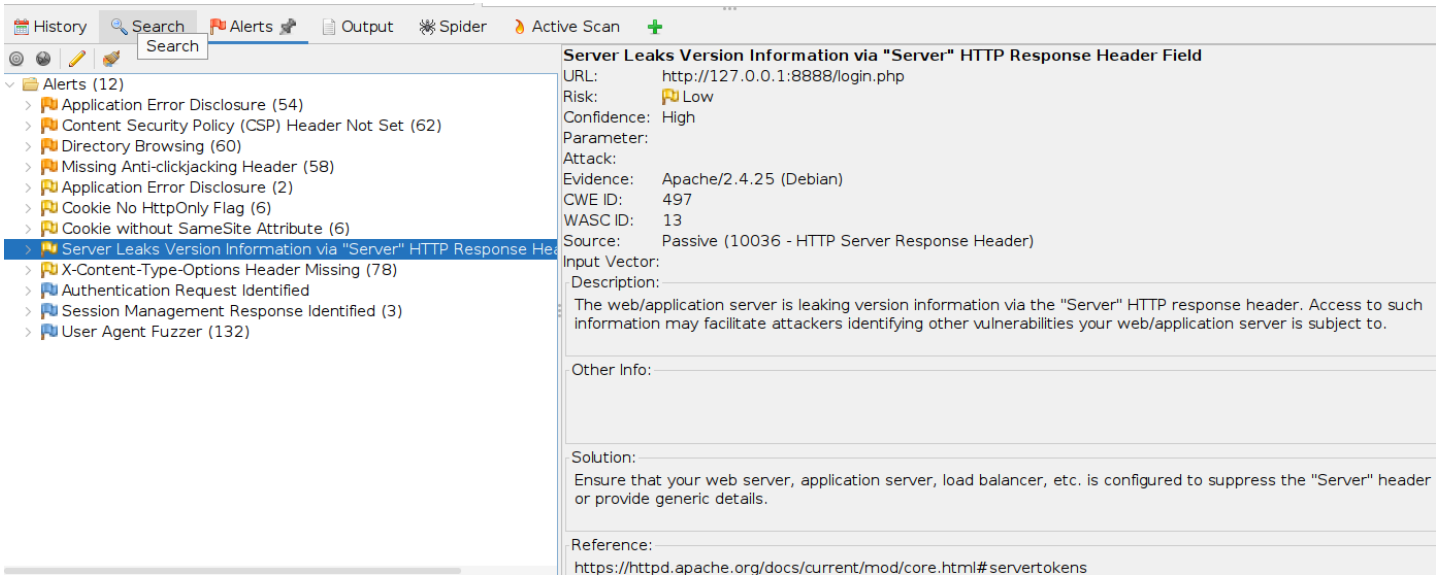

Figure 8: Server Version Disclosure in Response Header (OWASP ZAP)

Vulnerability 9: X-Content-Type-Options Header Missing

OWASP ZAP identified the absence of the X-Content-Type-Options HTTP header in the server's response. This header, when set to nosniff, prevents browsers from MIME-sniffing a response away from the declared content-type. Without it, certain browsers (particularly older versions of Internet Explorer and Chrome) may attempt to interpret the content type themselves, potentially exposing the application to Cross-Site Scripting (XSS) attacks if malicious content is incorrectly rendered. This issue also persists in error response pages like 401, 403, or 500, which often contain sensitive content.

The missing header increases the application's exposure to MIME-based attacks and content spoofing. Although rated as a low-risk vulnerability, it's a best practice to include this header in all HTTP responses for modern web applications. This finding aligns with WSTG-CLNT-05.

