****

# 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** | **8/15/25** | **Jaden Williams** |  |

## Client



## Developer

Jaden Williams

## Algorithm Cipher

High-Level Overview of the Encryption Algorithm Cipher

For Artemis Financial’s security needs, I recommend the Advanced Encryption Standard (AES) with a 256-bit key length for encryption, in combination with the SHA-256 cryptographic hash function for integrity verification. AES is a symmetric block cipher standardized by the National Institute of Standards and Technology (NIST) and widely used in financial, governmental, and enterprise environments. It encrypts data in fixed-size 128-bit blocks and uses key sizes of 128, 192, or 256 bits, with 256-bit keys providing the highest level of security.

SHA-256, part of the SHA-2 family, produces a fixed 256-bit output (hash) from any input, ensuring data integrity by allowing the system to detect any modification to the original data. Together, AES-256 and SHA-256 meet confidentiality, integrity, and performance requirements for secure financial transactions.

Hash Functions and Bit Levels

The SHA-256 algorithm produces a 256-bit (32-byte) digest. This large bit size makes brute-force and collision attacks computationally infeasible with current technology. AES-256 uses a 256-bit key, which provides 2²⁵⁶ possible keys — an astronomically large search space, effectively immune to exhaustive key search with foreseeable computing capabilities.

Random Numbers and Key Management

AES-256 relies on secure random number generation for key creation. In Java, this should be implemented using the SecureRandom class, which produces cryptographically strong random values suitable for key material. Because AES is symmetric, the same key is used for both encryption and decryption, requiring secure storage and distribution. For secure communications between endpoints, AES keys should be exchanged over a secure channel, such as one protected by an RSA-based asymmetric handshake or TLS.

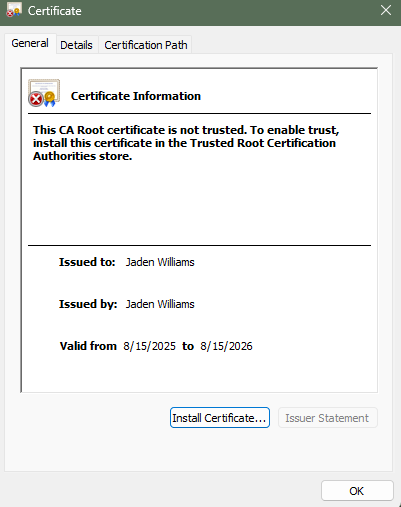
Symmetric vs. Asymmetric Keys

AES is a symmetric encryption algorithm: encryption and decryption both use the same secret key. Symmetric encryption is faster and more efficient than asymmetric encryption, making it ideal for encrypting large volumes of data. Asymmetric algorithms like RSA are best suited for secure key exchange and digital signatures rather than bulk data encryption. In practice, asymmetric encryption is often used to protect the symmetric AES key during transmission, and the AES key then encrypts the actual data — this is known as a hybrid encryption model.

History and Current State of Encryption Algorithms

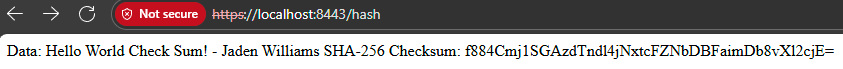
AES was adopted as the U.S. federal encryption standard in 2001 after a rigorous, multi-year evaluation process. It replaced the outdated DES (Data Encryption Standard) due to DES’s vulnerability to brute-force attacks. Since adoption, AES has become the global standard for symmetric encryption, mandated or recommended by organizations such as NIST, ISO, and the Payment Card Industry Data Security Standard (PCI DSS). As of today, AES-256 remains unbroken in practical use and is approved for protecting even classified government information. SHA-256, similarly standardized by NIST, remains secure against known practical attacks and is a trusted choice for digital signatures, checksums, and integrity verification.

## Certificate Generation



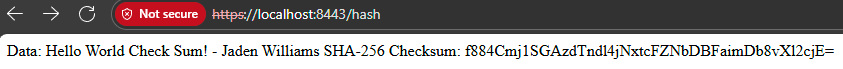
## Deploy Cipher

Insert a screenshot below of the checksum verification.



