

3.1 Data Analysis

The initial step involved a manual analysis, where notable vulnerabilities in the `ClientRequestPath` and unusual byte sizes in some requests were observed.

- **Tool Development:**

- **Country:** Filtered and analyzed the origin of requests to detect anomalies. No irregularities were found regarding countries.
- **Traffic:** Investigated patterns related to potential denial-of-service attempts. Identified multiple requests with suspicious byte sizes, primarily originating from ASN 396982 (data exported to a CSV file named `suspicious_traffic`).
- **requestpath_vulnerabilities:** Listed vulnerabilities in `ClientRequestPath`, uncovering 2,769 issues.
- **Aggroup:** Consolidated and organized the vulnerabilities for systematic remediation planning.

- **Outcome:** Detailed information and remediation measures were prepared for each identified vulnerability.

Note: All tools were properly documented with inline code comments for clarity and maintainability.

3.2 Risk Identification and Policy Development

Based on the traffic analysis, the following vulnerabilities were identified and mapped to the OWASP Top 10 2021:

- **SQL Injection (A03:2021 – Injection)**

- **Description:** Enables malicious SQL commands due to improper input validation. Can result in data leakage or modification.
- **Evidence:** Columns 22-114 in logs. Detected payloads like `user=admin'--`.
- **Policies:** Use parameterized queries, sanitize inputs, and enforce minimum database permissions.

- **Public Access to WordPress Admin Page (A07:2021 – Identification and Authentication Failures)**

- **Description:** Risk of brute force attacks or use of default credentials.

- **Evidence:** Columns 115-199, showing unauthorized traffic to admin pages.
- **Policies:** Restrict access, enforce account lockouts, and require strong, salted passwords.

• **Expression Language (EL) Injection (A03:2021 – Injection)**

- **Description:** Injection of commands into EL expressions, enabling server code execution.
- **Evidence:** Columns 200-286, revealing malicious payloads targeting EL.
- **Policies:** Disable EL functionality where unnecessary, validate and escape inputs.

• **Local File Inclusion (LFI) (A05:2021 – Security Misconfiguration)**

- **Description:** Exploitation of file paths to access sensitive files.
- **Evidence:** Columns 287-793, showing attempts to access restricted files.
- **Policies:** Validate file paths, use allowlists, and enforce strict file permissions.

• **Credentials in URLs (A02:2021 – Cryptographic Failures)**

- **Description:** Exposure of credentials in URL parameters, risking logs and cache leaks.
- **Evidence:** Columns 794-886, showing credentials in URL strings.
- **Policies:** Use secure headers for credentials and always enforce HTTPS.

• **Access to Suspicious Configuration Files (A05:2021 – Security Misconfiguration)**

- **Description:** Files like `robots.txt` or `phpinfo.php` expose sensitive information.
- **Evidence:** Columns 887-889, identifying traffic to such files.
- **Policies:** Remove unnecessary files and restrict access via server configurations.

• **Remote Code Execution via `shell.php` (A03:2021 – Injection)**

- **Description:** Exploitation of files like `shell.php` for arbitrary server commands.
- **Evidence:** Column 965, revealing the presence of malicious files.
- **Policies:** Validate file uploads and monitor the environment for malicious files.

• **Exposed `.git` Directory (A05:2021 – Security Misconfiguration)**

- **Description:** Access to `.git` directories can reveal source code.
- **Evidence:** Column 1085, showing attempts to access `.git`.
- **Policies:** Block `.git` access and avoid deploying version control files to production.

- **Null Byte (%00) Injection (A03:2021 – Injection)**

- **Description:** Exploits null bytes to bypass validations or execute attacks like LFI.
- **Evidence:** Detected control characters like `%00` in requests.
- **Policies:** Reject null bytes and use secure string manipulation functions.

- **Potential Denial of Service (A09:2021 – Security Logging and Monitoring Failures)**

- **Description:** Abnormal traffic patterns suggesting DoS attempts.
- **Evidence:** Log analysis revealed abusive request rates.
- **Policies:** Implement rate limiting and set up monitoring for unusual traffic.

- **Cross-Site Scripting (XSS) (A03:2021 – Injection)**

- **Description:** Unsanitized inputs allowing malicious script execution in browsers.
- **Evidence:** Columns 2098-2769, showing payloads like `<script>alert('XSS')</script>`.
- **Policies:** Escape outputs, validate inputs, enforce CSPs, and avoid insecure methods.

3.3 Implementation

The code implements the 11 security policies inspired by OWASP 2021 to analyze HTTP traffic logs and identify potential threats like SQL Injection, XSS, and LFI.

- **Key Components:**

- **POLICIES:** Dictionary containing regex patterns, descriptions, and mitigation measures for OWASP vulnerabilities.
- **check_security_policies:** Matches requests to defined policies, tracking anomalies like DoS attempts.
- **analyze_traffic:** Processes logs, associates requests with vulnerabilities, and identifies suspicious IPs.