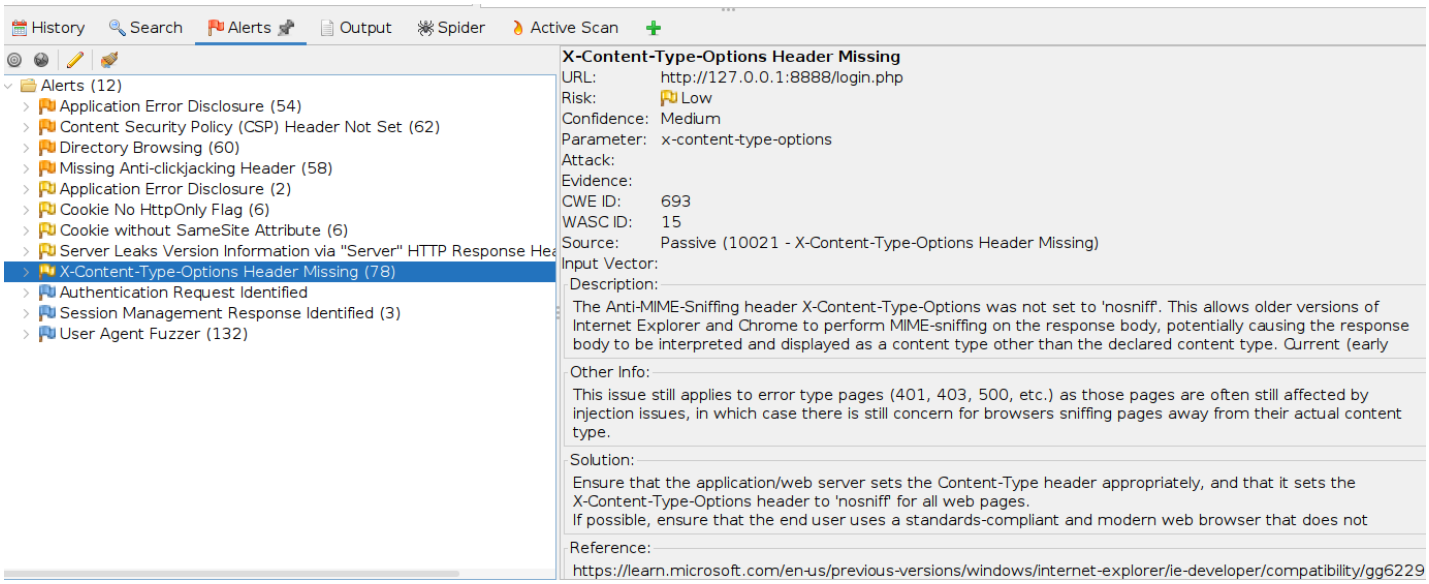

Figure 9: Missing X-Content-Type-Options Header Identified via OWASP ZAP

Vulnerability 10: Authentication Request Identified

OWASP ZAP identified a request containing authentication parameters such as userParam = Login and passwordParam=password in the login form. While this is not a vulnerability by itself, it is a useful discovery because it reveals where and how credentials are being transmitted. This can help in configuring ZAP for further authenticated scanning or for manual testing of login mechanisms.

This type of alert is categorized as informational and is generated through passive scanning. Although no direct risk is associated, it highlights an area that may need additional attention, such as validating secure transmission (e.g., via HTTPS) and ensuring proper authentication protections are implemented.This finding aligns with WSTG-AUTHN-02: Testing for User Enumeration and Guessable User Accounts.

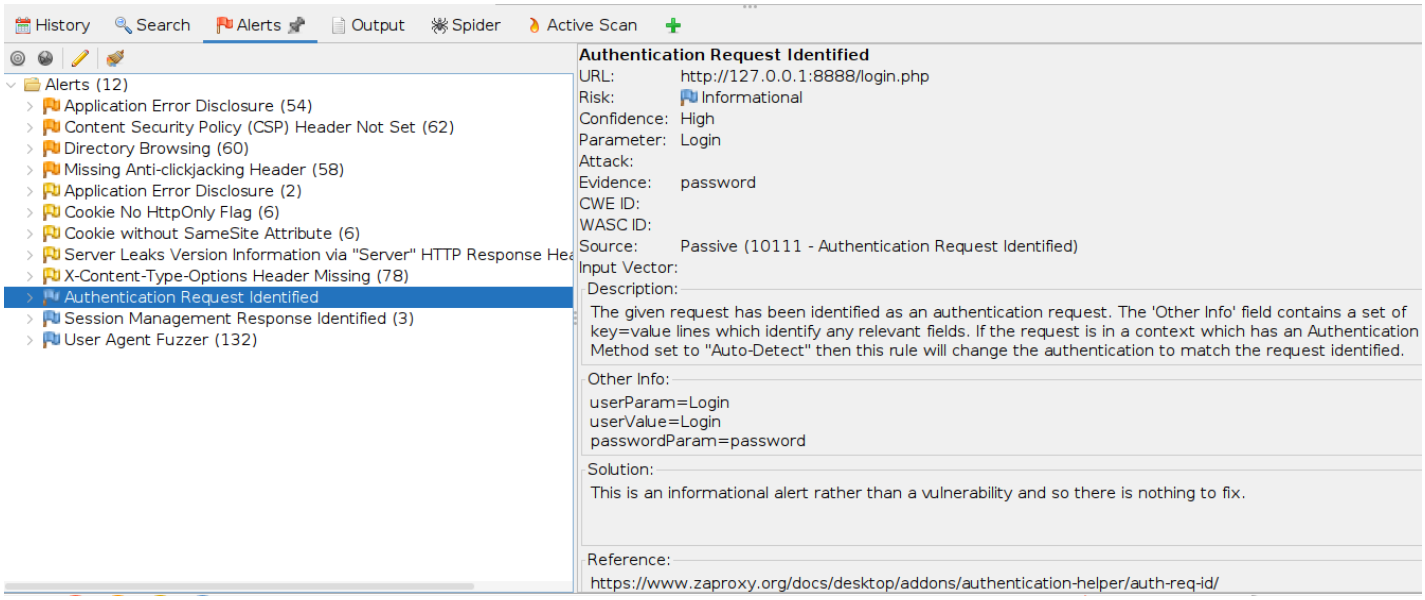

Figure 10: Authentication Parameters Detected via OWASP ZAP

4. Mitigation Strategies

Based on the vulnerabilities identified during the OWASP ZAP scan of DVWA, the following mitigation strategies are recommended to enhance the web application's security posture. These actions are aligned with OWASP best practices and the corresponding sections of the Web Security Testing Guide (WSTG):

