****

# 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** | **08/15/2024** | **Richard Wellman** | **N/A** |

## 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

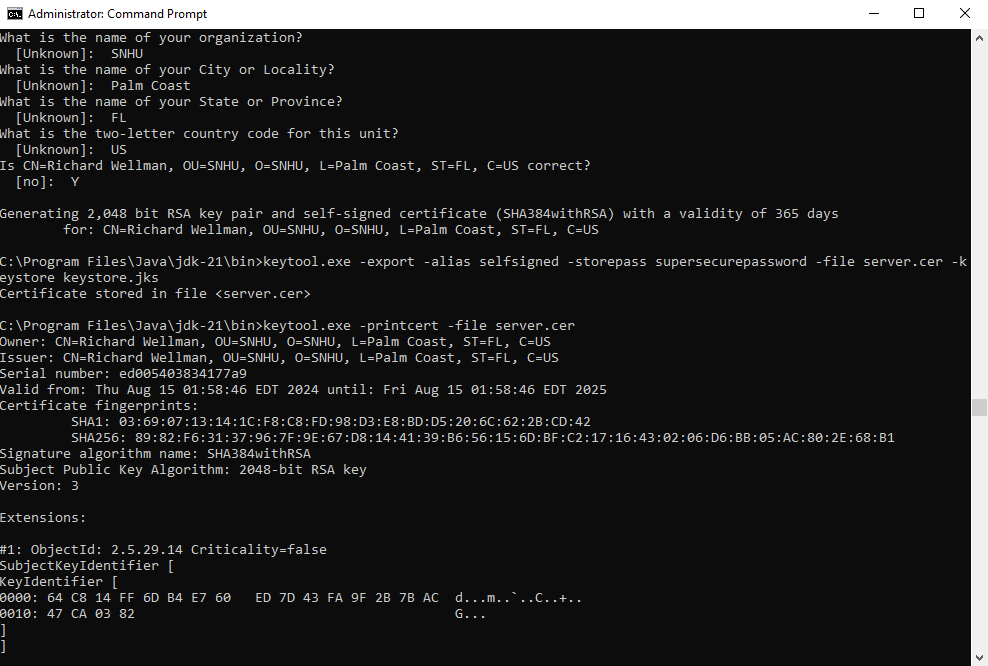
Richard Wellman

## Algorithm Cipher

SHA-256 with RSA key encryption is what I would recommend for Artemis Financial, the use of a SHA-256 hashing algorithm reduces the possibility of collisions. RSA is asymmetric, where the sender and receiver have different public and private keys and is used for sending and receiving signed public and private key pairs, which helps to add another layer of security.

## Certificate Generation

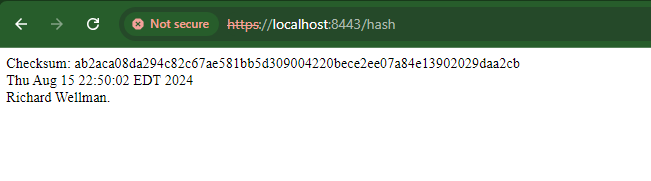
Insert a screenshot below of the CER file.



## Deploy Cipher

Insert a screenshot below of the checksum verification.

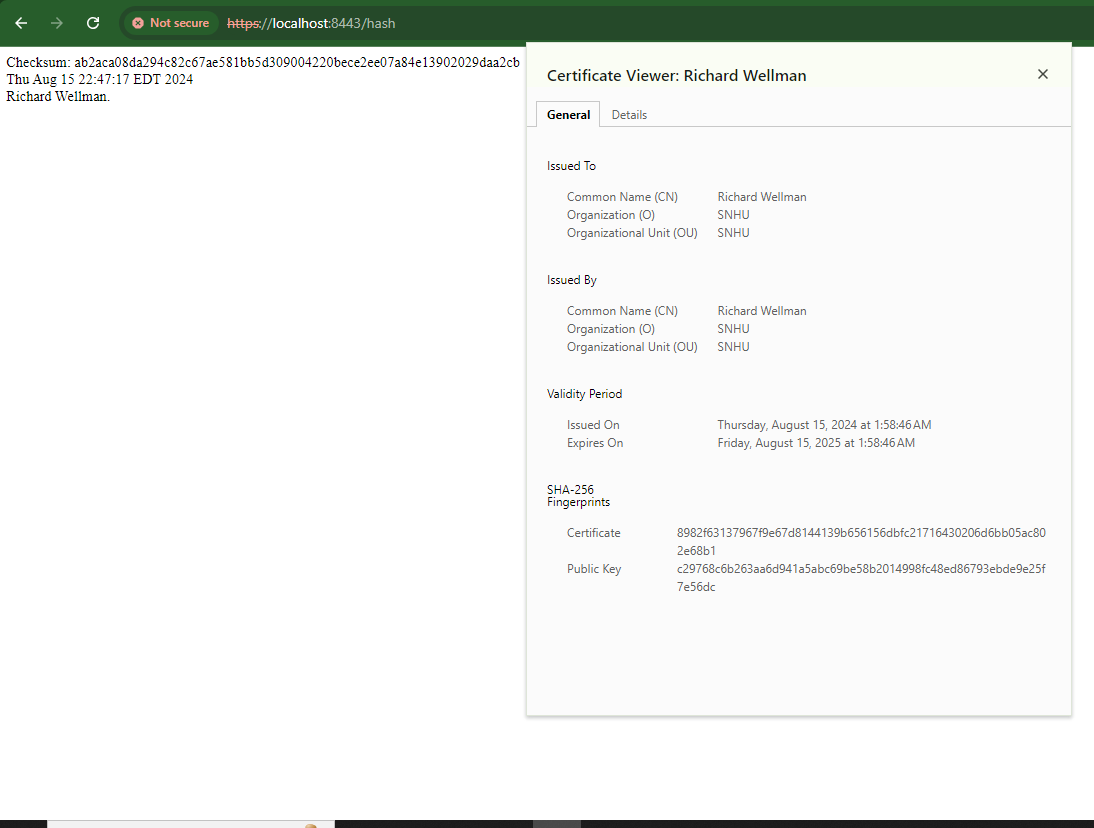
The screenshot below shows the checksum hash, the date, and my name.



