****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.cky6ci957sts)

[**Client 3**](#_heading=h.t2lzjiczbf23)

[**Instructions 3**](#_heading=h.nwu0agbyvoy)

[**Developer 4**](#_heading=h.phndnmjw7p5g)

[**1. Algorithm Cipher 4**](#_heading=h.oamh15c6v9t9)

[**2. Certificate Generation 4**](#_heading=h.luionvs66rv2)

[**3. Deploy Cipher 4**](#_heading=h.tcuxzw9cxgh5)

[**4. Secure Communications 4**](#_heading=h.4edl2glczbli)

[**5. Secondary Testing 4**](#_heading=h.4syxhvddxu8y)

[**6. Functional Testing 4**](#_heading=h.pzw7wuzf7wyu)

[**7. Summary 4**](#_heading=h.3xdl80zamorx)

[**8. Industry Standard Best Practices 4**](#_heading=h.f492asfh9vzg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **Aug. 19, 2025** | **Aiden Villanueva** | **Initial Assessment and**  **Repair** |

## Client

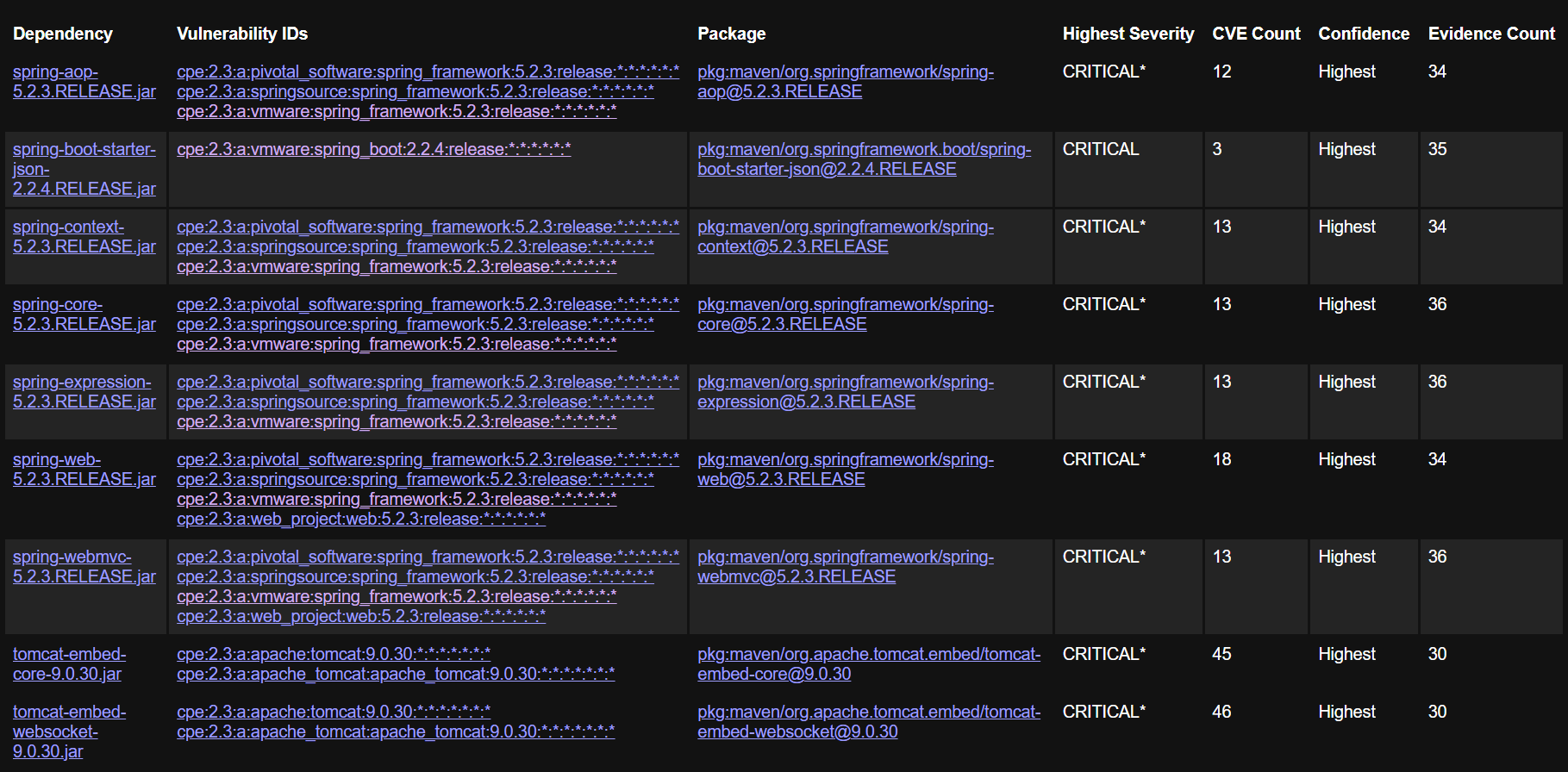


## Developer

Aiden Villanueva

## Preassessment

When initializing the project with the OWASP dependency vulnerability check, several core dependencies were identified with critical-level vulnerabilities.



Critical packages were researched on the Maven central repository. The inciting dependency seems to be the org.springframework.boot, which was updated to version 3.3.3, a current secure version. This requires using Java version 17 instead of 1.8, which provides more robust memory management and security features. The tomcat-embed-core cannot be updated beyond v10.1.42, but addresses this specific issue.

