Southern New Hampshire University

7-1 Project Two

Ryan Laird

Dr. Lyon

CS-305-T6616 Software Security 22EW6

8/14/2022

****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **20220814** | **Ryan Laird** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Ryan Laird

## 1. 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.

Considering Artemis Financials’ need for securing financial data, I would recommend using AES-256 encryption. The larger encryption size will somewhat increase decryption time; however, I would deem this a worthwhile exchange for the increased key complexity (Coffey, n.d.). AES depends on the length of its key for its security. A longer key increases the time and resources it would take to crack it (Manico & Detlefsen, 2014). This is important as AES is symmetric encryption; therefore, the private key is held by both ends of the data transfer (Manico & Detlefsen, 2014).

Hashing algorithms serve to convert text information into a set-length string of characters. This is commonly used to verify that the information you’re reading hasn’t been altered at any point since, if it were, the resulting hash string would no longer match what was provided (Crane, 2021). Within AES, multiple bit levels can be used. In this case, we are utilizing AES-256. This bit level determines the length of the key used, and as a result, the higher the bit level, the more resistant to brute force attacks the key will be.

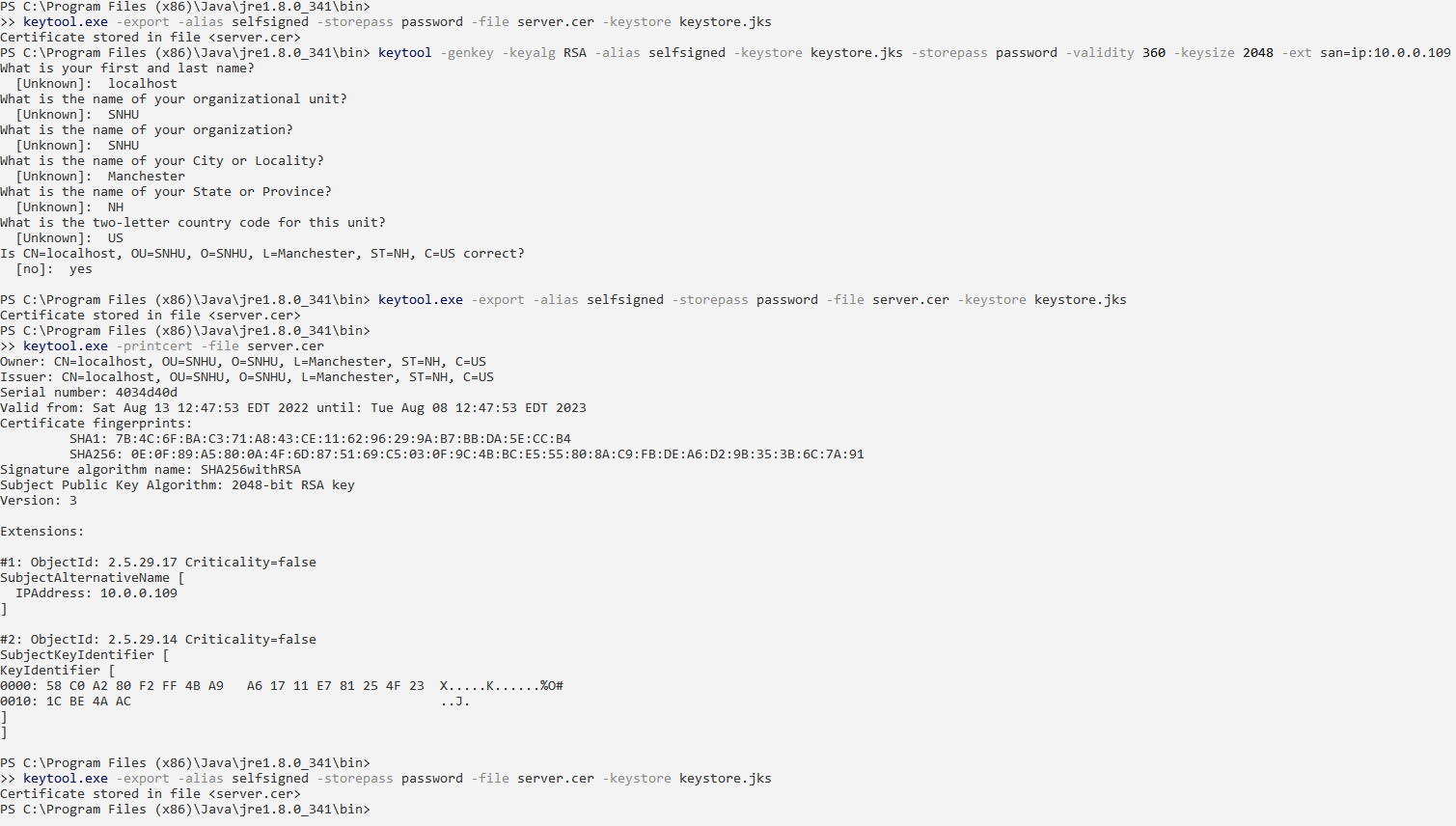
With any form of digital cryptography, the generation of random numbers is critically important. When generating a hash or encrypting data, random numbers serve to introduce unpredictable chaos into the generation of keys (Sunar, 2009). If an attacker can predict the sequencing of these random numbers at any point, the key should be considered insecure as the attacker would be able to predict the key itself (Manico & Detlefsen, 2014). The keys themselves come in two forms, symmetric and asymmetric keys. AES is symmetric encryption, and therefore a private key is held by both the sender of the data as well as the recipient. This private is used to both encrypt and decrypt the data in question (Manico & Detlefsen, 2014). On the other hand, asymmetric encryption operates by having both a public and a private key. The private key encrypts the data, while the public key verifies that signature before the data is decrypted (Manico & Detlefsen, 2014). Asymmetric encryption lowers the possibility of the private key being leaked as it is only held by one side of the transaction.

