Documentation WebGuard Framework

Documentation for WebGuard Framework **1**

1. Introduction

With the rapid growth of web applications in sectors such as banking, healthcare, and e-commerce, ensuring security is critical. Web applications are prime targets for cyberattacks, including SQL injection, cross-site scripting (XSS), and authentication bypasses. Current tools often lack the flexibility needed for customized testing. This project aims to design a flexible penetration testing framework capable of automating common security tests and providing developers with comprehensive analyses of vulnerabilities.

2. Problem Statement

Web applications are vulnerable to cyberattacks due to inadequate security measures and coding flaws. Existing penetration testing tools may not cover all aspects of web security or may be challenging to customize for specific needs. This project introduces a tailored framework that integrates various security testing techniques, allowing flexible execution to improve vulnerability detection and remediation.

3. Objectives

1. Develop a customizable framework for web application security testing.
2. Integrate tools and techniques to detect vulnerabilities such as SQL injection, XSS, and header security misconfigurations.
3. Provide detailed, automated reports on detected vulnerabilities.
4. Ensure the framework's modularity for future scalability and additional feature integration.

4. Scope of the Project

This project focuses on building a framework that can:

1. Crawl web applications to identify potential test parameters.
2. Detect SQL injection and XSS vulnerabilities using integrated tools such as sqlmap and OWASP ZAP.
3. Perform header security checks.
4. Generate automated reports with detailed findings and recommended solutions.
5. Integrate with other third-party security tools for enhanced scanning capabilities.

5. System Architecture

The system architecture consists of the following components:

1. **Web Crawler Module**: Collects URLs and input fields for potential testing.
2. **Vulnerability Scanning Module**: Detects SQL injection and XSS vulnerabilities using tools like sqlmap and ZAP.
3. **Security Header Checker**: Checks HTTP headers for secure attributes (e.g., Secure, HttpOnly).
4. **Reporting Module**: Generates a comprehensive report with all findings, solutions, and references.
5. **Controller Module**: The main orchestrator that coordinates the scanning processes and gathers results.

6. Module Descriptions

6.1 Web Crawler Module

- **Purpose**: Identifies endpoints and input fields within the target web application.
- **Implementation**: Uses a combination of Python libraries and tools to extract URLs and parameters.
- **Features**:
    - Parameter extraction
    - URL collection

6.2 Vulnerability Scanning Module

- **SQL Injection Detection:**
    1. Utilizes sqlmap for detecting SQL injection vulnerabilities.
    2. Execution: Runs sqlmap commands and parses results.

- **XSS Detection:**
    1. Uses OWASP ZAP for active XSS scanning.
    2. Integrates ZAP's spider and active scanning capabilities.

6.3 Security Header Checker

- **Purpose**: Checks HTTP headers for security configurations (e.g., Content-Security-Policy, X-Frame-Options).
- **Output**: Provides a detailed list of headers and whether essential security features are enabled.

6.4 Reporting Module

- **Purpose**: Compiles results into a structured report with solutions and references.
- **Features**:
    1. PDF generation with vulnerability details.
    2. Inclusion of solutions and references for remediation.

7. Implementation Details

- **Programming Language**: Python
- **Key Libraries/Tools**:
    1. sqlmap for SQL injection detection.
    2. OWASP ZAP API for automated scanning.
    3. requests and subprocess for module integration.
    4. pdfkit for report generation.
- **Configuration**:
    1. YAML configuration files for user-defined settings.
    2. Integration with APIs (e.g., ZAP API) for enhanced functionality.

8. Testing and Results

Testing Scenarios:

- Test on vulnerable web applications such as bWAPP.
- Simulated attacks for SQL injection and XSS.
- Header security checks on various test sites.

Results:

- Successful detection of common vulnerabilities.
- Detailed reporting with step-by-step remediation suggestions.

9. Conclusion and Future Work

Conclusion: The penetration testing framework effectively detects and reports vulnerabilities in web applications. Its modular structure ensures that it can be easily expanded to include additional tests and capabilities.

Future Work:

- Add modules for more vulnerability types (e.g., CSRF, authentication flaws).
- Integrate machine learning to improve scan accuracy.
- Enhance user interface for better usability.

10. References

- OWASP ZAP documentation
- sqlmap project repository
- Python documentation for subprocess, requests, and pdfkit
- Security best practices articles and guides