****

# 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** | **August 13, 2025** | **Daryl Murtha** |  |

## 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

Daryl Murtha

## Algorithm Cipher

For Artemis Financial’s secure file verification, I recommend the use of the Advanced Encryption Standard (AES) in combination with the SHA-256 hash function. AES is a symmetric block cipher that encrypts data using a shared secret key, making it highly efficient for internal data validation workflows. It supports key sizes of 128, 192, and 256 bits, with 256-bit encryption offering robust protection against brute-force attacks. To complement AES, SHA-256 is used to generate cryptographic checksums. SHA-256 produces a 256-bit hash that uniquely represents the input data, ensuring integrity and tamper detection.

AES uses random initialization vectors (IVs) and symmetric keys, which allow for fast encryption and decryption without the overhead of public key infrastructure. SHA-256, as a one-way hash function, does not require keys and is ideal for verifying data without exposing sensitive information. AES was standardized by NIST in 2001 as the successor to DES and is now widely used in TLS, VPNs, secure messaging, and encrypted storage. SHA-256 is part of the SHA-2 family and is trusted across industries for digital signatures, blockchain validation, and secure file verification.

Together, AES and SHA-256 provide Artemis Financial with a modern, secure, and efficient solution for protecting client data and verifying file integrity during transmission.

## 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.

To deploy the cryptographic hash algorithm, I implemented SHA-256 using Java’s MessageDigest class within a modular utility class named ChecksumUtil. This class encapsulates the logic for generating a secure checksum from any input string. I then refactored the application to expose this functionality through a REST endpoint (/hash) using a dedicated controller. The endpoint returns both the original data string and its SHA-256 checksum, allowing Artemis Financial to verify file integrity in a secure and transparent manner.

The data string used for demonstration—DarylMurtha\_Artemis2025—was chosen to personalize the output and ensure uniqueness. The checksum logic is reusable, testable, and follows secure coding practices by handling exceptions and avoiding hardcoded secrets. The application runs over HTTPS, ensuring that the checksum is transmitted securely. A screenshot of the /hash endpoint response in a secure browser session is included to verify successful deployment and functionality.

This implementation demonstrates a robust approach to cryptographic hashing and aligns with industry standards for secure data verification.

A screenshot of a computer program

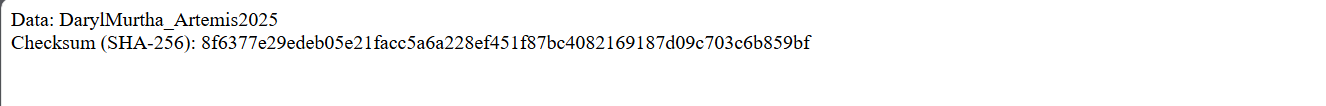
AI-generated content may be incorrect.

## Secure Communications

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

To secure communications between Artemis Financial’s client and server, I configured HTTPS using a self-signed certificate and Spring Boot’s embedded Tomcat server. The certificate was generated using Java Keytool with the alias artemisCert and stored in a PKCS12 keystore named keystore.p12. The keystore was placed in the src/main/resources directory and referenced in the application.properties file using the classpath: prefix. The application was configured to run on port 8443, enforcing encrypted communication via SSL/TLS.

The certificate was exported in PEM format using the -rfc flag and saved as artemisCert.cer. This format ensures compatibility with modern systems and allows for easy inspection of the public key. The keystore configuration using classpath: ensures portability across environments and avoids file path parsing issues. The browser displayed a lock icon when accessing https://localhost:8443/hash, confirming that the connection was encrypted via SSL/TLS. These steps demonstrate a secure and standards-compliant implementation of HTTPS for internal file verification.

