****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_1fob9te)

[Client 3](#_3znysh7)

[Instructions 3](#_2et92p0)

[Developer 4](#_tyjcwt)

[1. Algorithm Cipher 4](#_3dy6vkm)

[2. Certificate Generation 4](#_1t3h5sf)

[3. Deploy Cipher 4](#_4d34og8)

[4. Secure Communications 4](#_2s8eyo1)

[5. Secondary Testing 4](#_17dp8vu)

[6. Functional Testing 5](#_3rdcrjn)

[7. Summary 5](#_26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/08/2022** | **Chris Blair** |  |

## 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

Chris Blair

## 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.

[Include your findings here.]

I chose AES for my encryption cipher because it is one of the most secure and commonly used. It will work with a variety of different key sizes and it is extremely hard to crack without the proper key. It is nearly unbreakable, as it would take much much longer than the life of any human to even come close to brute-forcing.

AES uses substitution permutation, which basically means the data is passed through several rounds of encryption. There is also the aspect of bit-shifting, where in each round, the columns and tables of bits are shifted around and mixed to further enhance the encryption strength. AES-256 has 14 cycles.

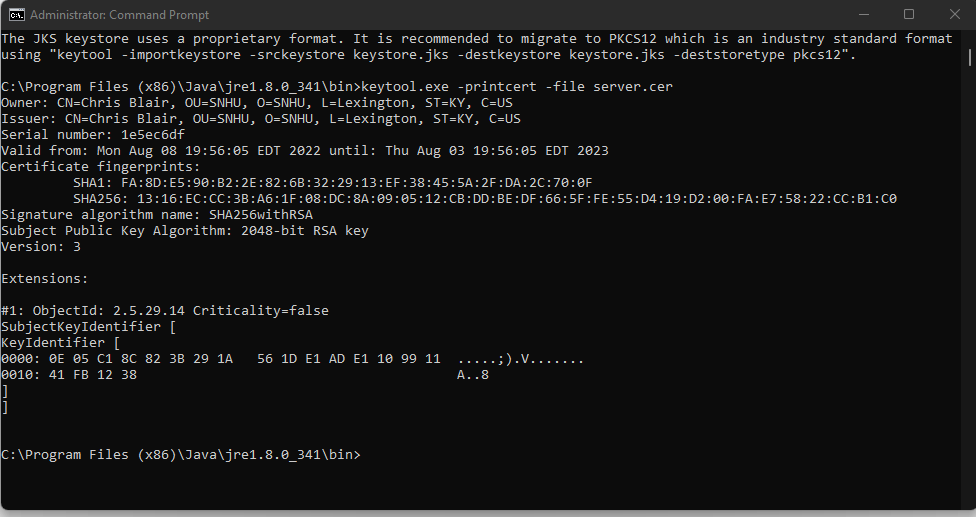
AES uses symmetric keys which means the same key used for encryption is used for decryption. This is good for our application because we want the server to be able to communicate the key with the client so that the client can successfully decrypt and read the contents of the encrypted data.

AES was first published in 1977, so it’s definitely been around long enough to prove itself as a viable option. It was also announced by the NIST in 2001, NIST is a government agency that handles technology standards and security, as well as became a US federal standard in 2002. It is also the only publicly available cipher that has been approved and used by the NSA. With this track record, you can see why it is widely used and one of the most common and secure algorithm ciphers available.

