****

# 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 6](#_Toc102040760)

[4. Secure Communications 6](#_Toc102040761)

[5. Secondary Testing 7](#_Toc102040762)

[6. Functional Testing 9](#_Toc102040763)

[7. Summary 10](#_Toc102040764)

[8. Industry Standard Best Practices 10](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2/23/25** | **Jennifer Swinton** |  |

## 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

Jennifer Swinton

## Algorithm Cipher

*Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:*

* *Provide a brief, high-level overview of the encryption algorithm cipher.*
* *Discuss the hash functions and bit levels of the cipher.*
* *Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.*
* *Describe the history and current state of encryption algorithms.*

Artemis Financial will require an algorithm that can be used to encrypt archive files long-term. The attacks will most likely be aimed at stealing personal financial information so we would need to utilize a method that will ensure that even if the data is stolen it will be unuseable.

The recommendation for this would be a SHA-256 hashing algorithm Since we are not addressing a file transfer, but encrypting stored data, there is no need for asymmetric keys. Eliminating the need to protect during transfer, we have a need to address only the data itself. SHA-256 is widely used and highly secure, providing a high level of encryption without severely impacting functionality. It generates a unique 256-bit checksum that verifies the authenticity of the file.

SHA-256 was developed by the National Security Agency (NSA) and first published in 2001. It uses random number generation to create checksums that validate the integrity of the data. This hash function is resistant to various forms of cryptographic attacks, including collision and preimage attacks. Currently, SHA-256 is the leading choice for secure hashing due to its proven reliability and security. (*The NSA … Algorithm*, n.d.)

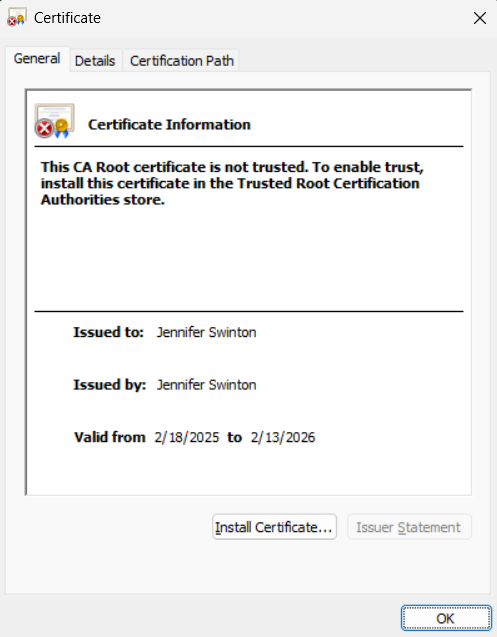
## Certificate Generation

Insert a screenshot below of the CER file.

*Generate appropriate self-signed certificates using the Java Keytool in Eclipse.*

1. *Complete the following steps to demonstrate that the certificate was correctly generated:*
   1. *Export your certificates as a CER file.*
   2. *Submit a screenshot of the CER file in your practices for secure software report.*



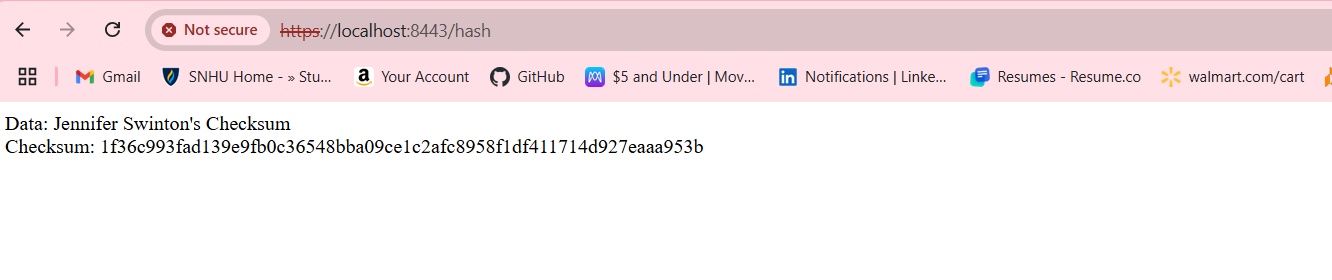


## Deploy Cipher

Insert a screenshot below of the checksum verification.