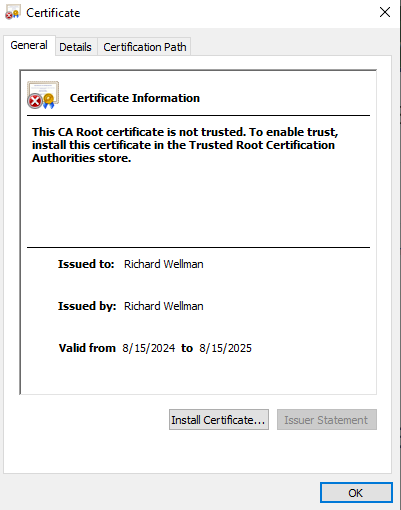
## Secure Communications

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

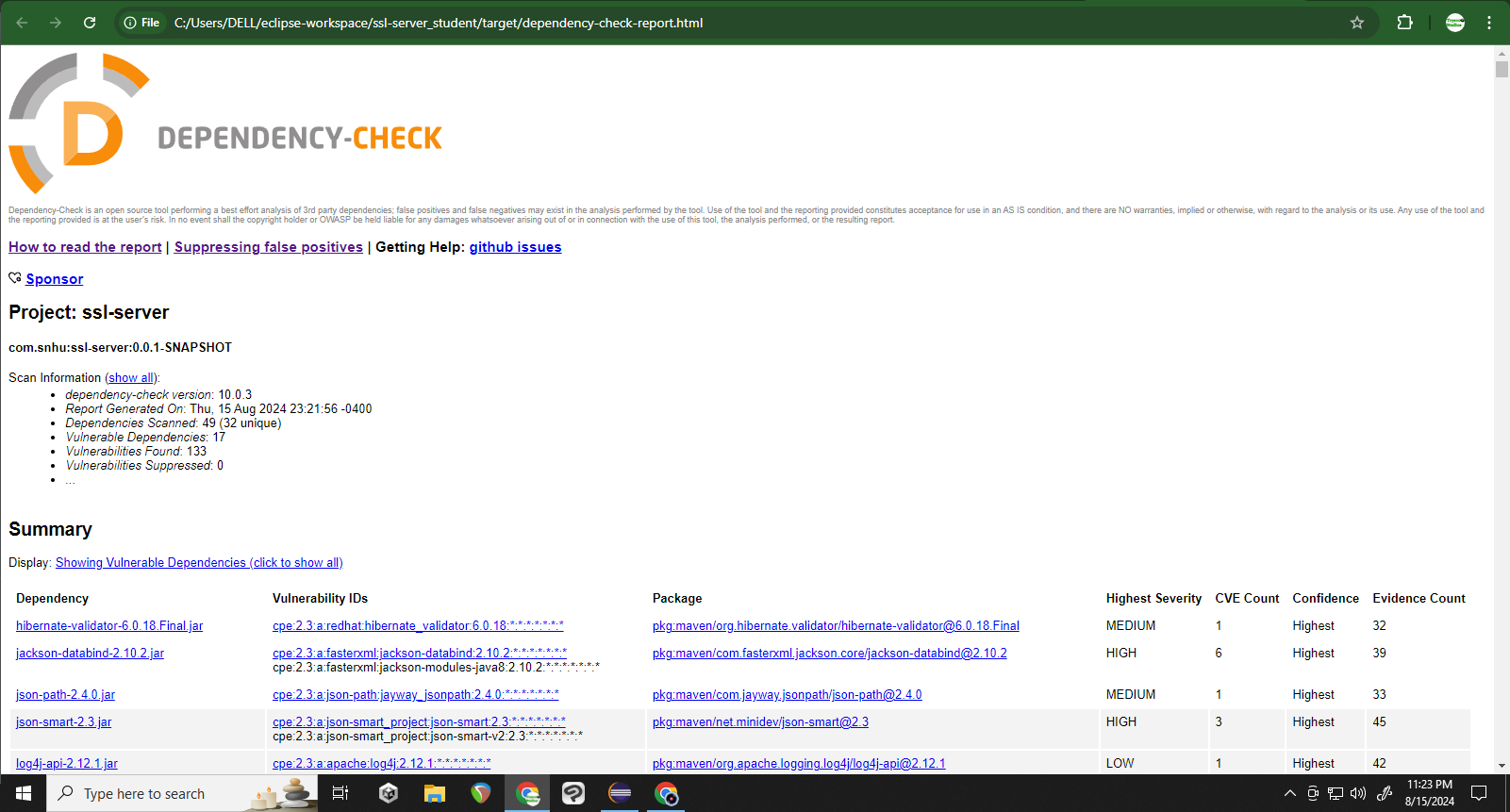
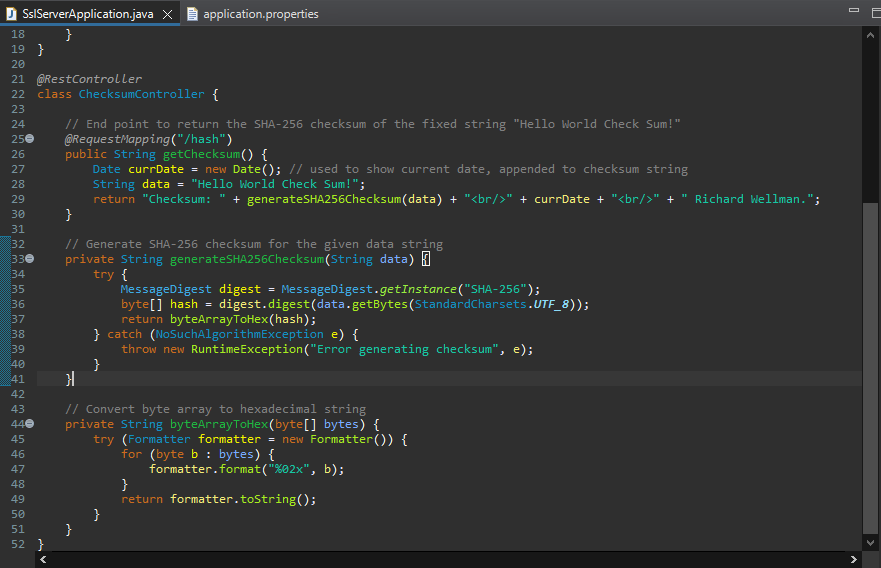
