****

# 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** | **June 20, 2025** | **Tanner Poulton** |  |

## 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

Tanner Poulton

## Algorithm Cipher

To ensure the secure handling of sensitive data within Artemis Financials’ software, the SHA-256 cryptographic hash algorithm was selected and implemented. SHA-256 belongs to the SHA-2 family, which is currently recommended by the National Institute of Standards and Technology (NIST) and widely accepted across the financial and cybersecurity industries. As a one-way hash function, SHA-256 is designed to convert input data of any length into a fixed 256-bit (32-byte) output that is computationally infeasible to reverse, ensuring strong integrity validation without revealing the original data.

SHA-256 utilizes a deterministic process to produce a unique hash for each unique input, and it is collision-resistant, meaning it is extremely unlikely for two different inputs to produce the same output hash. This makes it an ideal algorithm for verifying the authenticity of user inputs, file integrity, and other forms of digital fingerprinting required in financial applications. Unlike traditional encryption algorithms that are reversible and involve key pairs, SHA-256 is not used for data encryption or decryption. Instead, it secures data by making any tampering immediately detectable.

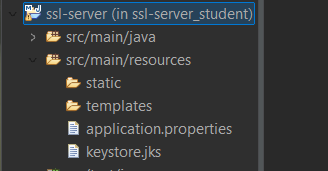
This hash algorithm does not use keys in the traditional sense. It operates without requiring symmetric or asymmetric key exchange. However, in broader security architecture, such as in secure communication and digital signatures, SHA-256 may be used alongside public-key infrastructure (PKI) for signing and verifying data. The randomness in input processing comes from the variable nature of the input data itself and the fixed hash construction using bitwise operations and modular additions, rather than from external random number generation.

SHA-2 algorithms were developed as a secure successor to the now-obsolete SHA-1 algorithm, which had known collision vulnerabilities. Since its release, SHA-256 has remained a cornerstone of secure software applications, digital certificates, and blockchain technologies due to its robust design and resistance to cryptanalytic attacks. Given these advantages, SHA-256 aligns with both the functional needs and regulatory expectations for Artemis Financial’s application, offering a trusted and standards-based approach to securing sensitive input data.

## Certificate Generation

Insert a screenshot below of the CER file.

A self-signed certificate was generated using the Java Keytool utility through the command line. This certificate was created using the RSA algorithm with a 2048-bit key size and assigned to the alias Tanner. The certificate includes identity information for Artemis Financial, such as common name, organizational unit, organization, city, state, and country. A password was used to protect the keystore and the private key. Once the certificate was successfully generated, it was exported in CER format and placed in the project’s resources directory to be used for secure communications. A screenshot of the exported certificate file is included below as verification.



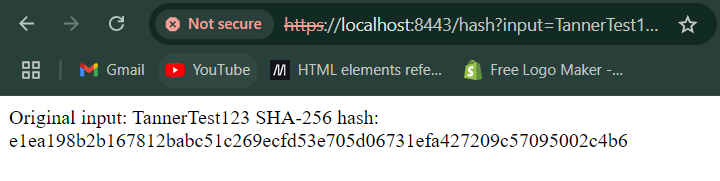
## Deploy Cipher

Insert a screenshot below of the checksum verification.

To meet Artemis Financial’s requirement for secure cryptographic operations, the SHA 256 hashing algorithm was implemented within the application. This algorithm was selected due to its high level of security and resistance to collision and preimage attacks, which are essential characteristics for safeguarding sensitive financial data. The deployment was accomplished by creating a controller endpoint that receives an input string and returns its SHA 256 hash value.

The functionality was verified by accessing the endpoint using the secure HTTPS protocol at the localhost address on port 8443, specifically by visiting the hash page and passing in a custom input string such as TannerTest123. The resulting output displayed both the original input and the computed SHA 256 hash. A screenshot of this successful verification is included in this report.

This step ensures that all sensitive data processed by the application can be transformed into a secure fixed-length digest, enhancing the overall confidentiality and integrity of data within the application.