*Deploy and implement the cryptographic hash algorithm by refactoring code. Demonstrate functionality with a checksum verification.*

1. *Submit a screenshot of the checksum verification in your practices for secure software report. The screenshot must show your name and a unique data string that has been created.*

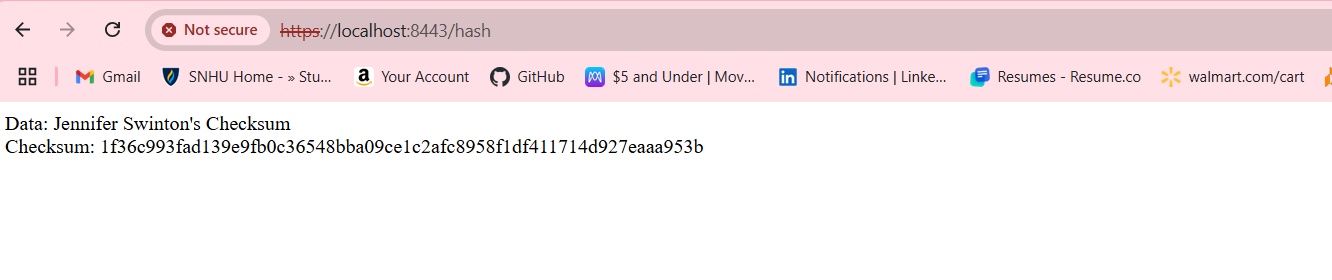


## Secure Communications

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

*Verify secure communication. In the application properties file, refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code. Once the server is running, type “*[*https://localhost:8443/hash*](https://localhost:8443/hash)*” in a new browser to demonstrate that the secure communication works.*

1. *Create a screenshot of the web browser that shows a secure webpage and include it in your practices for secure software report.*

