****

# 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** | **04/19/2025** | **Ruben Melikyan** |  |

## 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.

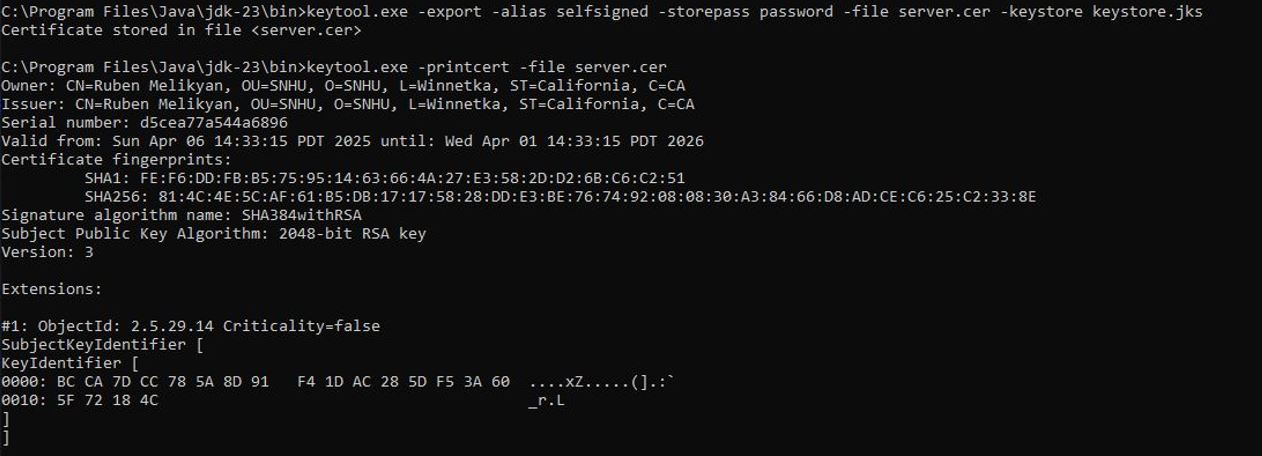
## Developer

Ruben Melikyan

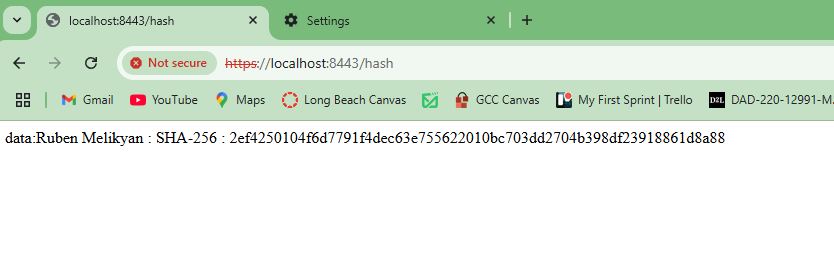
## Algorithm Cipher

Artemis Financial is requesting additional security for their web application to ensure secure communications. Given that the most likely attack vector for a financial institution is a malicious actor attempting to gain unauthorized access to sensitive financial data, encryption is the best line of defense, as it renders intercepted data useless without the correct decryption key. For securing communication, asymmetric encryption is the recommended approach, where the encryption key is public and the decryption key is private. In this context, the SHA-256 cipher algorithm is an ideal choice, offering strong 256-bit encryption that ensures a vast number of possible key combinations and a high level of security. SHA-256 generates a secure, non-reversible checksum using Java’s random number generator, making it highly effective in verifying the integrity of files and messages. The algorithm's hash function creates a fixed-length output (checksum) from input data, ensuring that even a minor change in the original content results in a completely different hash, thereby protecting against tampering. Historically, encryption algorithms have evolved significantly from basic ciphers like Caesar and Vigenère to modern algorithms such as AES, RSA, and SHA-2 family (which includes SHA-256) to keep pace with increasing cybersecurity demands. SHA-256 is currently considered one of the most robust encryption standards available, making it a suitable choice for Artemis Financial’s security needs.

