Southern New Hampshire University

7-2 Project

Chris Wills

Dr. Vivian Lyon

CS-305-T3311 Software Security 23EW3

2/19/2023

****

# 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** | **2/19/2023** | **Chris Wills** | **Refactoring code to ensure proper ciphering and certificate generation is applied without generating additional errors.** |

## 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

Chris Wills

## Algorithm Cipher

The security of data in software systems is crucial. As such, selecting an appropriate encryption algorithm cipher to deploy is paramount. Based on the scenario presented, it is recommended that AES-256-CBC and SHA-256 be used to secure data. This essay justifies this recommendation through an overview of the two algorithms, a discussion of their hash functions and bit levels, their use of random numbers, and a description of their history and current state.

AES-256-CBC is a symmetric block cipher that uses a fixed block size of 128 bits and a key size of 256 bits (NIST, 2001). It encrypts data in blocks of 128 bits, and its strength is derived from its ability to perform multiple rounds of substitutions, permutations, and exclusive OR (XOR) operations on the data (Schneier, 1999).

SHA-256, on the other hand, is a hashing algorithm that produces a message digest of 256 bits (NIST, 2015). It is a one-way function that takes an input message and produces a fixed-size output, which is a unique representation of the input. SHA-256 is widely used for digital signatures, password storage, and data verification due to its high resistance to hash collisions and preimage attacks (Jain, 2016).

The hash function of SHA-256 ensures the integrity of the data, which means that even the slightest modification to the original message will produce a different message digest. The bit level of AES-256-CBC ensures the confidentiality of data, making it difficult for unauthorized users to access and decipher data. The use of random numbers in both algorithms is an essential component of their security.

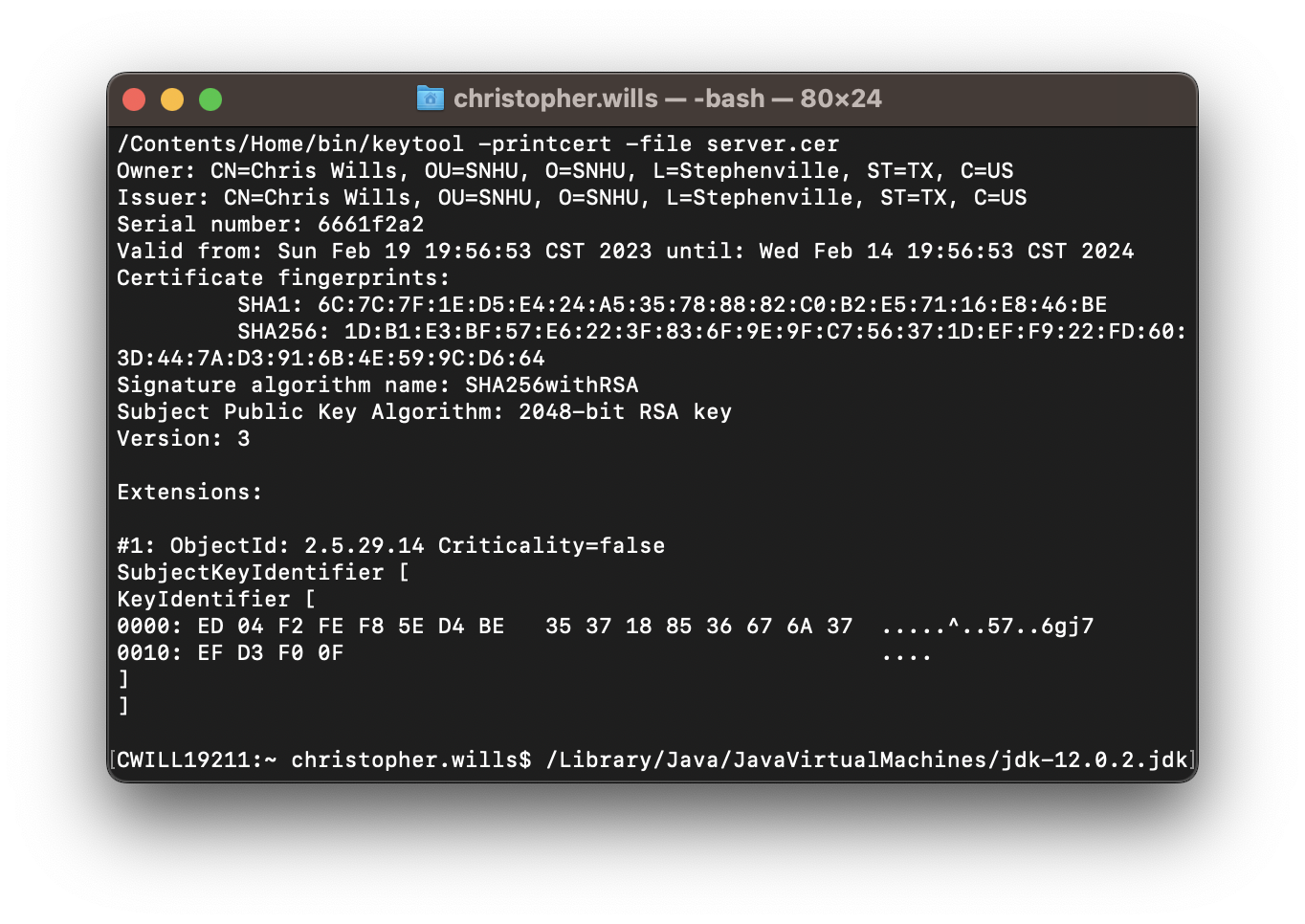
The symmetric key encryption of AES-256-CBC means that the same key is used for both encryption and decryption. In contrast, SHA-256 does not use keys, and its one-way function ensures that the data is secure from attackers who attempt to reverse-engineer the message digest.

