



SOFTWARE DESIGN SPECIFICATION

CYBERSECURITY  
  
Web Services Open to Malicious Attacks

|  |  |  |  |
| --- | --- | --- | --- |
| **Created By:** | M.D.Sandhiya | **Approved By:** | Harshada Topale. |
| **Created On:** | 6-04-2025 | **Approved On:** | DD-MMM-YYYY |

Page left blank intentionally

**INDEX**

[**1** **PURPOSE** 2](#_Toc142418236)

[**2** **PROJECT SCOPE** 2](#_Toc142418237)

[**3** **SYSTEM OVERVIEW** 2](#_Toc142418238)

[**4** **DESIGN CONSIDERATIONS** 2](#_Toc142418239)

[4.1 Requirements 3](#_Toc142418240)

[4.2 Assumptions 3](#_Toc142418241)

[4.3 Dependencies 3](#_Toc142418242)

[**5** **SYSTEM ARCHITECTURE** 3](#_Toc142418243)

[5.1 Architectural Strategies 4](#_Toc142418244)

[5.2 Structure & Relationships 4](#_Toc142418245)

[**6** **DETAILED DESCRIPTION OF COMPONENTS** 4](#_Toc142418246)

[**7** **INTEGRATION** 5](#_Toc142418247)

[**8** **APPENDICES** 1](#_Toc142418248)

[8.1 Appendix A – Detailed Description of Components 1](#_Toc142418249)

**General Instructions for using the Live Project POC Document**

* This template and the subsequent document created using this template is a confidential document and is the intellectual property of Cloud Counselage Pvt. Ltd. Circulating it outside of the organisation without the consent of Cloud Counselage Pvt. Ltd. is the breach of company policies and will lead to legal actions
* The Design Specification of a software forms the basis of development of software
* The **text between inequality (< >) is to be replaced** by relevant text
* Please **remove the yellow highlight on the Text** between the inequality (< >). This is done to help you notice the text to be changed/replaced
* The text in *italics* highlighted in grey is just for reference and should be removed after adding the relevant text

# **PURPOSE**

This document is created based on the requirement specification document. The purpose of this Software Design Specification (SDS) Document is to break down the project into components to describe in detail what the purpose of each component is and how it will be implemented. The SDS will also serve as a tool for verification and validation of the final product.

# **PROJECT SCOPE**

The scope of the **Web Services Open to Malicious Attack** project includes its distinct features, its benefits, and it limitations.The system's distinct features allow it to **identify vulnerabilities in web services and demonstrate data hiding techniques** by using **Python scripts, ApacheBench for load testing, and the Pillow library for image-based steganography**.  
The system enables the user to **analyze the resilience of a web application against DoS attacks and securely embed confidential information within images to protect sensitive data from unauthorized access**.

# **SYSTEM OVERVIEW**

This section will provide an outline of the various components and subsystems of “Web services Open to Malicious Attack”

A diagram of a model

Description automatically generated

# **DESIGN CONSIDERATIONS**

This section describes requirements, assumptions and dependencies to be addressed to devise a complete design solution.

## Requirements

The system requirements for the Web Services open to Malicious Attack projectt include:

\* A plubicly accessible website for testing.

\* Tools for security auditing .

\* Segnography funcionality using Python and PIL library

\* ApacheBench for DoS simulation testing.

\*Secure environment to run DoS tests witthout harming real services.

\* Basic reporting templates documenting vulnerabilities and results.

\* Linux or Windows operating system with administrative access.

Python environment set up with necessary modules.

## Assumptions

The design was based on the following assumptions:  
  
 \* The website provided is permitted for ethical security testing.  
 \* All tests, especially DoS, are conducted in a controlled, responsible manner.  
 \* The target server can handle basic load testing without operational impact.  
 \* Users running steganography or load tests have basic knowledge of Python and Linux commands.  
 \* Necessary permissions and approvals have been obtained before starting any tests.  
 \* There will be no legal consequences arising from penetration testing activities on approved sites.

## Dependencies

The success of the project relies on the following dependencies:  
  
 \*Availability of the website throughout the testing phase.  
 \*Proper installation and functioning of tools like OWASP ZAP, ApacheBench, Python, and PIL.  
 \*Internet connectivity for accessing web services.  
 \*No significant changes to the server configuration during testing.  
 \*Operating system compatibility with the tools used (Linux preferred for better tool support).  
 \*Timely feedback from mentors or supervisors regarding project deliverables.

