****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

[**1. Algorithm Cipher 4**](#_heading=h.tyjcwt)

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2/20/2025** | **cody Leveille** |  |

## 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

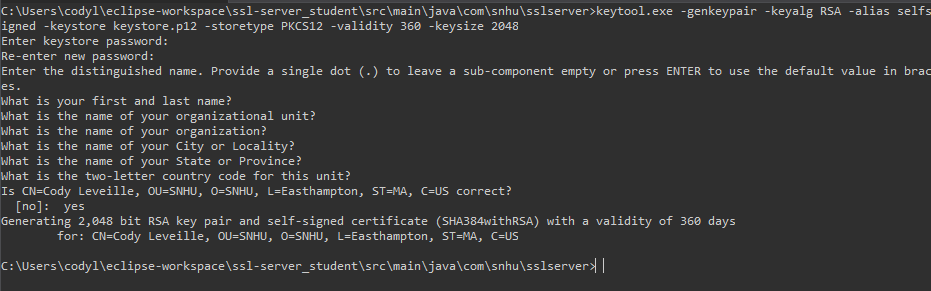
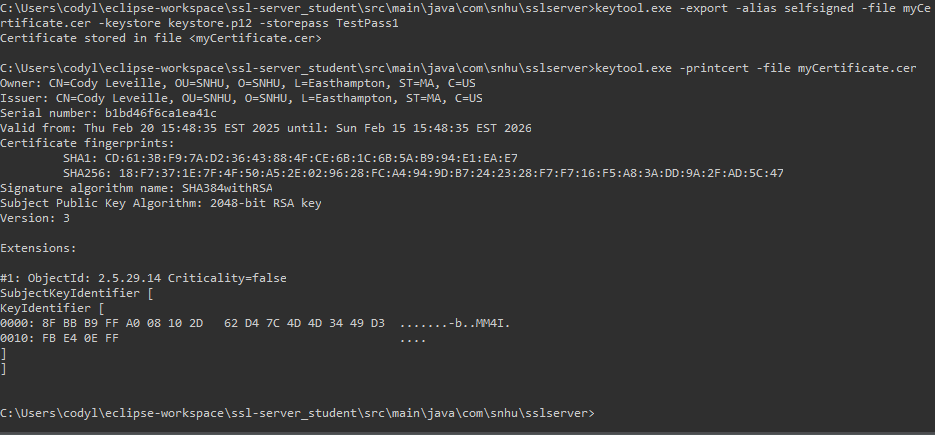
Cody Leveille

## Algorithm Cipher

1. Provide a brief, high-level overview of the encryption algorithm cipher.  
    The recommended Encryption that is recommended for Artemis Financial is to adopt AES-256. This will fit their needs and will be compliant with all federal demands since they are an organization that handles money. I recommend they utilize TLS 1.3 to handle secure client connections to their services that will provide protection against man in the middle attacks and to handle file transfer encryption I recommend SHA-256 to make sure data can not be stolen when transferred within the system. By implementing AES-256 for data encryption, TLS 1.3 for secure communication, and SHA-256 for data integrity, Artemis Financial can significantly enhance security, prevent data breaches, and meet compliance requirements for financial data protection.
2. Discuss the hash functions and bit levels of the cipher.  
    AES utilizes a symmetric encryption algorithm of 256 bits, this gives it a pool of keys of 2^256 keys this means that collision is highly unlikely and its brute force protection is the current industry standard. It is nearly impossible with modern computing power.   
    SHA-256 utilizes 256 bits for its hashing algorithm, this gives it a collision resistance of 2^128 so it is extremely unlikely there will ever be a collision within the system. Since SHA-256 is a one way function, once data is transformed, it cannot be reversed to its original form. With the implementation of SHA-256 it will allow for files to be verified that they have not been tampered with or altered, as well passwords will be able to be stored securely instead of them being stored as plaintext somewhere within the system they are transformed into a unique fixed length value to be stored securely. It is also compatible with TLS for certifications regarding web based communications.   
    TLS.13 handles the transport of data, it will make sure that there is an authentication and verify integrity it utilizes a cryptographic protocol that will secure the data over networks.
3. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.  
    Random numbers are utilized in encryption by making it so they are not easily guessed or replicated. AES-256 utilizes random numbers to make sure that its encryption remains secure. Most keys will be generated by a cryptographic method to be purely random and this will ensure that they are unpredictable. A symmetric Encryption will utilize one key for its encryption and decryption while a Asymmetric Encryption will utilize 2 keys, a public one for encryption and a private one for decryption. Symmetric encryption is faster and more efficient than Asymmetric encryption, this is because it only utilizes a single key for both needs. Symmetric Encryption is often used for data storage and transmission and will utilize AES and will require secure key management to do this while Asymmetric encryption is used for key exchange and authentication, it is more secure but it is more expensive overall.

