****

# 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** | **4/20/2025** | **Sabrina Ozburn** |  |

## 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

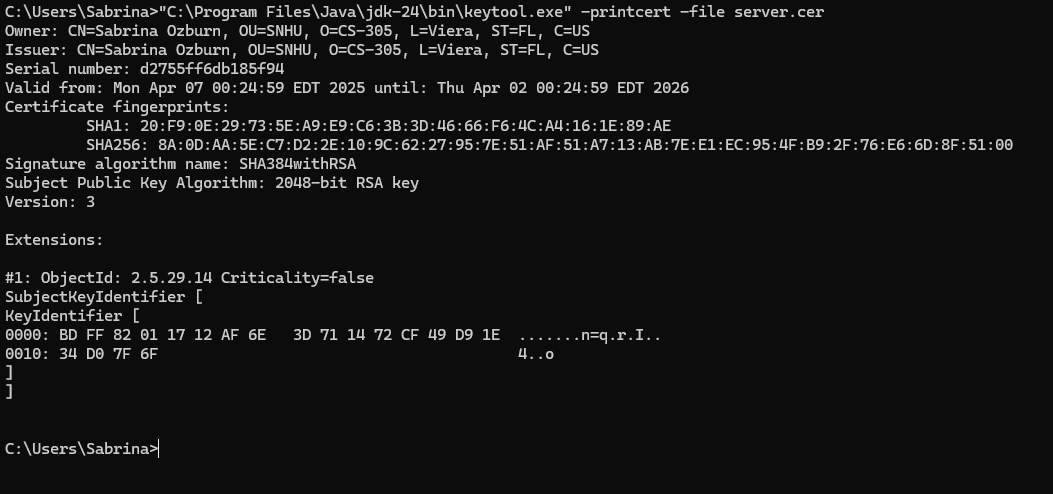
Sabrina Ozburn

## Algorithm Cipher

Artemis Financial is seeking an additional layer of security for their web application to safeguard communications. Given that the most probable threat to a financial institution is a malicious actor aiming to profit by accessing sensitive data, encryption would be the optimal solution. This would render the files useless to any potential attacker without the proper decryption key. Since the goal is to secure communication, I would recommend employing asymmetric encryption, where the encryption key is public, and the decryption key is private. To ensure the highest level of security, particularly for external transmissions, I suggest using the SHA-256 cipher algorithm with 256-bit keys for encryption. SHA-256 provides strong encryption with a vast number of potential key combinations due to its 256-bit key length. Moreover, SHA-256 relies on Java’s random number generator, ensuring robust security by creating a non-reversible checksum to verify the integrity of the file. The hash function will utilize the SHA-256 algorithm to generate a checksum for the given message.

## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

Insert a screenshot below of the checksum verification.

I am unable to get the localhost to work.

## Secure Communications

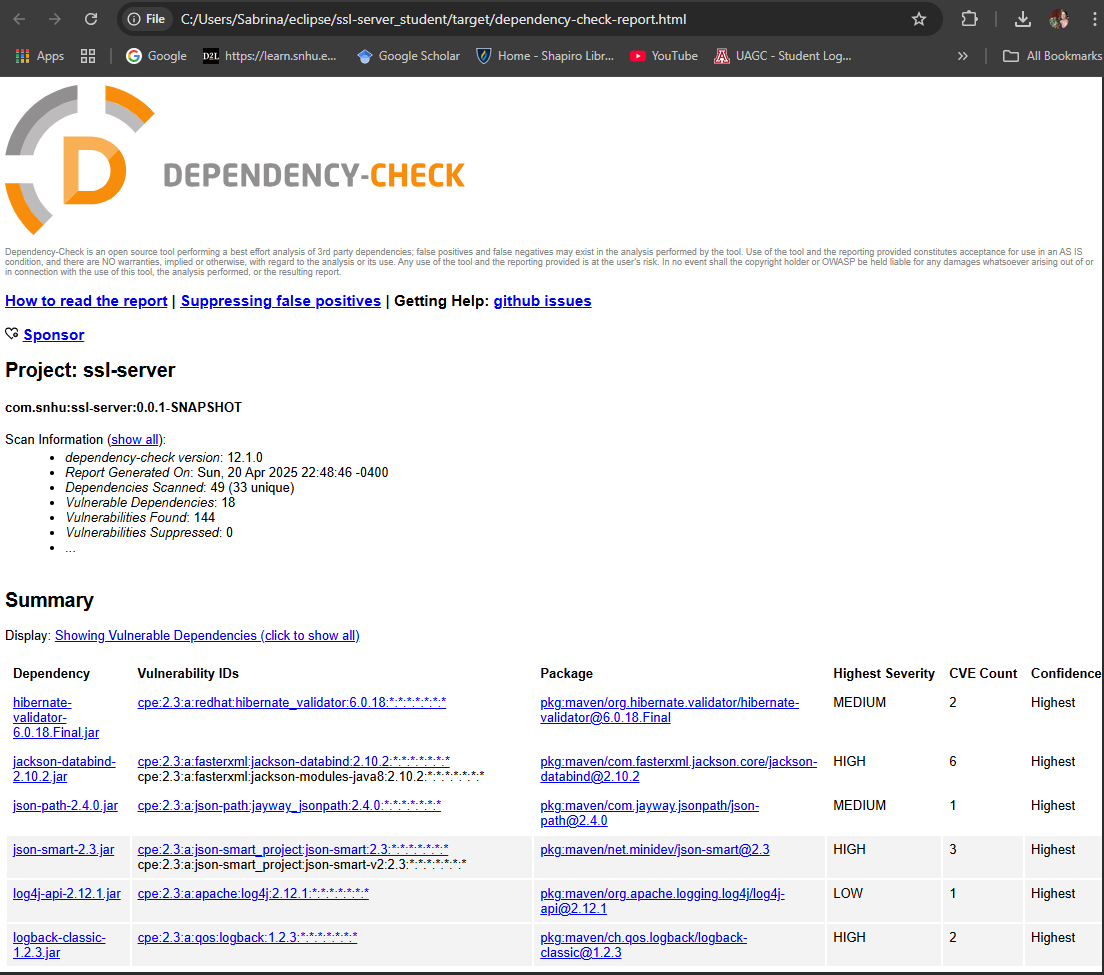
Insert a screenshot below of the web browser that shows a secure webpage.