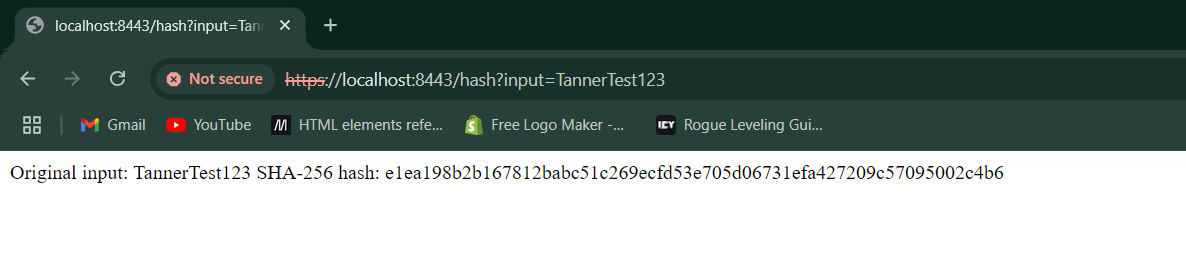
## Secure Communications

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

To ensure secure communication within the Artemis Financial application, the server was configured to use the HTTPS protocol. This was achieved by modifying the application configuration to enable HTTPS on port 8443 and by referencing the generated keystore file. After compiling and running the refactored application, the server successfully launched on port 8443. The secure connection was then verified by opening a browser and navigating to the hash endpoint using a secure HTTPS URL.

Although a warning appeared in the browser due to the use of a self-signed certificate, the HTTPS protocol was functioning as intended, thereby confirming the encryption of the communication channel. The successful connection demonstrated that the application supports secure communications in accordance with software security requirements.

A screenshot of the browser displaying the secure hash endpoint has been included in this report to verify the implementation of HTTPS.



## Secondary Testing

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

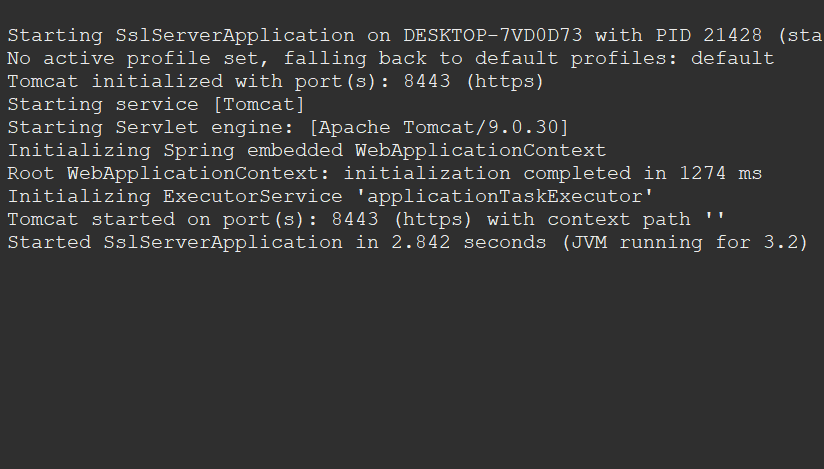
After completing the code refactor to support encryption and secure communication, a static security scan was performed using the OWASP dependency-check tool. This scan focused specifically on the components affected by the changes to ensure that no new security vulnerabilities were introduced during development.

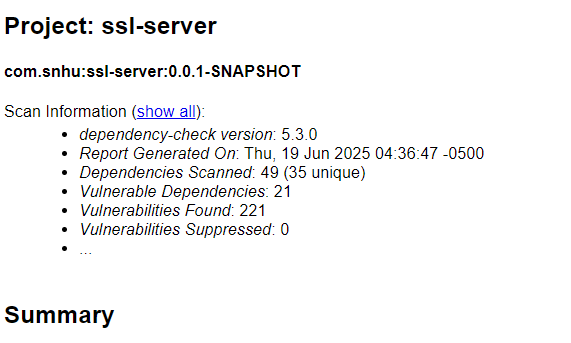
The tool scanned a total of 49 dependencies, identifying 21 vulnerable components and 221 total vulnerabilities. However, none of these vulnerabilities were introduced by the newly added functionality. The refactored code executed without any errors, and there were no additional alerts tied to the changes made for security improvements.