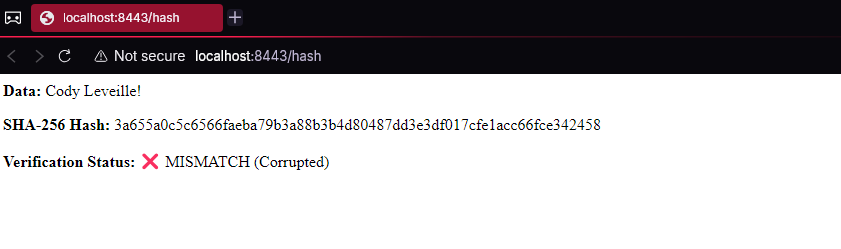
## Certificate Generation

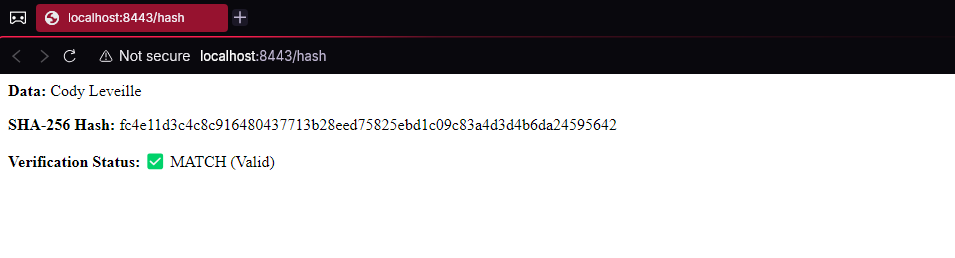
Insert a screenshot below of the CER file.

## Deploy Cipher

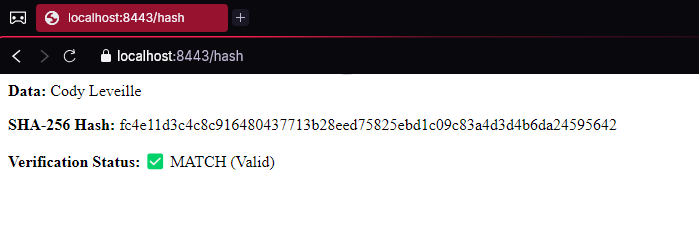
Insert a screenshot below of the checksum verification.

