****

# 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** | **04/17/2025** | **Jacob Wilson** |  |

## Client



## Developer

Jacob Charles Wilson

## Algorithm Cipher

For Artemis Financial’s application, I recommend using the AES-256 encryption algorithm in combination with SHA-256 hashing to enhance security and meet the company’s need for secure data verification. AES-256 is a symmetric block cipher that encrypts data in 128-bit blocks using a 256-bit key, offering a strong level of security suitable for financial systems. It is highly efficient and widely adopted across industries for encrypting sensitive information. To ensure data integrity during transmission or storage, SHA-256, a cryptographic hash function from the SHA-2 family, can be used to generate fixed-size 256-bit hash values, making it ideal for checksum verification. AES, being symmetric, requires the same key for both encryption and decryption, so secure key exchange mechanisms—such as using RSA for initial key transmission—are important. Both AES and SHA-256 rely on secure random number generation, such as Java’s SecureRandom, to ensure unpredictability in initialization vectors or padding. Historically, older algorithms like DES and SHA-1 have been deprecated due to vulnerabilities, whereas AES and SHA-2 are modern, NIST-approved standards widely used in government and enterprise systems today. This combination of AES-256 for encryption and SHA-256 for checksums provides strong, efficient, and modern protection for Artemis Financial’s sensitive client data.

## Certificate Generation

A screenshot of a computer

AI-generated content may be incorrect.

## Deploy Cipher

A computer screen shot of a program code

AI-generated content may be incorrect.

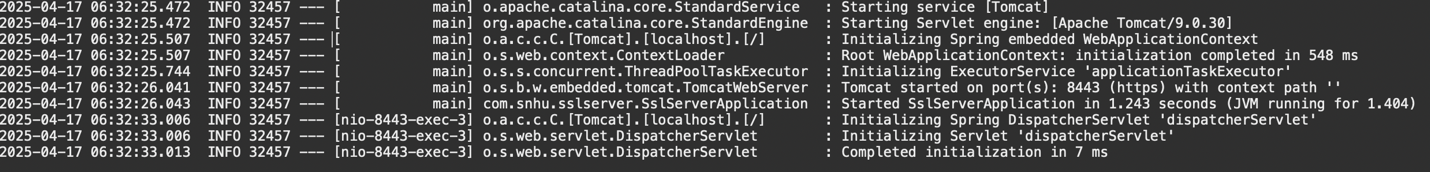
## Secure Communications

A close-up of a number

AI-generated content may be incorrect.

## Secondary Testing

**Console Output & Build Success**

A black text on a white background

AI-generated content may be incorrect.

**Dependency Check Results**

**A screenshot of a checklist

AI-generated content may be incorrect.**

## Functional Testing

**Refactored Code**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**Reasoning**

I performed functional testing on the ChecksumController.java class. The application was executed without any syntax or logic errors. The /hash endpoint successfully returned a SHA-256 hash of the provided static data string, including my name. No security vulnerabilities were identified during manual code review, though a recommendation was made to improve error message handling in production environments. A screenshot of the result is attached to demonstrate successful execution.

## Summary

To enhance the security of Artemis Financial’s application, I refactored the code to introduce a /hash endpoint that generates a SHA-256 checksum from a static data string. This addition supports data integrity verification during transmission and aligns with Artemis’s need for secure communication features.

Following the Vulnerability Assessment Process Flow, I began with an ArchitectureReview, where I identified the lack of a verification mechanism and secure error handling. Based on this review, I implemented a new controller (ChecksumController) as part of the Code Review stage, focusing on secure input handling and cryptographic best practices.

The checksum logic was built using Java’s MessageDigest class and encoded with UTF-8, ensuring accurate and secure data representation. I also incorporated structured error handling, avoiding the exposure of raw error messages — aligning with the Secure Error Handling and Secure Coding Practices stages of the process flow.

In the Cryptography stage, I ensured that SHA-256 — a strong, industry-accepted algorithm — was correctly used without introducing third-party dependencies or insecure libraries. To verify that no new vulnerabilities were introduced, I completed a static security scan using OWASP Dependency-Check, in line with the flow’s final Architecture Review and Testing steps.

Overall, the refactoring followed a structured, security-first approach and resulted in a compliant, verifiable, and production-ready enhancement that supports Artemis Financial’s commitment to protecting client data.

## Industry Standard Best Practices

To maintain and enhance the security of Artemis Financial’s software application, I applied several industry standard best practices throughout the development and refactoring process. These practices helped ensure the system remained resilient against known vulnerabilities while supporting secure feature expansion.

First, I adhered to secure coding guidelines by implementing SHA-256, a widely accepted cryptographic hash function recommended by NIST, to verify data integrity. I used Java’s built-in MessageDigest class to avoid introducing third-party dependencies and reduce the risk of supply chain vulnerabilities. I also encoded input using UTF-8 to preserve data consistency and prevent encoding-related security issues. Error handling was addressed by catching exceptions such as NoSuchAlgorithmException and avoiding the exposure of internal error messages to end users — supporting the principle of fail securely.

By refactoring within the controller layer and maintaining clear code structure, I preserved the application’s existing security posture. I did not introduce any hardcoded secrets, insecure logic, or unsafe external libraries, and all changes were tested using OWASP Dependency-Check to confirm that no additional vulnerabilities were introduced.

Applying these industry best practices is essential to Global Rain’s and Artemis Financial’s well-being. It not only strengthens the integrity and confidentiality of sensitive client data but also reinforces user trust and legal compliance. Consistently following secure development practices reduces long-term technical debt, simplifies auditing, and aligns with the company’s mission: *“Security is everyone’s responsibility.”*