****

# 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** | **[Date]** | **[Your Name]** |  |

## 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

Trevor Mayo

## Algorithm Cipher

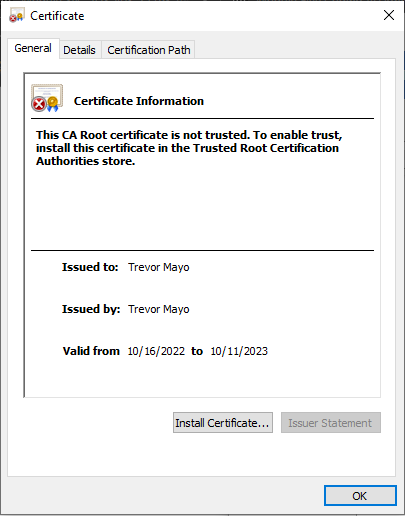
AES-256

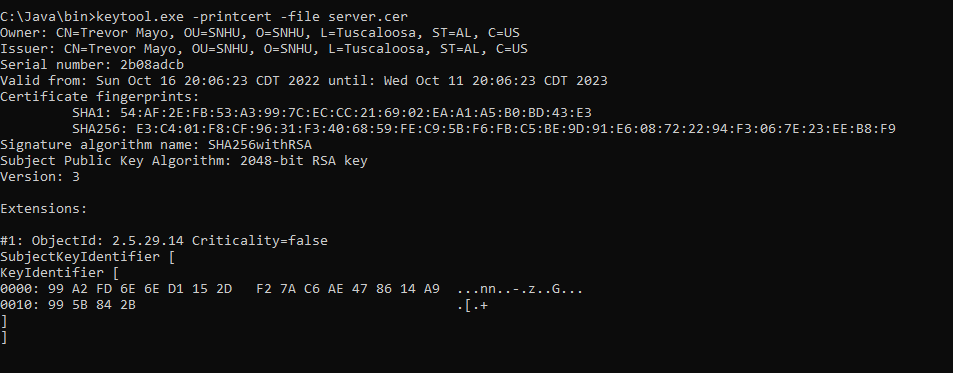
Provide a brief, high-level overview of the encryption algorithm cipher.

I’m going to be using the AES-256 symmetric cipher given the security vulnerabilities of Artemis Financial. AES-256 is a symmetric block cipher that uses one key to encrypt and decrypt data. On a high-level view, the data is split into bit-blocks and reorganized/changed according to the algorithm and key used within the cipher. The blocks are then combined back together to form the final cipher text. Each block goes through the following process before the final cipher text: sub bytes in the matrix for other bytes using the S-Box, shift the rows in the matrix according to algorithm, and then multiply the byte matrix by another corresponding bye matrix. After each block has been adjusted, the output is XORed onto the corresponding round key. Symmetric keys ensure that one key has the power to encrypt and decrypt messages and files, while asymmetric keys also contain a public key that can encrypt messages that are decrypted by the private key. The use of a symmetric key for a checksum ensures that fake files cannot be created to fake identities within the application. Symmetric ciphers are also much faster than asymmetric ciphers as well. Encryption algorithms have evolved since their release because of the strength and quickness of technology today. Encryption started off as a way for people to deliver messages using a simple key leading to easily broken ciphers. Nowadays, encryption is a multi-step process that ensures complete anonymity through sent data.

## Certificate Generation

Insert a screenshot below of the CER file.

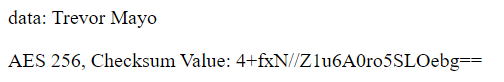




## Deploy Cipher

Insert a screenshot below of the checksum verification.

