Alden A. Quiñones

IT Protection and Security: MIT 264

Security Audit of PNexus Web Application and its Deployment Environment.

Table of Contents

[PHASE 2: DATA COLLECTION & ANALYSIS 1](#_Toc195080060)

[**1.** **System/Network Reconnaissance** 1](#_Toc195080061)

[**2.** **USE OF SECURITY TOOLS FOR DATA GATHERING** 2](#_Toc195080062)

[**A.** **APPLICATION-LEVEL VULNERABILITIES** 2](#_Toc195080063)

[**B.** **DEPLOYMENT ENVIRONMENT VULNERABILITIES** 2](#_Toc195080064)

[**3.** **Risk and Vulnerability Analysis** 2](#_Toc195080065)

[**A.** **INFRASTRUCTURE LEVEL ANALYSIS** 2](#_Toc195080066)

[**B.** **APPLICATION-LEVEL ANALYSIS** 3](#_Toc195080067)

[**4.** **Comparison with Industry Security Standards** 6](#_Toc195080068)

[**A.** **INFRASTRUCTURE LEVEL** 6](#_Toc195080069)

[**B.** **APPLICATION LEVEL** 8](#_Toc195080070)

# PHASE 2: DATA COLLECTION & ANALYSIS

1. System/Network Reconnaissance

Given the highly secure network setup in the PNexus Web Application environment, a full network reconnaissance was not applicable. However, a local application-level reconnaissance scan was conducted using ZAP to assess the security posture of the system. The scan targeted the application on http://127.0.0.1, simulating real-world attack scenarios and attack surfaces that could be exposed through the web application. This approach ensured a comprehensive identification of web-based vulnerabilities, despite the limited scope of the assessment at the network level.

The local reconnaissance scan focused on evaluating potential security risks associated with the application infrastructure. While the network itself was secured via the GlobalProtect VPN, which restricts access to authenticated users and adds a layer of security, it was still important to analyze the web application for vulnerabilities that could be exploited even when protected by the VPN.

The PNexus Web Application, developed using CodeIgniter 3 on the back-end, was assessed for potential exposure to SQL injection vulnerabilities, due to the lack of built-in ORM support. Additionally, the front-end components, including jQuery, HTML5, and Bootstrap 4, were evaluated for common vulnerabilities like cross-site scripting (XSS) and cross-site request forgery (CSRF), especially since the application lacks proper configuration to mitigate these risks.

On the security mechanisms such as CAPTCHA and Two-Factor Authentication (2FA) were not implemented, which significantly increases the risk of bot attacks and account compromises. Although the system integrates with Active Directory for user authentication, the integration with the web application was also assessed for potential weaknesses. Given that the application operates over HTTP, an insecure protocol, this raised concerns about the integrity and confidentiality of the data transmitted.

The scan results indicated that while the network setup itself provided strong protection, the application’s vulnerabilities, including lack of encryption, insecure authentication mechanisms, and missing security headers, were clear areas for immediate improvement.

***Note:*** *The scan was performed in a development or staging environment (not production) to evaluate the application's web security posture in isolation from the network-level security mechanisms.*

1. USE OF SECURITY TOOLS FOR DATA GATHERING
2. Application-Level Vulnerabilities

A security assessment was conducted using OWASP ZAP (Zed Attack Proxy) v2.16.1 as the primary tool for data gathering. ZAP was configured to perform both passive and active scans on the application running in an isolated server environment. The scanning process was designed to identify potential vulnerabilities across various endpoints of the application. This comprehensive scan included a detailed analysis of HTTP headers, JavaScript libraries, form handling mechanisms, hidden files, session and cookie management, and response metadata. The scan specifically targeted key application endpoints, such as login forms, APIs, and session management features. Several vulnerabilities were detected during the scan, and they were subsequently categorized based on their severity level.

***Note:*** *The scan focused on publicly exposed endpoints and did not include internal endpoints that are not accessible via the web, which were intentionally excluded from the scope of this assessment.*

1. Deployment Environment Vulnerabilities

Wireshark was used to analyze the network traffic in the deployment environment. It helped identify vulnerabilities such as sensitive data being transmitted without encryption (e.g., passwords and session tokens) that could be intercepted. It also detected the use of outdated protocols like old SSL/TLS versions, which are insecure.