- **save_suspicious_traffic:** Exports analysis to `suspicious_traffic_log.csv`, detailing threats and recommended actions.

- **Highlights:**

- Each rule maps directly to a specific OWASP vulnerability, ensuring systematic monitoring.
- The output report aids in tracking and remediating vulnerabilities effectively.

This structured implementation demonstrates adherence to OWASP guidelines and provides actionable insights for mitigating security risks.

- **Você disse:**

Não quero que mexa no texto, só arrume a formatação do texto

- **O ChatGPT disse:**

3.1 Data Analysis

The first step was a superficial manual analysis, where I noticed several vulnerabilities in the `ClientRequestPath` and the number of bytes sent in some requests.

I developed a tool named “**Country**” to filter the data and analyze the origin of these requests to check if anything unusual was found; apparently, there were no irregularities with the countries.

Next, I used the tool named “**Traffic**” to investigate patterns of denial-of-service attempts. Here, I identified multiple requests with suspicious byte sizes, especially originating from ASN 396982 (data exported to the CSV file named `suspicious traffic`).

Finally, I used the tool named “**requestpath_vulnerabilities**” to list vulnerabilities found in the `ClientRequestPath`, surprisingly identifying 2,769 issues. Due to the large dataset, I created another tool, “**Aggroup**”, to consolidate and organize these issues. From this point, I started gathering information and planning remediation measures for each identified vulnerability.

Note: All tools are properly documented with comments in the code.

3.2 Risk Identification and Policy Development

Based on the analysis of the provided traffic for testing, the following vulnerabilities were identified and classified according to the OWASP Top 10 2021:

- **SQL Injection (A03:2021 – Injection)**

- **Description:** This vulnerability allows malicious SQL commands to be injected into database queries due to a lack of input validation or sanitization. It can lead to data leakage, modification, or deletion.
- **Evidence:**
 - Columns 22 - 114: The request to the `login.jsp` endpoint demonstrated injection using the payload `user=admin' --`, among other malicious payloads identified during the test.
- **Policy Development:**
 - Use parameterized queries (Prepared Statements) or secure ORMs.
 - Validate and sanitize all user inputs.
 - Enforce minimum permissions in the database to limit potential impacts.
- **Reference:** OWASP A03:2021 – Injection

- **Public Access to WordPress Admin Page (A07:2021 – Identification and Authentication Failures)**

- **Description:** Unauthorized access to the WordPress admin panel exposes the application to risks such as brute force attacks or the use of default credentials.
- **Evidence:**
 - Columns 115 - 199: Detection of traffic directed to the WordPress admin page.
- **Policy Development:**
 - Restrict public access to the admin page.
 - Apply account lockout policies after multiple failed login attempts.
 - Use strong, unique passwords secured with salted hashing (e.g., bcrypt).
- **Reference:** OWASP A07:2021 – Identification and Authentication Failures

- **Expression Language (EL) Injection (A03:2021 – Injection)**

- **Description:** Malicious commands injected into EL expressions can execute arbitrary code on the server.
- **Evidence:**
 - Columns 200 - 286: Requests with payloads related to EL Injection were identified.
- **Policy Development:**
 - Disable or limit EL functionality in applications that do not require it.
 - Escape and validate all user inputs.
- **Reference:** OWASP A03:2021 – Injection

- **Local File Inclusion (LFI) (A05:2021 – Security Misconfiguration)**

- **Description:** Local File Inclusion allows access to sensitive files on the server.
- **Evidence:**
 - Columns 287 - 793: Requests exploiting paths to access restricted files.
- **Policy Development:**
 - Validate and normalize file paths.
 - Use allowlists to restrict accessible files.
 - Configure appropriate file permissions on the server.
- **Reference:** OWASP A05:2021 – Security Misconfiguration

• **Credentials in URLs (A02:2021 – Cryptographic Failures)**

- **Description:** Sending credentials through URLs can expose sensitive information in logs and caches.
- **Evidence:**
 - Columns 794 - 886: Requests captured contained credentials in URL parameters.
- **Policy Development:**
 - Transmit credentials only in request bodies or secure headers (e.g., Authorization Header).
 - Always use HTTPS to protect data in transit.
- **Reference:** OWASP A02:2021 – Cryptographic Failures

• **Access to Suspicious Configuration Files (A05:2021 – Security Misconfiguration)**

- **Description:** Files such as `robots.txt` or `phpinfo.php` expose sensitive application information.
- **Evidence:**
 - Columns 887 - 889: Requests for vulnerable files were identified.
- **Policy Development:**
 - Remove unnecessary files from the production environment.
 - Restrict access to critical files via server configurations.
- **Reference:** OWASP A05:2021 – Security Misconfiguration

• **Remote Code Execution via `shell.php` (A03:2021 – Injection)**

- **Description:** Files like `shell.php` allow remote execution of commands on the server.
- **Evidence:**
 - Column 965: Identification of a malicious file enabling arbitrary execution.

