**<!clientName!>**

**<!clientName!>** **SAST report**

<!title!>

Scan date: <!finishDate!>

**Table of Contents**

[1. Executive Summary 3](#_Toc155193501)

[1.1 Introduction 3](#_Toc155193502)

[1.2 Engagement Overview 3](#_Toc155193503)

[1.3 Limitations 3](#_Toc155193504)

[1.4 Summary of findings 4](#_Toc155193505)

[2. Application Assessment Details 5](#_Toc155193506)

[2.1 Identified Code Vulnerabilities 5](#_Toc155193507)

[2.2 Assessment of Vulnerable Library Versions 6](#_Toc155193508)

[3. Remediation Strategies 7](#_Toc155193509)

[3.1 General Remediation Guidelines 7](#_Toc155193510)

[3.2 Specific Recommendations for Code Vulnerabilities 7](#_Toc155193511)

[3.3 Remediation Plan for Vulnerable Libraries 7](#_Toc155193512)

# Executive Summary

## Introduction

The testing team conducted a comprehensive code analysis of <!clientName!>'s <!title!>This analysis was focused on identifying potential security vulnerabilities and weaknesses within the source code. The objective of this SAST report is to detail the findings from the code examination, emphasizing the identification of coding flaws, security gaps, and areas of non-compliance with best coding practices.

This report synthesizes the findings from the comprehensive review performed by the SAST team. The focus of the analysis was on identifying potential security weaknesses inherent within the code itself. The vulnerabilities highlighted in this report were uncovered through a detailed inspection of the source code, taking into account the specific security risks and implications associated with each identified issue. The recommendations in this report are aimed at guiding the remediation efforts to bolster the security and integrity of the application, ensuring robust protection against potential cybersecurity threats.

## Engagement Overview

The objective of the Static Application Security Testing (SAST) for <!clientName!>'s <!title!> application was to proactively identify and address potential security vulnerabilities within the source code. This SAST engagement was structured to ensure a thorough analysis of the application's codebase, with a focus on detecting weaknesses that could be exploited by potential attackers.

The following objectives for the engagement were co-developed with {company name}:

* **In-Depth Code Security Review:** Conduct a thorough examination of the application's source code to identify security vulnerabilities, coding flaws, or non-compliance with secure coding practices, with a particular focus on detecting vulnerabilities susceptible to common security threats such as improper data handling, insecure storage practices, and potential backdoors.
* **Evaluation against Industry Best Practices:** Assess the application's source code in relation to industry-standard security practices and guidelines, particularly those recommended by OWASP, to determine the application's alignment with best security practices.
* **Formulation of Remediation and Improvement Strategies:** Based on the analysis, provide comprehensive recommendations for the remediation of identified vulnerabilities and advice for enhancing the overall security posture of the application at the code level.

<!#isLimitation!>

## Limitations

We encountered the following limitations during this assessment: :<!#limitations!>

* <!desc!><!/limitations!>

<!/isLimitation!>

## Summary of findings

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Observations | | | | |  | |
|  | **Critical** | **High** | **Medium** | **Low** | **Informational** | | **Total** |
| Application | <!criticalCount!> | <!highCount!> | <!mediumCount!> | <!lowCount!> | <!infoCount!> | | <!totalCount!> |
| Total | <!criticalCount!> | <!highCount!> | <!mediumCount!> | <!lowCount!> | <!infoCount!> | | <!infoCount!> |

|  |  |
| --- | --- |
|  |  |

# Application Assessment Details

## Identified Code Vulnerabilities

This section details specific vulnerabilities identified within the application's source code. It encompasses a range of issues, from critical security flaws to minor coding inefficiencies. Each vulnerability is described in terms of its nature, potential impact, and the context within which it was found. This part of the report not only highlights the severity of each issue but also provides insights into how these vulnerabilities might be exploited by malicious entities<!#isCritical!> <!#criticals!>

|  |  |  |
| --- | --- | --- |
| **Critical** | |  | | --- | | <!issueType!> | |
| **Code Example** | <!code!> |
| **Details** | <!details!> |
| **Instances** | |  |  | | --- | --- | | File Location | Line | | <!#instances!><!location!> | <!line!><!/instances!> | |

<!/criticals!> <!/isCritical!> <!#isHigh!> <!#highs!>

|  |  |  |
| --- | --- | --- |
| **High** | |  | | --- | | <!issueType!> | |
| **Code Example** | <!code!> |
| **Details** | <!details!> |
| **Instances** | |  |  | | --- | --- | | File Location | Line | | <!#instances!><!location!> | <!line!><!/instances!> | |