Invalid cipher when you change expected data

  
Valid cipher with correct data

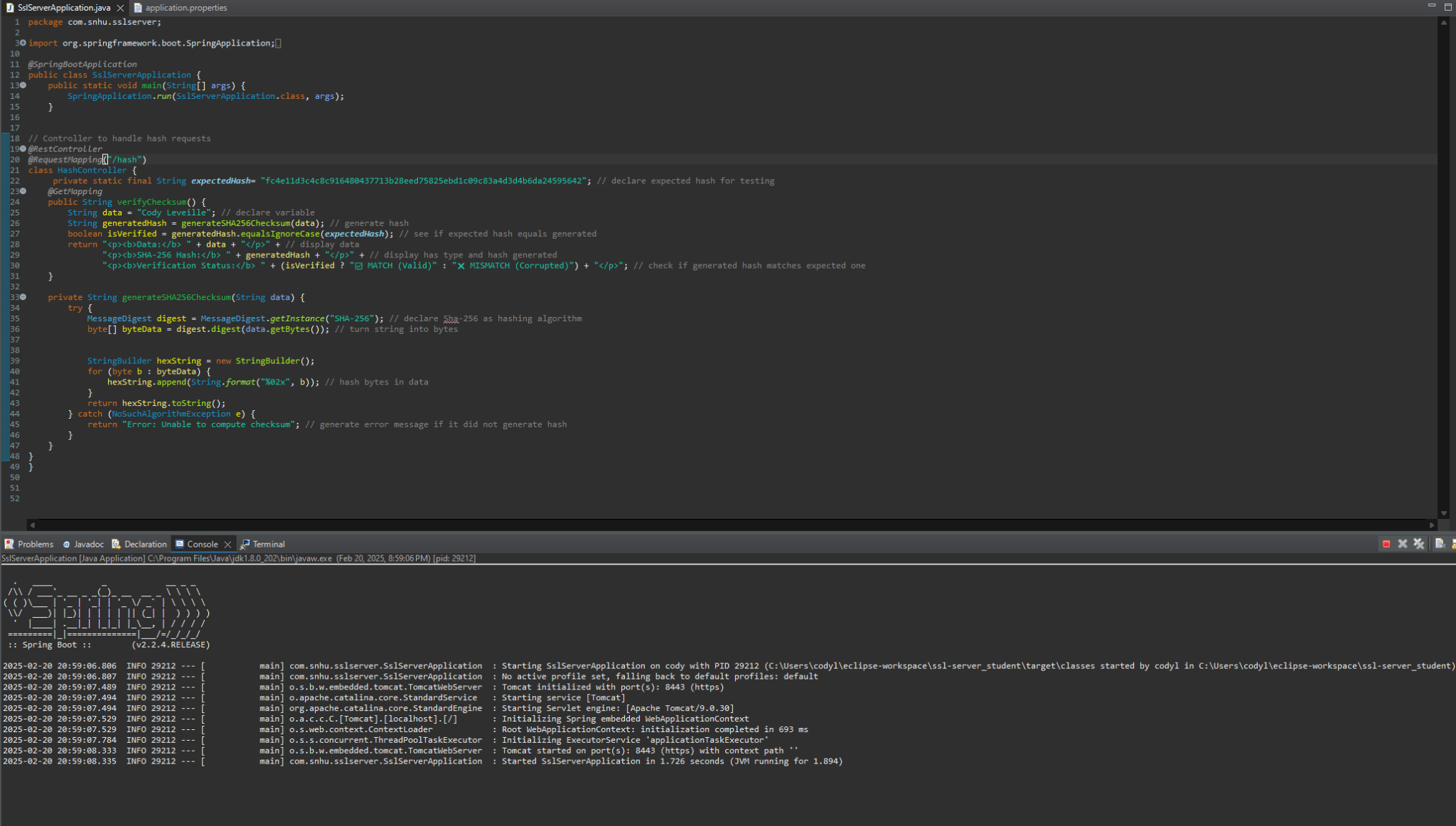
## Secure Communications

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

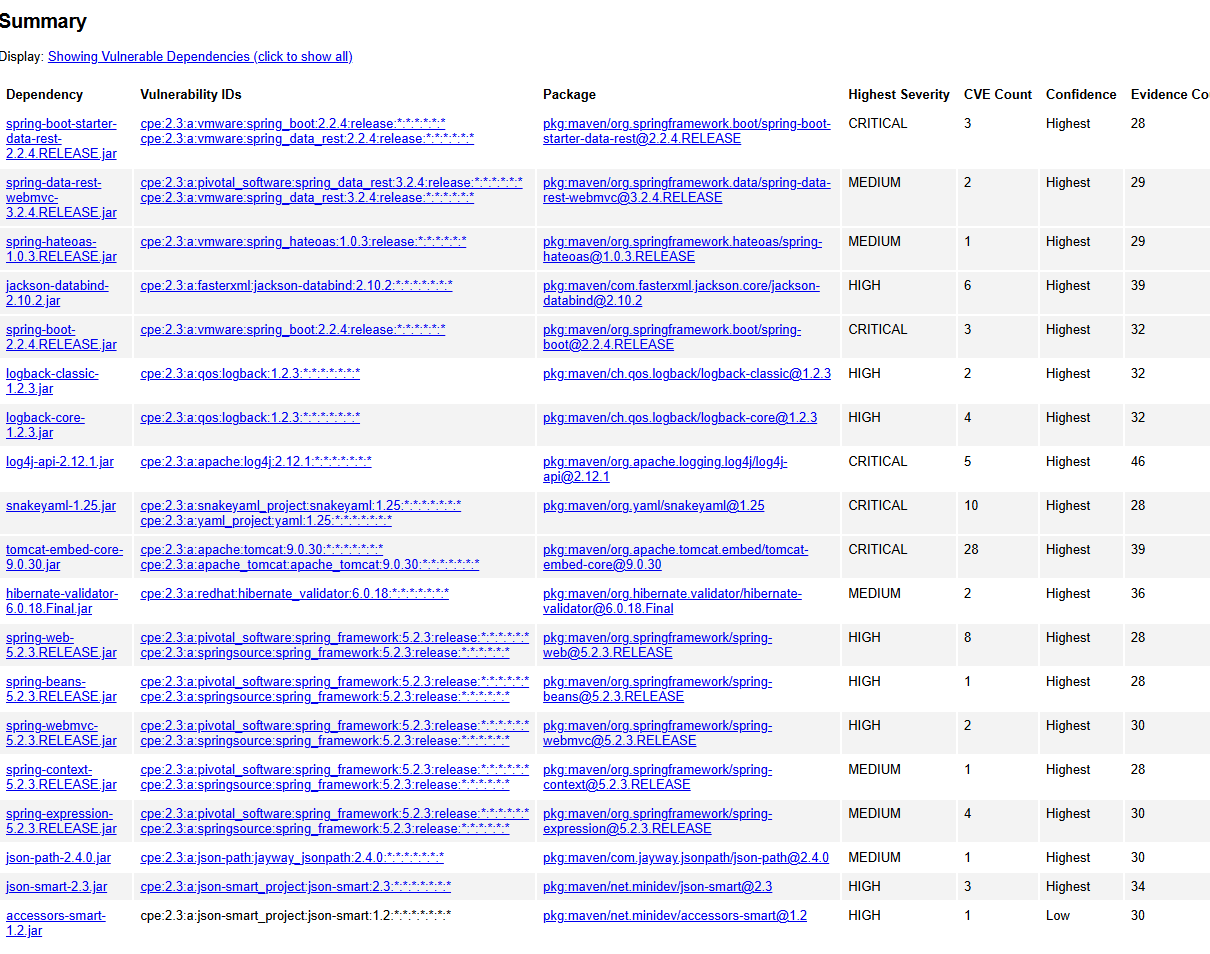


## Secondary Testing

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

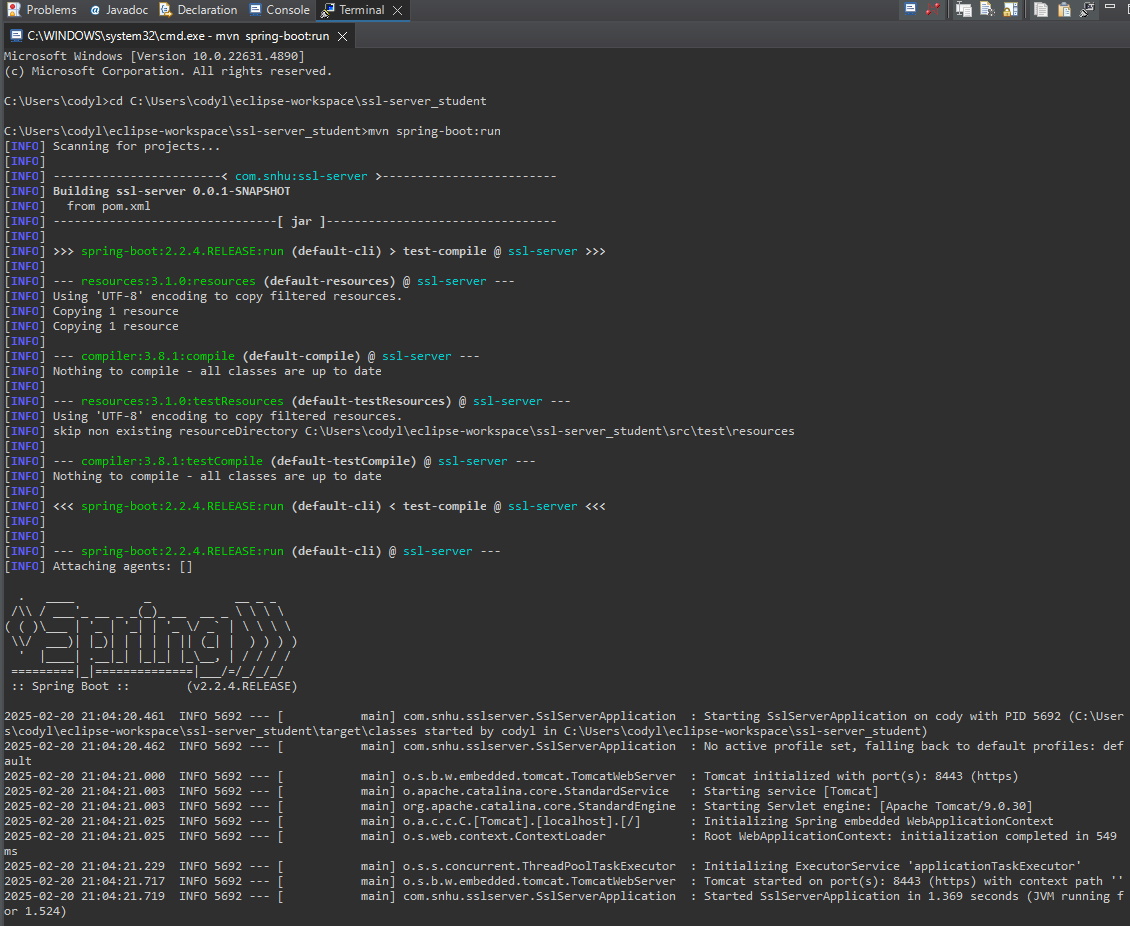


refactored code running without errors

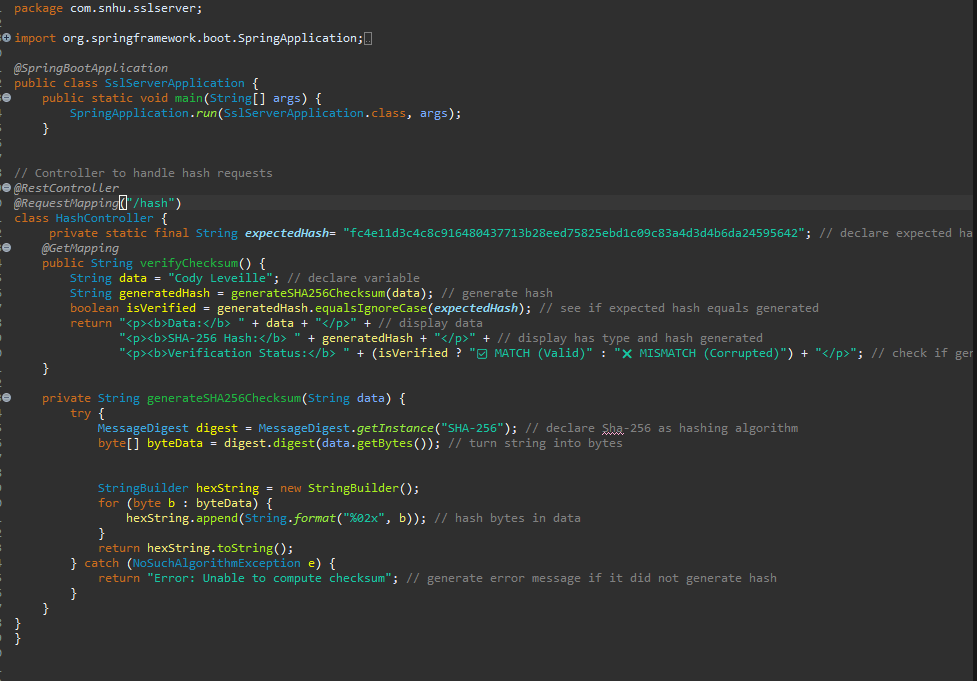
dependency check snapshot

## Functional Testing

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



program booting up with no errors



refactored code with no errors

## Summary

From the original code that contained a fixme portion, after refactoring it I implemented a rest API with /hash to do the checksum verification. as well I implemented SHA-256 to verify and hash data. Since I wanted to verify data I did a test with what was expected for the SHA-256 hash for Cody Leveille, this then checks