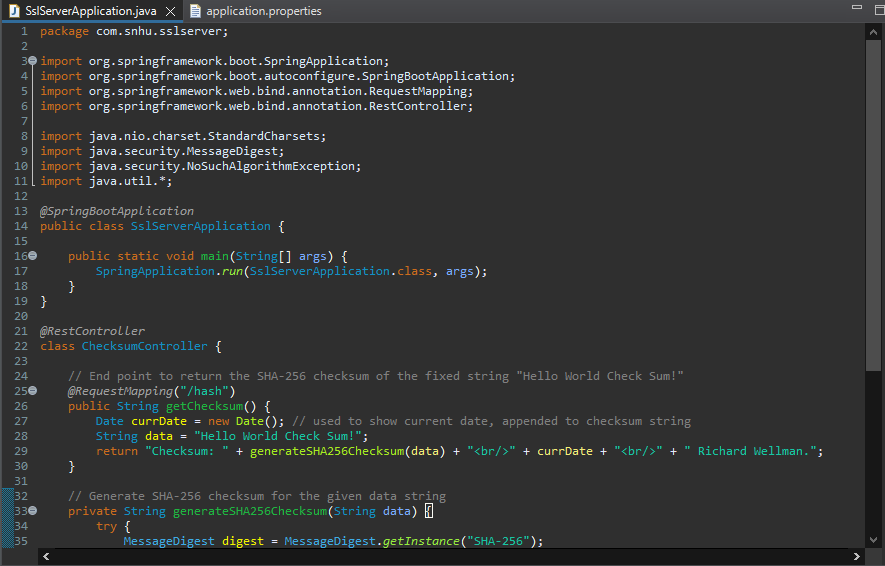
The screenshot below shows the self-signed certificate as it appears in the browser. It shows not secure because it is a self-signed certificate and is of course not verified.



