

# 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** | **April 19, 2025** | **Jonathan Brugh** | **Added algorithm cipher.** |
| **1.1** | **April 19, 2025** | **Jonathan Brugh** | **Added certification generation and deployed cipher.** |
| **1.2** | **April 19, 2025** | **Jonathan Brugh** | **Added proof of secure connection, secondary testing, and functional testing.** |
| **1.3** | **April 20, 2025** | **Jonathan Brugh** | **Completed Summary and Industry Best Practices sections.** |

## Client



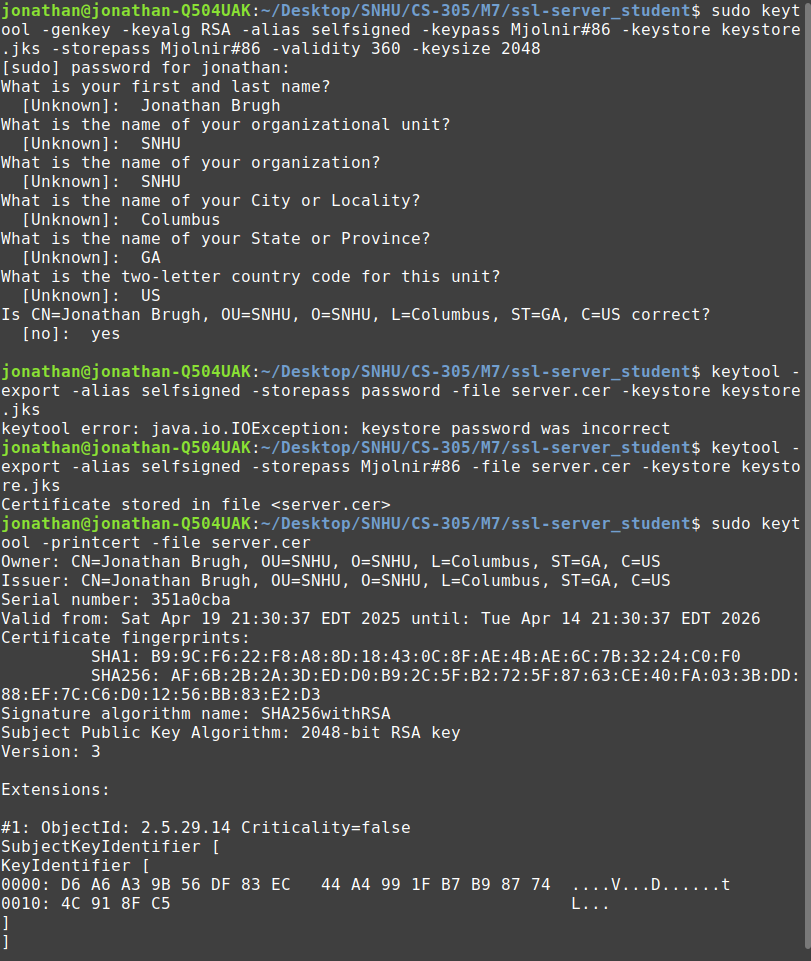
## Developer

Jonathan Brugh

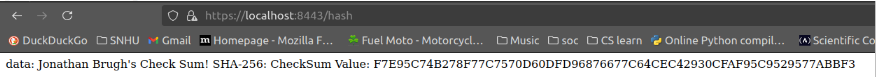
## Algorithm Cipher

The algorithm recommended is the SHA-256 algorithm. SHA-256 is a one-way function which makes it virtually impossible to discover the original input. The algorithm is highly resistant to collisions minimizing the possibility that two inputs would generate the same hash value. Because of these two aspects, this algorithm is perfect for verifying data integrity securely. The hashing algorithm first pads the message to be exactly 64 bits less than a multiple of 512. The padding bits begin with one and the remaining bits are zeros. The 64 bits of padding characters are determined by taking the modulus of the original plain text without padding. The padded plain text then goes through rounds of bit, row, and column shifting using different keys for each round initialized by the output of the previous round. Random numbers provide security by allowing for the creation of unpredictable keys and initialization vectors to be used in cryptography. Symmetric keys encrypt data using a secret key that is also required for decryption. Asymmetric keys encrypt data using a secret key that is not needed for decrypting the data later. Instead, a publicly known key is used to decrypt the data. Random numbers ensure that a secret key in both cases can be created with a minimal chance of being reverse engineered. Cryptographic algorithms have been sought after for centuries to keep information secure from adversaries whether that be nation-states, competitors, or today’s cybercriminals. Today’s algorithms are having to be designed with the development of quantum computation in mind. This is the current threat to encryption algorithms.