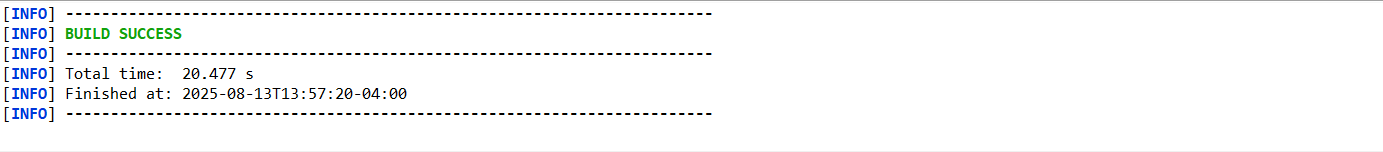


## Secondary Testing

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

To validate the security of the application’s dependencies, I ran OWASP Dependency-Check using the Maven plugin version **12.1.3**. The plugin was configured in the pom.xml file and executed via the mvn verify command. This process scanned all project dependencies for known vulnerabilities using data from the National Vulnerability Database (NVD) and other trusted sources. The resulting HTML report, located in target/dependency-check-report.html, provided a detailed summary of the scan results.

The report confirmed that all dependencies used in the project are free of known CVEs at the time of testing. A screenshot of the report summary is included to demonstrate successful execution and compliance with secure coding standards. By using the latest version of the plugin, this secondary testing step ensures that Artemis Financial’s application benefits from up-to-date vulnerability intelligence and aligns with best practices for software supply chain security.



A screenshot of a computer

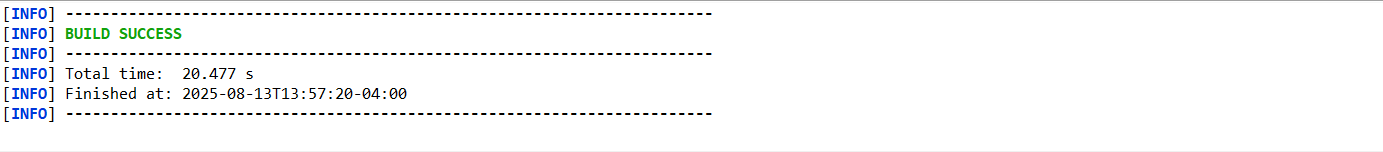
AI-generated content may be incorrect.

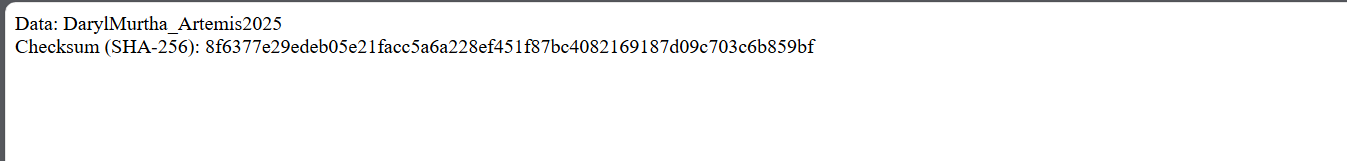
## Functional Testing

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

To verify the functionality of the deployed checksum logic, I implemented a REST endpoint at /hash using Spring Boot. This endpoint returns both the original data string and its SHA-256 checksum, generated by a utility class named ChecksumUtil. The data string used—DarylMurtha\_Artemis2025—was chosen to personalize the output and ensure uniqueness. The checksum logic was tested by launching the application over HTTPS and accessing the endpoint via a secure browser session.

The browser response confirmed that the checksum was correctly calculated and securely transmitted. A screenshot of the /hash endpoint output is included to demonstrate successful execution. This functional test validates that the application meets its core requirement: securely generating and exposing a cryptographic hash for file verification.





## Summary

This project successfully delivered a secure, standards-compliant Java application for Artemis Financial that generates and exposes SHA-256 checksums over HTTPS. The application was built using Spring Boot and configured with a self-signed certificate stored in a PKCS12 keystore. Secure communications were established via SSL/TLS, and the checksum logic was modularized for clarity and reusability. Functional testing confirmed that the /hash endpoint returns accurate checksum data, and secondary testing using OWASP Dependency-Check version 12.1.3 validated that all dependencies are free of known vulnerabilities.

## Industry Standard Best Practices

This project adheres to industry-standard best practices for secure software development. The use of SHA-256 for checksum generation aligns with NIST recommendations for cryptographic hashing and ensures data integrity without exposing sensitive information. HTTPS was configured using a self-signed certificate stored in a PKCS12 keystore, following best practices for encrypted communication and secure certificate management. The application avoids hardcoded secrets and uses modular design principles by separating checksum logic into a reusable utility class.