For clarity, here is the server certificate generated on my PC in the Java folder.

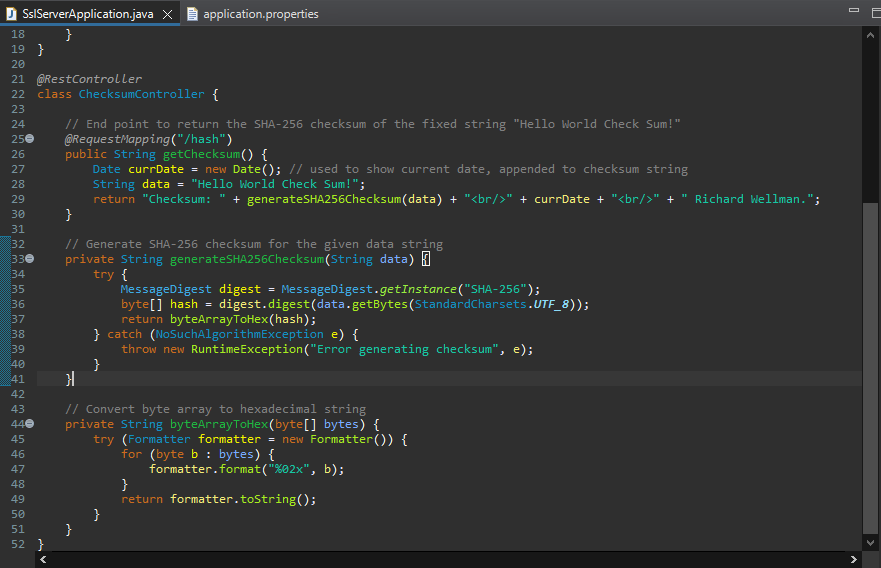
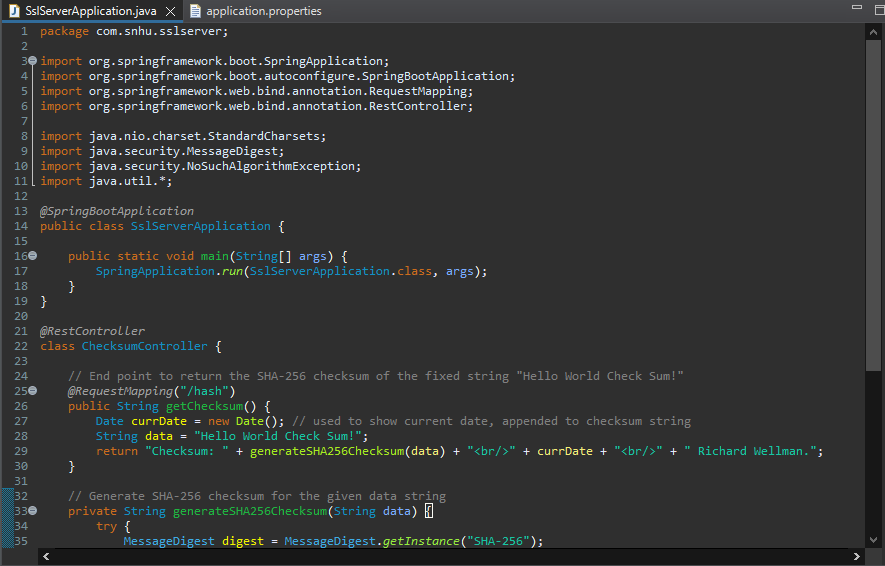


## Secondary Testing

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

The app does not accept input. SHA-256 was used to create a hash for the checksum, which has a low probability of collision and so is secure. The certificate was created with SHA-256 and RSA as well and is encrypted. I kept classes private by default and only made public those classes that absolutely needed it. For instance, the generateSHA256Checksum is set to private, so how it works is unknown to anyone not directly viewing the code. There is a try catch used to ensure errors are handled if the generate checksum function fails to work. Forward facing public trust can be reinforced with constant updates, training, and security checks. It only takes one breach for a company to lose trust in its customers and employees.

## Industry Standard Best Practices

I made sure to follow code modularity so functions can be reused as needed and it also keeps what needs to be kept private, private, and only allows public accessibility - in terms of classes - as needed. Maintaining code modularity as the app grows is going to help with security. Regular security testing, keeping dependencies up to date, and constantly keeping staff up to date on the latest vulnerabilities and threats will help with security. Making sure to write only code that is necessary, commenting clearly, and thinking of future proof code, such as not leaving in commented out code that could lead to a security breach or issue.

**References**

Manico, J., & Detlefsen, A. (2014). Iron-Clad Java. McGraw Hill Computing.