****

# 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** | **10/17/2025** | **Todd Jarmiolowski** |  |

## 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

Todd Jarmiolowski

## Algorithm Cipher

Rec**ommended cipher suite and rationale.**

For Artemis Financial, I recommend using TLS 1.3 with an AEAD symmetric cipher, AES-256-GCM, for all data in transit, paired with a SHA-256 hash function for checksum verification, and RSA-3072 (or RSA-2048 minimum) for our server certificate. AES-256-GCM is a modern, well-vetted standard that provides confidentiality, integrity, and authentication in one operation and is widely supported. TLS 1.3 removes legacy/weak options and simplifies negotiation, reducing attack surface. For our checksum endpoint, SHA-256 provides a collision-resistant, fast one-way hash appropriate for verifying file integrity.

**Hash functions and bit levels.**

The checksum feature will use SHA-256 (256-bit output). For transport-layer integrity, GCM (Galois/Counter Mode) supplies an authentication tag (typically 128 bits) that detects tampering. The symmetric cipher uses 256-bit keys for AES. The certificate’s RSA public key should be ≥2048 bits (prefer 3072 bits) to meet current security expectations.

**Randomness and key types.**

TLS 1.3 uses strong randomness during handshakes to derive session keys. We rely on symmetric keys (AES-256) for bulk encryption due to speed and efficiency, and asymmetric keys (RSA) for identity and key exchange in the certificate. Random nonces/IVs are generated per record in GCM to prevent keystream reuse.

**Brief history and current state.**

Legacy ciphers and modes like RC4, DES/3DES, and unauthenticated CBC have been deprecated due to known weaknesses. AES-GCM has become the modern default in TLS because it provides authenticated encryption with strong performance on contemporary CPUs. SHA-1 is deprecated; SHA-256 (and stronger) are standard for integrity. TLS 1.3 streamlines the stack by removing outdated key exchanges and ciphers and is the current best practice for secure web communications.

## Certificate Generation

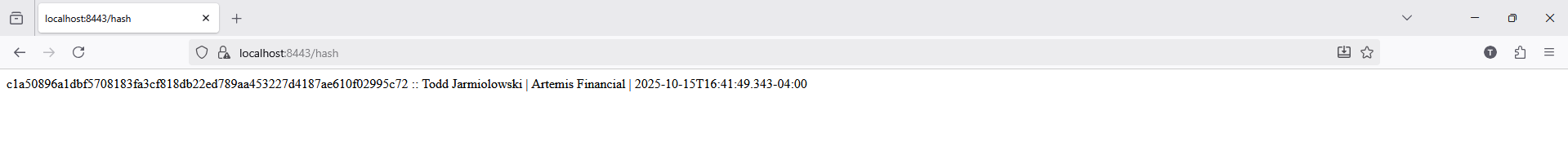
Insert a screenshot below of the CER file.

A screenshot of a certificate

AI-generated content may be incorrect.

