****

# 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** | **2/23/2025** | **Jared Bickler** |  |

## 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

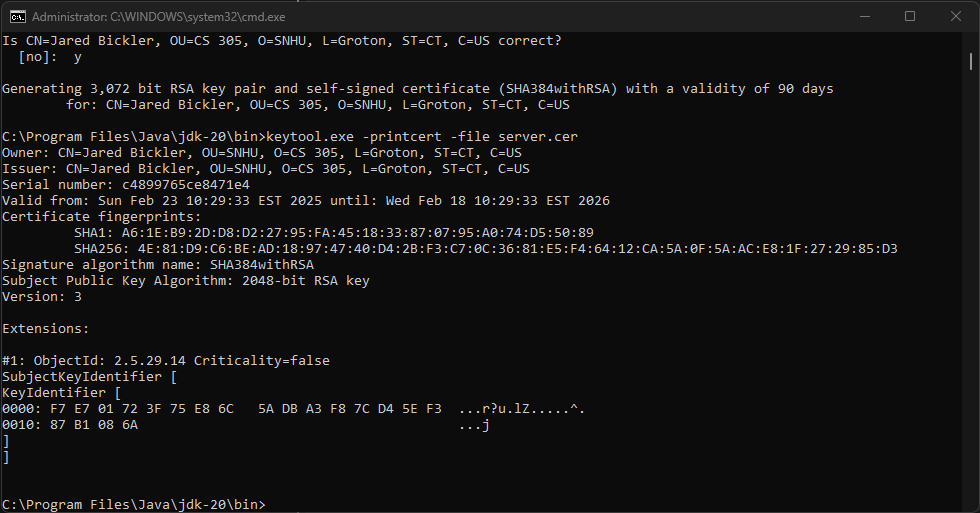
Jared Bickler

## Algorithm Cipher

The most likely threat to a financial institution involves malicious actors attempting to access sensitive information for financial gain, encryption is the best solution. This would render any intercepted data useless without the proper decryption key. Since the firm aims to secure communications, I recommend using an asymmetric encryption algorithm, where a public key is used for encryption, and a private key for decryption. I suggest SHA-256 as a cipher algorithm with 256-bit keys. SHA-256 provides robust encryption due to its extensive range of possible key combinations. Additionally, the algorithm leverages Java’s random number generator, enhancing security by generating a non-reversible checksum that verifies file integrity.

## Certificate Generation

Insert a screenshot below of the CER file.



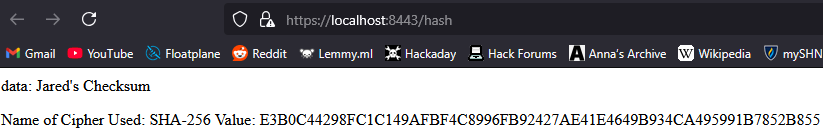
## Deploy Cipher

Insert a screenshot below of the checksum verification.