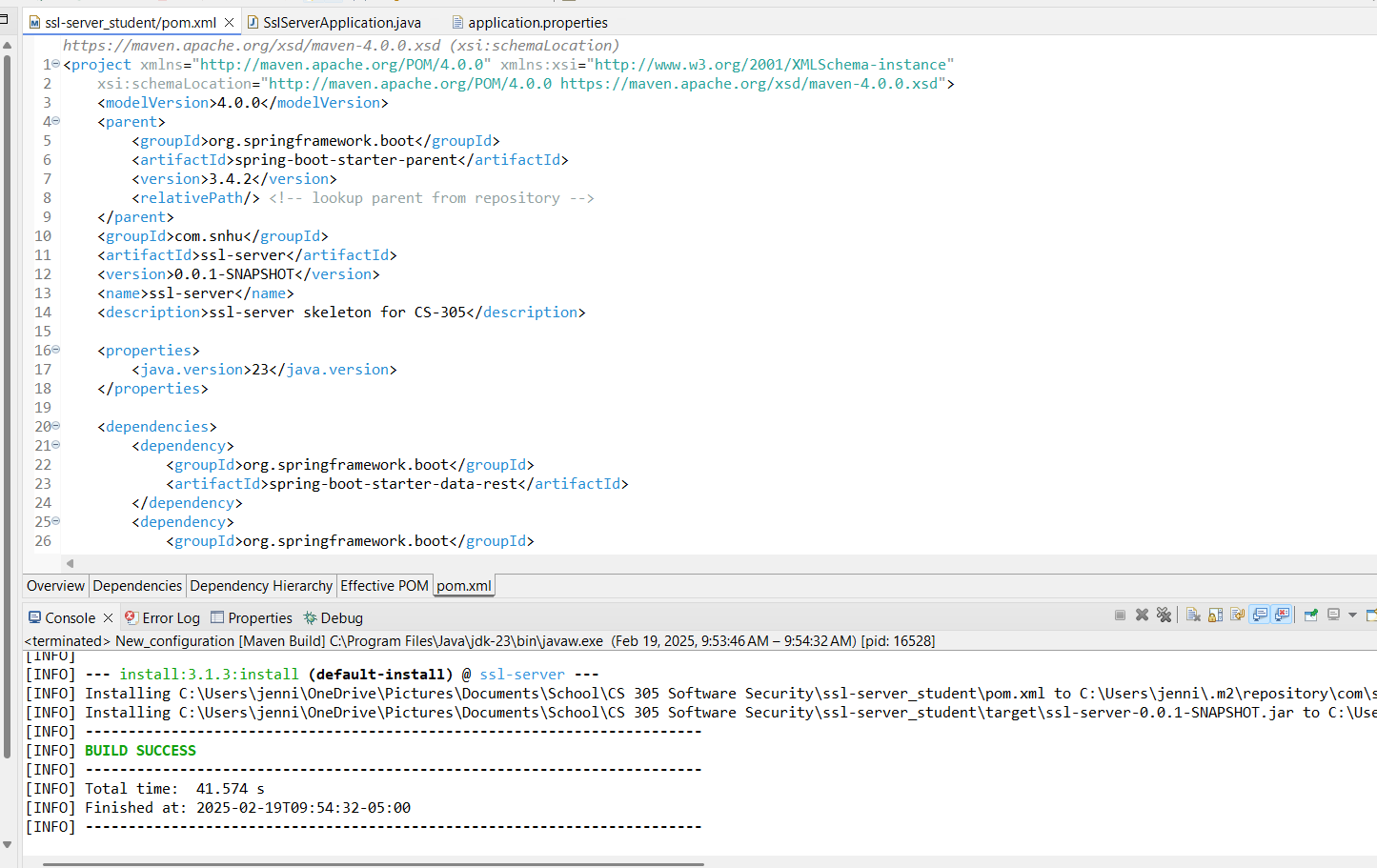


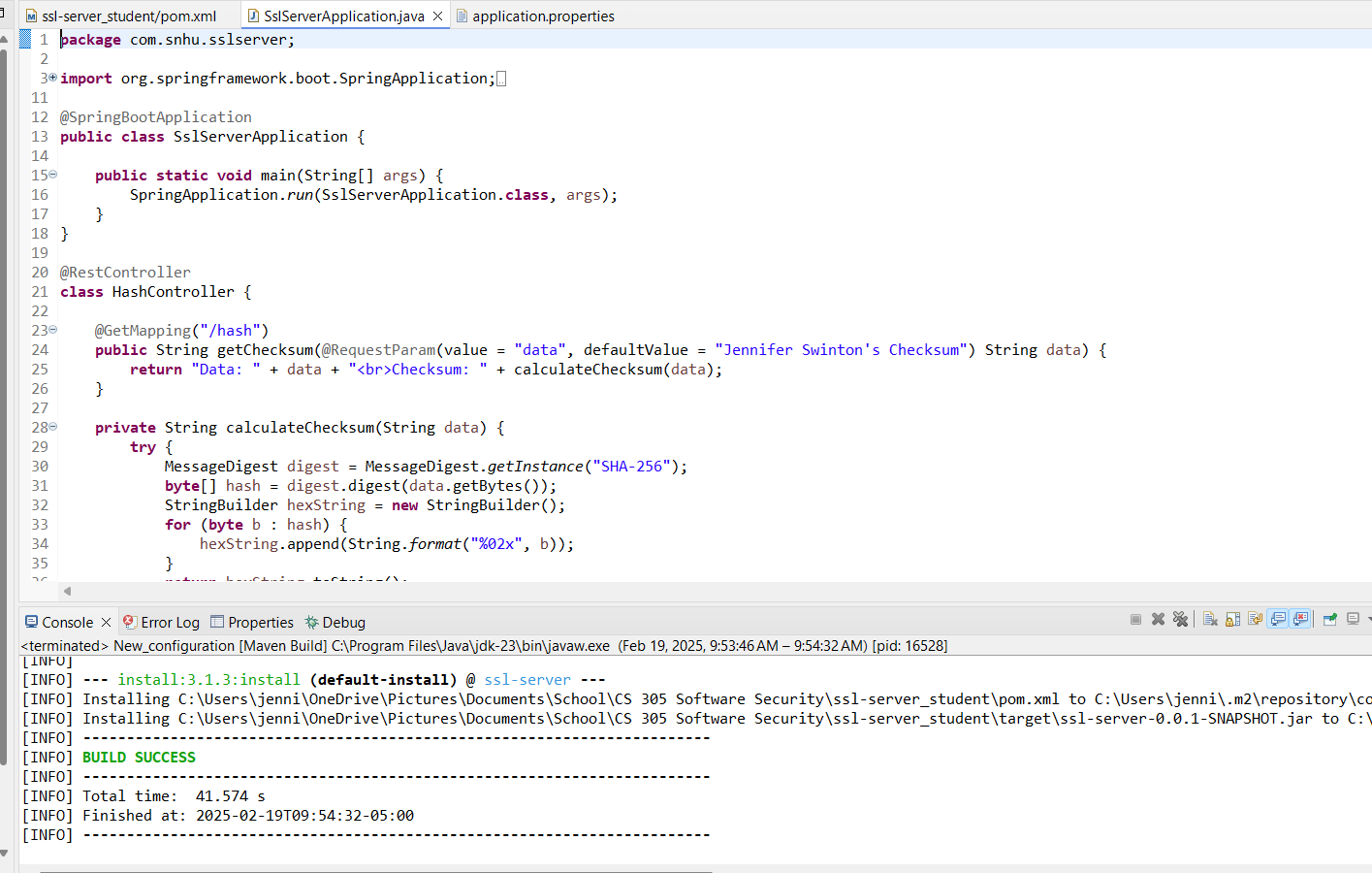