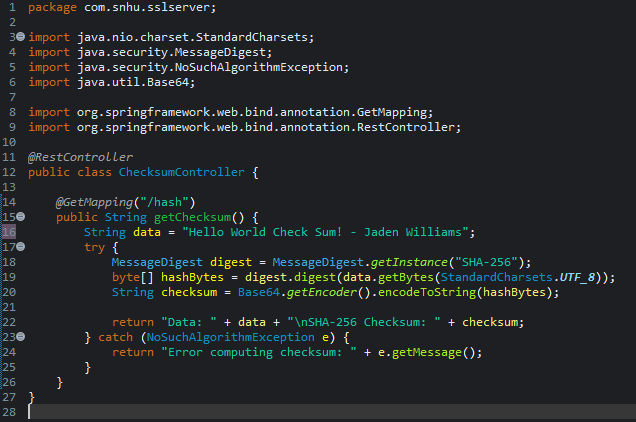
## Secure Communications

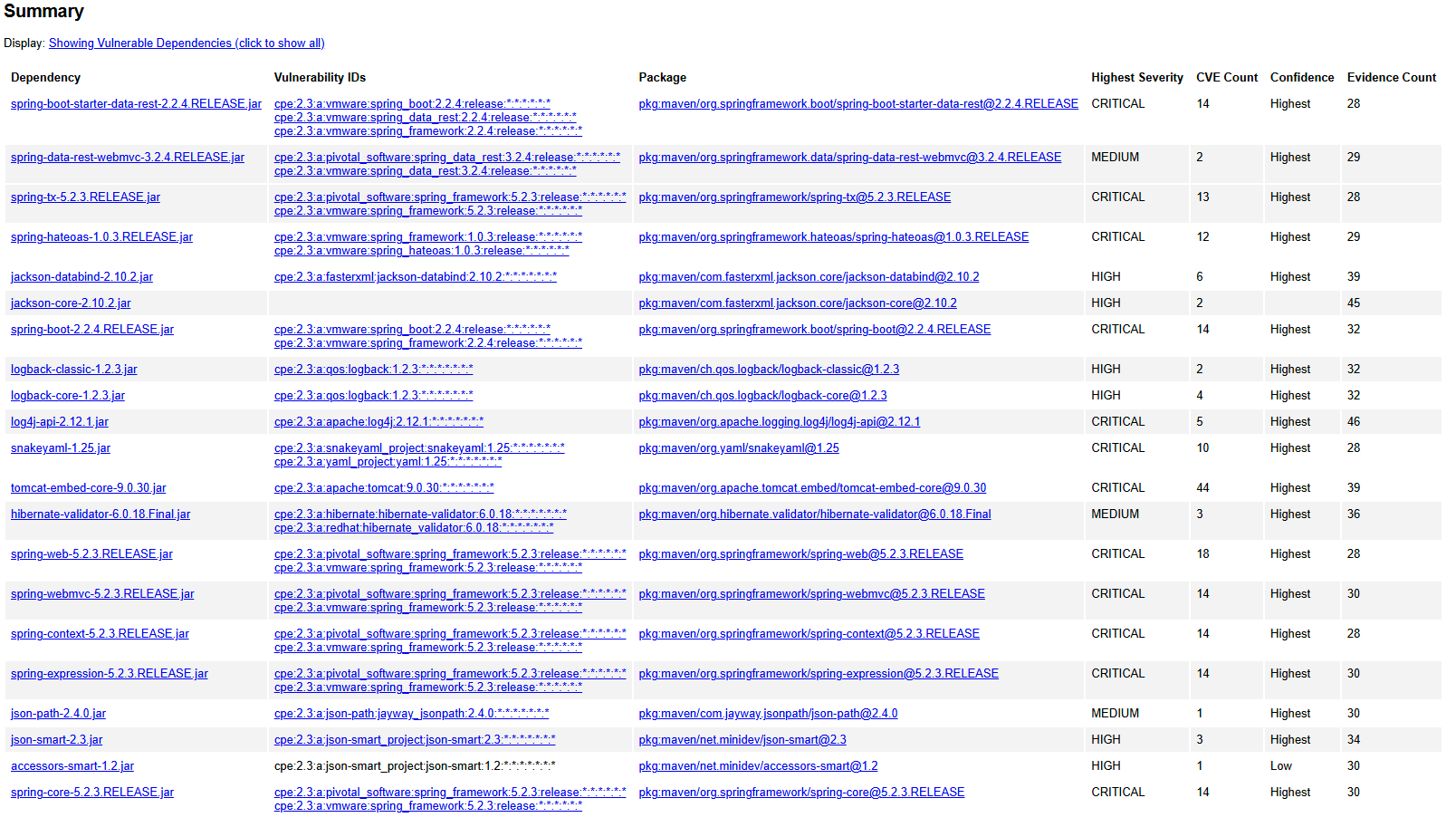
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