1. **Anti-clickjacking Header Missing**
   To prevent clickjacking attacks, it is recommended to implement the X-Frame-Options: DENY header or use a more flexible Content-Security-Policy (CSP) directive like frame-ancestors 'none'. These headers prevent the application from being embedded in iframes, thereby protecting users from UI redress attacks (WSTG-CLNT-09).

2. **HttpOnly Cookie Flag Not Set**
   All session cookies should include the HttpOnly attribute to restrict access from client-side scripts. This reduces the risk of session hijacking through XSS vulnerabilities by preventing JavaScript access to the cookie data (WSTG-SESS-02).

3. **SameSite Cookie Attribute Missing**
   To reduce the risk of cross-site request forgery (CSRF), the SameSite -Strict attribute should be applied to all cookies. This ensures that cookies are not sent with cross-origin requests, enhancing session integrity (WSTG-SESS-05).

4. **Missing Content-Security-Policy Header**
   To mitigate XSS and other content injection attacks, a well-defined Content-Security-Policy should be implemented. The policy should limit sources for scripts, styles, and other potentially dangerous content (WSTG-CLNT-01).

5. **Server Version Information Leakage**
   Suppress or modify the Server HTTP response header to prevent attackers from identifying the server version. This reduces the potential for targeted exploits based on known vulnerabilities in specific software versions (WSTG-INFO-01).

6. **X-Content-Type-Options Header Missing**
   To prevent MIME-sniffing attacks, configure the application to send the X-Content-Type-Options: nosniff header. This ensures that browsers interpret content strictly based on the declared Content-Type header (WSTG-CLNT-05).

7. **Session Token in Response**
   Session tokens should not be included in HTML or URLs. Instead, ensure they are securely transmitted via headers and stored in secure cookies. This reduces the risk of session leakage and fixation attacks (WSTG-SESS-01).

8. **Authentication Fields in Plain HTML Forms**
   Login forms should use secure POST methods and credentials should be encrypted in transit using HTTPS. Furthermore, sensitive fields should be obfuscated or tokenized to avoid exposure during transmission (WSTG-AUTHN-02).

9. **Directory Browsing Enabled**
   Directory listing should be disabled in the web server configuration. Exposing directory contents may lead to information disclosure and give attackers insights into application structure or sensitive files (WSTG-CONF-06).

10. **Error Message Disclosure**
    Detailed system or debug messages should not be displayed to end users. Implement custom error pages and ensure that sensitive information is logged internally while users see generic error messages (WSTG-ERRH-01).

All recommended mitigations follow the OWASP WSTG framework and are supported by additional guidance in the OWASP Cheat Sheet Series, which provides implementation-level advice for securing web applications.

4. Conclusion and Reflection

In conclusion this assessment demonstrated the critical role of web application security testing in identifying and addressing common vulnerabilities. Through the use of OWASP ZAP, a comprehensive scan was performed on the Damn Vulnerable Web Application (DVWA), revealing multiple security issues such as missing headers, session management weaknesses, and unnecessary information disclosures. Each identified vulnerability was analyzed and mapped to the corresponding OWASP Web Security Testing Guide (WSTG) scenario, ensuring that mitigation efforts align with widely accepted best practices.

The hands-on application of ZAP combined with the structured methodology of the WSTG provided a clear understanding of both automated and manual approaches to vulnerability discovery. This experience reinforces the value of proactive security assessments in development workflows, especially in ensuring the implementation of protective headers, secure cookie configurations, and proper error handling.

Reflecting on the assessment, it is evident that even basic misconfigurations in web applications can lead to significant security exposures if left unchecked. By following the WSTG and OWASP recommendations, organizations can significantly improve their security posture, reduce attack surfaces, and build trust in their applications

5. References

1. OWASP Foundation. (2020). *OWASP Web Security Testing Guide* (v4.2). https://owasp.org/www-project-web-security-testing-guide/
2. Mozilla Developer Network. (n.d.). *Content Security Policy (CSP)*. https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP
3. OWASP ZAP Project. (n.d.). *OWASP ZAP User Guide*. https://www.zaproxy.org/docs/
4. OWASP Foundation. (n.d.). *OWASP Cheat Sheet Series*. https://cheatsheetseries.owasp.org/