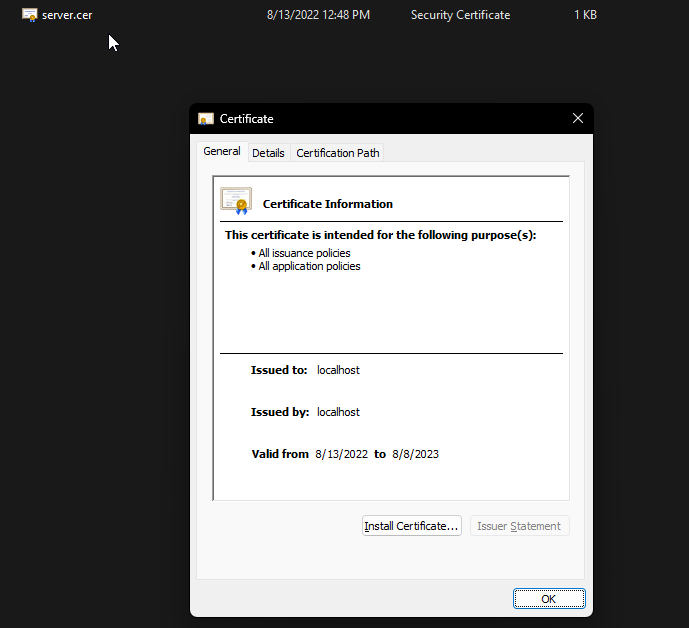
Cryptography in its simplest form can be dated back to ancient Egypt, where hieroglyphs were intentionally obfuscated in order to disguise the true meaning. Cryptography would remain simplistic until World War One, where cryptography became an instrumental tool to disguise information being sent across an army (Damico, 2009). Cryptography continued to adapt to the modern age and grew in complexity with digital encryption. The current standard for encryption is AES, a symmetric encryption standard utilized to secure data within the government (Dworkin et al., 2001).