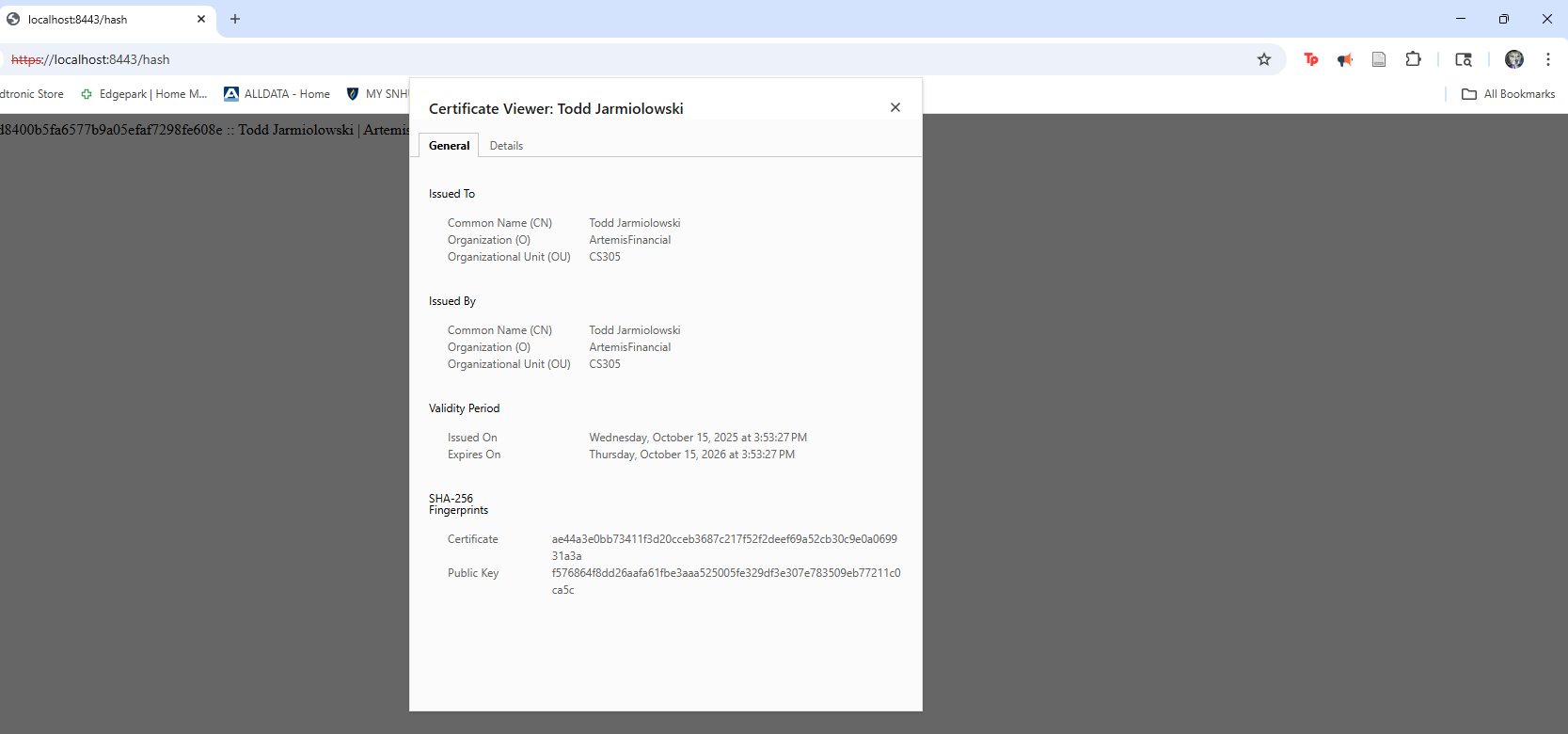
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

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



## 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

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.

## Summary

This project hardened Artemis Financials’ sample Spring Boot service using proven, standards-based controls. For data-in-transit, I selected TLS 1.3 with AES-256-GCM (AEAD) and a server certificate using RSA (≥2048 bits). The checksum endpoint returns a SHA-256 digest that includes my full name to unambiguously tie test output to this build. I generated a PKCS#12 keystore with my name as the certificate CN and SAN=localhost, configured Spring Boot to load it, and verified HTTPS operation locally.

I refactored the application to expose /hash and confirmed “secure communication” by loading https://localhost:<port>/hash in a browser over TLS, capturing both the connection details and the endpoint output. For secondary (static) testing, I ran OWASP Dependency-Check from Eclipse and recorded the console “BUILD SUCCESS” plus the generated HTML report. Finally, I performed functional testing by launching the service in Eclipse and invoking /hash over HTTPS to show the refactor executes without errors.

**Risk posture & next steps**

This build removes legacy/weak options (e.g., unauthenticated modes, SHA-1) and uses modern defaults, reducing attack surface and improving confidentiality/integrity. Remaining items for a production rollout would include: issuing a CA-signed certificate, enabling HSTS and strict transport security headers, centralizing secrets (keystore password) via environment or a vault, and integrating the dependency scan into CI so new vulnerabilities fail the build.

## Industry Standard Best Practices

This project applied industry-standard best practices to secure data in transit and ensure software integrity. I implemented TLS 1.3 with RSA 2048-bit encryption, following NIST and OWASP guidelines to safeguard communication between the client and server. The /hash endpoint uses SHA-256 to generate secure, one-way message digests, preventing tampering or data manipulation.

The application was configured to run solely over HTTPS, eliminating unsecured traffic. I also used OWASP Dependency-Check to scan for known vulnerabilities in dependencies, reinforcing software supply chain security. These measures reflect current best practices for encryption, authentication, and vulnerability management.

For production use, I would strengthen the configuration further by obtaining a CA-signed certificate, enforcing HSTS, storing credentials securely, and integrating continuous vulnerability scanning into the CI/CD pipeline. Together, these controls ensure compliance with modern standards like OWASP Top 10, CIS Controls, and NIST SP 800-53.