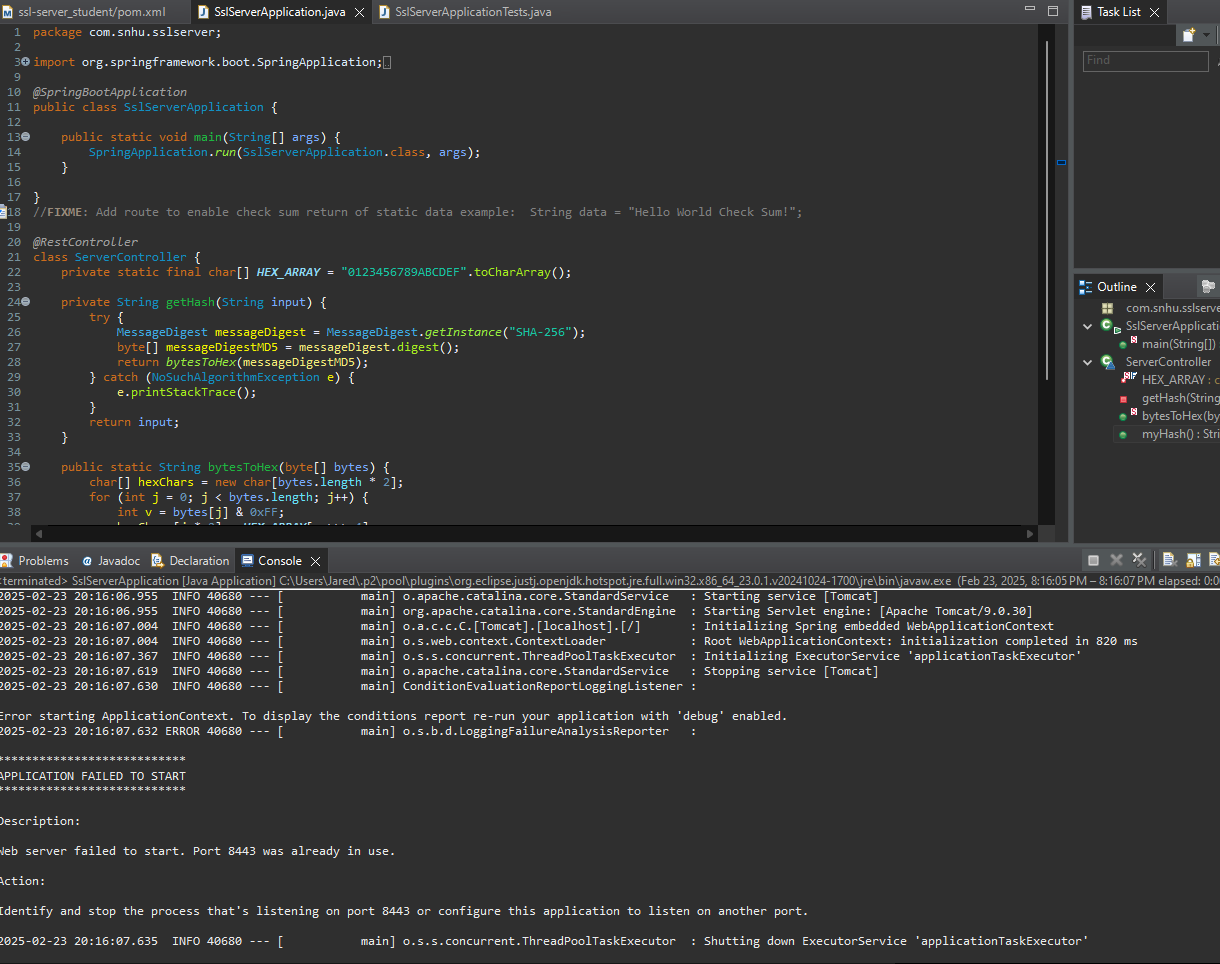
## Secure Communications

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

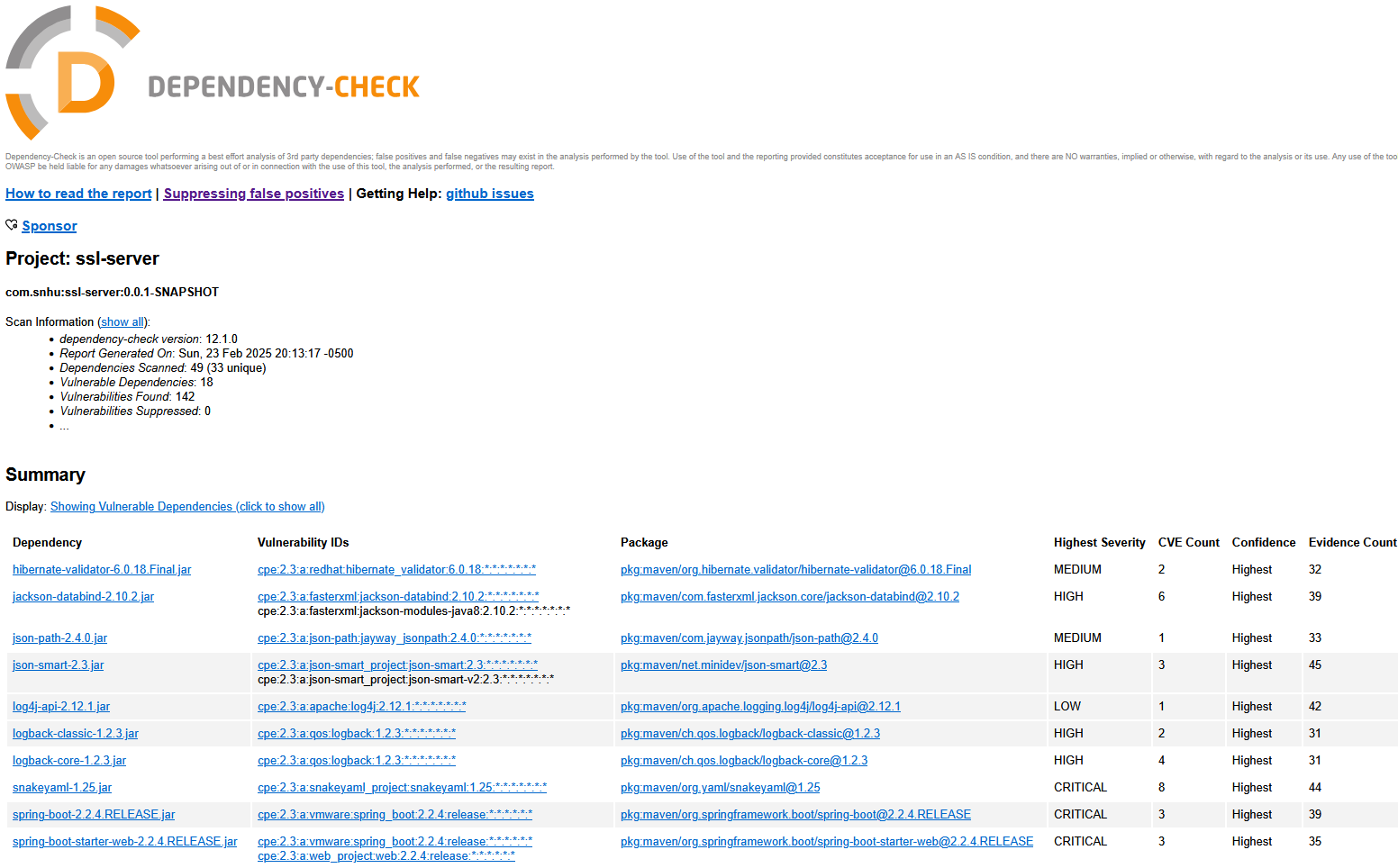


## Secondary Testing

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



(Executed without errors; had to run again to get the console to display and it was already running)



## Summary

While refactoring my code, I added a secured RestController to serve as the secure endpoint for my program’s hash-based RESTful operations. The ServerController class is designed to address the issues identified in the vulnerability assessment diagram. I opted to use the SHA-256 hashing algorithm due to its strong security and minimal risk of collisions. To maintain the application’s security, I recommend conducting dependency checks once or twice a month to stay updated on potential vulnerabilities, helping to safeguard the company and its sensitive data. Additionally, keeping the plugins within the pom.xml file up to date will ensure the latest versions are used, enhancing overall security.