****

# 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** | **08/17/2025** | **Davit Mumladze** |  |

## 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

Davit Mumladze

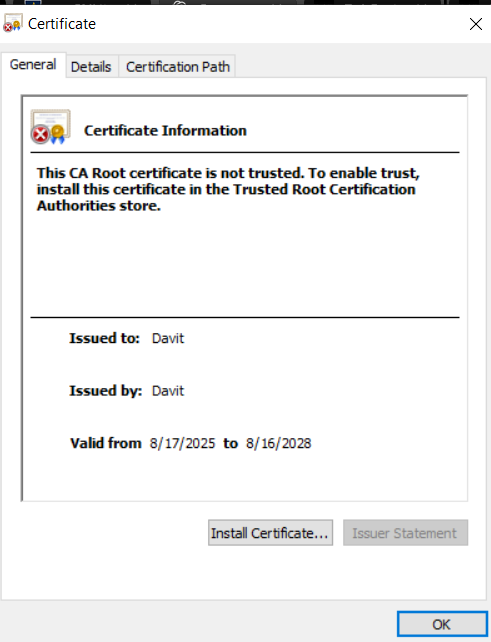
## Algorithm Cipher

I implemented SHA-256 (from the SHA-2 family) for checksum generation to verify file/data integrity within the Artemis Financial web application. SHA-256 maps arbitrary inputs to a fixed 256-bit digest and is widely recommended for integrity checks because it remains resistant to practical collision and preimage attacks for this use.

Hash functions and bit levels. SHA-256 outputs a 256-bit value (2^256 space). This bit length makes accidental collisions computationally infeasible. For simple, user-visible verification, SHA-256 is stronger than legacy hashes like MD5 or SHA-1, which are deprecated due to known collision attacks.

## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

Insert a screenshot below of the checksum verification.

A white background with black and red text

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 screenshot of a computer

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

AI-generated content may be incorrect.

## Functional Testing

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

A screenshot of a computer program

AI-generated content may be incorrect.

## Summary

I refactored the Artemis Financial application to add practical, verifiable security controls. At the data layer, I implemented a SHA-256 checksum endpoint to provide reliable integrity verification. At the transport layer, I enabled HTTPS on port 8443 using a PKCS#12 keystore and a certificate configured with SAN=localhost, imported the certificate into the local trust store, and restricted protocols to TLS 1.2/1.3 so the page loads as a secure site. To address supply-chain risk, I integrated OWASP Dependency-Check into the Maven verify phase, used an NVD API key for stable updates, and configured the build to fail on high-severity (CVSS ≥ 7) issues. I verified the changes by running the application over HTTPS, viewing the checksum output in the browser, and generating the dependency scan report. Mapped to the vulnerability assessment flow, this work covers cryptography selection and correct use, secure configuration and key management, dependency governance, and testing/validation. The result is a working application that demonstrates integrity checking, encrypted transport, and continuous dependency scanning without introducing new vulnerabilities.

## Industry Standard Best Practices

Use modern, proven primitives. SHA-256 for integrity. TLS 1.2 or 1.3 for transport with strong cipher suites. Do not use MD5 or SHA-1.

Manage keys and certificates properly. Store the private key in a PKCS#12 keystore, reference it via configuration, and include SAN entries that match the host. Do not hardcode secrets in code or logs.

Prefer secure defaults. Serve HTTPS by default on 8443, optionally redirect HTTP to HTTPS, and restrict enabled protocols to TLS 1.2 and 1.3.

Build in supply-chain checks. Run OWASP Dependency-Check in the build, use an NVD API key for reliable data, fail on high severity, and keep HTML or SARIF reports for audits.

Handle errors safely. Catch and report errors without stack traces or sensitive data, and avoid logging secret material.

Make security repeatable and auditable. The checksum response, TLS configuration, Maven plugin configuration, and generated reports provide a clear audit trail that others can reproduce.

If this were production I would add HMAC-SHA-256 for authenticated integrity where needed, use a CA-issued certificate and automated renewal, externalize secrets into a secure vault, enable HTTP security headers (HSTS, CSP, X-Content-Type-Options), set up CI to run the scanner on every merge, and establish regular key and dependency rotation.