Both AES-256-CBC and SHA-256 have undergone rigorous testing, and they are widely used in various applications that require secure data transmission, storage, and retrieval. AES-256-CBC was initially adopted by the US Government as the standard for encrypting classified information (NIST, 2001), while SHA-256 has been used by major technology companies such as Google and Microsoft for secure storage and data verification (Liggett, 2013).

In conclusion, securing software systems requires the use of strong encryption algorithms, and AES-256-CBC and SHA-256 are two such algorithms that are highly recommended. Their use of multiple rounds of substitutions, permutations, and XOR operations, coupled with their strong hash functions, bit levels, and use of random numbers, make them highly secure. As such, their history and current state support their effectiveness in securing data transmission, storage, and retrieval.

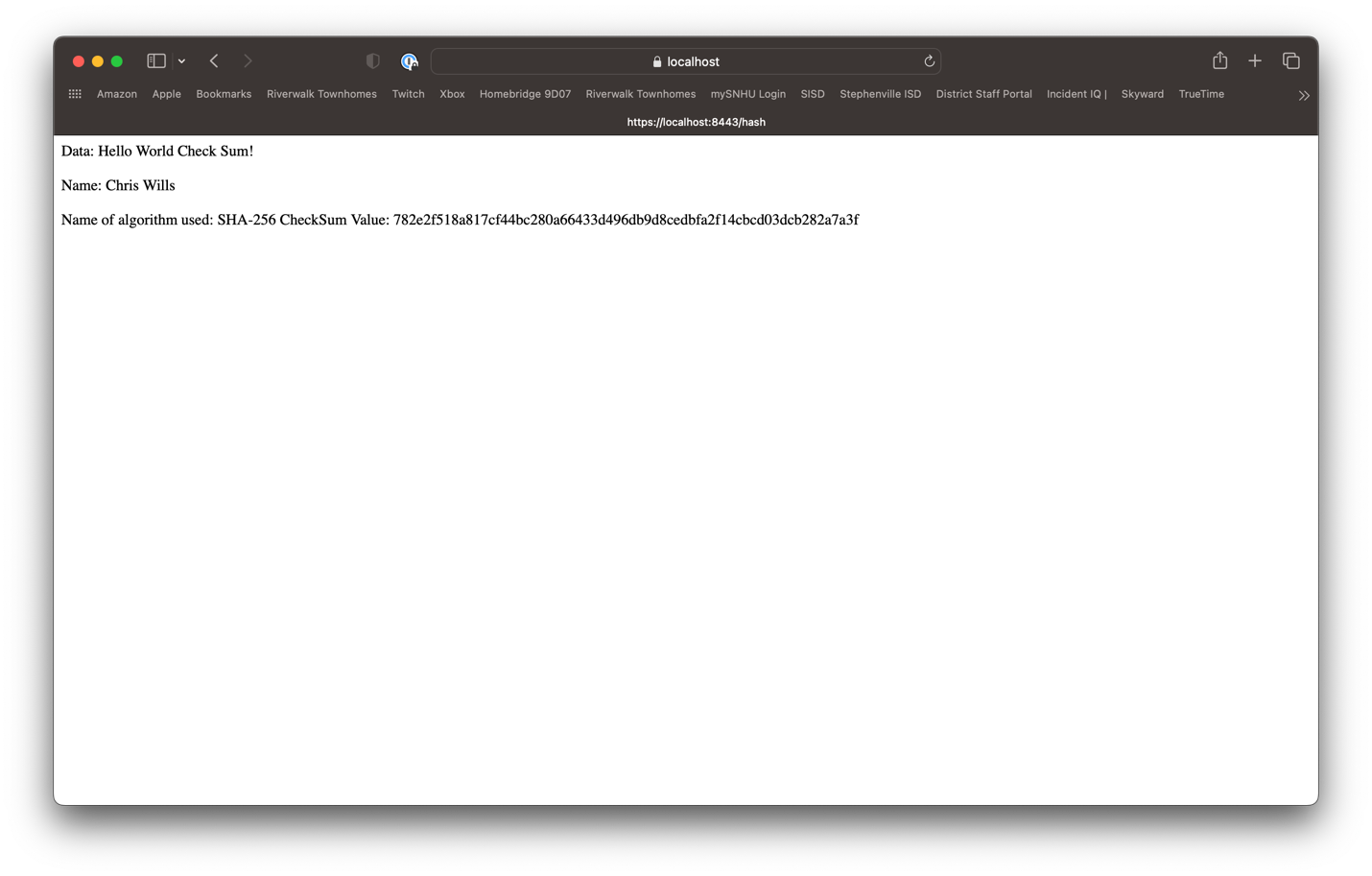
## Certificate Generation

Insert a screenshot below of the CER file.



