**User Manual: Automated Web Vulnerability Scanner**

**Version 1.0**  
**(ToniaVulnScanner - As per your User-Agent)**  
**May 2025**

**Table of Contents**

1. Introduction  
   1.1 Purpose of the Scanner  
   1.2 System Requirements  
   1.3 Key Features
2. Getting Started  
   2.1 Installation (Conceptual - as your setup is local)  
   2.2 Launching the Application
3. User Interface Overview
4. Performing a Scan  
   4.1 Initiating a New Scan  
   4.2 Understanding Scan Parameters
5. Viewing Scan Results  
   5.1 Accessing Results  
   5.2 Interpreting Results  
   5.3 Data Gathering Results  
   5.4 Vulnerability Detection Results  
   5.4.1 SQL Injection Potential  
   5.4.2 Cross-Site Scripting (XSS) Potential  
   5.4.3 HTTP Security Headers  
   5.4.4 Open Ports
6. Database Management (Conceptual)
7. Troubleshooting
8. Best Practices & Limitations
9. Contact Information

**1. Introduction**

**1.1 Purpose of the Scanner**  
This Automated Web Vulnerability Scanner is designed to help website developers and security professionals identify common security flaws in web applications. It automates the process of information gathering, port scanning, and detection of prevalent vulnerabilities such as SQL Injection (SQLi), Cross-Site Scripting (XSS), insecure HTTP security headers, and open ports. The goal is to provide accurate and in-depth analysis to enhance web security.

**1.2 System Requirements (Based on your dissertation)**

* **Operating System:** Windows 10 Pro (or similar capable of running Python)
* **RAM:** 16GB (Recommended for smooth operation)
* **Processor:** Intel Core i7 (or equivalent)
* **Software:**
  + Python (version used in development, e.g., 3.7+)
  + Web Browser (e.g., Chrome, Firefox, Edge) for accessing the user interface.
  + Required Python packages (Flask, Flask-CORS, Requests, BeautifulSoup4, etc. - assumed to be bundled or installed via a requirements.txt if distributed).

**1.3 Key Features**

* Automated scanning of web applications.
* **Data Gathering:** Collects information about target assets, including subdomains and IP addresses.
* **Port Scanning:** Identifies open ports and associated services on the target.
* **Vulnerability Detection:**
  + SQL Injection (SQLi) potential.
  + Cross-Site Scripting (XSS) potential (reflected XSS).
  + Insecure HTTP Security Header configurations.
* Web-based user interface for easy operation.
* SQLite database for storing scan results and project information.

**2. Getting Started**

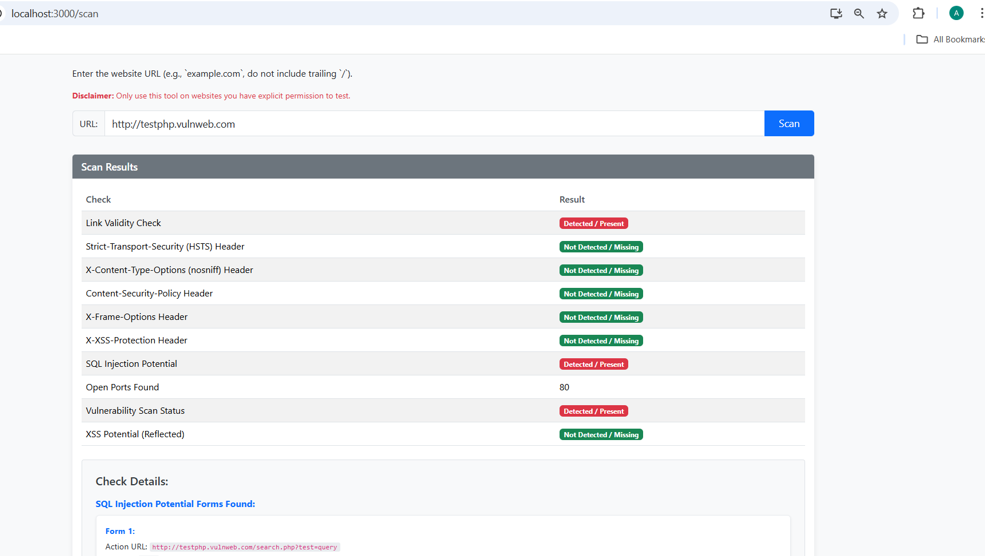
**2.1 Installation (Conceptual for current prototype)**  
*Currently, the scanner is run from its development environment.*

