****

# 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** | **[Date]** | **[Your Name]** |  |

## 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

Ayomide Akingbemisilu

## Algorithm Cipher

Recommendation: Use **SHA-256** (from the SHA-2 family) for checksum verification. SHA-256 is a cryptographic hash (one-way function) that outputs a 256-bit digest and is widely accepted by NIST. It replaces deprecated options like MD5 and SHA-1 for integrity checks.

**Overview**: A hash function maps data of arbitrary length to a fixed-length output. Because it is one-way and collision-resistant, it is ideal for verifying file integrity without exposing secrets.

**Hash functions & bit levels**: SHA-256’s 256-bit digest makes preimage and collision attacks computationally infeasible for current threat models. Longer variants (SHA-384/512) exist, but 256-bit is a strong, performant default for web apps.

**Random numbers and keys**: Hashing alone does not require random numbers or keys. However, the application also uses TLS (HTTPS), which relies on secure randomness for key exchange and session keys. If authenticity of the producer is needed in addition to integrity, use HMAC-SHA-256 with a secret key or a digital signature with an asymmetric key.

**Symmetric vs. asymmetric**: The checksum (SHA-256) is not encryption. For transport security and server authentication we rely on asymmetric cryptography via X.509 certificates; for message authentication within services, we can use symmetric HMAC.

**History & current state**: Earlier algorithms (MD5/SHA-1) are considered broken/weak; current best practice is SHA-2 or SHA-3. For this project, SHA-256 provides excellent compatibility and security for file verification.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer program

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

AI-generated content may be incorrect.

## 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.

## Secondary Testing

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

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

A screenshot of a computer

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

AI-generated content may be incorrect.

## Summary

I refactored Artemis Financial’s application to add layered defenses that align with the Vulnerability Assessment Process Flow Diagram. Key changes include:

1. A **SHA-256 checksum utility and endpoint** to verify file/data integrity.
2. **HTTPS/TLS enablement** using a self-signed X.509 certificate to protect data in transit.
3. **Configuration hardening** in application properties (TLS keystore).
4. **Continuous static analysis** using OWASP Dependency-Check to ensure no new vulnerable libraries were introduced.

Functional testing confirmed the application builds and runs without errors, the checksum endpoint returns the expected digest for a unique test string containing my name, and the site is reachable over https://localhost:8443. Dependency-Check verified the refactor did not raise new high-severity findings.

Collectively, these changes improve **integrity, confidentiality, and authenticity** while maintaining application functionality.

## Industry Standard Best Practices

* **Use strong, approved primitives**: SHA-256 for integrity, RSA 2048+ for certificates, and TLS for transport security. Avoid deprecated algorithms (MD5, SHA-1) and weak key sizes.
* **Enforce secure defaults**: Redirect/require HTTPS and load the server keystore via configuration rather than hard-coding secrets in code.
* **Least privilege & secret management**: Keep keystore passwords out of source control (use environment variables or secure config stores in production).
* **Validate dependencies continuously**: Integrate OWASP Dependency-Check in Maven builds and fail on critical CVEs to prevent vulnerable libraries from shipping.
* **Defensive coding**: Clear error messages without leaking sensitive details; input handling that avoids injection and deserialization risks; and consistent logging for auditability.
* **Test and iterate**: Build → scan → fix → retest until no high-risk issues remain. This reduces operational risk and protects client data—critical for a financial services context.