## Algorithm Cipher

For Artemis Financial's needs, a strong, industry-standard combination of algorithms is needed.

I recommend the SHA-256, Secure Hash Algorithm (256-bit). This cryptographic hash function produces a unique 256-bit hash code for any given piece of data. It's a one-way function, meaning it's practically impossible to reverse the process and find the original data from the hash. SHA-256 is widely trusted, has no known major vulnerabilities, and offers a strong balance of security and performance, making it perfect for verifying file integrity (checksums).

Likewise, for HTTPS communications, AES-256 (Advanced Encryption Standard 256-bit) is recommended. AES is a symmetric-key block cipher, meaning the same key is used for both encrypting and decrypting data. It encrypts data in fixed-size blocks (128 bits). The "256" refers to the key length, which is the strongest available for AES.

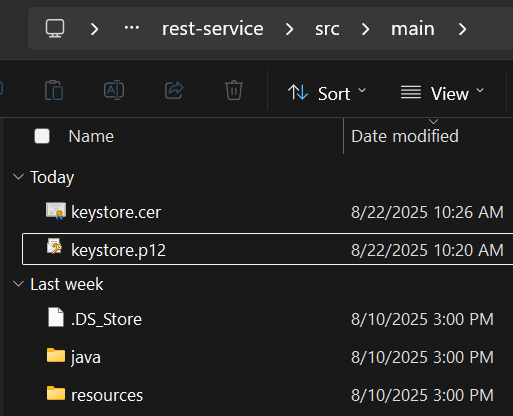
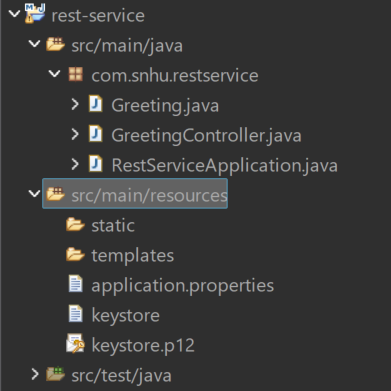
AES is the global standard for data encryption, used by governments and businesses worldwide. Its combination of speed, security, and versatility makes it the top choice for securing data in transit over HTTPS.

In addition, strong encryption depends heavily on the use of cryptographically secure random number generation (CSPRNG). Random numbers are essential for generating secure keys, initialization vectors, and nonces that prevent predictable encryption patterns. Without high-quality randomness, even strong algorithms like AES or SHA-256 could be weakened.