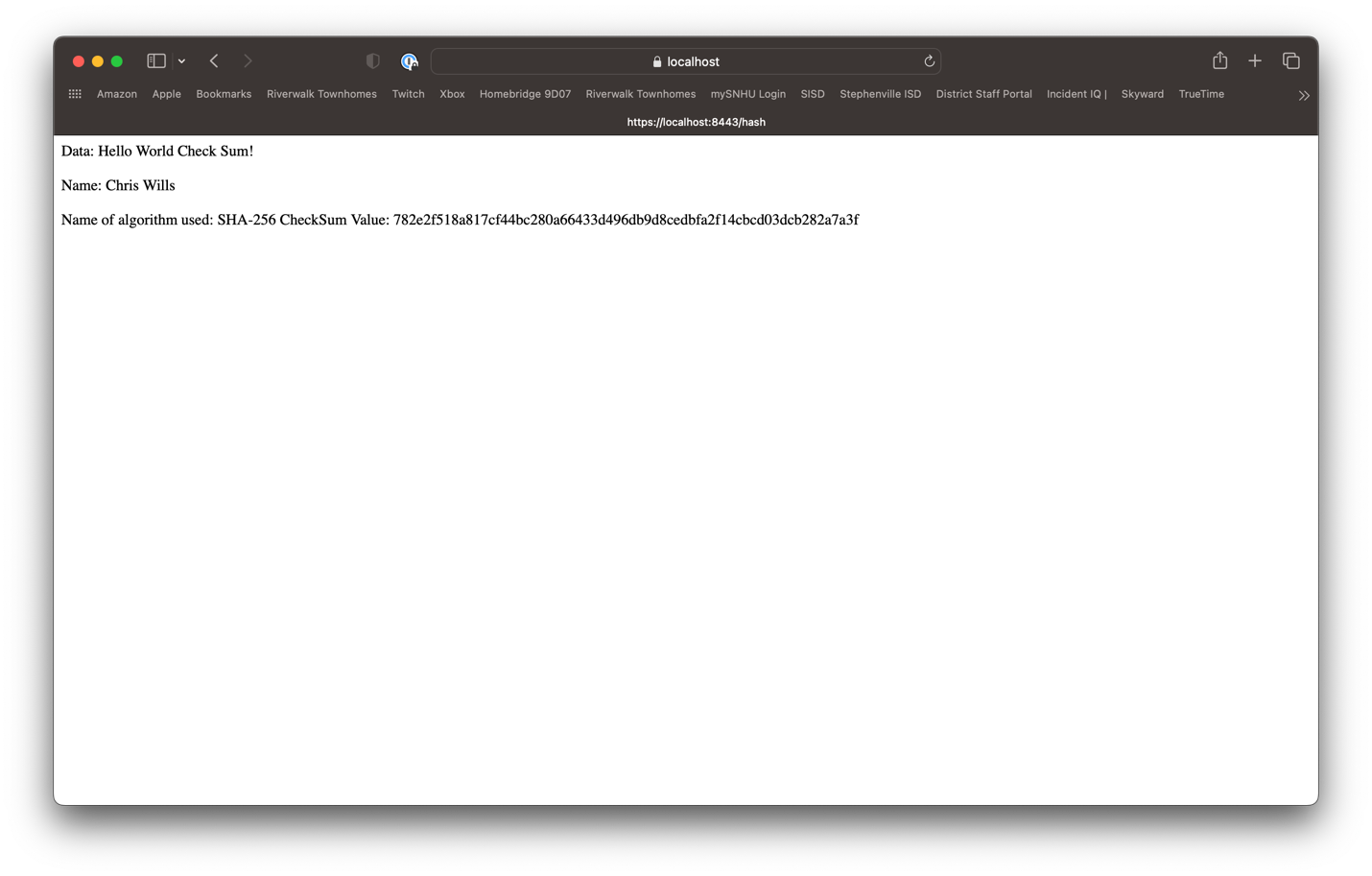
## Deploy Cipher

Insert a screenshot below of the checksum verification.