## Secondary Testing

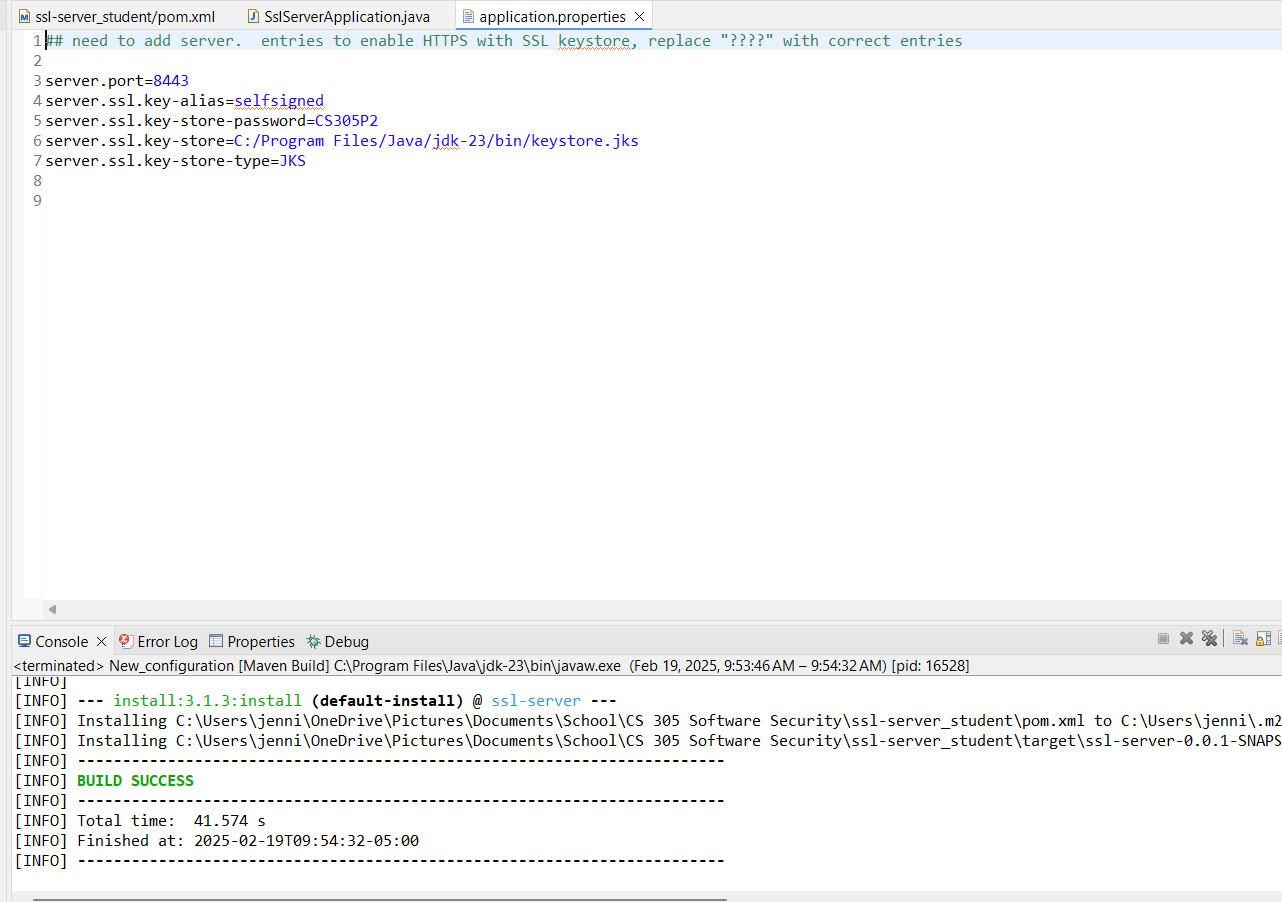
Insert screenshots below of the refactored code executed without errors and the dependency-check report.

