****

# 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** | **8/17/2025** | **Thomas Sweet** |  |

## 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

Thomas Sweet

## Algorithm Cipher

For this project, I selected the SHA-256 cryptographic hash algorithm for checksum verification. SHA-256 is a secure hashing algorithm from the SHA-2 family, providing 256-bit encryption. It is considered highly secure for most applications and resistant to known attacks such as collision or preimage attacks.

SHA-256 uses a fixed-size 256-bit (32-byte) hash. It is deterministic and irreversible, making it ideal for verifying data integrity. It operates on blocks of data and produces unique hash values for different inputs, reducing the risk of collisions.

This cipher is symmetric in nature for hashing use cases but does not require a key, as it is not an encryption algorithm. Instead, it creates a fingerprint of the data. The use of secure random number generation, though not needed in the checksum step, is important in key-based systems such as AES, RSA, and others. The SHA-256 algorithm is widely used in digital certificates, secure applications, and blockchain technology, proving its robustness and reliability.

## 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.

[Insert screenshots here.]

## Secure Communications

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

A screen shot 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 program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

[Insert screenshots here.]

## Summary

To meet Artemis Financial’s security needs, I refactored the existing code to include modern encryption practices and secure communication protocols. I implemented SHA-256 hashing to verify data integrity and used Java’s Keytool to generate a self-signed certificate, enabling HTTPS communication.

Additionally, I reconfigured the Maven pom.xml file to integrate OWASP's dependency-check plugin. I created a suppression.xml file to handle false positives related to known but unresolvable vulnerabilities, ensuring that future developers won't be distracted by unnecessary warnings.

The system was retested after implementing these changes, and no new vulnerabilities were introduced. The application now runs securely, with encrypted traffic and checksum validation functioning correctly.

## Industry Standard Best Practices

Throughout this project, I followed industry best practices for secure software development. By using SHA-256 and enabling HTTPS, I helped protect client data in transit and verified file integrity. Running a static code analysis with the OWASP dependency-check ensured third-party library risks were minimized.

Suppressing known false positives in the suppression.xml file also demonstrates good DevSecOps hygiene, acknowledging risks while ensuring focus remains on actionable vulnerabilities. These practices align with NIST and OWASP guidelines, helping ensure Artemis Financial's system adheres to modern security expectations and builds stakeholder trust.