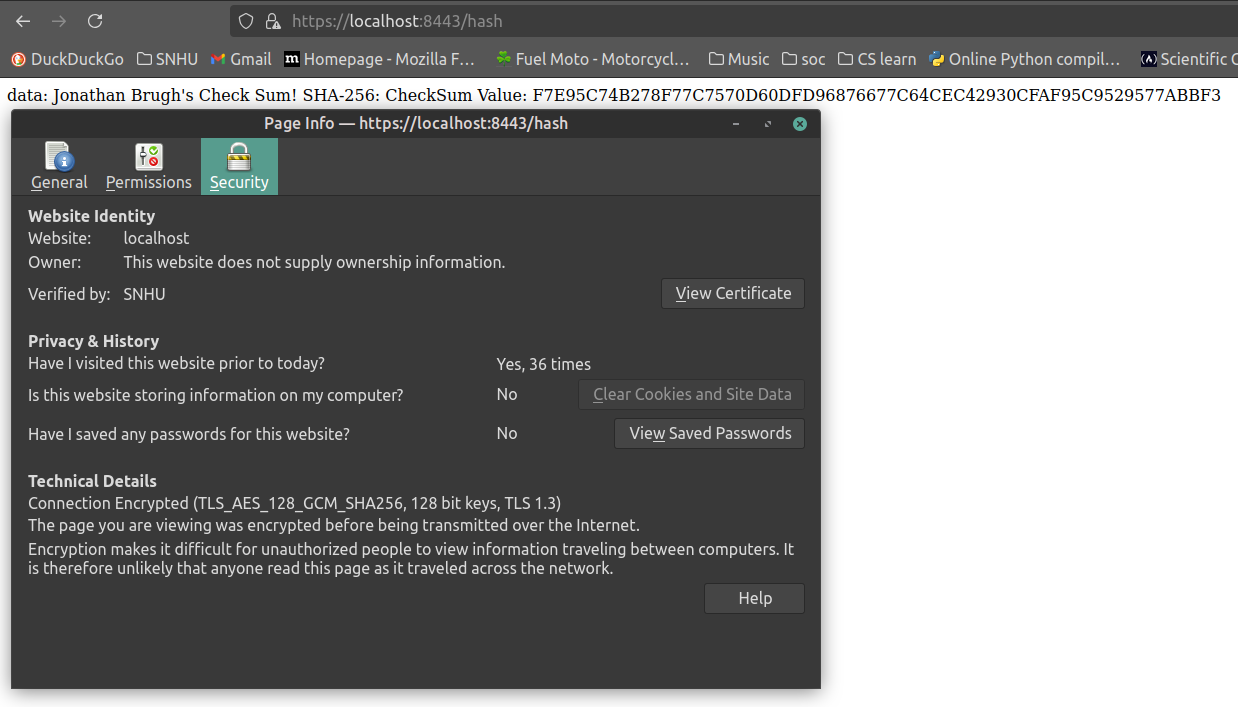
## Certificate Generation



## Deploy Cipher



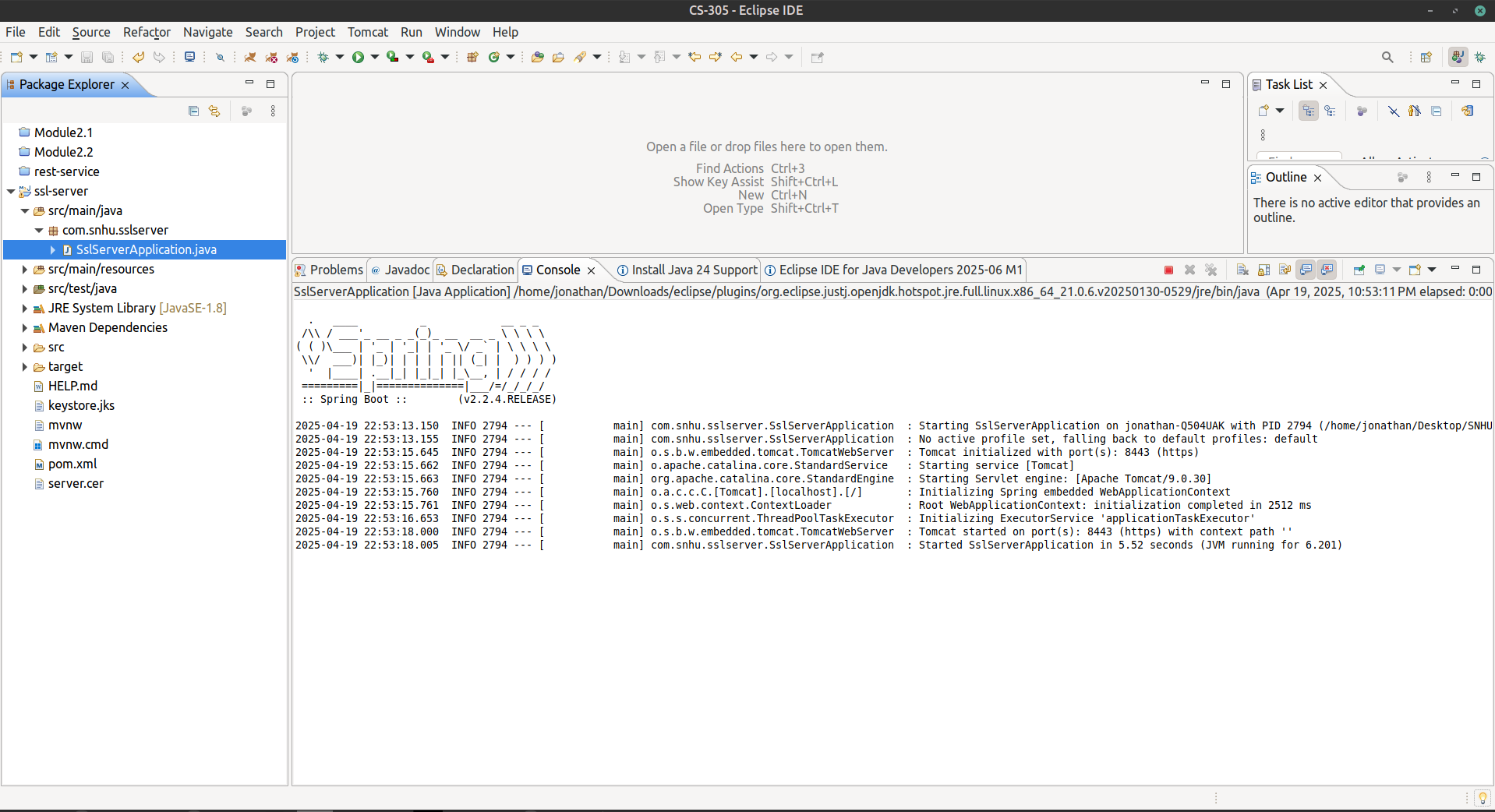
## Secure Communications



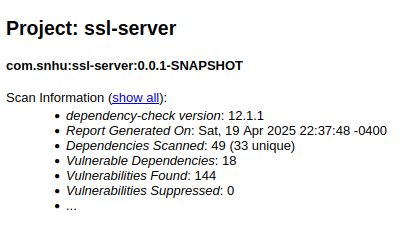
My browser did not appreciate my self-signed certificate.

## Secondary Testing

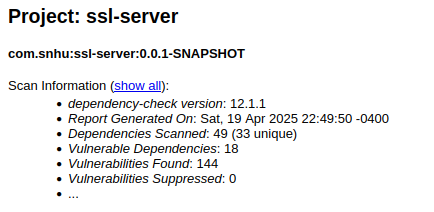
Executed code with no errors:



Initial dependency check:

