****

# 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]** | **[Juan Matos]** |  |

## 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

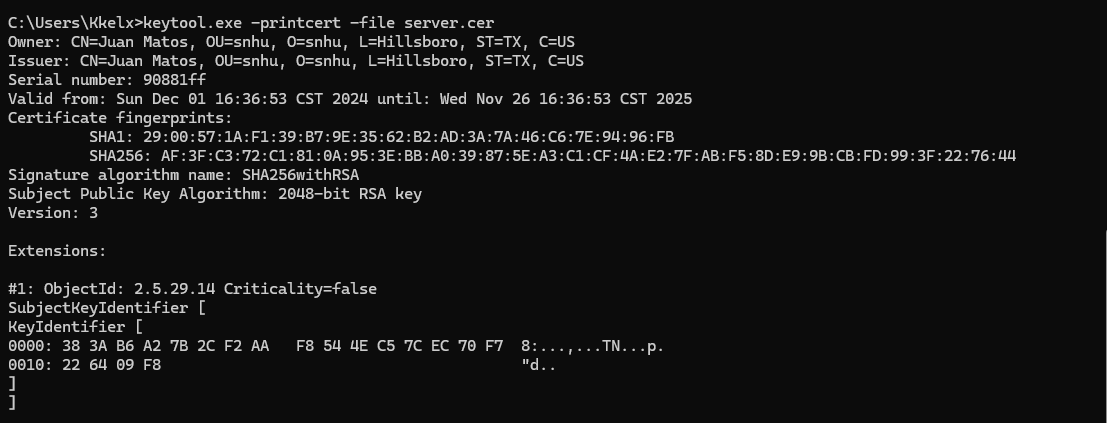
[Juan Matos.]

## Algorithm Cipher

Since financial institutions are often targeted by hackers trying to steal sensitive information, encryption is the best solution. This will make files unreadable to attackers without the correct key. For secure communication, I recommend using Asymmetric encryption, where one key is used to encrypt the data and a different, private key is used to decrypt it. To provide strong security, especially for data sent outside the company, I suggest using the SHA-256 encryption method with 256-bit keys. SHA-256 is a strong encryption system with many possible key combinations, making it highly secure. It also uses Java’s random number generator to create a secure checksum, which ensures the file’s integrity and that it hasn’t been tampered with. The hash function will use SHA-256 to create a checksum for the message.

## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

Insert a screenshot below of the checksum verification.

A computer screen shot of a program code

Description automatically generated

A screenshot of a computer

Description automatically generated

## Secure Communications

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

A screenshot of a computer

Description automatically generated

Since the certificate is self-signed it’s not recognized as secured.

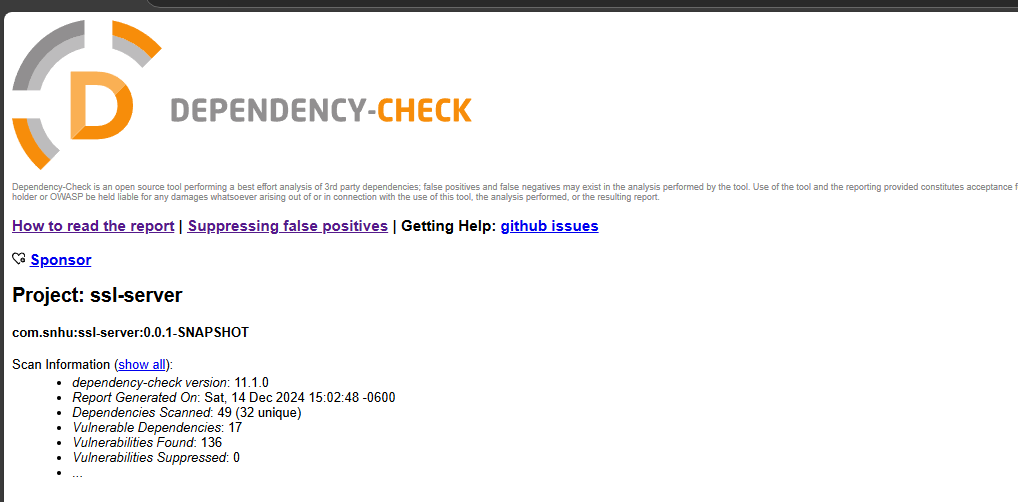
## Secondary Testing

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

A screen shot of a computer program

Description automatically generated

The build was successful, there were no errors.



## Functional Testing

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

A screen shot of a computer program

Description automatically generated

I also refactored my pom file and upgraded the spring boot version and the java version and was able to bring down the vulnerabilities form 17 to 12.

A screenshot of a computer

Description automatically generated

## Summary

The refactored code improves security by using constants for sensitive data, such as the hashing algorithm (SHA-256) and the data string ("Hello I’m Juan :D"). By defining these as constants, the risk of unintended changes or errors is reduced, and managing the algorithm securely across the code becomes easier. Additionally, the logic for generating the hash and formatting the response has been separated into distinct methods, promoting clarity and maintainability while making it less likely that vulnerabilities will be introduced. Finally, by isolating the hashing logic and centralizing the string formatting, the exposure of sensitive data is minimized, making the application more secure. These changes result in more modular, maintainable, and secure code by reducing repetition, avoiding hardcoded values, and improving the organization of sensitive operations.

## Industry Standard Best Practices

I applied industry-standard best practices by using constants for sensitive data like the hashing algorithm and data string, reducing errors and minimizing the risk of changes. Separating the hashing logic and response formatting improves readability and reduces the chances of introducing vulnerabilities. These practices follow secure coding guidelines, enhancing the software’s security and maintainability.

Applying these best practices helps the company by reducing security risks and making the code more reliable. This ensures the software is secure, easier to maintain, and adaptable to future changes, contributing to the company's long-term success.