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

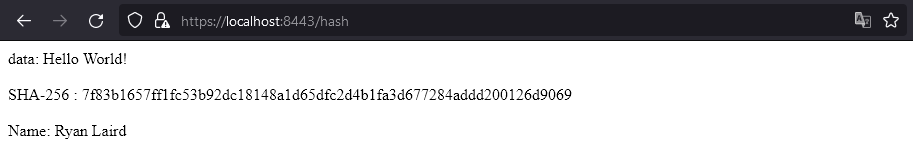




## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

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

Graphical user interface, text, application

Description automatically generated

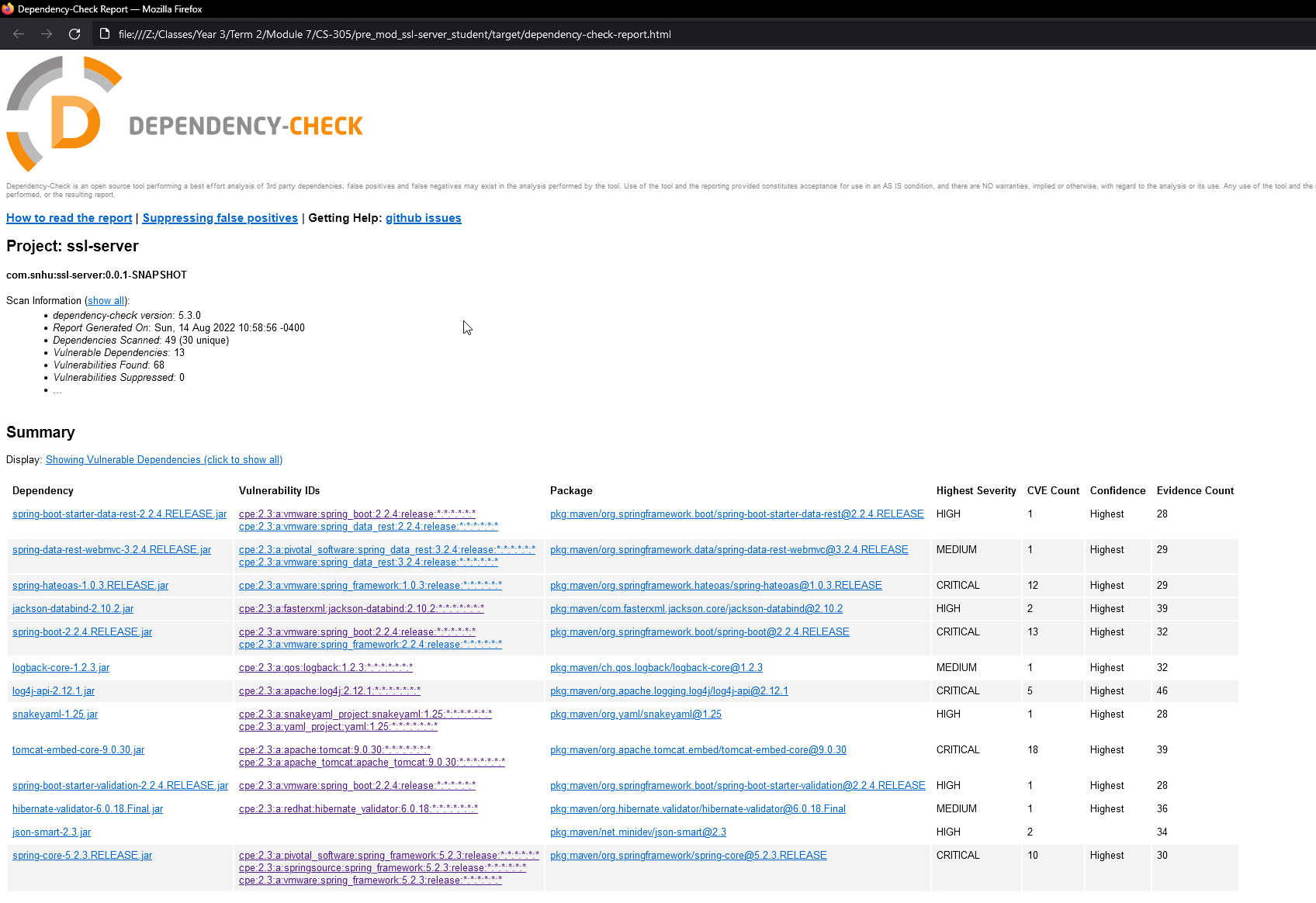
I attempted to get my browser to trust the certificate I generated, but it gave back an error related to the certificate being self-signed. I could connect through HTTPS, but the browser did not see the certificate as trustworthy.