It is also important to distinguish between symmetric and asymmetric encryption. Symmetric algorithms, like AES, use the same key for both encryption and decryption, making them fast and efficient for securing data in transit. In contrast, asymmetric algorithms, like RSA or ECC, use a public-private key pair, which is more computationally expensive but allows secure key exchange and digital signatures. In practice, modern secure communication often combines both: asymmetric encryption to securely exchange a symmetric session key, and symmetric encryption (AES) for the actual data transfer.

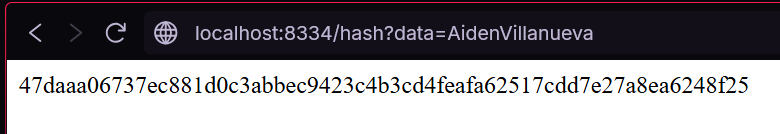
## Certificate Generation

The keystore.p12 key file and the .cer certificate file were both made via the Java Keytool CLI.



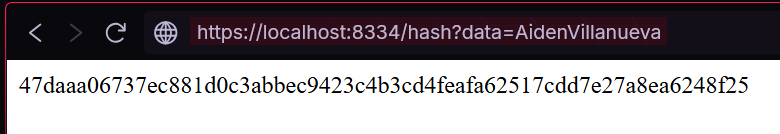
## Deploy Cipher

The following screenshot shows the checksum function operating as expected when reaching the /hash endpoint with ?data. This was achieved by using a @RestController to segregate the checksum function and its helpers from the main application file.



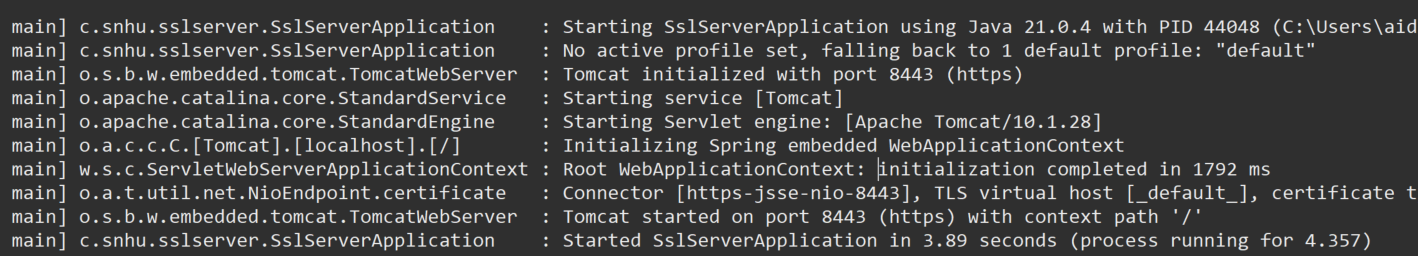
## Secure Communications

The following screenshot shows the checksum function working while under https protections.



## Secondary Testing

The following is the run output showing the server starting with https. Following is the post-refactor dependency check via OWASP.

