****

# 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** | **10/19/2025** | **Everett De Bree** | **Finalized Report** |

## 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

Everett De Bree

## Algorithm Cipher

The selection of SHA-256 for Artemis checksum purposes was based on its ability to offer both effective collision protection and its compatibility with various platforms and libraries. SHA-256 continues to serve as a recommended integrity verification tool because it remains free from the documented collision attacks which affect MD5 and SHA-1. A hash is a one-way function (no key), so it’s ideal for verifying that data hasn’t changed. When a secret must be protected (confidentiality) rather than just verified, encryption such as AES (symmetric) is appropriate; and when identity/non-repudiation is needed, asymmetric cryptography (e.g., RSA, ECDSA) with certificates is used. The SHA-256 algorithm provides suitable security protection through simple deployment for this project's checksum functionality.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

AI-generated content may be incorrect.

## Secure Communications

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

A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

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

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect.

## Functional Testing

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

A computer screen shot of a program

AI-generated content may be incorrect.

## Summary

In this project I implemented several security enhancements to the existing Java Spring Boot application. I started by implementing SHA-256 checksum functionality which produces secure hash values for verifying data integrity. Next, I created a self-signed certificate using the Java Keytool in Eclipse and configured the application to serve requests over HTTPS (8443). I verified the /hash route operated through HTTPS before running OWASP Dependency-Check to find new vulnerabilities and I confirmed the build process finished correctly. The application ran without errors during functional testing and the checksum endpoint produced the correct output when accessed through HTTPS.

The process follows the secure-software development life cycle through its design and implementation stages and verification phases. The application becomes more secure through each defensive layer which includes data integrity encryption and SSL/TLS transport security and dependency analysis for vulnerability protection and manual functional testing. The system now follows industry-standard best practices for confidentiality and integrity and availability after the refactoring process.

## Industry Standard Best Practices

The project implementation followed multiple industry-approved security standards which supported both OWASP Top Ten and secure software development principles. The system protects data through SHA-256 and TLS 1.2 cryptographic algorithms which enable secure data transmission and maintain data integrity. The application configuration followed the principle of least privilege by limiting access to only necessary resources and avoiding the exposure of sensitive information. The OWASP Dependency-Check tool scanned all dependencies while the suppression file contained known false positives to ensure transparency and auditability. The system demonstrated its security and reliability through static and functional testing which confirmed its adherence to current security standards and safeguarded both present and future system trustworthiness and integrity.