## 5. Secondary Testing

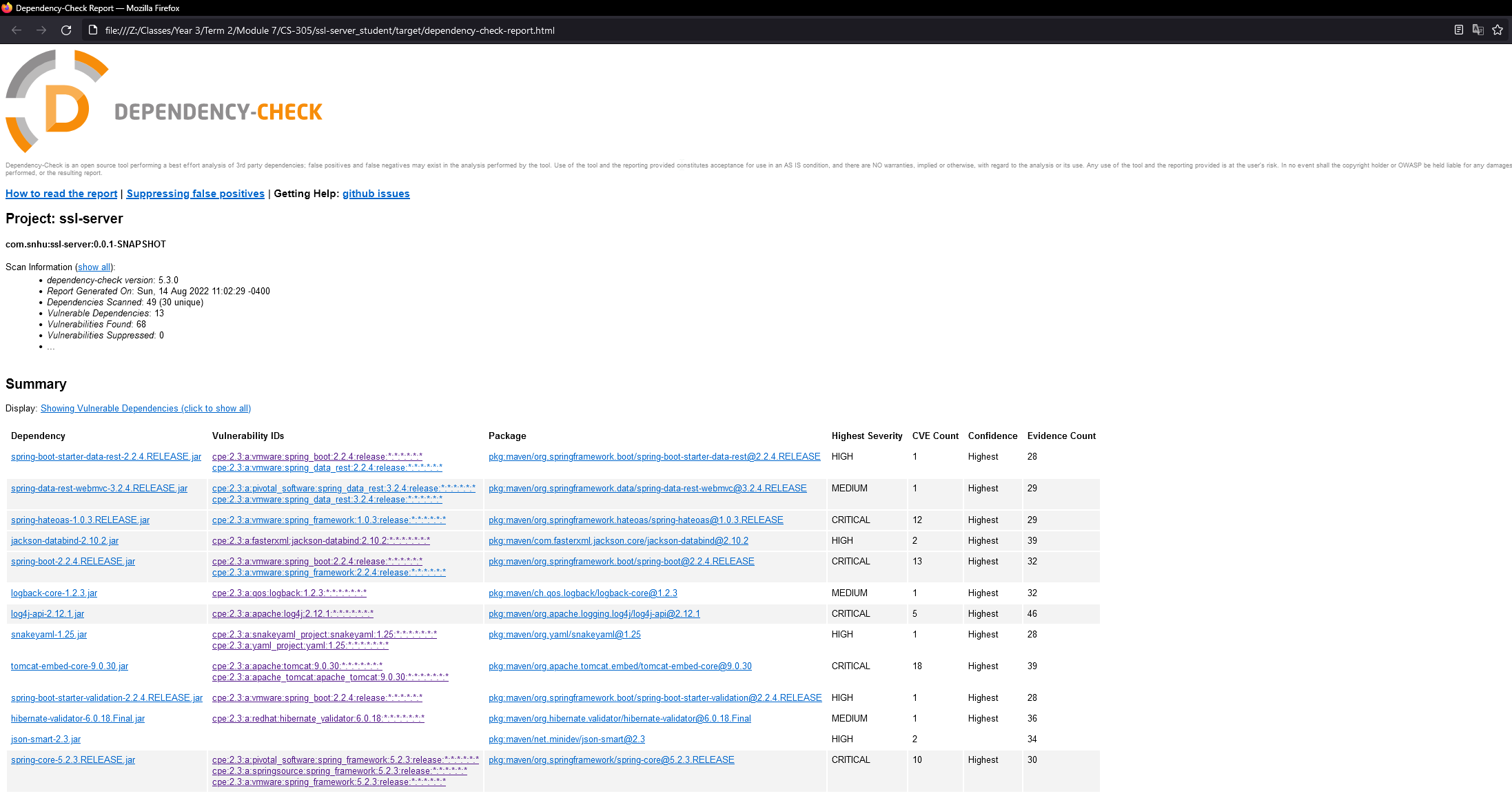
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