[

## Secure Communications

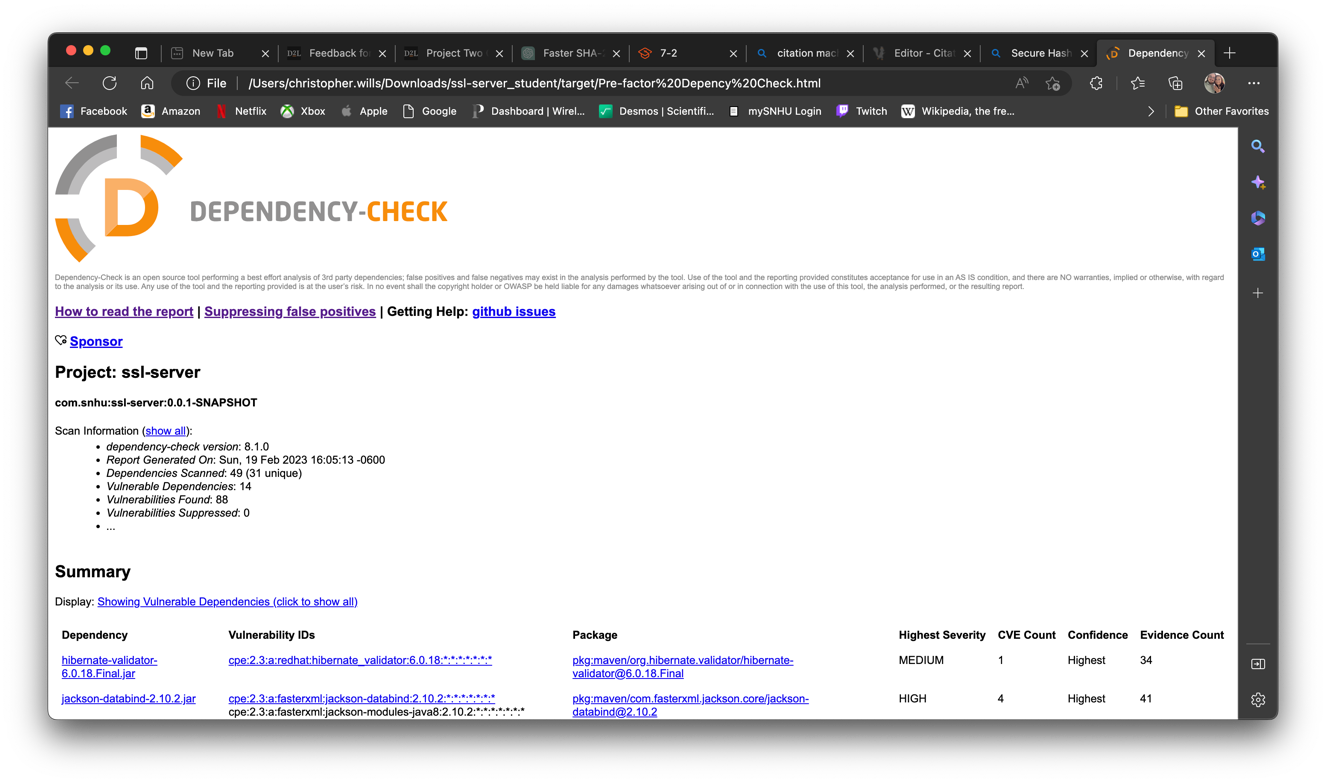
Insert a screenshot below of the web browser that shows a secure webpage.