I am unable to get the localhost to work

## Secondary Testing

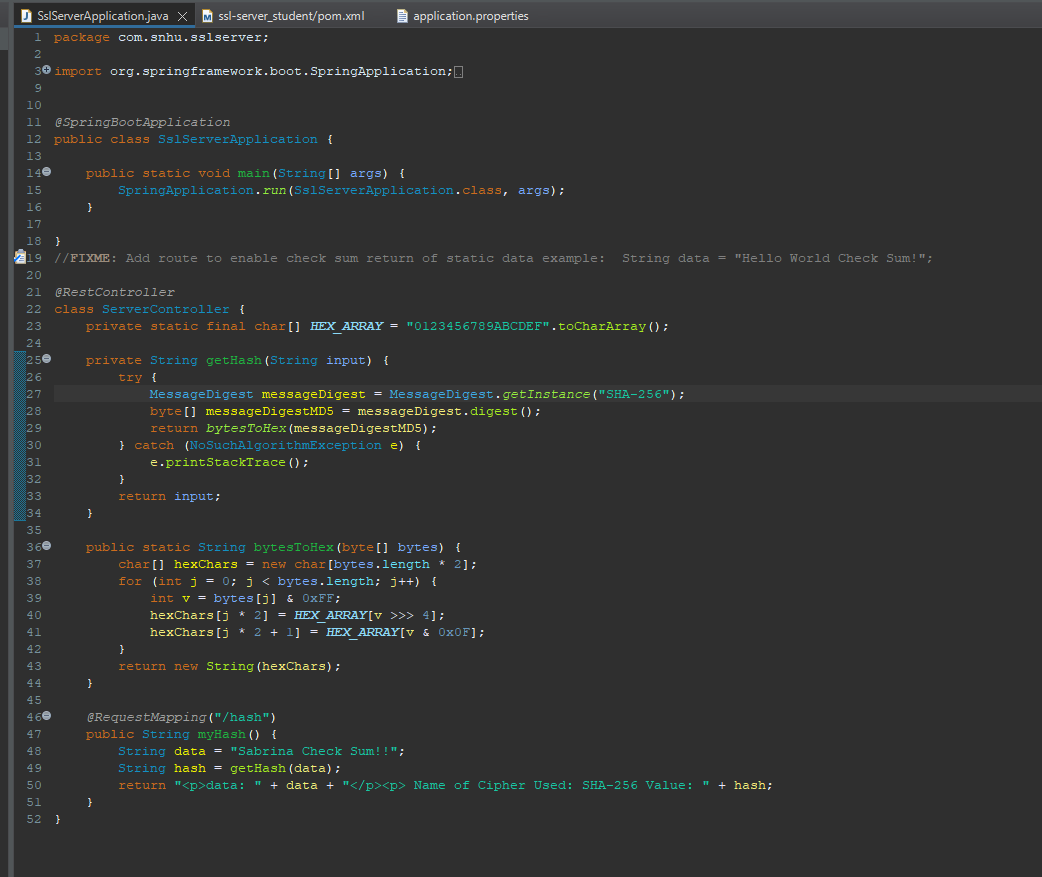
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

In refactoring my code, I’ve implemented a secure RestController to serve as the dedicated secure controller for the program's hash RESTful endpoint. The ServerController class addresses the issues identified in the vulnerability assessment diagram. I also selected the SHA-256 hashing algorithm because of its strong security and minimal risk of collisions. To maintain the application's current security posture, I recommend performing dependency checks on the application once or twice a month to stay up to date with potential vulnerabilities, which will help protect both the company and its sensitive data. Additionally, keeping the plugins in the pom.xml file updated will ensure that the latest versions of the plugins are used, providing the highest level of security.

## Industry Standard Best Practices

To mitigate known security vulnerabilities and maintain the software application's security, I applied several industry-standard best practices for secure coding. I implemented encryption, specifically using the SHA-256 hashing algorithm, to ensure the integrity and confidentiality of sensitive data. This algorithm is highly secure and minimizes the risk of collisions, making it a reliable choice for safeguarding critical information. Additionally, I employed asymmetric encryption for secure communication, where the encryption key is public and the decryption key is private, preventing unauthorized access. I also established a routine of regular dependency checks to stay up to date on potential vulnerabilities in third-party libraries, addressing them proactively to avoid security risks. Another key practice was secure configuration management, which involved ensuring proper management of security settings, such as secure RESTful endpoints and authentication mechanisms, to prevent misconfigurations that could lead to exploitation. Applying these best practices offers significant value to the company’s overall well-being. By protecting sensitive data and minimizing risks, these measures help preserve the company’s reputation and foster customer trust. Regular vulnerability assessments and updates reduce the likelihood of attacks, ensuring that the application remains resilient against emerging threats. Ultimately, adopting secure coding practices strengthens the company’s security posture, preventing costly data breaches and maintaining compliance with industry standards, all of which contribute to long-term operational success and business continuity.