1. **Risk and Vulnerability Analysis**
2. **INFRASTRUCTURE LEVEL ANALYSIS**

The deployment environment of the PNexus Web Application was also assessed to identify potential infrastructure-level risks. This evaluation focused on configuration weaknesses, absence of recommended production-grade practices, and the overall security posture of the application hosting environment.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **#** | **Vulnerability** | **Description** | **Risk Level** | **Impact** | **Recommendation** |
| 1 | Unencrypted HTTP Traffic | **Wireshark** detected sensitive data (like., passwords, session tokens) transmitted over HTTP instead of HTTPS. | High | Sensitive information could be intercepted by attackers, leading to data theft or session hijacking. | Implement HTTPS encryption using SSL/TLS for all sensitive data transmissions. |
| 2 | Outdated SSL/TLS Protocols | Wireshark identified the use of outdated SSL/TLS versions during secure communications, making them vulnerable to attacks. | High | Older SSL/TLS protocols are vulnerable to exploits like **POODLE** or **Man-in-the-Middle** attacks. | Update to the latest version of SSL/TLS and disable outdated protocols. |
| 3 | XAMPP Default Configuration | Traffic analysis revealed that **XAMPP** was running with default settings, exposing services like MySQL and Apache to potential attacks. | Medium | Attackers could exploit default credentials or weak configurations to gain unauthorized access. | Secure **XAMPP** by changing default passwords, disabling unnecessary services, and applying security patches. |
| 4 | Plaintext Session Cookies | Session cookies sent in plaintext were identified by Wireshark, which could be intercepted by attackers. | High | Attackers could hijack sessions and gain unauthorized access to user accounts. | Set the **Secure** and **HttpOnly** flags on cookies, and enforce HTTPS for all session traffic. |

Above table shows a detailed vulnerability assessment table for the Deployment Environment, formatted to be report-ready. Each item includes the vulnerability, description, risk level, impact, and recommendation for remediation.

1. Application-Level Analysis

**Figure 1:** the number of alerts for each level of risk and confidence

|  | | Confidence | | | |
| --- | --- | --- | --- | --- | --- |
| **High** | **Medium** | **Low** | **Total** |
| Risk | **High** | 0 (0.0%) | 1 (6.2%) | 0 (0.0%) | 1 (6.2%) |
| **Medium** | 1 (6.2%) | 2 (12.5%) | 2 (12.5%) | 5 (31.2%) |
| **Low** | 1 (6.2%) | 5 (31.2%) | 1 (6.2%) | 7 (43.8%) |
| **Informational** | 0 (0.0%) | 3 (18.8%) | 0 (0.0%) | 3 (18.8%) |
| **Total** | 2 (12.5%) | 11 (68.8%) | 3 (18.8%) | 16 (100%) |

The table provides a distribution of alerts categorized by Risk and Confidence levels. It shows the number of alerts for each combination of these two categories and represents the proportion of each combination in relation to the total alerts. The data helps to analyze where the system has uncertainty in its risk assessments and how confident it is in identifying those risks.

A key observation is that Low Confidence is prevalent across all risk categories. Of the total 16 alerts, 11 (68.8%) are classified as Low Confidence. This suggests that the system is uncertain about the risk levels of most alerts, particularly for Low Risk and Medium Risk categories. This could indicate that the alerts in these categories are harder to classify with high certainty, which may point to a need for further refinement in the risk-assessment algorithms or additional data for better accuracy.

The High Risk category, with only 1 alert (6.2%) in the entire dataset. Interestingly, there are no High Confidence or User Confirmed High Risk alerts, further highlighting the uncertainty around high-risk alerts. This suggests that either high-risk scenarios are less frequent, or the system struggles to classify high-risk situations with confidence, making them harder to verify.

Both the Medium Risk and Low Risk categories have similar distributions. While the Medium Risk category has a mix of Low Confidence (12.5%) and User Confirmed (12.5%) alerts, the Low Risk category contains a larger portion of Low Confidence alerts (31.2%). Despite this, Low Risk still represents the largest number of alerts in the dataset (7 alerts or 43.8%). This indicates that the system frequently identifies low-risk situations but often with low certainty, potentially due to the nature of the data or the complexity of the risk classification process.

Finally, Informational alerts make up 18.8% of the total, with all three of them categorized under Low Confidence. Informational alerts typically don’t indicate an immediate risk but might provide valuable context or background information. The low confidence in these alerts suggests that they may be harder to categorize or verify, yet they still represent a meaningful portion of the overall alert system.

In summary, the table shows that Low Confidence is a dominant factor across the alert categories, particularly in Low Risk and Medium Risk situations. This shows a potential area for improvement in the alert system’s confidence level. It might be beneficial to further assess and refine the risk classification algorithms to increase the system's certainty in identifying and verifying risks. Additionally, investigating the cause of low-confidence alerts, especially for Low Risk items, could help improve the system’s overall performance.