to make sure that is the intended data that was hashed and will inform the user if it was. I implemented an HTTPS connection with TLS for secure communication and verified this with the selfsigned certificate. I did a maven snapshot of before and after I refactored my code and no new vulnerabilities were introduced. From the Original Pom.xml it is recommended that a few libraries be updated. such as Springboot to 2.7.12, Log4j to 2.17.1 and Tomcat from 9.03 to 9.08 this would address the existing vulnerabilities. Through the addition of the application.properties file I enabled a check for the selfsigned certification as well as implemented a password on that key. This will ensure data integrity and data verification and will prevent man in the middle attacks and data injection.

## Industry Standard Best Practices

Through the use of SHA-256 it makes the program compliant with hash security standards. As well as the data is compared to the expected values to make sure data is authorized and secure. By enforcing HTTPS connections it makes sure anyone who connects has a secure connection and with the use of HTTPS/TLS it will prevent people from stealing data between endpoints. With the creation of keystore.p12 it can protect sensitive data and it makes it so things are not hardcoded where they are not needed to be and requires a password for the SSL certificate. By running the dependency check it also shows what things are needed for an update and what vulnerabilities you may have within your program. With these security measures in place it will prevent unauthorized access to the system and with error handling it will not display an error message that may give attackers an insight into the system. SHA-256 gives security strength, TLS encryption makes sure data on the connection is secure and by implementing the updates the vulnerabilities originally on the application will be fixed.