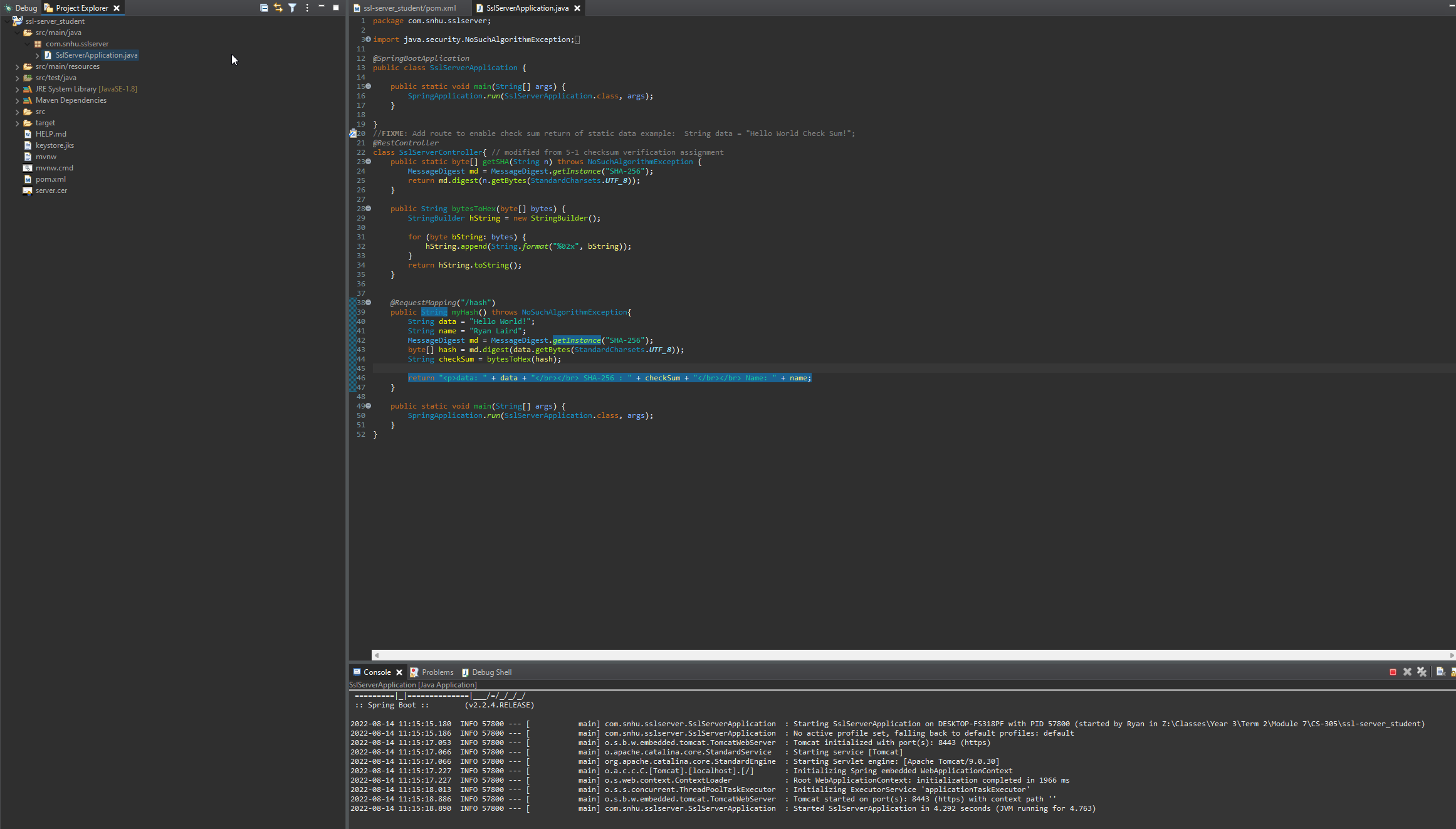
Pre-refactoring dependency check



Post-refactoring dependency check



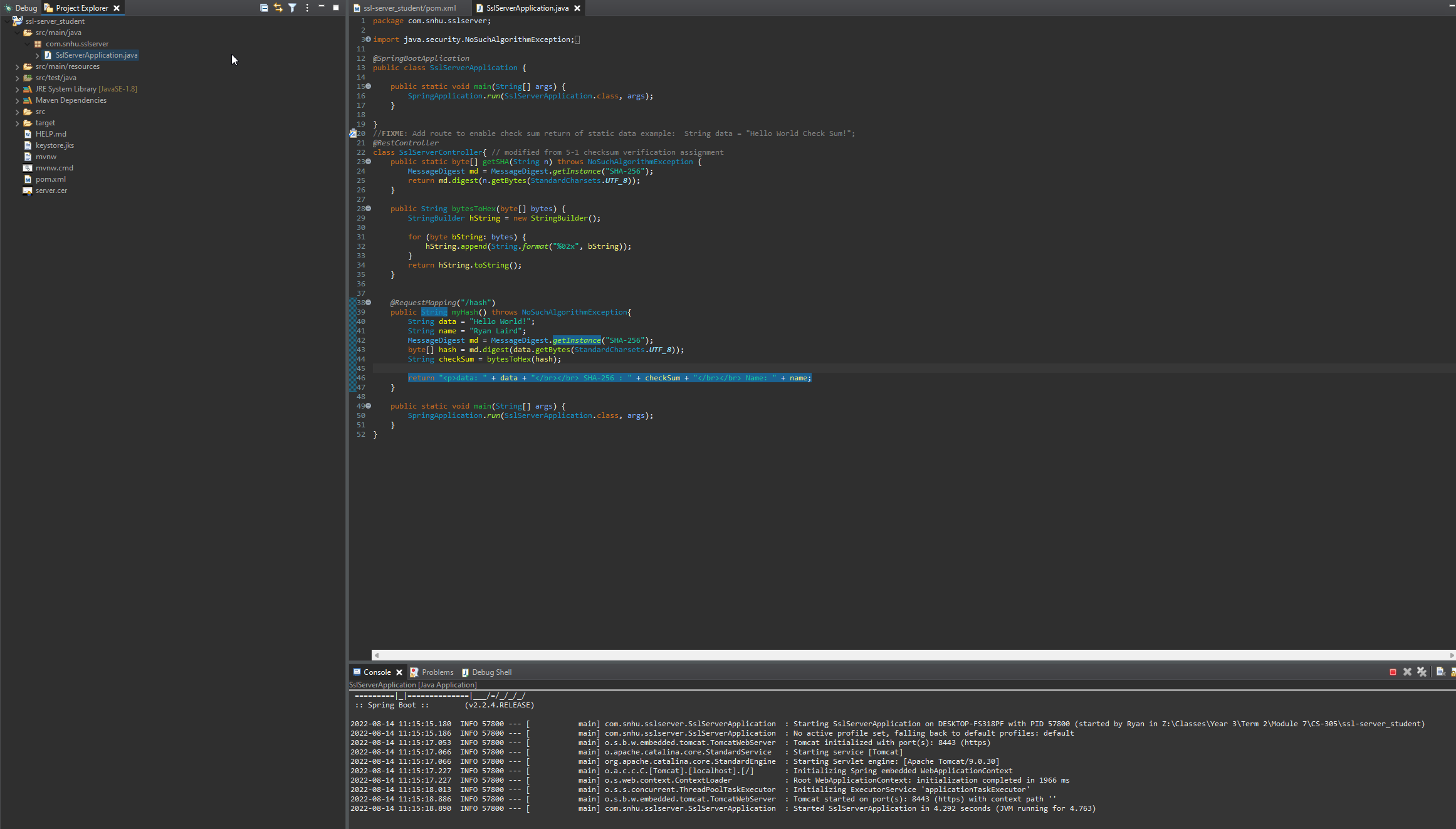
Refactored code

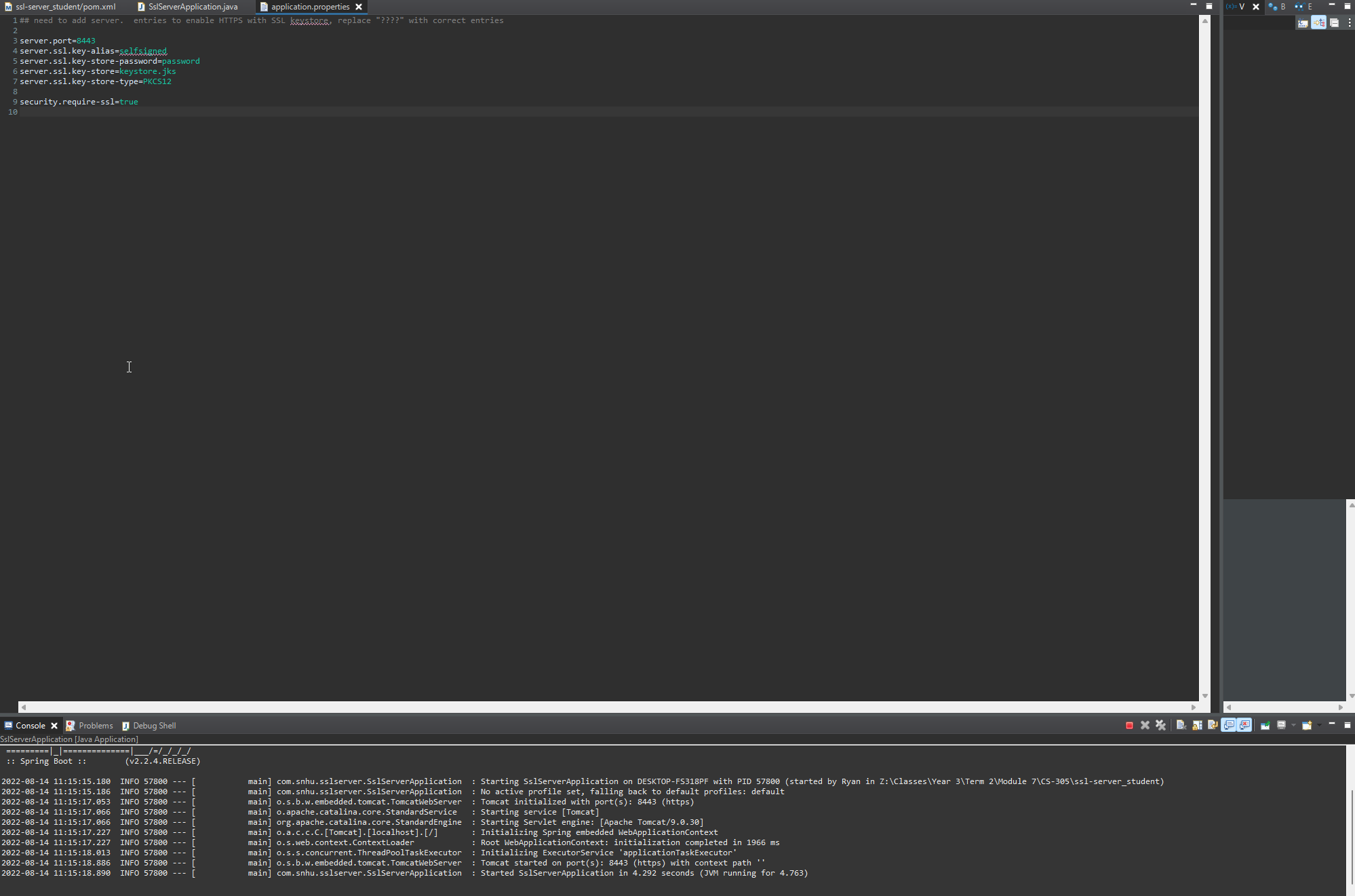


## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.





The core issue I see within the code is the use of plaintext fields for data. For example on line 5 of application.properties, the password is in plaintext and anyone with source code access can easily obtain the password for the certificate. In addition, on lines 40 and 41 of SslServerApplication.java The data string and my name are in clear plaintext. To resolve these issues, I would recommend utilizing Jasypt. Jasypt is a library that allows for text encryption within the source code itself and would allow for the removal of the plaintext secure text within the code (*Java Simplified Encryption - Jasypt: Java Simplified Encryption - Encrypting Application Configuration Files*, n.d.).

## 7. Summary

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