## Certificate Generation



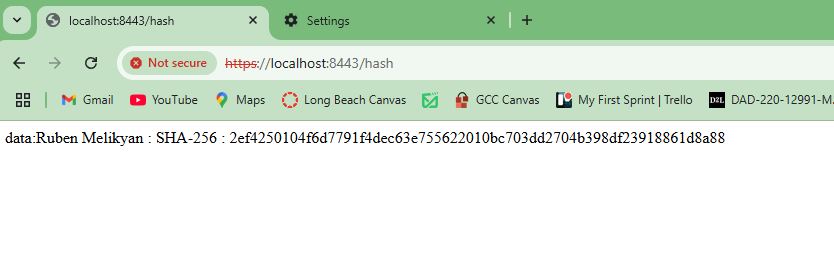
## Deploy Cipher

Insert a screenshot below of the checksum verification.

## Secure Communications

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

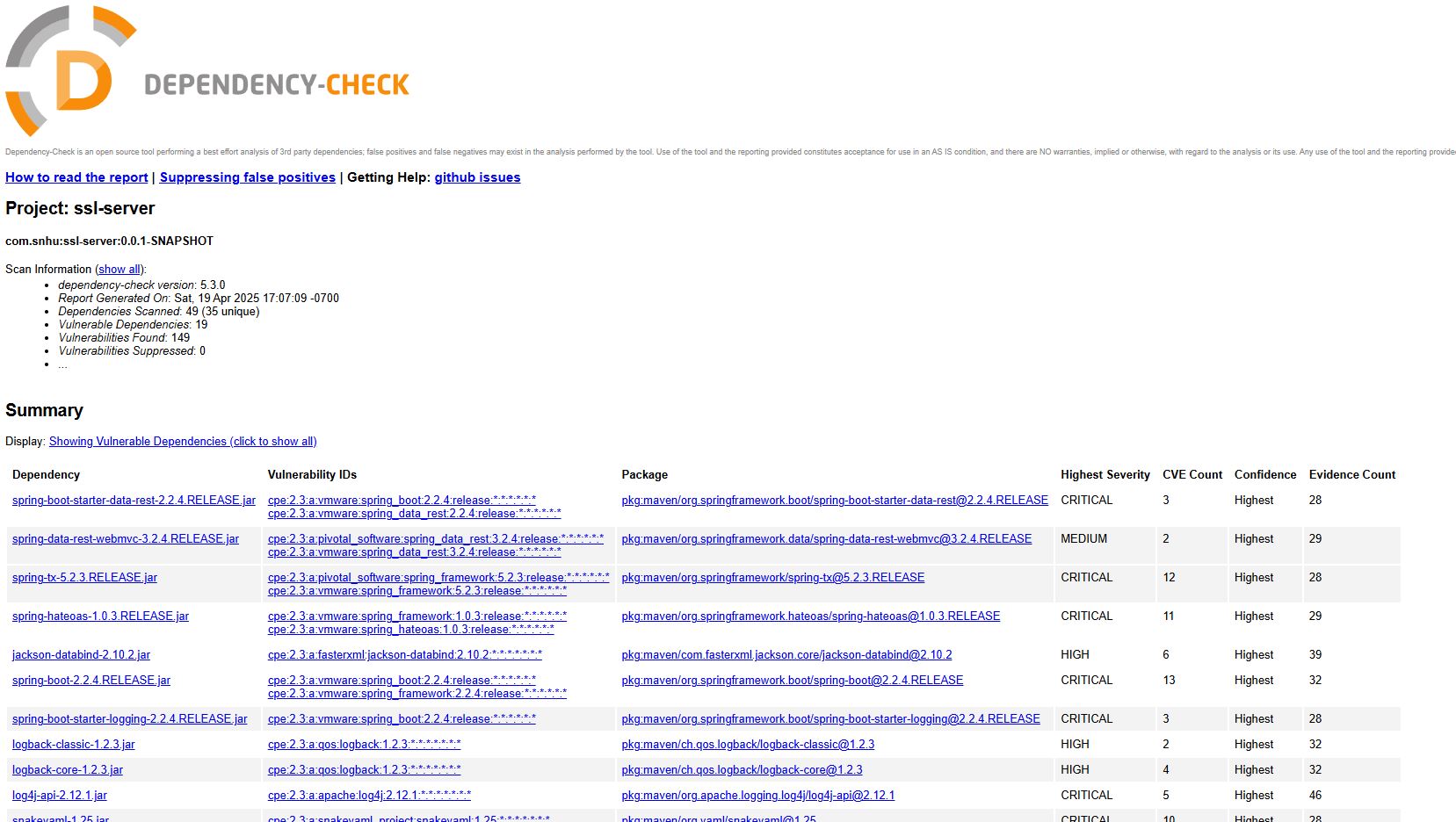
Showing the HTTPS is working but that my Cert isn’t official because it’s self signed.