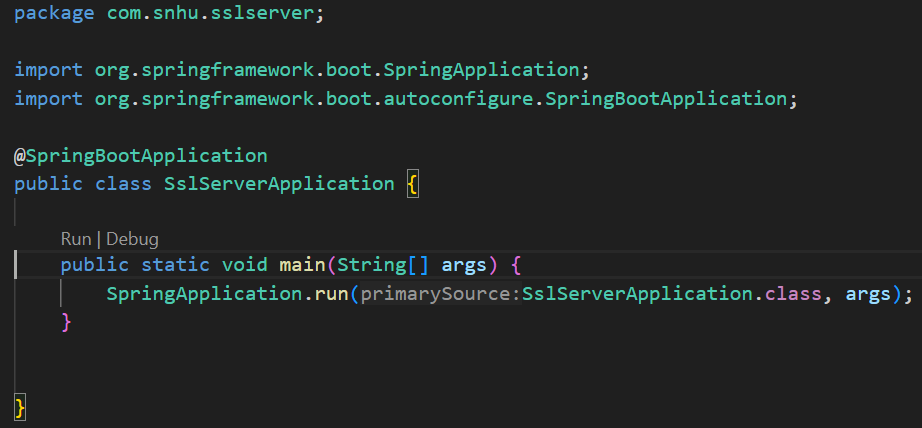


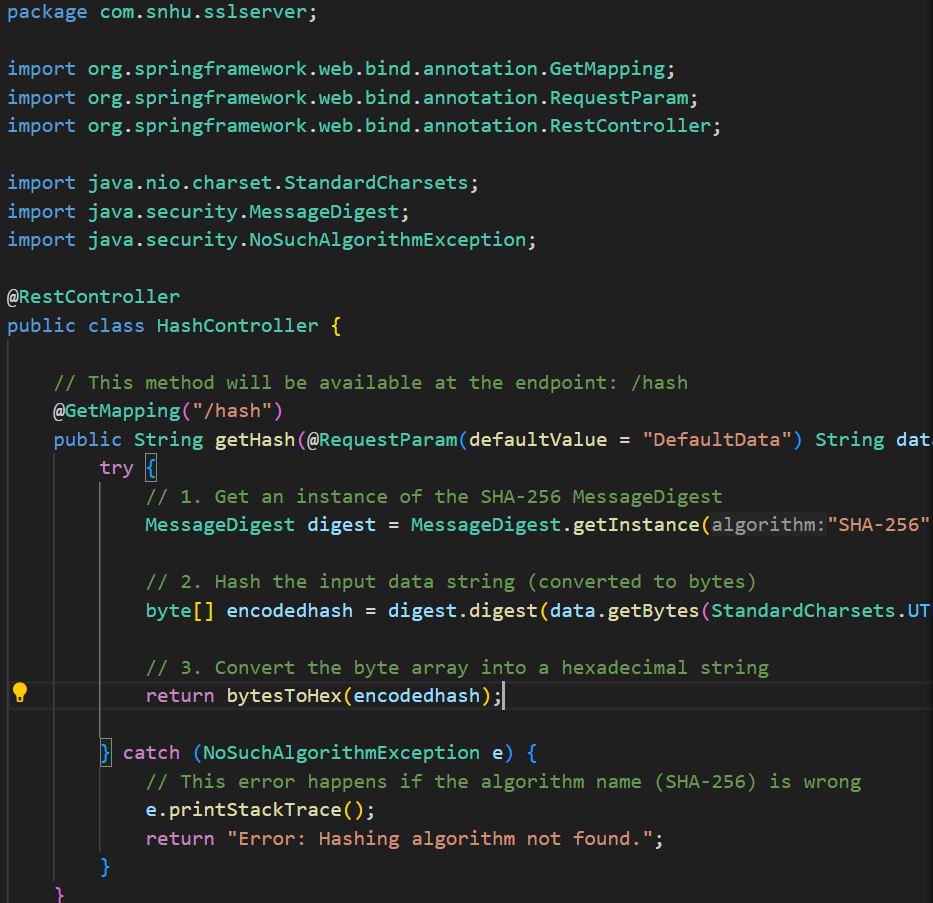


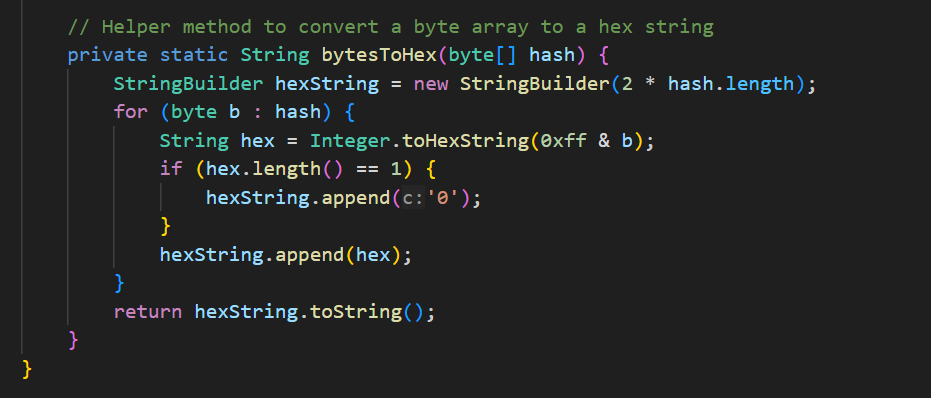
As shown, the only remaining critical vulnerability is the unavoidable tomcat-embed-core version 10.1.28 required by the springframework.boot v3.3.3.