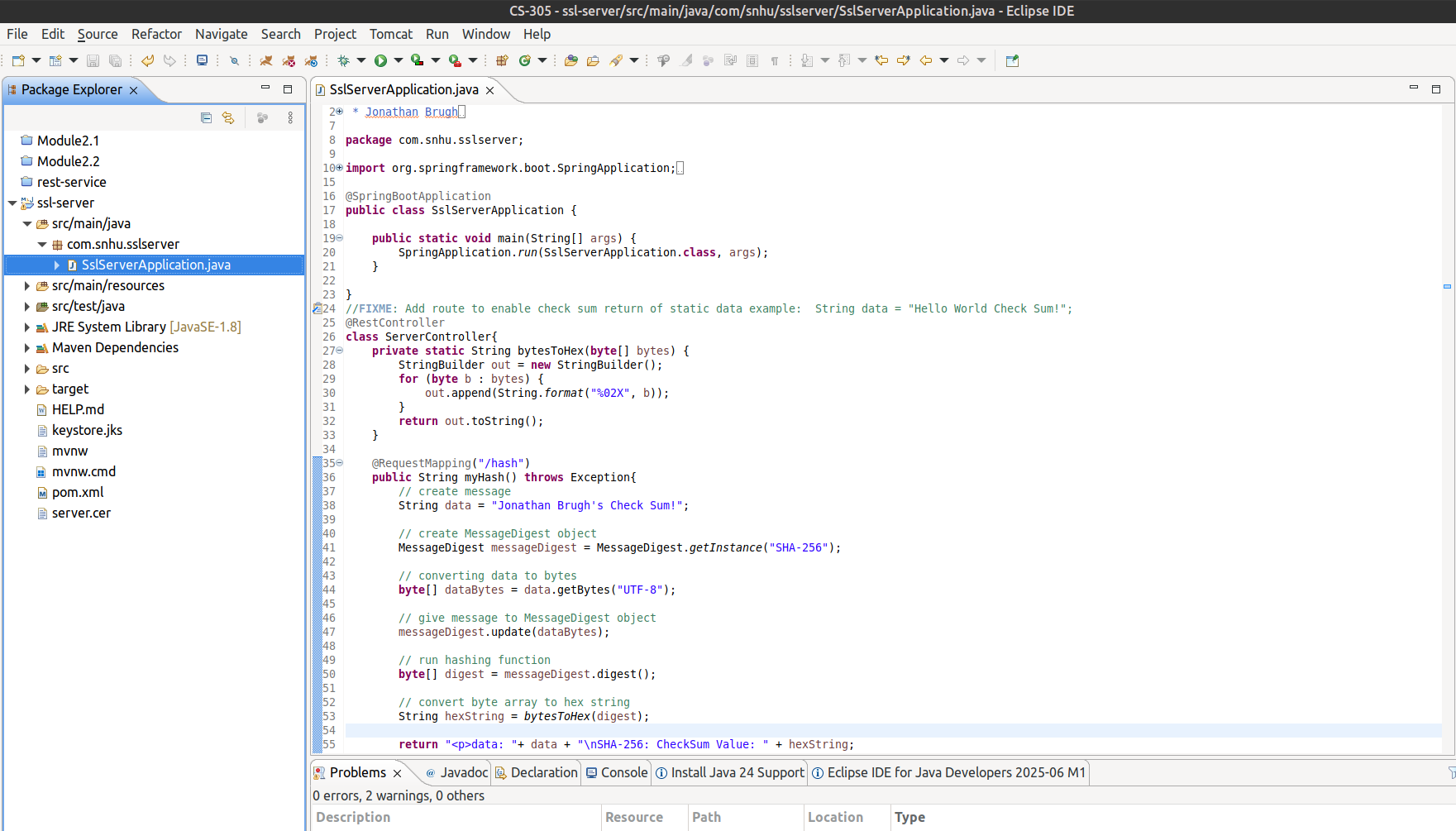


Secondary dependency check:

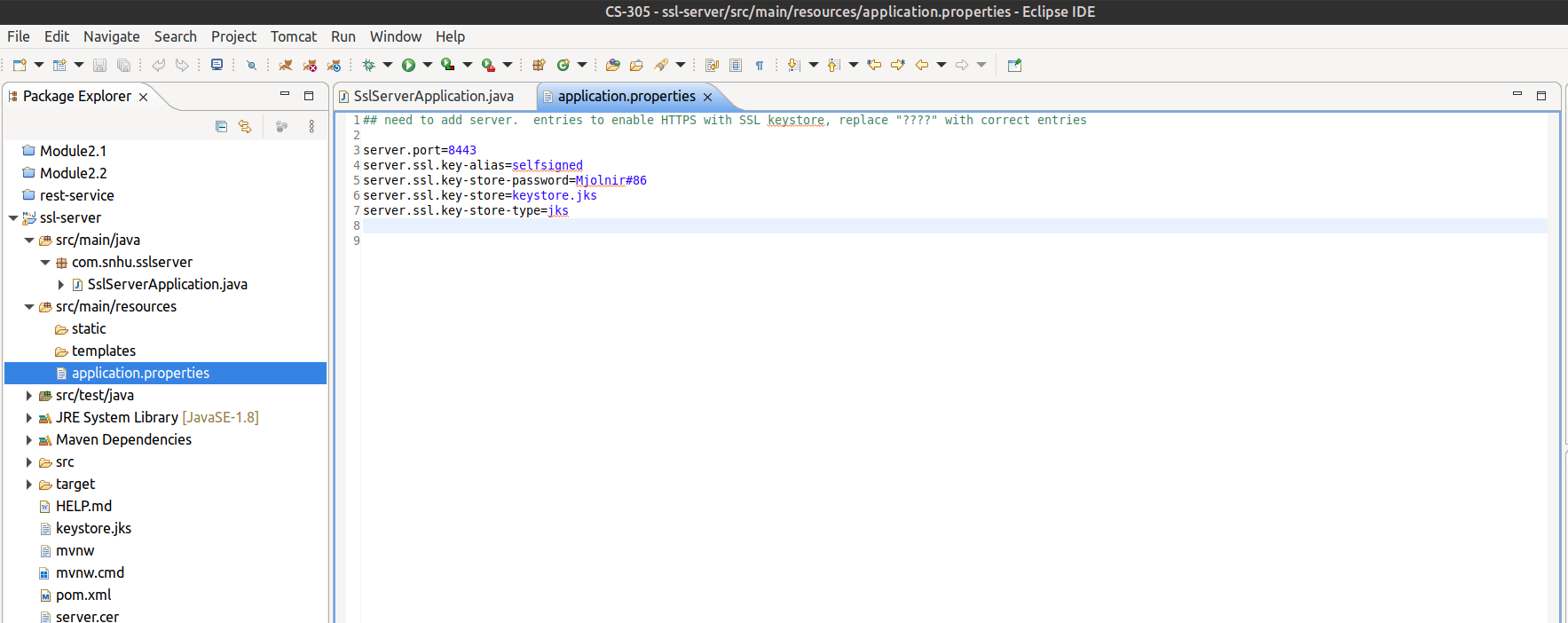


## Functional Testing

Main:



Application properties:



## Summary

The areas of vulnerability to be assessed in the refactored code are APIs, cryptography, client/server, code errors, and code quality. The code deploys a RESTful API that provides a defined path to the hash algorithm. Security was added to the data in the form of a self-signed certificate, security keys, and a cryptographic hashing algorithm, SHA-256. The certificate validates the key generated for use in the SHA-256 algorithm, the algorithm encrypts the message, and the hash value generated from the algorithm can be used in verifying the data integrity. The hashing algorithm is written from line 35-55. The certificate is generated using Java’s built-in key manager, keytool, and the data to access the SSL key store is found in the application properties file on lines 4-7. Verifying the data integrity ensures that the data was not compromised by outside entities or otherwise corrupted. The client/server communication was encrypted and can only be accessed by using HTTPS which validates the security of the connection between the server and the client. If the hash path’s function call fails, an exception is thrown but not exposed to a user attempting to access the data on the server. This is important as it does not allow the user to understand what failed, which could allow them to exploit the error. Finally, the quality of the code does not allow unvlidated inputs to be parsed to functions or to be concatenated to an existing variable. Each data input is parsed as a variable itself to avoid manipulation which can be seen in the myHash function on lines 38, 44, 50, 53, and 55. There is no user input in this code, but it is best practice to maintain data/input validation for variable use. Additional coding quality was employed by using comments in the code to aid in maintainability.

## Industry Standard Best Practices

For Artemis Financial’s application, I employed industry standard best practices by ensuring data was not parsed in the application by keeping control of variables to avoid manipulation. This needs to be maintained when user input is required to eliminate a user from exploiting weaknesses in an applications input/variable handling. This could lead to injection attacks. Applying a security certificate for secure HTTPS connection prevents outside entities from seeing any data in transit. The integrity of the data can be verified by examining a generated hash value received by the client compared to the hash value given by the server. The hash value is generated by using an asymmetric algorithm and should follow industry standards for the data being sent. Since Artemis Financial is dealing with highly sensitive data, an algorithm that has not been cracked and is generally accepted by the industry will ensure the data’s integrity. I chose SHA-256 for Artemis’ application because it is considered by the industry to be nearly impossible to reverse engineer the input and is highly collision resistant. These two factors can prevent false checksums to be generated for data that may be malicious.