## Secondary Testing

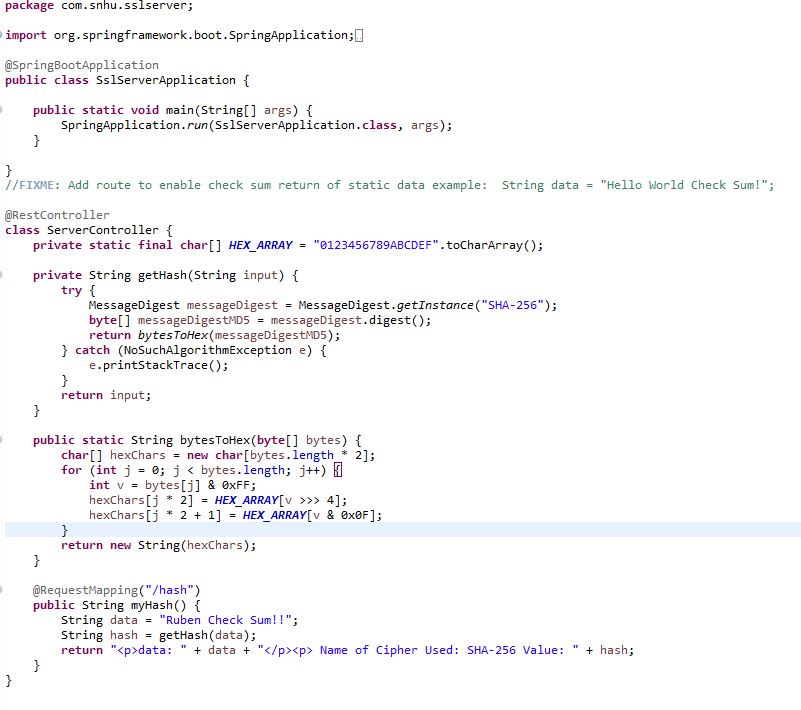
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

Refactoring in my code I have added a secured RestController to work as the secure controller for my programs hash RESTful stop. The ServerController class works to match the problems presented by the vulnerability assessment diagram. I additionally chose to work with the SHA-256 hashing cipher as it’s very secure and runs a very small chance at collisions. To best maintain the current security of the application I would suggest once or twice monthly dependency checks of the application to keep the most up to date on potential vulnerabilities this will help to protect the company and their sensitive data. Keeping the plugins within the pom.xml additionally would do well to keep the latest iterations of the plugins running ensuring the highest security.

## Industry Standard Best Practices

To address known security vulnerabilities and uphold the integrity of the software application, I implemented industry-standard secure coding practices. In addition to the changes outlined in the Summary section, it is crucial to apply key security measures to ensure ongoing protection. These include input validation to guard against threats such as SQL injection, cross-site scripting (XSS), and command injection by properly validating and sanitizing user inputs. Secure authentication and password management should also be enforced through strong password policies, hashing algorithms, and multi-factor authentication (MFA) to strengthen access control. Adhering to the principle of least privilege helps minimize risk by granting users and applications only the permissions necessary for their roles. Sensitive data should be encrypted both at rest and in transit using robust encryption algorithms and secure communication protocols like HTTPS. Additionally, the software and its dependencies must be regularly updated and patched to address vulnerabilities and reduce the risk of exploitation. Proper error handling and secure logging practices should be in place to prevent data leakage and support threat detection. Embracing these secure coding standards not only protects sensitive data but also supports regulatory compliance, reduces costs associated with breaches, builds customer and partner trust, and enhances the organization’s overall reputation.