1. Ensure Python and pip are installed.
2. Clone the project repository: git clone https://github.com/anthoniat/tfortscan.git
3. Navigate to the project directory: cd Tonia-vuln-scanner
4. Install required Python packages: pip install -r requirements.txt (assuming a requirements.txt file exists)
5. The frontend (React) would typically require a separate build step if not served by Flask. (e.g., npm install && npm start in the frontend directory).

**2.2 Launching the Application**

1. **Start the Backend (Flask Server):**
   * Navigate to the backend directory.
   * Run the main application file (e.g., python app.py ).
   * By default, the server might start on http://localhost:5000.
2. **Start the Frontend (React Development Server - if separate):**
   * Navigate to the frontend directory.
   * Run npm start (or similar command). This usually opens the application in your default web browser, often on http://localhost:3000.
3. Access the scanner's user interface by opening your web browser and navigating to the appropriate URL (e.g., http://localhost:3000 or http://localhost:5000 if Flask serves the frontend).

**3. User Interface Overview**

**The main interface typically consists of:

* **Input Area/Scan Configuration:**
  + **Target URL Field:** Where you enter the URL of the web application to scan (e.g., your Figure 4.15).
  + **Scan Button:** To initiate the scanning process.
* **Results Display Area:**
  + Where scan progress and findings are displayed. This might be divided into sections for different types of results.
* **Navigation (if any):** Links to previous scan results, settings, or help.
* **Information Panel:** May display general information, logs, or vulnerability details.

**4. Performing a Scan**

**4.1 Initiating a New Scan**

1. Navigate to the main page of the scanner in your web browser.
2. In the **Target URL field**, enter the full URL of the web application you wish to scan (e.g., http://testphp.vulnweb.com).
   * Ensure the URL is accessible from the machine running the scanner.
3. Click the **"Scan"** button

**4.2 Understanding Scan Parameters (Current Scope)**

* **Target URL:** The primary input.
* **Scan Scope (Implicit):** The scanner will attempt to gather information related to the provided domain and then test the discovered assets and the primary URL for vulnerabilities.
* **Vulnerabilities Checked (Default):**
  + SQL Injection potential
  + Reflected XSS potential
  + Common HTTP Security Headers
  + Common Open Ports (e.g., 80, 443, 21, 22, etc. - as defined in your port\_scanner.py)

**5. Viewing Scan Results**

**5.1 Accessing Results**

* Once a scan is initiated, results will typically be displayed dynamically on the interface.
* For completed scans, there might be a history or project section to retrieve past results (conceptual, as your current dissertation focuses on single scan runs for evaluation).

**5.2 Interpreting Results**  
Results are typically categorized. Pay attention to severity indicators if provided (though your current focus is detection).

**5.3 Data Gathering Results (Based on Figure 4.16, prior to detailed vulnerability list)**

* **Target Information:** The initial URL provided.
* **IP Address:** Resolved IP address of the target domain.
* **Discovered Assets/Subdomains:** (If implemented beyond the IP for the main domain).

**5.4 Vulnerability Detection Results (Based on Figure 4.16 and subsequent details)**

* **Example Entry Format:**
  + **Vulnerability Type:** (e.g., SQL Injection Potential, XSS Potential, Missing Security Header, Open Port)
  + **Location/Details:** Specific URL, parameter, header name, or port number.
  + **Status/Finding:** (e.g., "Potential SQLi forms found", "XSS payload reflected", "X-Frame-Options: Missing", "Port 80: Open")

**5.4.1 SQL Injection Potential**

* + **Indicates:** The scanner found HTML forms or input parameters that might be susceptible to SQL Injection.
  + **Details Provided (as in Figure 4.17):**
    - Action URL of the form (e.g., http://testphp.vulnweb.com/search.php?test=query)
    - Form method (GET/POST)
    - Input field names within the form.
  + **Action Recommended:** These are *potential* findings. Manual verification or tools like SQLMap are recommended to confirm exploitability (as demonstrated in your research).

