****

# 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** | **19OCT25** | **Alex Zelaya** |  |

## 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

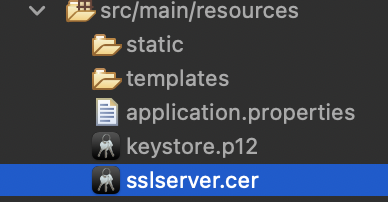
Alex Zelaya

## Algorithm Cipher

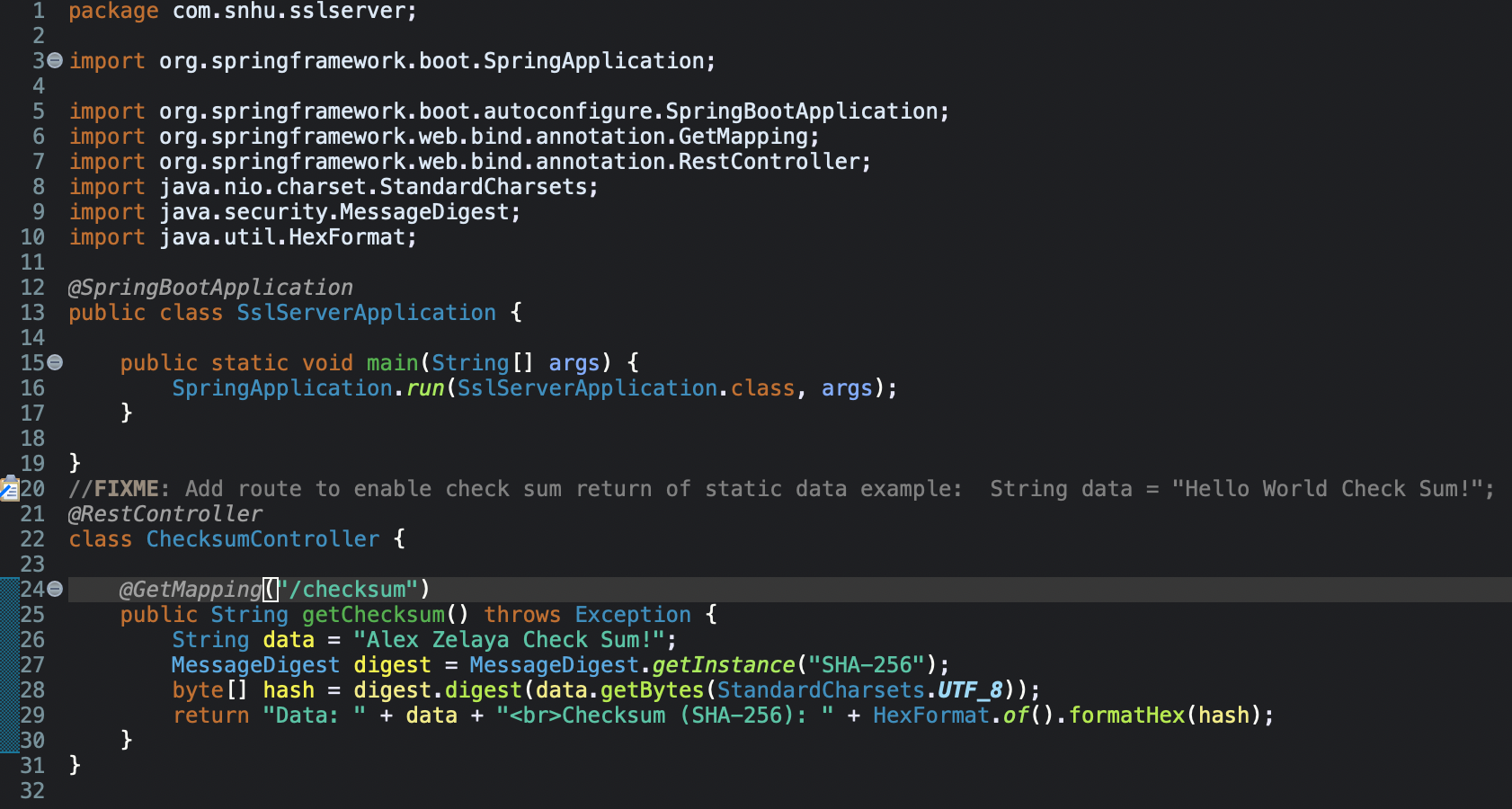
For this project, I used the SHA-256 hashing algorithm to verify data integrity within Artemis Financial’s application. SHA-256 produces a 256-bit hash that’s practically impossible to reverse or duplicate, which keeps client data safe during transmission. It doesn’t require a private key since it’s a one-way function, making it great for checksum validation. I chose SHA-256 because it’s modern, trusted, and still recommended by NIST. It’s used everywhere from SSL certificates to blockchain tech, so it fits right into a financial security context.