***Note:*** *The percentages in brackets represent the count as a percentage of the total number of alerts included in the report, rounded to one decimal place.*

**Figure 2:** ZAP reported a total of 16 security alerts, broken down as follows:

The security scan of the PNexus Web Application, conducted using ZAP, identified a total of 16 vulnerabilities, which span various levels of severity and confidence. These vulnerabilities are categorized into high, medium, low, and informational risks, as outlined below:

**Figure 3:** Vulnerabilities classified as high, medium, low, or informational risks.

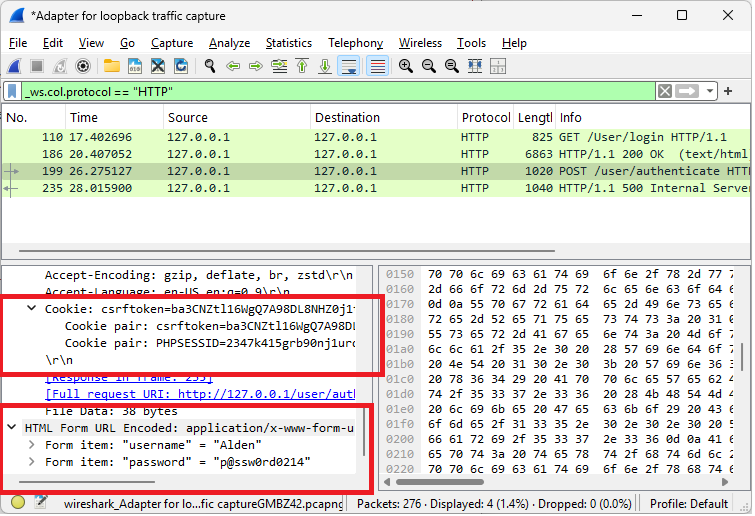
|  |  |  |  |
| --- | --- | --- | --- |
| **Risk Level** | **Count** | **Description** | **Recommendation** |
| **High** | 1 | - Vulnerable JavaScript Library Detected (File: moment.min.js)  - Risk: Contains known security issues that may be exploited by attackers.  - Info: The use of an outdated version of the moment.js library exposes the application to known security vulnerabilities that could potentially be exploited by attackers. | It is recommended to replace this library with the latest, secure version to mitigate the risk. |
| **Medium** | 5 | * **Missing Content Security Policy (CSP) Header**: A missing CSP header can allow attackers to inject malicious content into the application. Implementing a CSP header will mitigate the risk of XSS attacks by restricting the sources from which content can be loaded. * **Missing Anti-clickjacking Header (X-Frame-Options)**: The absence of the X-Frame-Options header leaves the application vulnerable to clickjacking attacks, where an attacker could trick users into clicking on hidden or invisible buttons. Setting this header to "DENY" or "SAMEORIGIN" will prevent such attacks. * **Missing Anti-CSRF Tokens**: The lack of anti-CSRF tokens in the application increases the risk of cross-site request forgery (CSRF) attacks, where an attacker can trick authenticated users into making unwanted requests. Implementing anti-CSRF tokens will protect against these types of attacks. * **Hidden File Detected (.hg)**: The detection of a hidden file (.hg) suggests that version control data might be exposed to unauthorized access. It is essential to remove any unnecessary or sensitive files from the public directory. * **Outdated JavaScript Library (bootstrap.bundle.min.js)**: An outdated version of Bootstrap exposes the application to potential vulnerabilities. Updating to the latest version of the library will ensure that known security issues are addressed | Implement the missing security headers and anti-CSRF tokens. Investigate and address the hidden file and outdated library. |
| **Low** | 7 | * **Information Disclosure via Headers and Timestamps**: The application is disclosing potentially sensitive information, such as server version numbers, in HTTP headers and timestamps. Removing this information will make it harder for attackers to identify the underlying server technology. * **Debug/Error Messages Visible in Responses**: Debugging or error messages exposed in the application responses could provide attackers with information about the system that can be exploited. These messages should be suppressed in the production environment. * **Cross-domain Script Inclusion**: Cross-domain script inclusion vulnerabilities could allow an attacker to inject malicious scripts into the application. Review and secure the implementation to prevent cross-origin resource sharing (CORS) issues. | Review and remove sensitive information in headers and responses. Investigate and address the cross-domain script inclusion. |
| **Informational** | 3 | * **Suspicious Comments Found in Source Code**: The source code contains suspicious comments that could provide useful information to potential attackers. These comments should be removed or sanitized to prevent leakage of sensitive information. * **Modern Web Application Detected**: The application uses modern web technologies, which may require periodic review to ensure they adhere to current security best practices. Although this finding is informational, ongoing assessment of emerging threats is recommended. * **Session Management Behavior Identified**: The application's session management behavior was identified, which could potentially be optimized to ensure secure handling of user sessions. No immediate action is required unless further vulnerabilities in session management are discovered. | Review and remove any sensitive or unnecessary comments. (No specific action needed for the other informational findings). |

