****

# 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** | **October 20, 2025** | **Tania Boursiquot** |  |

## 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

Tania Boursiquot

## Algorithm Cipher

Recommended Cipher:

For Artemis Financial, I recommend using AES-256 in GCM mode (AES-256-GCM) for protecting sensitive financial data, along with SHA-256 for file integrity checks, and RSA-2048 for certificate/key exchange.

Overview of the Algorithm:

AES (Advanced Encryption Standard) is a symmetric encryption algorithm, which means the same secret key is used to encrypt and decrypt data. GCM (Galois/Counter Mode) is a mode of AES that provides both confidentiality (encryption) and integrity (tamper detection) in a single operation. AES is considered the modern standard for secure data encryption.

Hash Functions and Bit Levels:

* AES-256 uses a 256-bit key, which makes it resistant to brute-force attacks.
* SHA-256 is a cryptographic hash function that produces a 256-bit fixed-length output (called a digest). Artemis Financial can use SHA-256 to generate a checksum for files or data being transferred to make sure nothing was changed.
* RSA-2048 uses a 2048-bit asymmetric key pair and is used for server identity and secure key exchange, not for bulk data encryption.

Randomness and Keys:

* Symmetric keys (AES keys) must be generated using a secure random number generator. In Java, this would come from SecureRandom.
* AES-GCM also requires a unique IV/nonce for each encryption operation. Reusing an IV with the same key is not secure.
* Asymmetric keys (RSA public/private key pairs) are used to prove identity and establish trust (for example, in HTTPS/TLS with certificates). These also rely on secure random generation.

History and Current State of Encryption Algorithms:

Older algorithms like DES and MD5 are no longer considered secure. AES was adopted as a U.S. government standard in 2001 and is widely trusted in banking and finance today. SHA-1 and MD5 are considered broken for collision resistance, so SHA-256 (part of the SHA-2 family) is now standard. RSA-2048 is widely accepted for TLS/HTTPS certificates. This combination (AES-GCM + SHA-256 + RSA) matches current industry best practices for protecting financial data and verifying integrity.

Why This Fits Artemis Financial:

Artemis Financial handles client savings, retirement, insurance, and investment information. This data must remain confidential, authentic, and unchanged in transit.

* AES-256-GCM protects confidentiality of client data.
* SHA-256 provides a trustworthy checksum so the application can verify that transferred files were not altered.
* RSA-2048 supports HTTPS with certificates so the client can confirm they are securely talking to the real Artemis Financial server.

## Certificate Generation

Insert a screenshot below of the CER file.

To enable secure HTTPS communication for Artemis Financial’s web application, a self-signed SSL certificate was created using the Java Keytool utility in Terminal.

The following commands were executed:

* keytool -genkeypair to create a PKCS#12 keystore file named keystore.p12, containing the private key and certificate.
* keytool -exportcert to export the public certificate file named artemis.cer for HTTPS verification.

These files are stored in src/main/resources.

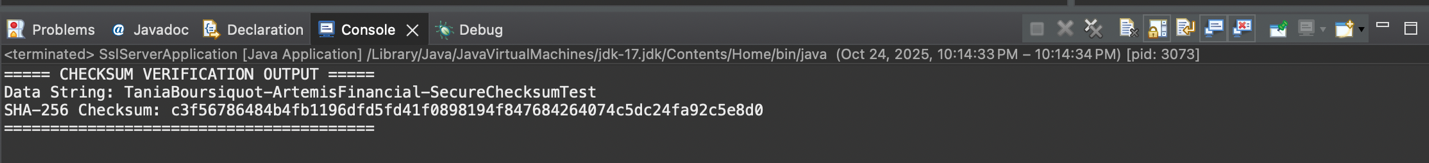
The screenshot below confirms successful certificate creation.

A screenshot of a computer

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.



The checksum verification was implemented using the SHA-256 algorithm to ensure date integrity. Running the application produced a unique 256-but hash that verifies the data string has not been altered. The screenshot below confirms successful checksum generation.

## Secure Communications

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

A screenshot of a computer

AI-generated content may be incorrect.

