**Automated Vulnerability Scanning and Reporting Tool**

**1. Project Overview:**

This project is about creating a tool that helps check websites for security problems automatically. Instead of doing manual testing (which takes a lot of time and effort), this tool uses a popular security scanner called **OWASP ZAP** and the **Python** programming language to do the work. It also has a simple **web interface** made using **Flask**, where users can enter a website URL and get a **PDF report** showing any vulnerabilities found.

The main goal is to make security testing easy for students, developers, and anyone who wants to keep their website safe without needing deep cybersecurity knowledge.

**2.** **Problem Statement:**

Today, websites are often attacked by hackers who take advantage of security issues like:

* SQL Injection (getting into databases),
* XSS (injecting harmful scripts), and
* CSRF (unauthorized actions on behalf of a user).

Finding and fixing these issues is important, but manual testing is difficult and time-consuming. Not all developers are security experts, and many don’t have the time or tools to perform proper testing.

This project solves this problem by creating a tool that:

* Automatically scans websites,
* Uses a strong security scanner (OWASP ZAP), and
* Provides results in a simple report format.

**3. Implementation Details:**

This project includes four main parts:

**1. zap\_scanner.py**

This part connects to the OWASP ZAP tool and tells it to scan a website. It waits until the scan is complete and collects all the security warnings (alerts).

**2. generate\_report.py**

This part takes the scan results and turns them into a neat PDF report using a Python library called ReportLab. The report includes details like the type of issue, how risky it is, and what can be done to fix it.

**3. app.py**

This is the **Flask web app**. It shows a simple form where the user can:

* Enter one website URL, or
* Upload a text file with many URLs.  
  When the form is submitted, it starts the scan process and generates the report.

**4. run\_tool.py**

This file puts everything together and runs the complete process – scanning and reporting.

**4.** **Challenges & Solutions:**

**Challenge 1: Connecting with OWASP ZAP tool**

It was tricky to send commands to ZAP using code.  
 *Solution*: Used the official **ZAP Python API** to send instructions like "start scan," "check progress," and "get alerts."

**Challenge 2: Creating easy-to-read reports**

Just printing results in a file looked messy.  
 *Solution*: Used **ReportLab** to generate clean, well-organized PDF reports.

**Challenge 3: Scanning multiple websites**

Scanning one site was easy, but handling many websites at once needed careful handling.  
 *Solution*: Allowed users to upload a **.txt file with multiple URLs** and ran scans for each.

**Challenge 4: User-friendly interface**

Not all users are comfortable with command-line tools.  
 *Solution*: Built a **simple Flask web page** so users can just enter a URL and click a button.

5. **Conclusion and Future Enhancements:**

**Conclusion:**

This project successfully provides a **quick and simple way** to find security issues in websites. By combining OWASP ZAP with Python and Flask, we built a tool that:

* Saves time,
* Is easy to use,
* And creates helpful reports.

It is useful for students, developers, and small companies who want to test their websites for security without hiring experts.

**Future Enhancements:**

In the future, the tool can be improved by:

* Adding **login system** so that each user has their own account.
* Including more types of checks like **SSL, headers, and cookie security**.
* Letting users **schedule scans** and receive reports through email.
* Giving options to download reports in **CSV or HTML formats**.
* Making it easy to run the tool on any system by using **Docker**.