[

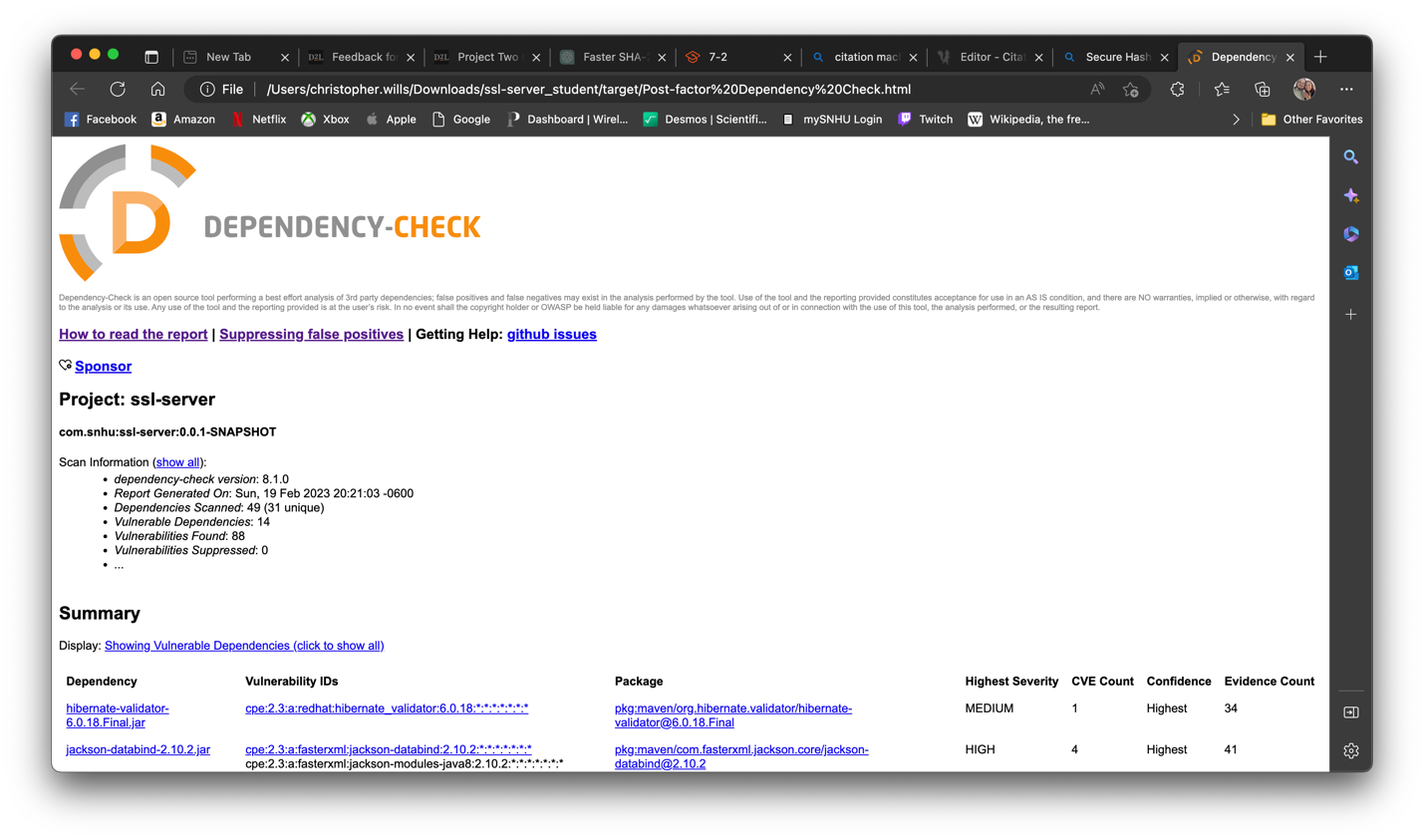
## Secondary Testing