# **SYSTEM ARCHITECTURE**

The software system architecture for the project Web Services Open to Malicious Attack refers to the logical organization of different software components used to audit, analyze, and simulate attacks on a web service. It defines how these components interact with each other, ensuring the system achieves its intended functionality.  
An effective architecture has been designed to enable agility, save time and cost, and identify design risks early. It serves as a conceptual blueprint for development and project management throughout the project's lifecycle.   
  
The software architecture for this project:   
  
Defines the structure of the security auditing and testing system.   
  
Defines the behavior of vulnerability scanning, load testing, and steganography processes.  
  
Defines component relationships, such as between the auditing tool, the testing tool, and the reporting mechanism.   
  
Defines communication structure between user inputs, analysis tools, and outputs.  
  
Balances stakeholder needs (educational purpose, ethical testing).   
  
Influences team structure by organizing activities (testing, reporting) into clear responsibilities.  
  
Focuses on significant elements like security checks, DoS resistance, and data hiding.  
  
Captures early design decisions related to technology stack and workflow.   
  
Important Architecture Characteristics Considered:   
Operational Architecture Characteristics:   
Availability: Tools and systems selected are open-source and available for local and offline usage if needed.   
  
Performance: Tools like ApacheBench simulate high request loads to measure website response performance.   
  
Reliability: Processes ensure reliable extraction of vulnerabilities and performance metrics.  
  
Low fault tolerance: The system ensures that any interruption during load testing does not cause a full crash.   
  
Scalability: The approach allows scaling up (e.g., larger request batches in load tests) based on server capacity.   
  
Structural Architecture Characteristics:   
Configurability: Scanning and load-testing parameters (like request number) are easily configurable.  
  
Extensibility: The system can be extended with more testing tools (e.g., Burp Suite, Nikto) if needed.   
  
Supportability: Python scripts and auditing tools used are widely documented and community-supported.  
  
Portability: All components can work across different operating systems (Linux, Windows).  
  
Maintainability: Scripts and configurations are simple, modular, and easy to update or debug.  
  
Cross-Cutting Architecture Characteristics:   
Accessibility: The tools and code are accessible to the development team without cost or licensing issues.   
  
Security: Testing is conducted ethically and only on permitted websites, ensuring secure operation.  
  
Usability: Tools have user-friendly command-line or GUI interfaces (e.g., OWASP ZAP).  
  
Privacy: Testing respects the privacy of the system being tested, without unnecessary data extraction.   
  
Feasibility: The design and selected components ensure the project remains achievable within available time and resources.

## Architectural Strategies

The major components of the system architecture for the project Web Services Open to Malicious Attack are:  
  
1. Vulnerability Assessment Component  
Performs security audits on the target website.  
Uses tools like OWASP ZAP and manual analysis to identify common vulnerabilities (e.g., SQL injection, XSS).  
Focuses on authentication policies, server configuration issues, and outdated technologies.  
  
2. Performance and Load Testing Component  
Simulates heavy traffic to test the server’s resistance to Denial of Service (DoS) attacks.  
Uses ApacheBench (ab) to send multiple concurrent requests and analyze server response times and behavior under load.  
Identifies performance bottlenecks and server overload risks.  
  
3. Steganography Module  
Implements basic steganography techniques to hide sensitive information in images.  
Demonstrates how malicious actors could use steganography for covert data transmission through websites.  
  
4. Audit and Reporting Component  
Consolidates findings from vulnerability scans and load testing.  
Generates detailed audit reports including vulnerabilities discovered, performance results, and recommendations for improvement.  
Supports documentation for stakeholders to understand risks and fixes.  
  
5. User Interface / Interaction Layer  
Provides a basic command-line interface or interaction layer.  
Allows users (testers) to configure the tests, run audits, simulate attacks, and review output.  
Future scope: Could be extended to a web-based dashboard.  
  
6. Security and Ethical Control Layer  
Ensures that all actions are performed within ethical guidelines.  
Implements safeguards to prevent accidental harm to unintended systems.  
Focuses only on permitted or dummy websites for ethical hacking practice.