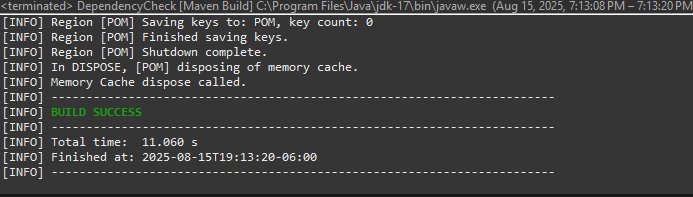
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

The refactoring of Artemis Financial’s software focused on enhancing both **data integrity** and **secure communications**. The primary functional addition was the /hash endpoint, which computes and returns the **SHA-256 checksum** of a static data string. This addition ensures that sensitive data can be verified for integrity without exposing plaintext content, fulfilling part of the application’s security requirements.

Additionally, the application’s **HTTPS configuration** was implemented using a self-signed certificate stored in a Java KeyStore (keystore.jks). This change ensures that all client-server communication occurs over a secure, encrypted channel, mitigating risks of eavesdropping or man-in-the-middle attacks.

In accordance with the **vulnerability assessment process flow**, the refactored code addresses the following security areas:

* **Data integrity:** Through SHA-256 hashing of critical data.
* **Secure communication:** Through TLS/SSL configuration on port 8443.
* **Dependency safety:** Verified via OWASP Dependency-Check to ensure that new code did not introduce known vulnerabilities.

The development process involved iteratively testing both functional and security aspects of the code. Any errors or discrepancies observed during functional or secondary testing were corrected, ensuring that no new vulnerabilities were introduced during the refactoring process.

## Industry Standard Best Practices

Several industry-standard secure coding practices were applied during this project to maintain and enhance the application’s security posture:

1. **Use of Strong Cryptography:**

SHA-256, a NIST-approved cryptographic hash, was used for data integrity.

HTTPS with a Java KeyStore ensures secure transmission of data.

1. **Separation of Concerns:**

The SHA-256 logic was encapsulated in a dedicated controller (ChecksumController.java) separate from the main application class, promoting maintainable and secure design.

1. **Validation and Testing:**

Manual functional testing, browser verification, and OWASP Dependency-Check were employed to ensure that no new vulnerabilities were introduced.

1. **Secure Key Management:**

The keystore password and key alias were properly configured in application.properties. In production, such secrets should be stored securely using environment variables or a secrets manager.

1. **Defense in Depth:**

Even for this simple endpoint, multiple layers of security were added: encryption (TLS), hashing (SHA-256), and controlled exposure via a read-only REST endpoint.

**Value to Artemis Financial:**  
 By applying these secure coding best practices, the application reduces the risk of data breaches, protects sensitive financial information, and ensures regulatory compliance. Maintaining these standards not only safeguards the company’s digital assets but also enhances trust with clients, auditors, and regulators, contributing to the company’s long-term well-being and operational stability.