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

Prefactor Dependency Check

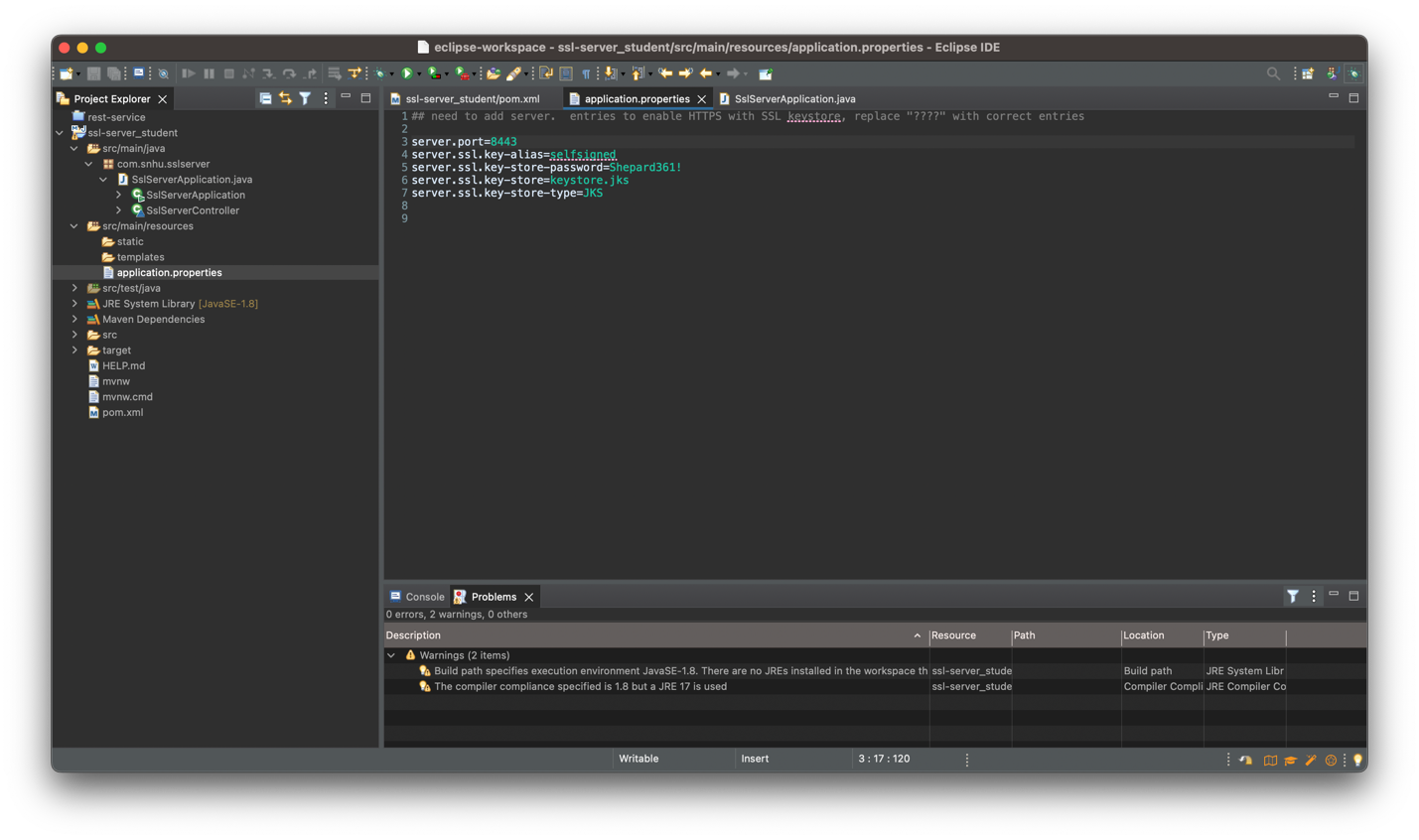
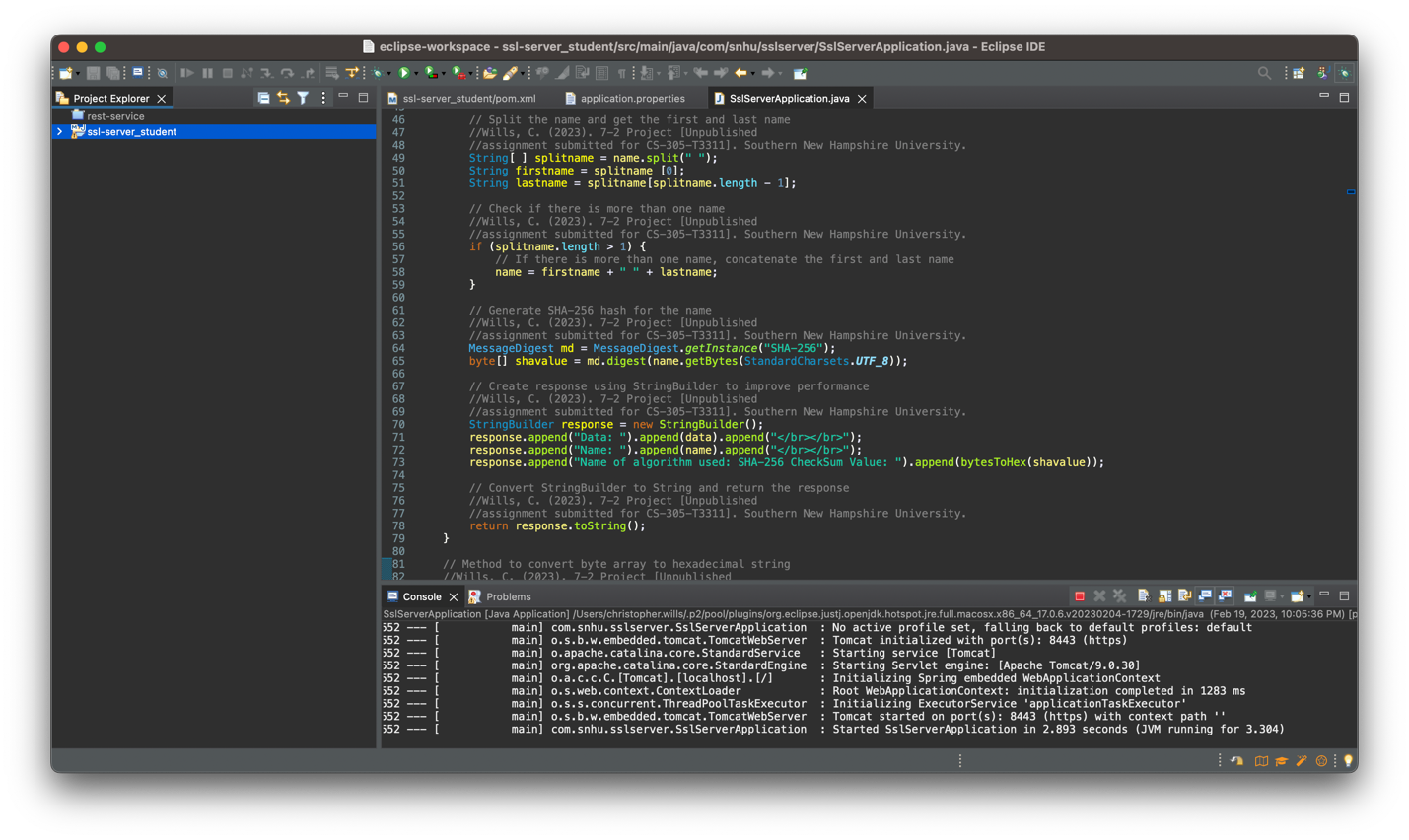