## Structure & Relationships

<Create a flowchart to show the structure and relationships between the components listed above >

A diagram of a software system

Description automatically generated

# **DETAILED DESCRIPTION OF COMPONENTS**

For detailed description of the components, please refer **Appendix A – Detailed Description of Components**

The below template will be used to specify the details of all the components

**Table 1: Detailed Design Specification Template**

|  |  |
| --- | --- |
| **Identification** | The unique name for the component and the location of the component in the system. |
| **Type** | A module, a subprogram, a form, a data file, a control procedure, a class, etc. |
| **Purpose** | Function and performance requirements implemented by the design component, including derived requirements. Derived requirements are not explicitly stated in the SRS - but are implied or adjunct to formally stated SDS requirements. |
| **Subordinates** | The internal structure of the component, the constituents of the component, and the functional requirements satisfied by each part. |
| **Dependencies** | How the component’s function and performance relate to other components. How this component is used by other components. The other components that use this component. Interaction details such as timing, interaction conditions (such as order of execution and data sharing), and responsibility for creation, duplication, use, storage, and elimination of components. |
| **Interfaces** | Detailed description of all external or internal interfaces as well as of any mechanism for communicating through messages, parameters, or common data areas. All error messages and error codes should be identified. All screen formats, interactive messages, and other user interface components (originally defined in the SRS) should be given here. |
| **Resources** | A complete description of all resources (hardware or software) external to the component but required to carry out its functions. |
| **Processing** | A full description of the functions presented in the Function subsection. Pseudocode can be used to document algorithms, equations, and logic. |
| **Data** | For the data internal to the component, describes the representation method, initial values, use, semantics, and format. |

# **INTEGRATIONS**

The "Web Services Open to Malicious Attack" project integrates with several tools and platforms to perform vulnerability assessments, load testing, and reporting. Integration details are as follows:   
  
ApacheBench (ab) is used to perform load testing on the web server. It communicates directly with the server endpoints to simulate multiple concurrent requests and measure server performance.   
  
OWASP ZAP (Zed Attack Proxy) is integrated as the primary tool for vulnerability scanning. It acts as a proxy server to intercept and scan web traffic for security flaws like SQL injection, cross-site scripting, and insecure authentication mechanisms.  
  
Nikto is another tool integrated for web server vulnerability scanning. It helps in identifying outdated server software, dangerous files, and configuration issues by directly connecting to the web service.   
  
Burp Suite Community Edition is used for manual testing and verification of vulnerabilities. It communicates with the website and enables in-depth testing of input validation, session management, and access controls.   
  
Linux Terminal/Command Line Tools are used for executing penetration testing commands, load tests, and scanning activities. Tools like nmap are integrated for network and port scanning.   
  
Google Chrome Developer Tools are used for inspecting network activities and identifying potential client-side security risks through manual testing.   
  
Reporting Tools like Microsoft Word and Excel are integrated manually for preparing audit reports, traceability matrices, and documentation of the security analysis findings.

# **APPENDICES**

## Appendix A – Detailed Description of Components

|  |  |
| --- | --- |
| Identification | Vulnerability Scanner |
| Type | Tool/Module |
| Purpose | Performs automated vulnerability scanning on the targer web applicationto identify common security issues like XSS,SQL injection and weak authentication. |
| Subordinates | * OWASP ZAP * Nikto |
| Dependencies | Requires the web application to be live and accessible during testing |
| Interfaces | Command-line interface for nikto, GUI and proxy settings for OWASP ZAP. |
| Resources | OASP ZAP tool installed  Nikto installed on Linux system |
| Processing | Scan request are sent to the server;responses are analyzed for vulnerabilities. |
| Data | Scan results including identified vulnerabilities, response code , and endpoint data. |

|  |  |
| --- | --- |
| **Identification** | Reporting System |
| **Type** | Document Preparation |
| **Purpose** | To compile test results, nobservations, screenshorts , and recommendation into an organized report for stakeholders |
| **Subordinates** | MS word  MS Excel |
| **Dependencies** | Completion of vulnerability scans and load tests before starting documentation. |
| **Interfaces** | Document files and spreadsheets created manually. |
| **Resources** | Microsoft Office Suite  Scan output files |
| **Processing** | Summarization ofvulnerabilities, severity levles, performance metrics and proposed security improvements. |
| **Data** | Vulnerability scan results  Load test results  Screenshots  Recommendations |