1. **Comparison with Industry Security Standards**
2. INFRASTRUCTURE LEVEL

This table shows a comparison of the deployment environment’s vulnerabilities against industry standards, along with recommendations.

|  |  |  |  |
| --- | --- | --- | --- |
| **Industry Standard** | **Relevant Control/Recommendation** | **Vulnerabilities in Deployment Environment** | **Recommendation** |
| NIST Cybersecurity Framework | Secure data in transit and improve network monitoring. | Sensitive data (e.g., passwords) transmitted over unencrypted channels; inadequate monitoring. | Implement TLS/SSL for encryption of sensitive data in transit. Improve monitoring and incident detection capabilities. |
| CIS Controls | Secure configuration for hardware and software. | Default configurations in XAMPP exposing services like MySQL and Apache. | Apply secure configuration practices, disable unnecessary services, and update default credentials. |
| CIS Controls | Vulnerability management. | Outdated software versions running in the environment, leaving systems open to exploitation. | Regularly apply patches and updates to all exposed services to fix known vulnerabilities. |
| ISO/IEC 27001 | Protect sensitive assets and manage access. | Lack of encryption for sensitive data and weak session management. | Implement proper access controls, encrypt sensitive data (e.g., using TLS) and secure session management. |
| ISO/IEC 27001 | Ensure timely detection and response to incidents. | Insufficient network traffic monitoring, delaying detection of suspicious activity. | Enhance network traffic monitoring and incident response capabilities to quickly detect and mitigate threats. |

**Figure 5:** Screenshot showing unencrypted packet of user credential sniffed using wire shark

****

1. **APPLICATION LEVEL**

The identified vulnerabilities were assessed in relation to widely recognized cybersecurity frameworks and standards to determine their severity, prevalence, and recommended remediation strategies. Notably, many findings align with critical areas outlined in the OWASP Top 10 (2021), Common Weakness Enumeration (CWE), and Web Application Security Consortium (WASC) Threat Classification.

One of the most critical findings—the use of vulnerable and outdated JavaScript libraries such as moment.min.js and bootstrap.bundle.min.js—falls under OWASP A06:2021 – Vulnerable and Outdated Components. Using known-vulnerable components in a production environment can lead to severe exploits, including remote code execution or cross-site scripting. This issue also maps to CWE-1104: Use of Unmaintained Third-Party Components.

Another major area of concern is the absence of essential HTTP security headers, such as:

* Content-Security-Policy (CSP)
* X-Frame-Options
* Strict-Transport-Security (HSTS)
* X-Content-Type-Options
* Referrer-Policy