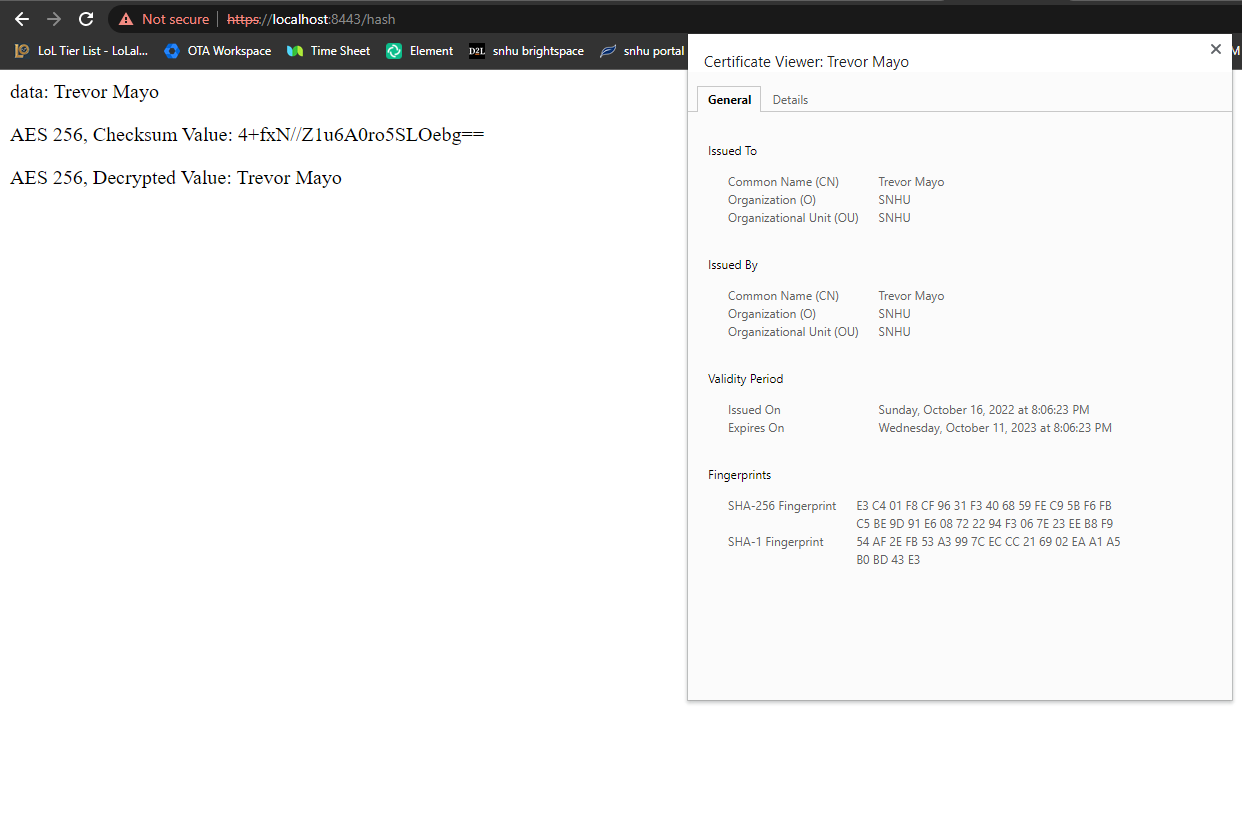
[Insert screenshots here.]



## Secure Communications

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

[Insert screenshots here.]

