****

# 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** | **6.20.25** | **Charity Deel** |  |

## 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

Charity Deel

## Algorithm Cipher

For this project, I selected the SHA-256 encryption algorithm due to its proven strength, industry-wide adoption, and compatibility with modern software security practices. SHA-256 belongs to the SHA-2 family and produces a 256-bit fixed-length hash. This output is irreversible and ensures data integrity without exposing the original content. It operates on 512-bit blocks and is built with robust avalanche properties, where even the smallest change in the input produces a drastically different hash. This is critical for financial systems that require reliable verification. Although SHA-256 itself does not use keys, it can be paired with keyed mechanisms such as HMAC to enable symmetric-style authentication. Unlike symmetric or asymmetric encryption, hashing is a one-way process, meaning it is useful for validating that data has not changed. Random numbers play a role when generating salts or in other cryptographic processes layered around hash functions to prevent predictable outputs. Historically, SHA-2 was developed in response to the increasing vulnerabilities in SHA-1. Today, SHA-256 is widely used in secure APIs, cryptocurrency, and financial systems, making it a natural fit for Artemis Financial's upgrade requirements.

## Certificate Generation

A screenshot of a computer

AI-generated content may be incorrect.

## Deploy Cipher

A screenshot of a computer

AI-generated content may be incorrect.

## Secure Communications

A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

## Functional Testing

A screenshot of a computer

AI-generated content may be incorrect.

## Summary

In this project, I reviewed Artemis Financial’s existing software and identified security gaps related to data integrity and encrypted communication. To address these issues, I refactored the application to include SHA-256 checksum generation using Java’s built-in MessageDigest class. This ensures that any transmitted input can be verified for integrity using a secure, one-way hash function. I also implemented secure communication protocols by generating a self-signed certificate and enabling HTTPS via Spring Boot’s SSL configuration. This protects sensitive data in transit and prevents man-in-the-middle attacks. The code was tested for correct output, error-free execution, and dependency integrity. A manual dependency check confirmed the application uses secure and up-to-date libraries. These enhancements follow the flow of identifying vulnerabilities, applying secure refactors, and verifying effectiveness through controlled testing.

## Industry Standard Best Practices

Throughout the project, I followed industry standard best practices to ensure Artemis Financial’s application meets modern software security expectations. This included using SHA-256, a cryptographic hash algorithm recommended by NIST, to implement data verification via checksums. The hashing was done using Java’s MessageDigest API, ensuring compatibility with secure, native JDK functions. For secure communication, I used HTTPS by enabling SSL/TLS via a self-signed certificate. Although not signed by a certificate authority, this still demonstrates the encrypted handshake and the use of public/private key infrastructure, both of which are critical in real-world deployments. I also followed Spring Boot’s recommended configuration practices for integrating security features and manually verified all project dependencies. This helps avoid known third-party vulnerabilities, a practice aligned with OWASP dependency management recommendations. By applying these standards, the application now prevents plaintext data transmission, verifies data integrity using strong cryptographic methods, and avoids introducing third-party risks. These practices not only secure the application technically but also support Artemis Financial’s business need to protect sensitive client data.