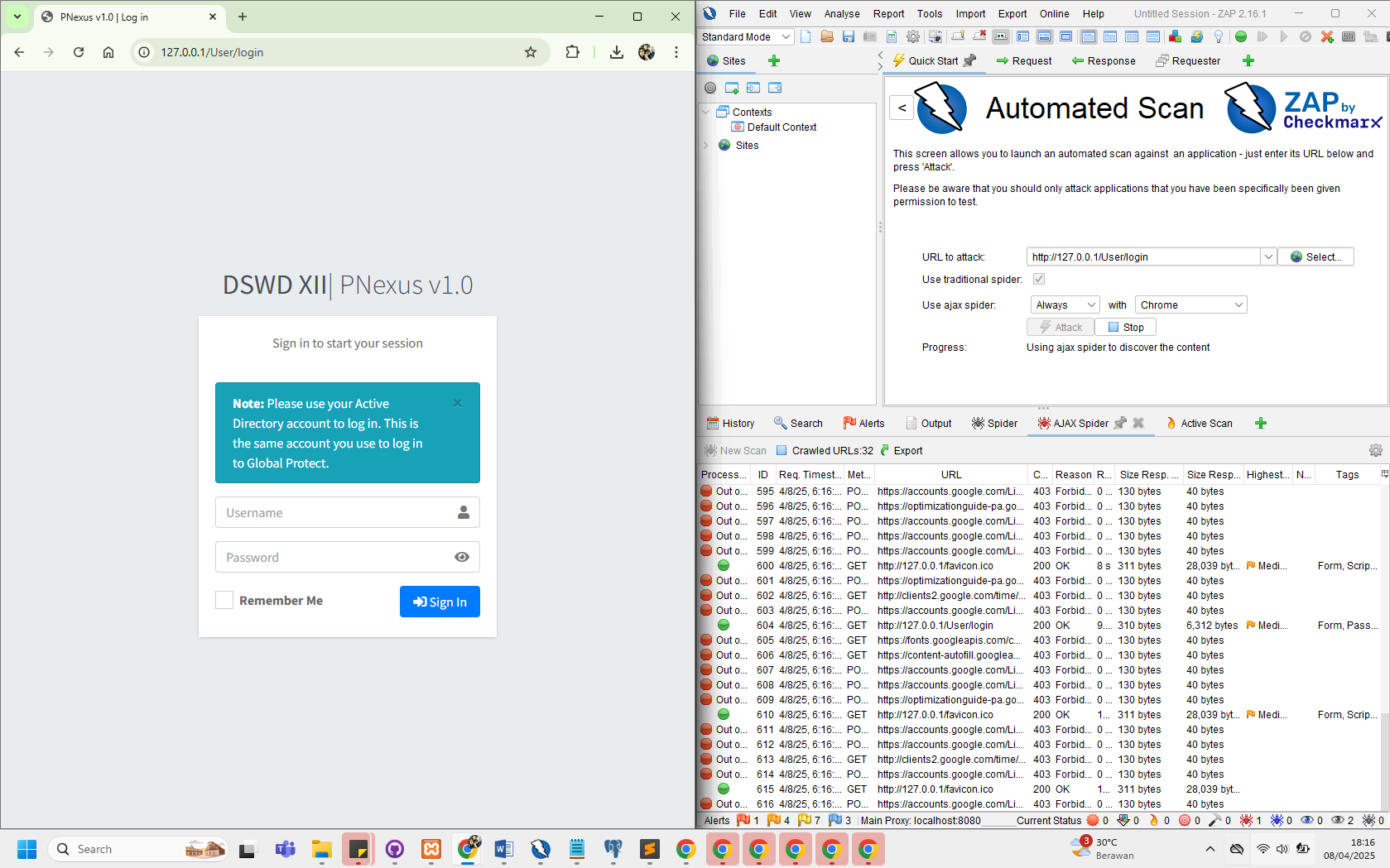
These missing headers expose the application to potential clickjacking, MIME type sniffing, and information leakage. These gaps conflict with recommendations from OWASP’s Secure Headers Project and also align with CWE-693: Protection Mechanism Failure.

The table below shows how the findings relate to industry security standards:

|  |  |  |  |
| --- | --- | --- | --- |
| **Vulnerability** | **Related OWASP Top 10** | **CWE** | **WASC** |
| Vulnerable JS Library | A06: Vulnerable Components | CWE-1104 | WASC-20: Improper Input Handling |
| Missing CSP Header | A05: Security Misconfiguration | CWE-693 | WASC-15: Application Misconfiguration |
| Missing X-Frame-Options | A05: Security Misconfiguration | CWE-1021 | WASC-15 |
| Missing Anti-CSRF Tokens | A01: Broken Access Control | CWE-352: Cross-Site Request Forgery | WASC-9: CSRF |
| Hidden .hg Directory | A05: Security Misconfiguration | CWE-200: Exposure of Sensitive Info | WASC-13: Info Leakage |
| Suspicious Comments/Debug Info | A03: Injection / A05: Misconfiguration | CWE-615: Debug Info Exposure | WASC-13 |

The analysis clearly shows that multiple findings correspond to major categories of application vulnerabilities commonly exploited in real-world attacks. These gaps not only pose technical risks but also reduce compliance with standards such as ISO 27001:2013 (Annex A.14 – System Acquisition, Development, and Maintenance) and NIST SP 800-53 (System and Information Integrity SI-10, SI-2). Aligning the system with industry standards requires addressing these vulnerabilities through component updates, HTTP header configuration, and secure development lifecycle (SDLC) improvements. Regular vulnerability assessments and automated dependency checks should be integrated into the development and deployment pipeline to maintain compliance and reduce risk exposure over time.

**Figure 6:** Screenshot showing automated vulnerability scan using Owasp Zap



**Figure 7:** Screenshot showing high-risk vulnerability assessment result.