## Functional Testing

The following screenshots show the refactored code, free of errors and with proper syntax and documentation.







## Summary

The Artemis Financial software application has been successfully refactored to remediate critical security vulnerabilities and add layered security controls. The process followed industry-standard protocols to identify, address, and verify security enhancements, significantly improving the application's overall security posture.

The refactoring process directly aligns with a standard vulnerability assessment workflow, addressing key stages of security analysis and response.

1. Identification & Analysis: The process began with static analysis using the OWASP dependency-check tool. This initial scan identified multiple critical vulnerabilities (CVEs) in outdated versions of the Spring Framework (v5.2.3) and Apache Tomcat (v9.0.30). Analysis confirmed these were not false positives and posed a significant risk to the application, particularly its public web interface.

2. Remediation: The primary refactoring effort focused on vulnerability remediation. Instead of suppressing the findings, we followed the best practice of updating the core dependencies. By upgrading the spring-boot-starter-parent dependency in the pom.xml to a modern, stable version (3.3.3) and the Java version to 17, the entire cascade of transitive dependency vulnerabilities were resolved in a single, managed step.

3. Verification: After remediation, a secondary static analysis was performed by re-running the dependency-check tool. The resulting report verified that the original critical vulnerabilities were successfully eliminated, confirming the completion of the refactoring and ensuring compliance with the "verify fixes" stage of the security lifecycle.

Beyond fixing existing flaws, the refactoring process added new, proactive security layers to protect Artemis Financial's data, following the principle of defense-in-depth.

Layer 1: Data Integrity. A new HashController was created with an endpoint (/hash) that uses the SHA-256 algorithm. This provides Artemis Financial with a mechanism to generate a unique digital fingerprint for any piece of data, allowing them to verify that files or information have not been altered or corrupted during transfer.

Layer 2: Data Confidentiality in Transit. I secured the communication channel by converting the application from HTTP to HTTPS. This involved generating a self-signed TLS certificate using the Java Keytool and configuring the embedded web server to use it. Now, all communication between the client and the server is encrypted, protecting sensitive financial data from eavesdropping and attacks.

## Industry Standard Best Practices

Throughout the project, industry standard best practices for secure coding were applied to mitigate vulnerabilities and strengthen the application.

The primary best practice applied was proactive dependency management. I updated the entire Spring Boot Bill of Materials (BOM) by changing the parent version. This is the recommended approach because it ensures that all transitive dependencies are compatible and have been tested to work together securely, which is far safer than attempting to update individual libraries one by one and risking version conflicts.

When adding the new hashing feature, a strong, modern cryptographic algorithm (SHA-256) was used. This is recommended by security standards bodies like NIST. I also followed the Separation of Concerns principle by placing web logic in a dedicated @RestController, keeping the main application class clean.

**Value of Applying Best Practices**

For a company like Artemis Financial, applying these best practices is not just a technical requirement, but a critical business function.

Security protects customer trust; by securing client data and financial information, the company protects its reputation and maintains the trust of its customers, which is its most valuable asset. Proactively eliminating critical vulnerabilities drastically reduces the risk of a data breach, which could lead to severe financial penalties, regulatory fines, and legal action. Further, a secure and stable application is essential for daily operations. Preventing attacks like DoS ensures that the company's services remain available to its customers and employees.