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

The refactoring of the code focused on cryptography, Input validation, and client/server interactions. Part of the refactoring process was to generate a hash from plaintext data through cryptography. This process of checksum verification serves as a system to validate input by checking the generated hash against what data is being shown. The other portion of the code refactoring was designed to allow for the use of a certificate to validate client/server interactions.

I took these two refactoring steps separately. I had experience with generating a checksum for this purpose from a previous assignment, so I accomplished that task first. Checksum verification like what is seen in the SslServerApplication is important to a business that wishes to validate that the info they are seeing is unmodified. Generating a checksum server side allows for the recipient to compare the data they received with the checksum provided to ensure that a man-in-the-middle attack did not alter the data in some way (Crane, 2021). On the other hand, utilizing certificates to properly utilize HTTPS protocol is crucially important to any business that operates through the web. A certificate allows the client to verify that the website they are seeing is verified as the true website and therefore can be trusted.

In order to maintain the security of the code base, the recommendation to encrypt the plain text fields using Jasypt should be taken into consideration (*Java Simplified Encryption - Jasypt: Java Simplified Encryption - Encrypting Application Configuration Files*, n.d.). In addition, many vulnerabilities already existed within the code base prior to refactoring, which should be addressed. Many of these vulnerabilities stem from out-of-date frameworks and libraries. In order to remedy this, packages should be updated to their latest secure versions as soon as possible and should be kept updated throughout the development process.

**References**

Coffey, N. (n.d.). *RSA key lengths*. Javamex. https://www.javamex.com/tutorials/cryptography/rsa\_key\_length.shtml

Crane, C. (2021). *What is a hash function in cryptography? A beginner’s guide*. The SSL Stpre. https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/

Damico, T. M. (2009). A brief history of cryptography. *Inquiries Journal*, *1*(11).

Dworkin, M., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., & Dray, J. (2001). *Advanced encryption standard (AES)*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD. https://doi.org/https://doi.org/10.6028/NIST.FIPS.197

*Java Simplified Encryption - Jasypt: Java simplified encryption - encrypting application configuration files*. (n.d.). Jasypt. http://www.jasypt.org/encrypting-configuration.html

Manico, J., & Detlefsen, A. (2014). *Iron-Clad java: building secure web applications (oracle press)*. McGraw-Hill Education Group.

Sunar, B. (2009). True random number generators for cryptography. In *Cryptographic Engineering* (pp. 55–73). Springer.