- **Policy Development:**
 - Remove detected malicious files.
 - Validate and sanitize inputs, especially during file uploads.
 - Continuously monitor the environment for signs of compromise.
- **Reference:** OWASP A03:2021 – Injection

- **Exposed .git Directory (A05:2021 – Security Misconfiguration)**

- **Description:** Access to the .git directory can expose sensitive source code.
- **Evidence:**
 - Column 1085: Requests to access the .git directory were detected.
- **Policy Development:**
 - Configure the server to block access to the .git directory.
 - Avoid deploying version control files in production environments.
- **Reference:** OWASP A05:2021 – Security Misconfiguration

- **Null Byte (%00) Injection (A03:2021 – Injection)**

- **Description:** Exploiting null bytes (%00) can bypass validations or execute attacks like LFI or SQL Injection.
- **Evidence:**
 - Identification of payloads containing null bytes in requests.
- **Policy Development:**
 - Reject control characters, including null bytes, in user inputs.
 - Use secure functions for string manipulation.
- **Reference:** OWASP A03:2021 – Injection

- **Potential Denial of Service (A09:2021 – Security Logging and Monitoring Failures)**

- **Description:** Anomalous traffic behavior may indicate exploitation for DoS attacks.
- **Evidence:**
 - Traffic patterns indicated DoS attack attempts.
- **Policy Development:**
 - Implement rate limiting to restrict abusive requests.
 - Set up monitoring systems to detect anomalous traffic.
- **Reference:** OWASP A09:2021 – Security Logging and Monitoring Failures

- **Cross-Site Scripting (XSS) (A03:2021 – Injection)**

- **Description:** XSS occurs when unsanitized inputs allow malicious scripts to be injected and executed in victims' browsers.

- **Evidence:**
 - Columns 2098 - 2769: Requests contained payloads like `<script>alert('XSS')</script>`.
- **Policy Development:**
 - Escape outputs for special characters (`<`, `>`, `'`, `"`).
 - Validate and sanitize user inputs.
 - Implement Content Security Policies (CSP).
 - Avoid using insecure methods like `innerHTML` or `eval`.
- **Reference:** OWASP A03:2021 – Injection

3.3 Implementation

The code implements the 11 security policies inspired by OWASP 2021 vulnerabilities. It analyzes HTTP traffic logs to detect malicious patterns, vulnerabilities, and potential attacks like SQL Injection, XSS, and LFI.

Key Components:

- **POLICIES:** A dictionary defining the 11 security policies, with regex patterns, descriptions, and actions to mitigate threats.
- **check_security_policies:** Validates requests against the policies, detecting malicious patterns using regex and tracking anomalies like excessive requests (DoS).
- **analyze_traffic:** Processes logs, matches requests to the 11 policies, tracks IP activity, and identifies suspicious entries linked to vulnerabilities.
- **save_suspicious_traffic:** Exports results to a CSV file, detailing IP addresses, detected policy violations, and recommended actions.
- **main:** Orchestrates the workflow: loads traffic logs, performs analysis, and exports findings.

The program outputs a report (`suspicious_traffic_log.csv`) summarizing detected threats, their causes, and actions to prevent exploitation. This implementation provides a practical application of OWASP 2021 guidelines for monitoring and mitigating security threats.

3.5 Innovation

For innovation, I decided to create a graphical interface using HTML, CSS, PHP, and JS, which presents the spreadsheet in a more 'visual' way to make it easier to see which attacks are happening more frequently. This would help in taking more effective measures, focusing defense on the most common attacks. I also thought about adapting the tool from 3.3 Implementation to act on real-time network traffic, blocking these types of threats more simply, and even creating an artificial intelligence to learn about these attack patterns and block them automatically, but this things will take more time to do, so i make something more simple.

The tool currently works as follows:

1. **CSV File Reading:** The tool reads a CSV file containing network traffic logs, identifying different types of vulnerabilities (such as SQL Injection, XSS, and others) in the data.
2. **Vulnerability Processing and Counting:** PHP processes the CSV file, counting the frequency of each vulnerability type found. These counts are stored in a format that can be returned to the frontend.
3. **Graphical Interface:** The graphical interface is built using HTML, CSS, PHP, and JavaScript. It displays the vulnerabilities in a bar chart, making it easier to visualize the frequency of each attack type. The total number of vulnerabilities is also shown on the interface.
4. **Data Display:** Using JavaScript (with the Chart.js library), the tool generates interactive charts that present the count of each vulnerability type clearly and visually.