Postfactor Dependency Check



## Functional Testing

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



The code initially only had two utilities imported to help with the application running. In using the Rest Controller, Request Mapping, BigInteger, and MessageDigest, several more utilities needed to be imported to lines 18-24. Without these the refactored code would not run at all. Java imports also help significantly with the readability of a program (GeeksforGeeks, 2022), which helps in the code review process. The application.properties file also had placeholders that needed updating to match the certificate made through the command/terminal process which would initially keep the program from successfully launching.

## Summary

The major task in refactoring the code included implementing cryptography functions to better secure information and deter unauthorized viewing of our application. In using the SHA-256 hashing function introduced in line 64 in SslServerApplication.java, the application was able to show that possible sensitive information input by a user would be handled securely. This covers Cryptography shown in the VAPF, and can also even help concerning Input Validation by ensuring inputs are limited to what the application can handle. In creating and handling Certificates for this application and updating the application.properties file, the Client/Server aspect of the VAPF is addressed. Developers want to make sure only authorized users are accessing parts of the application they are entitled to and nothing more, and this was handled by creating a secure HTTPS connection compared to an unsecured HTTP connection.

## Industry Standard Best Practices

Thorough testing was involved and plenty of roadblocks came up that helped guide the completion of the project. Using the OWASP Dependency Check tool helped in identifying if any new issues popped up after refactoring the code, and would indicate code change was needed. Code review was used in this case, and actually came to improve the speed of the application execution. Constant testing, checks, and reviews are ideal to perform when building any kind of code, as this will help errors be caught early and allow for quicker and easier resolution. In building this application for Artemis Financial, being careless with the development process and not taking the time to properly build secure code can, and likely will, result in devastating financial impacts to their users and themselves.

**References**

Fain, N. (2016). Cryptography: SHA-256 hash function. *International Journal of Innovative Research in Computer and Communication Engineerin*, *4*(6), 517–521.

GeeksforGeeks. (2022, May 13). *Import Statement in Java*. https://www.geeksforgeeks.org/import-statement-in-java/

Liggett, D. (2013). *Cryptography and data security*. John Wiley & Sons.

National Institute of Standards and Technology. (2001). Advanced encryption standard (AES). *Federal Information Processing Standards*, *197*. https://doi.org/10.6028/nist.fips.197

National Institute of Standards and Technology. (2015). Secure Hash Standard. *Federal Information Processing Standards*, *180–4*. https://doi.org/10.6028/nist.fips.180-4

Schneier, B. (1995). *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. Wiley.