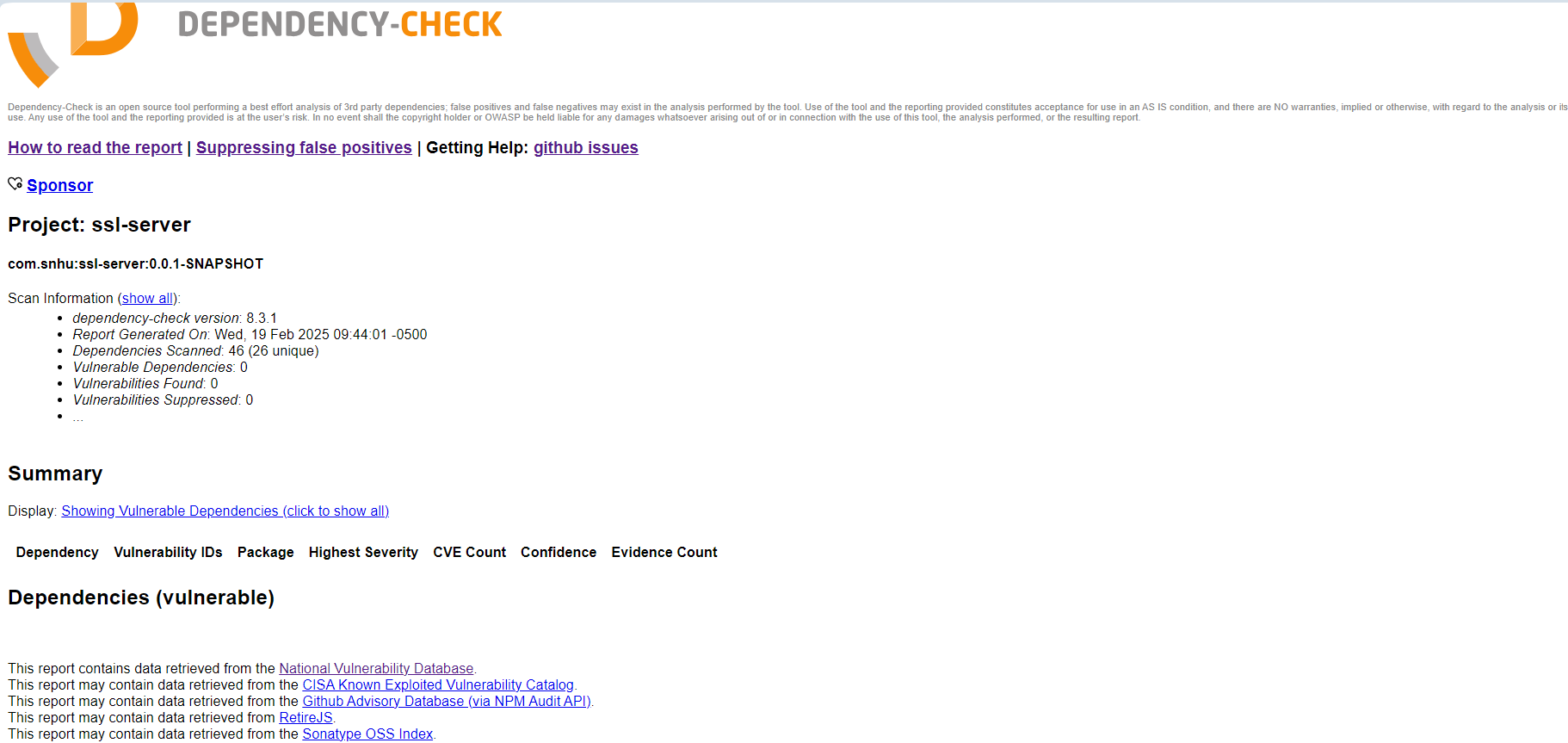
*Run a secondary static testing of the refactored code using the dependency-check tool to make certain the code complies with software security enhancements. You need to focus only on the code you have added as part of the refactoring. Complete the dependency check and review the output to make certain you did not introduce additional security vulnerabilities. Refer to the resources in the module’s Resources section for help on this action. In your practices for secure software report, include the following items:*