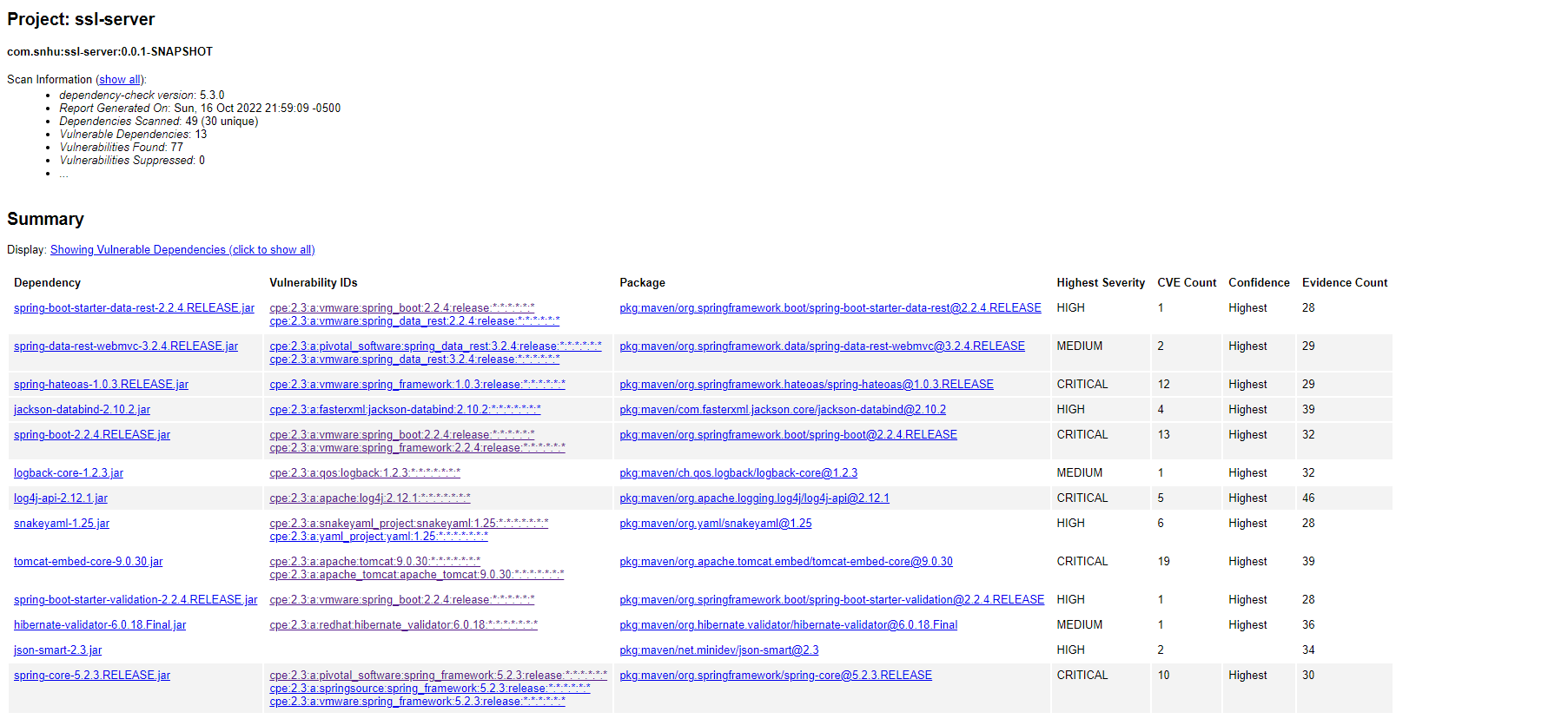


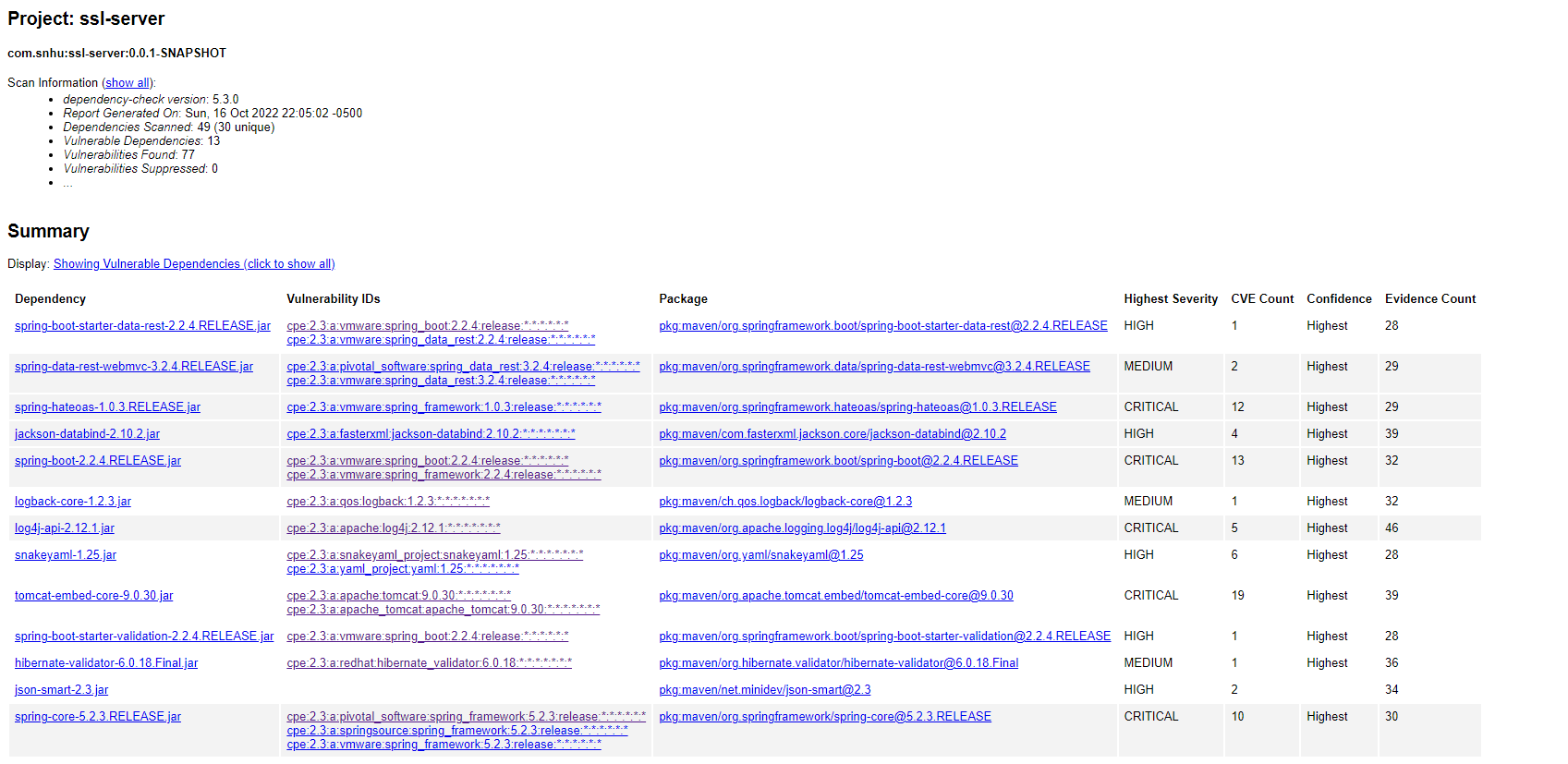
## Secondary Testing

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

[Insert screenshots here.]

Before:

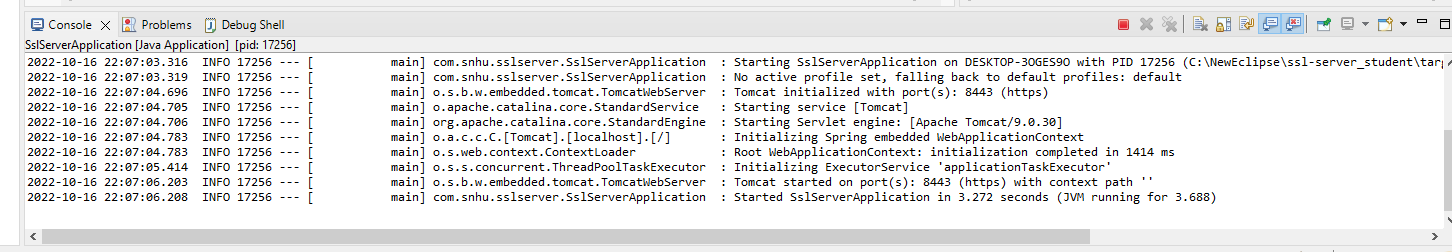
After:



## Functional Testing

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

[Insert screenshots here.]



## Summary

When refactoring the code, I added two get mappings to the springboot application. One uses my name to cipher through AES 256 and encrypt/output the string. After, the function decrypts the encrypted string using the key and outputs that as well. The other function just uses the test string provided within the code base to do the exact same thing. Both of the functions use a try/catch to validate any errors within the AES cipher. I also wrote two encryption/decryption functions for the AES protocol that were provided within another class. There wasn’t any input validation required because I’m not taking in any information that’s provided by a third party. The API’s were all using a secure springboot framework along with the AES cipher. The code was secure with error handling and quality control. The data structures were also encapsulated correctly without any room for errors breaking the application.

I mainly focused on catching errors within the application because of the lack of unsafe information coming in from a third party. I made sure to use quality code throughout the whole process to ensure a readable and efficient codebase as well.

## Industry Standard Best Practices

To ensure the applications continued security, I used OWASP to test for vulnerabilities throughout the coding process. I also made sure to catch and handle errors correctly throughout the entire process to avoid shutting down the system.

Applying industry standard best practices for secure coding ensures that the current and future developers will have an easier time working with and debugging future problems within the application. The company in turn will spend less money fixing problems and dealing with security breaches if security protocols are followed correctly. The process of secure coding within a company also helps improve overall quality within the application which can extend to the perception of the company as a whole.