<!/highs!> <!/isHigh!> <!#isMedium!> <!#mediums!>

|  |  |  |
| --- | --- | --- |
| **Medium** | |  | | --- | | <!issueType!> | |
| **Code Example** | <!code!> |
| **Details** | <!details!> |
| **Instances** | |  |  | | --- | --- | | File Location | Line | | <!#instances!><!location!> | <!line!><!/instances!> | |

<!/mediums!> <!/isMedium!>

## Identified Vulnerable Library Versions

This section focuses on the vulnerabilities associated with third-party libraries and dependencies used within the application. It lists outdated or insecure versions of libraries that are known to have security flaws. For each library, the report includes details such as the current version in use, the latest secure version available, and the specific vulnerabilities associated with the version in use. This section is crucial for maintaining the overall security of the application, as vulnerabilities in third-party components can be critical entry points for attackers.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module Name | Current version | Latest secure version | Unsupported/Outdated | CVE Identifier |
| <!#libs!><!name!> | <!version!> | <!fixedIn!> | <!version!> | <!notes!><!/libs!> |

# Remediation Strategies

## General Remediation Guidelines

This section provides general guidelines for remediation aligned with OWASP best practices. These guidelines are intended to enhance the security posture of the application by addressing the common types of vulnerabilities typically identified in SAST assessments:

**Secure Coding Practices:** Adhere to secure coding standards as recommended by OWASP, focusing on preventing common vulnerabilities like code injections and cross-site scripting. Regularly update coding guidelines to reflect the latest security advancements and threats.

**Input Validation and Sanitization:** Implement stringent input validation and sanitization to prevent injection attacks. All user-supplied data should be validated against a defined set of rules for type, length, format, and range.

**Consistent Error Handling:** Develop a uniform error handling strategy that avoids exposing sensitive information. Ensure that error messages are generic and logs do not contain confidential data.

**Authentication and Authorization:** Strengthen authentication mechanisms using multi-factor authentication where feasible. Implement comprehensive authorization checks throughout the application to control access to different user levels.

**Encryption and Secure Data Practices:** Utilize strong encryption for data in transit and at rest. Adopt secure data handling practices to protect sensitive information from unauthorized access and leaks.

**Regular Security Reviews and Dependency Management:** Conduct regular security reviews of the codebase and maintain up-to-date third-party libraries and dependencies. Use dependency management tools to monitor and manage external code used in the application.

**Security Education and Awareness:** Engage in continuous security training for the development team. Stay updated with the latest security threats and encourage a security-first mindset in the development process.

**References:**

<https://owasp.org/www-project-top-ten/>

<https://cheatsheetseries.owasp.org/index.html>

<https://owasp.org/www-project-developer-guide/draft/>

## Specific Recommendations for Code Vulnerabilities

This section outlines the remediation plan for the vulnerabilities identified in the test, as detailed in Section 2.1: Identified Code Vulnerabilities. Addressing these vulnerabilities is crucial to maintain the security and integrity of the application.

### Hard-coded Passwords

* Avoid hard-coding passwords directly in the code. Instead, use secure credential management solutions such as environment variables, configuration files, or dedicated credential vaults.
* Store sensitive information in a centralized and secure location, separate from your source code.
* Utilize environment variables to store sensitive information, including passwords. This helps keep sensitive data outside of the source code and allows for easier management in different environments.

### SQL injection

Recommendations for this finding.

## Recommendations for Vulnerable Libraries

This section outlines the remediation plan for the vulnerabilities identified in the test, as detailed in Section 2.2: Identified Vulnerable Library Versions. Addressing these vulnerabilities is crucial to maintain the security and integrity of the application.

**Update to Latest Secure Versions:** Updating vulnerable libraries to their latest secure versions is a critical step in mitigating security risks. This process involves identifying the libraries flagged as vulnerable in the assessment and upgrading them to the newest versions that have patched known vulnerabilities. The process should be prioritized based on the severity of the vulnerabilities, with high-risk libraries being updated first. Post-update, it's essential to conduct thorough testing to ensure that the updates do not disrupt the application's functionality or introduce new dependencies.

**Implement Dependency Management Tools:** Introducing or enhancing dependency management and monitoring tools is a proactive approach to maintaining the security of third-party libraries. These tools automate the tracking of library versions, alerting developers to outdated or vulnerable components. By integrating these tools into the development workflow, teams can continuously monitor for new vulnerabilities and updates, ensuring that the application remains protected against emerging threats. These tools also aid in documenting dependencies, making future audits and updates more efficient.