**5.4.2 Cross-Site Scripting (XSS) Potential**

* + **Indicates:** The scanner injected a test payload into URL parameters, and this payload was reflected in the server's response.
  + **Details Provided:**
    - Vulnerable parameter.
    - Indication of reflection (e.g., "Payload reflected in response text," "Payload found within <script> tags").
  + **Action Recommended:** Investigate the identified parameter and sanitize user input. Test manually in a browser to confirm.

**5.4.3 HTTP Security Headers**

* + **Indicates:** Checks for the presence and configuration of important security headers.
  + **Details Provided:**
    - Header Name (e.g., X-Frame-Options, Content-Security-Policy, Strict-Transport-Security, X-XSS-Protection, X-Content-Type-Options).
    - Status (e.g., "Present," "Missing," "Misconfigured").
  + **Action Recommended:** Implement or correctly configure missing/misconfigured headers according to security best practices.

**5.4.4 Open Ports**

* + **Indicates:** Ports that are open and listening for connections on the target IP address.
  + **Details Provided:** Port number and sometimes the service commonly associated with it (if service fingerprinting is basic).
  + **Action Recommended:** Review open ports. Ensure only necessary ports are open to the internet. Unnecessary open ports increase the attack surface.

**6. Database Management (Conceptual)**

* The scanner uses an SQLite database to store information for each "project" or scan.
* This includes:
  + Asset collection details (target title, IP, banner).
  + Open port numbers and services.
  + Vulnerability detection results (targets, scripts used, vulnerability types).
* *Currently, direct user interaction with the database management module via the UI is assumed to be for developers or future enhancements. Scan results are primarily viewed through the main results interface.*

**7. Troubleshooting**

* **Scanner Not Starting:**
  + Ensure Python and all dependencies (Flask, etc.) are correctly installed.
  + Check that no other application is using the same port (e.g., 5000 or 3000).
* **"Could not resolve domain: [domain]" Error:**
  + Verify the target URL is correct and the domain exists.
  + Check your machine's internet connectivity and DNS settings.
* **Scan Takes Too Long / No Results:**
  + The target website might be very large, complex, or slow to respond.
  + Check for network connectivity issues between your machine and the target.
  + The default timeout for requests is 10 seconds; slow servers might exceed this for some requests.
* **False Positives/Negatives:**
  + Automated scanners are not infallible. Always manually verify critical findings.
  + This scanner focuses on specific patterns; sophisticated WAFs or unusual application behavior might affect detection.

**8. Best Practices & Limitations**

* **Authorization:** Only scan web applications for which you have explicit, written permission from the owner. Unauthorized scanning is illegal and unethical.
* **Scope:** Be mindful of the potential impact of scanning on production systems. Prefer scanning in staging or test environments.
* **Verification:** Always manually verify findings, especially those identified as "potential."
* **Regular Updates:** (Conceptual for this prototype) Security threats and techniques evolve. Keep any scanning tools and their vulnerability definitions updated.
* **Limitations of this Scanner (v1.0):**
  + Primarily detects a specific set of common vulnerabilities (SQLi, Reflected XSS, Headers, Open Ports).
  + May not detect all variants of these vulnerabilities or more complex/advanced attack vectors (e.g., Stored XSS, Second-Order SQLi, DOM XSS, CSRF are not explicitly covered in the described implementation).
  + Relies on patterns and may be evaded by sophisticated defenses or obfuscation.
  + Limited JavaScript rendering capabilities (if any), which can affect XSS detection in modern JS-heavy applications.

**9. Contact Information**

For support or inquiries regarding this scanner (within the context of the MSc project):

* Anthonia
* [t033188n@student.staffs.ac.uk](https://www.google.com/url?sa=E&q=mailto%3At033188n%40student.staffs.ac.uk)