1. *A screenshot of the refactored code executed without errors*
2. *A screenshot of the report of the output from the dependency-check static tester*

**

**

**

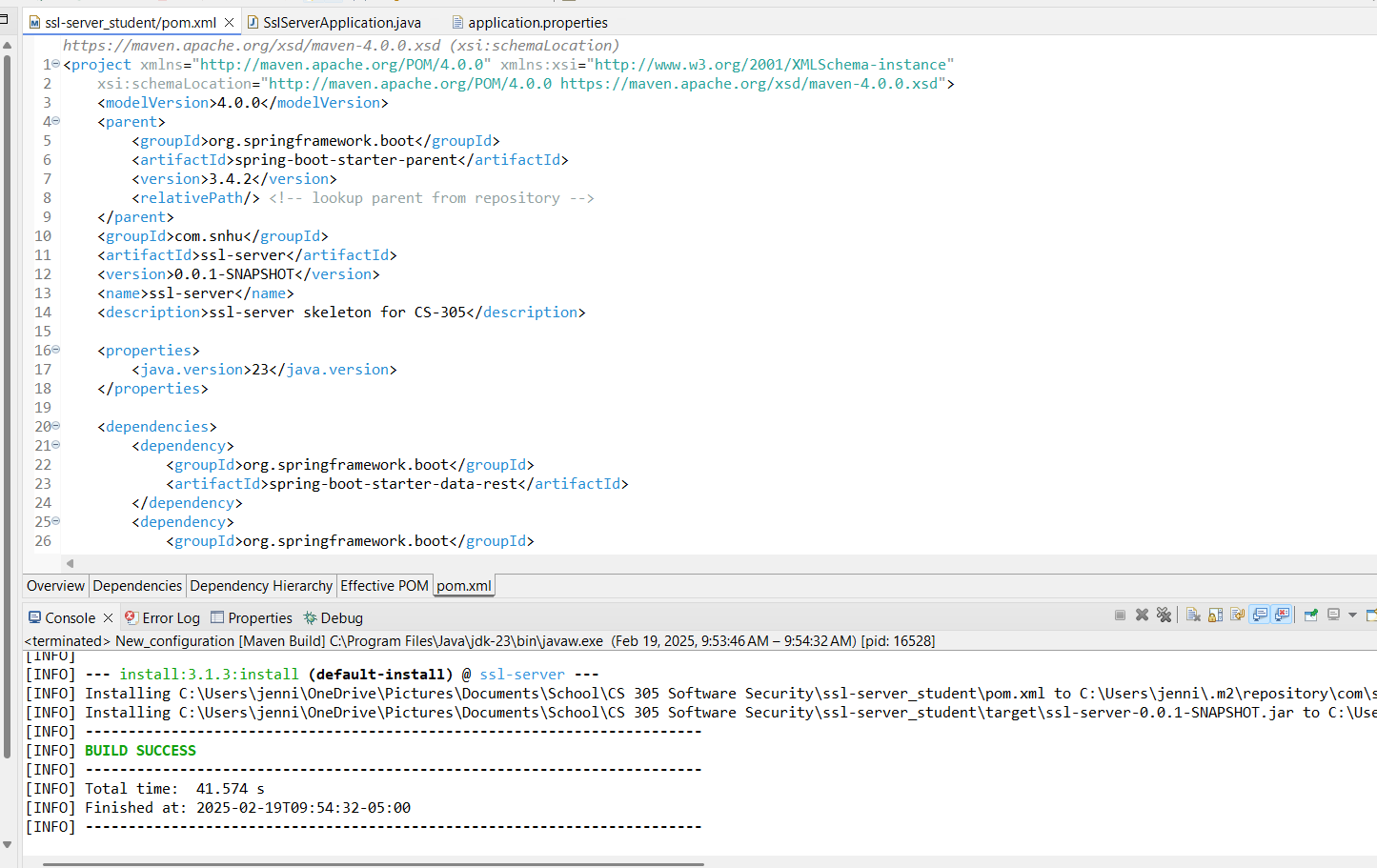


## Functional Testing

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

*Identify the software application's syntactical, logical, and security vulnerabilities**by manually reviewing the code.*

1. *Complete this functional testing and include a screenshot of the refactored code, executed without errors, in your practices for secure software report.*

**

## Summary

*Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:*

1. *Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.*
2. *Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.*
3. *Point out best practices for maintaining the current security of the software application to your customer.*

I’ve updated the versions of several items in the pom.xml file to get a more updated system and reduce the vulnerabilities. These updates include: updating the Java version to 23, spring boot to 3.4.2, Maven dependency check to 8.3.1, and the JSON smart artifact to 2.5.2.

Our application now utilizes a self-signed certificate which enables the use of https, a key security enhancement. To use HTTPS we first needed to make sure the certificates were created appropriately. By making sure that our website is secure and users can be sure they are dealing with us and not a scam, this security improves the trust of our business. The next step was ensuring that our hashing function operated correctly and checking this with the checksum. By giving us peace of mind that the data of our consumers is correctly scrambled and difficult to recover, we can ensure the trust and security of our customers.

To maintain the security of the application, the dependency checker should be run at least once a month to check for new vulnerabilities that may need to be updated. This can also help ensure that the plugins noted in the pom.xml file remain up to date. Implementing automated security testing during the build process can help catch potential issues early, maintaining the current security of the software application.

## Industry Standard Best Practices

*Explain how you applied industry standard best practices for secure coding to mitigate known security vulnerabilities. Be sure to address the following items:*

1. *Explain how you used industry standard best practices to maintain the software application’s existing security.*
2. *Explain the value of applying industry standard best practices for secure coding to the company’s overall well-being.*

To maintain the existing security of the software application, industry standard best practices were followed diligently throughout the development and refactoring processes. A SHA-256 hashing algorithm, a cipher that is widely recognized for its resistance to cryptographic attacks and provides protection for stored data.

Self-signed certificates were generated using the Java Keytool in Eclipse to ensure secure communications using HTTPS protocol**.**

The cryptographic hash algorithm was implemented with proper checksum verification.

The code was refactored to convert HTTP to HTTPS, which encrypted data transmitted between the server and client thereby preventing unauthorized access to sensitive information.

The secondary static testing of the refactored code was performed using the dependency-check tooland verified that we did not introduce additional vulnerabilities.

Manual review of the code was conducted to identify and address any syntactical, logical, and security vulnerabilities. This thorough testing process maintained the overall security of the software application.

By using robust encryption algorithms, secure certificate generation, and secure communications, the company ensures that sensitive data is well-protected which fosters trust among clients and customers.

When users know that their data is safe, they are more likely to engage with the company's services and recommend them to others.

Adhering to industry best practices helps the company comply with various data protection regulations and standards minimizing the risk of legal issues and penalties.

Regular dependency checks and automated security testing help identify and mitigate vulnerabilities early. This reduces the likelihood of security incidents and enhances stability.

With fewer security incidents and data breaches, the company can maintain uninterrupted operations and focus on growth and innovation.

References:

*The NSA and Bitcoin: Origins of the SHA-256 hashing algorithm*. (n.d.). Supra. https://supra.com/academy/the-nsa-and-bitcoin-origins-of-the-sha-256-hashing-algorithm/?form=MG0AV3