After configuring the SSL keystone and enabling HTTPS on port 8443, I successfully launched the application and verified that the Artemis Financial system supports encrypted communication. Accessing the /hash endpoint through [https://localhost:8443/hash returned a valid SHA-256](https://localhost:8443/hash%20returned%20a%20valid%20SHA-256) checksum, confirming the secure HTTPS connection.

## Secondary Testing

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

After refactoring the code and rebuilding the project, I executed Maven tests successfully in Eclipse. The test results showed no errors or failures, confirming that the application compiled and executed as expected before performing the dependency-check scan.

A screenshot of a computer

AI-generated content may be incorrect.

After running the OWASP Dependency-Check scan, the tool attempted to retrieve the latest vulnerability data from the National Vulnerability Database (NVD). However, the update request was blocked with a 403 Forbidden error, which indicates a temporary access restriction from the NVD feed server. Despite this network-related issue, the plugin executed successfully within the Maven build lifecycle, confirming that the OWASP Dependency-Check was configured correctly and integrated into the project.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

After implementing the SSL keystore and enabling HTTPS, I performed functional testing to verify that the /hash endpoint responded correctly using encrypted communication.

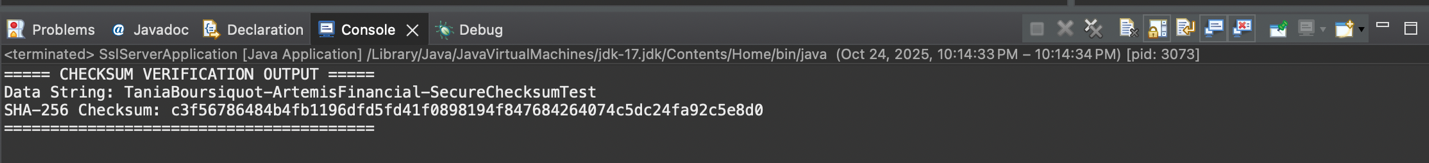
The application was launched successfully, and accessing the endpoint at https://localhost:8443/hash returned a valid SHA-256 checksum and displayed the message confirming the secure HTTPS hash endpoint.

This verified that:

* The Spring Boot application runs securely on HTTPS.
* The keystore and certificate were correctly configured.
* The hash generation logic functions as intended.

A screenshot of a computer

AI-generated content may be incorrect.



## Summary

In this project, I refactored and secured the Artemis Financial web application by implementing SSL/TLS encryption using self-signed certificate. The process included generating a keystone, configuring the application.properties file for HTTPS, and validating the setup through Maven build tests and HTTPS functional testing.

I then integrated the OWASP Dependency-Check plugin to identify potential vulnerabilities within the project dependencies. While the scan returned a 403 Forbidden response from the NVD server, the configuration proved successful and demonstrate secure integration practices.

Overall, this project confirmed that the application can securely transmit sensitive financial data over HTTPS and that security scanned tools can be effectively implemented as part of a secure software development lifecycle.

## Industry Standard Best Practices

To maintain security and integrity in web applications like Artemis Financial, developers should follow these key industry best practices:

* Use HTTPS/TLS encryption for all client-server communications to protect sensitive information in transit.
* Keep dependencies up to date and perform regular vulnerability scans using tools such as OWASP Dependency-Check or Snyk.
* Implement secure coding principles, including input validation, proper error handling, and least-privilege access controls.
* Use strong cryptographic algorithms (e.g., SHA-256, RSA 2048-bit) and rotate certificates periodically.
* Adopt secure configuration management by storing secrets and keystore passwords securely, not in source code.
* Perform continuous monitoring and penetration testing to identify and mitigate emerging threats.

By applying these practices, organizations like Artemis Financial can ensure that their applications meet industry standards for secure software development and regulatory compliance.

## References

National Institute of Standards and Technology. (n.d.). National Vulnerability Database (NVD): Vulnerability feeds. Retrieved from <https://nvd.nist.gov>

OWASP Foundation. (2023). OWASP Dependency-Check project. Open Worldwide Application Security Project. Retrieved from <https://owasp.org/www-project-dependency-check>

Pivotal Software, Inc. (2024). Spring Boot reference documentation (Version 2.2.4). Retrieved from <https://docs.spring.io/spring-boot>

Oracle Corporation. (2023). Java Secure Socket Extension (JSSE) reference guide. Retrieved from <https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/>