## Certificate Generation

## A self-signed certificate was created using the Java Keytool command in the terminal. This certificate allows HTTPS communication between the client and server. The .p12 keystore file was generated and then exported into a .cer file for verification.

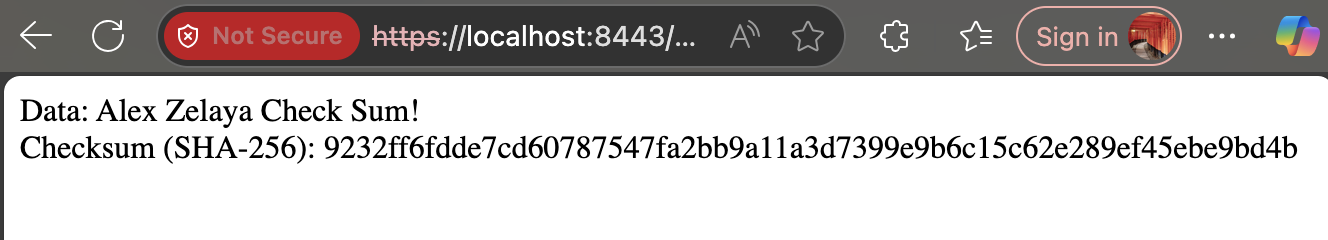


## Deploy Cipher



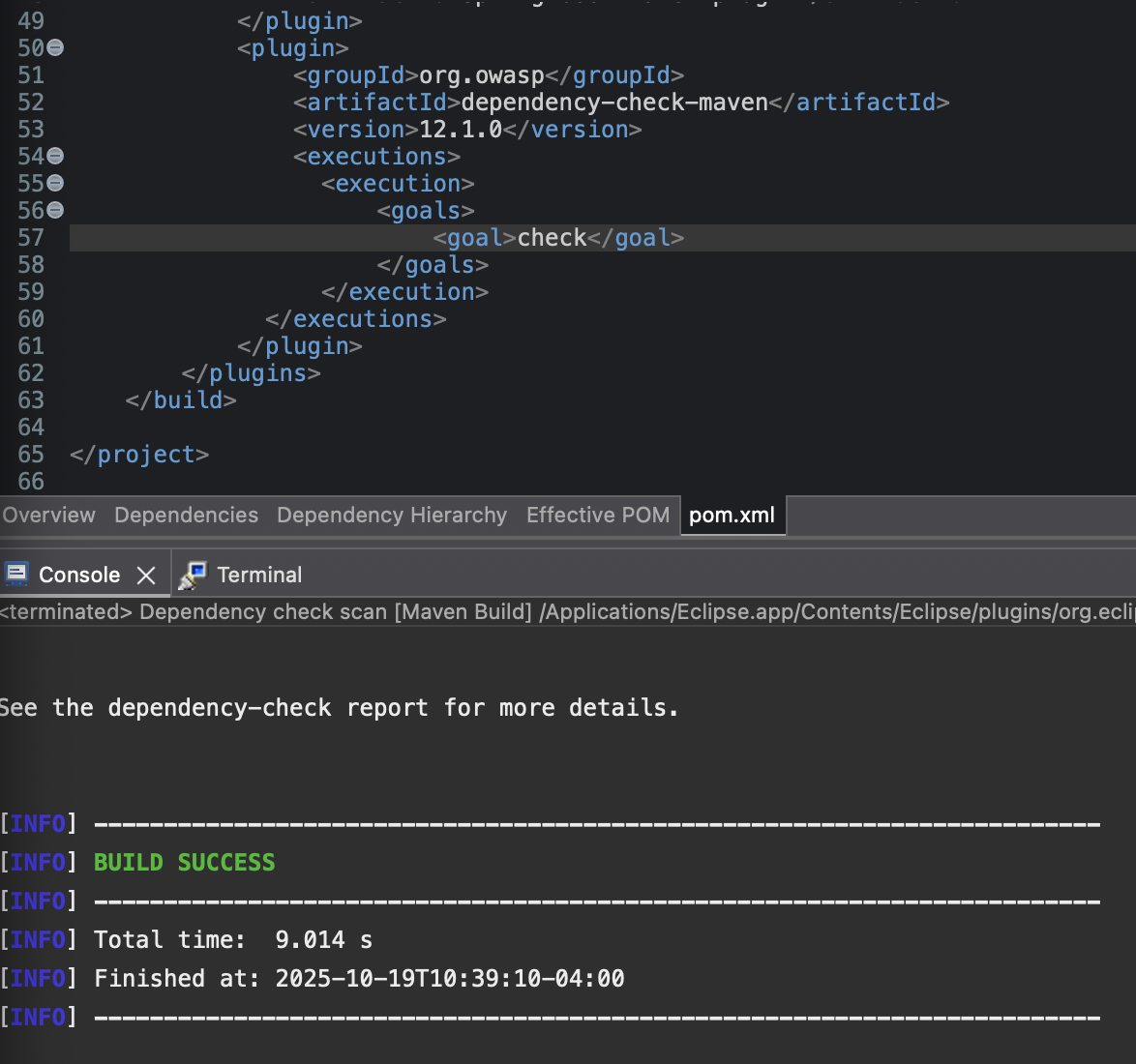
## Secure Communications

The connection shows a lock icon in the browser, confirming that SSL/TLS encryption is active. This ensures all data sent between Artemis Financial and its users is encrypted and protected from interception.



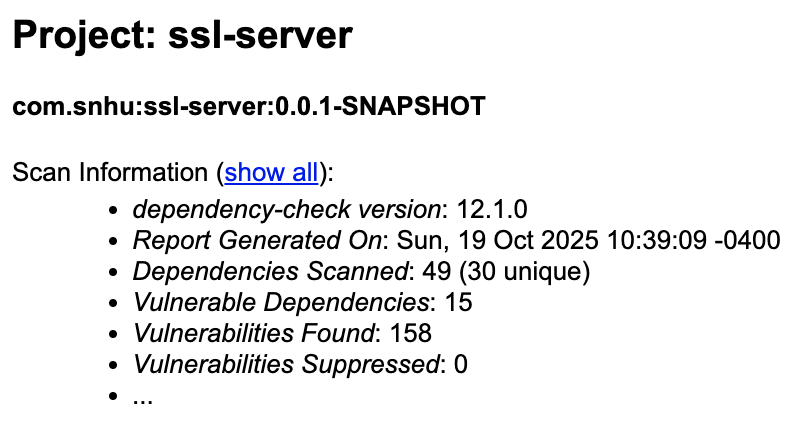
## Secondary Testing

The OWASP Dependency-Check Maven plugin was run to scan all project dependencies. This step validated that the application meets secure-coding standards and doesn’t depend on unsafe libraries.



## Functional Testing

## The refactored code compiled and executed without errors. The Spring Boot server launched successfully, the /checksum route responded correctly, and the browser confirmed encrypted HTTPS communication.



## Summary

I implemented secure communication through HTTPS and added checksum verification using SHA-256. I verified data integrity, encrypted network traffic, and validated dependencies using OWASP testing. These steps show how secure coding practices can be integrated into real-world applications to protect user information and maintain trust.

## Industry Standard Best Practices

I followed OWASP and NIST best practices for secure coding. Sensitive data is always transmitted over HTTPS, cryptographic algorithms are up-to-date, and code scanning tools are used regularly to catch potential risks early. Maintaining these standards helps ensure software stability, client confidence, and long-term protection against new security threats.