****

# 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** | **6/22/2025** | **Mark Christof** | **Initial submission** |

## Client



## Developer

Mark Christof

## Algorithm Cipher

For Artemis Financial’s application, I implemented the **SHA-256** algorithm, which is part of the SHA-2 family of cryptographic hash functions. SHA-256 is widely adopted for validating data integrity and verifying file authenticity (National Institute of Standards and Technology [NIST], 2002).

* **Overview**: SHA-256 produces a 256-bit fixed hash from an input string. It is deterministic, collision-resistant, and non-reversible.
* **Hashing & Bit Levels**: SHA-256 always generates a 256-bit hash output.
* **Symmetry**: It is a non-symmetric, one-way function, meaning no key is used for encryption or decryption.
* **Randomness**: It uses deterministic hashing, not random values.
* **History**: Developed by the NSA and published by NIST in 2002, SHA-2 replaced earlier, less secure algorithms like MD5 and SHA-1 due to identified vulnerabilities (NIST, 2002).

## Certificate Generation

A self-signed X.509 certificate was generated using Java’s keytool utility, part of the Java Development Kit (Oracle, n.d.-a). The certificate was exported as a .cer file and integrated into the key store to support HTTPS communication.

A screenshot of a certificate

AI-generated content may be incorrect.

## Deploy Cipher

Using the MessageDigest class in the java.security library, I implemented SHA-256 hashing in the Java application. A custom string (Mark Christof ArtemisSecure) was hashed and converted to hexadecimal using a byte array utility method, in accordance with standard cryptographic API guidance (Oracle, n.d.-b).

A screen shot of a computer

AI-generated content may be incorrect.

## Secure Communications

To implement HTTPS, the application was configured to use the generated certificate over port 8443 in Spring Boot. The application.properties file was updated to enable secure access at https://localhost:8443/hash (Allamaraju, 2014). A secure lock icon was verified in the browser, confirming valid SSL setup.



## Secondary Testing

A static vulnerability scan was conducted using the OWASP Dependency-Check Maven plugin (version 12.1.0). This was integrated into pom.xml and executed using the Maven install goal, validating that no CVEs were introduced by refactored code (OWASP, 2024).

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

After implementing all security improvements, the application was executed via Spring Boot with no errors encountered. Manual code review confirmed that no logic or syntax vulnerabilities were introduced.

A close-up of a computer screen

AI-generated content may be incorrect.

## Summary

Using the **Vulnerability Assessment Process Flow Diagram** provided by SNHU (2025), several critical security layers were addressed:

* **Cryptography**: SHA-256 hash algorithm deployed for data integrity.
* **Client/Server**: HTTPS configured using a self-signed certificate.
* **Code Error**: Handled with try-catch logic around hashing and key generation.
* **Code Quality**: Refactored methods with encapsulated logic and better structure.
* **Encapsulation**: Used secure data structures and minimized public exposure.

This approach improved the confidentiality and integrity of Artemis Financial’s communications and provided a secure foundation for data exchange.

## Industry Standard Best Practices

Throughout the project, I applied secure coding principles outlined by OWASP and trusted industry sources:

* I followed OWASP recommendations for secure cryptographic storage and input validation (OWASP, 2023).
* SSL was configured based on best practices in Java security (Oracle, n.d.-a).
* Static analysis through OWASP’s Dependency-Check ensured no new vulnerabilities were introduced (OWASP, 2024).
* The refactoring process aligned with defense-in-depth strategy and the principle of least privilege (Erickson, 2014).

Applying these best practices enables organizations like Artemis Financial to meet regulatory expectations, mitigate security risks, and maintain customer trust.

**References**

Allamaraju, S. (2014). *RESTful Web Services Cookbook*. O'Reilly Media.

Erickson, J. (2014). *Hacking: The Art of Exploitation* (2nd ed.). No Starch Press.

National Institute of Standards and Technology. (2002). *Announcing the Secure Hash Standard (SHS)* (FIPS PUB 180-2). U.S. Department of Commerce. <https://csrc.nist.gov/publications/detail/fips/180/2/final>

Oracle. (n.d.). *Java Security Standard Algorithm Names*. <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html>

Oracle. (n.d.). *keytool - Key and Certificate Management Tool*. <https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html>

OWASP. (2023). *OWASP Top 10 Web Application Security Risks*. https://owasp.org/www-project-top-ten/

OWASP. (2024). *Dependency-Check Maven Plugin*. <https://jeremylong.github.io/DependencyCheck/>

Southern New Hampshire University. (2025). *CS 305 Vulnerability Assessment Process Flow Diagram*.