****

# 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** | **06/22/2025** | **Andrew Lemus** |  |

## Client



## Developer

Andrew Lemus

## Algorithm Cipher

For this project, I implemented AES-256 encryption using Java’s javax.crypto package. AES-256 is a symmetric encryption standard that uses a 256-bit key and encrypts data in 128-bit blocks. It offers strong security and is efficient for applications like secure data storage.  
I also paired this with SHA-256 hashing to verify data integrity through a checksum. SHA-256 is a cryptographic hash function that produces a 64-character hexadecimal string from input data. Together, AES-256 and SHA-256 meet modern security standards for encryption and verification.

## Certificate Generation

## Figure 1: Exported self-signed certificate server.cer created from keystore.p12

A screenshot of a certificate

AI-generated content may be incorrect.

## Deploy Cipher

## Figure 2: SHA-256 checksum verification shown via the /hash endpoint

A black numbers on a white background

AI-generated content may be incorrect.

## Secure Communications

## Figure 3: Secure communication using HTTPS via self-signed certificate

A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

## Figure 4: OWASP Dependency Check output showing third-party vulnerabilities identified but not introduced by refactored code

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

## Figure 5: IntelliJ console showing successful startup of refactored Spring Boot application

A black background with white and blue text

AI-generated content may be incorrect.

## Summary

This project involved refactoring a Java-based secure server application for Artemis Financial to meet modern software security standards. AES-256 encryption was implemented to protect data confidentiality, and SHA-256 hashing was used for checksum verification. A self-signed certificate was generated and integrated to enable HTTPS for secure communication. OWASP Dependency Check was used to validate the security of third-party libraries, and the application was tested to ensure it executed without functional or security errors. The entire development, configuration, and testing process was completed using IntelliJIDEA, which provided integrated support for Maven and Spring Boot.

## Industry Standard Best Practices

In secure software development, it is critical to follow industry standards that prioritize data protection, integrity, and threat mitigation. AES-256 is widely recognized as a strong encryption standard for protecting data at rest, while SHA-256 ensures data integrity through hashing. Enabling HTTPS via SSL/TLS certificates helps secure data in transit. Dependency scanning with tools like OWASP Dependency Check allows early identification of vulnerable third-party components. Modern IDEs like IntelliJ IDEA support these best practices through plugin integration, static code analysis, and ease of running Maven-based security tools. Following these practices ensures that applications remain secure and compliant with current cybersecurity standards.