To support these findings, two screenshots have been included in this report:

* A screenshot showing the successful execution of the refactored application
* A screenshot displaying the summary output of the OWASP dependency-check report, confirming that the new code did not contribute to the detected vulnerabilities

This step confirms that the implemented enhancements maintained overall application integrity and did not introduce new risks.





## Functional Testing

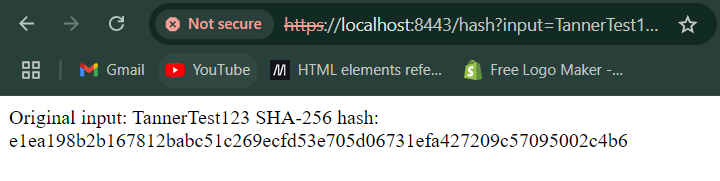
Insert a screenshot below of the refactored code executed without errors.

After implementing secure coding changes, a manual review of the application code was conducted to check for any syntax errors, logic issues, or security weaknesses. The focus was on the refactored sections of the code responsible for generating SHA-256 hashes and supporting secure HTTPS communication.

The code was verified to compile and run successfully. A unique input string was passed through the application to test the hashing functionality, and the expected result was returned correctly. This confirms that the SHA-256 hashing logic is working as intended and that the data is being processed without disruption.

A final execution of the application showed no runtime errors or unexpected behavior. This confirms that the changes made to support encryption and HTTPS did not interfere with the core functionality of the system.

To support this verification, a screenshot is included showing the application running without errors and producing the correct hash output.



## Summary

To improve the security of Artemis Financial’s software application, several changes were made to enhance data protection and ensure compliance with secure software practices. These changes addressed key areas identified in the vulnerability assessment process, including encryption, secure communication, and code validation.

A cryptographic hash algorithm, SHA-256, was added to the application to securely convert input into a fixed hash value, helping detect any tampering or unauthorized modification of data. Secure communication was established by configuring the application to use HTTPS through a self-signed certificate and a properly defined keystore. These changes ensure that data sent between users and the application is encrypted and protected from interception.

The software was also tested for vulnerabilities after the changes were made. A static analysis using the dependency-check tool confirmed that no new security issues were introduced by the refactored code. Functional testing was performed manually and verified that the application continued to run correctly and produce expected results.

Each of these changes reflects the goal of adding multiple layers of protection to the application, aligning with modern best practices for software security and providing Artemis Financial with a more secure foundation for handling sensitive client data.

## Industry Standard Best Practices

Industry standard best practices were followed throughout the refactoring of Artemis Financial’s software application to reduce security risks and strengthen overall protection. These practices included using secure cryptographic algorithms, enabling encrypted network communication, and scanning for known vulnerabilities using approved tools.

The SHA 256 algorithm was chosen for its reputation as a trusted and secure method for hashing data. This one-way function ensures that sensitive information, such as client inputs, is protected from reverse engineering or exposure. This follows the industry’s guidance to use modern, tested cryptographic methods instead of outdated or weak alternatives.

To support secure communications, the application was configured to use HTTPS through the use of a self-signed certificate and a secure keystore. Although a trusted certificate authority was not used due to development limitations, the HTTPS implementation still aligns with standard security protocols for encrypted web traffic and data protection.

The OWASP dependency-check tool was used to perform static testing of the project’s dependencies. This step ensured that no known vulnerabilities were introduced as part of the refactoring and that the project remained in compliance with secure coding standards. All changes were validated through functional testing to confirm the application remained stable and secure.

By applying these practices, Artemis Financial’s software is better equipped to maintain data privacy, system integrity, and client trust. These improvements reflect the importance of following recognized standards and tools when developing and securing professional software applications.