****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/16/24** | **Joel** | **Security implementation** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Joel Hays

## Algorithm Cipher

SHA-256

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A computer screen with a black border

Description automatically generated

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

A screenshot of a computer

Description automatically generated

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A screenshot of a computer

Description automatically generated

A screenshot of a computer error

Description automatically generated

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

A computer screen shot of a hand pointing

Description automatically generated

## Summary

## The main focus of this project was systematic refactoring to remove identified security vulnerabilities. Steps and best practices as defined by industry standards governed the process in the vulnerability assessment. Cryptographic strength stood out as one of the key areas of concern, for which SHA-256 was used as the cryptographic hash function to ensure data integrity and security. Through collision resistance from SHA-256, secure information could be attained. I also implemented communication security by refactoring the application to communicate over HTTPS instead of the default HTTP. I did that by creating a self-signed certificate through Java Keytool and configuring the application to use it in communication. With HTTPS, any piece of data sent between the client and server is safe because it is encrypted, which ensures that only the communicating parties can read or tamper with the data being transmitted. A self-signed certificate was properly generated using Java Keytool and embedded in the Spring Boot application, with secure and encrypted communication.

## Dependency management also utilized the OWASP Dependency-Check tool for a dependency check. All vulnerabilities found in third-party libraries had to be suppressed based on careful analysis in order not to let them impact the security posture of the application, and custom suppression rules were implemented whenever required to keep a good analysis of security, free of false positives. Among other steps taken to increase layers of security into the software application was enhancing encryption using SHA-256 for data integrity verification. This made sure that the data is hashed to a unique checksum, which can be verified to detect unauthorized changes.

## This helped configure SSL/TLS properly, to use the certificate, which will be self-signed and generated within this project, for the encryption of all data in transit. Vulnerabilities were managed by tools such as OWASP Dependency-Check, proactively checking and managing vulnerabilities in third-party libraries. It comprised validating each identified vulnerability for applicability to the application, suppressing irrelevant ones, and planning to address the rest. Lastly, through the right setup of the application properties, it became possible to enforce all parameters related to secure communication that significantly reduced the risk of misconfiguration.

## Industry Standard Best Practices

By adapting industry-accepted best practices, such as using strong cryptographic algorithms and conducting robust dependency checks, the project mitigated the risk of security breaches and ensured compliance with modern regulatory requirements. These measures not only enhance the security posture of the application but also build trust with stakeholders and provide a sustainable foundation against emerging threats.