## 2. Certificate Generation

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

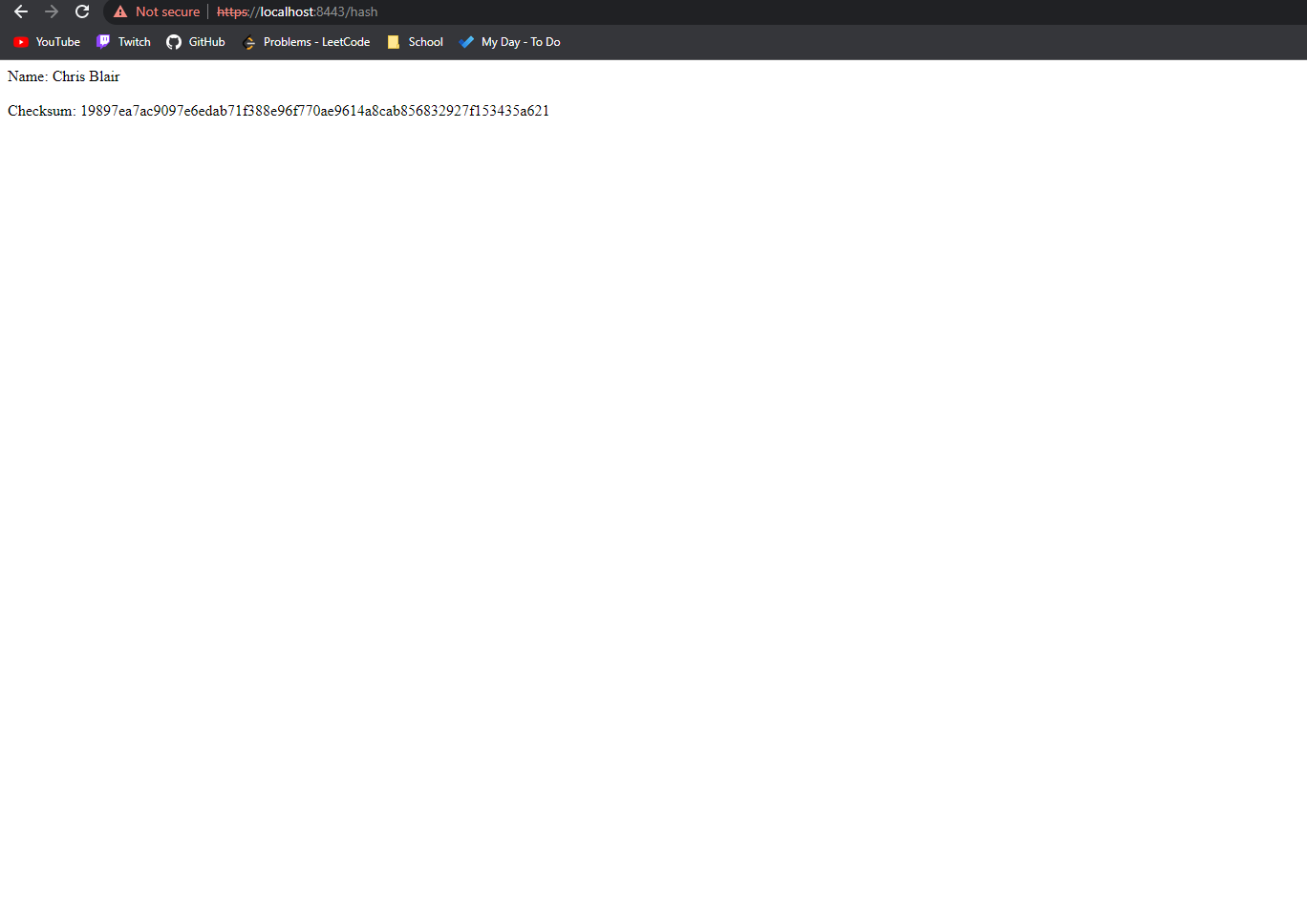
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

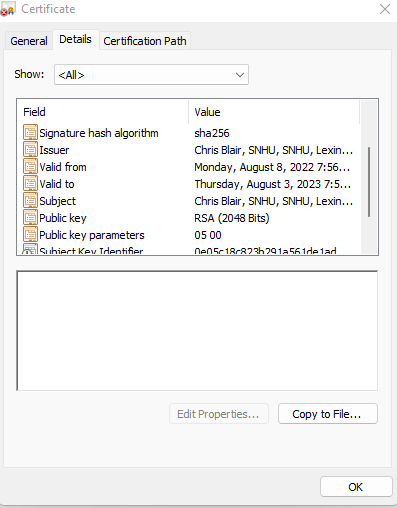
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.



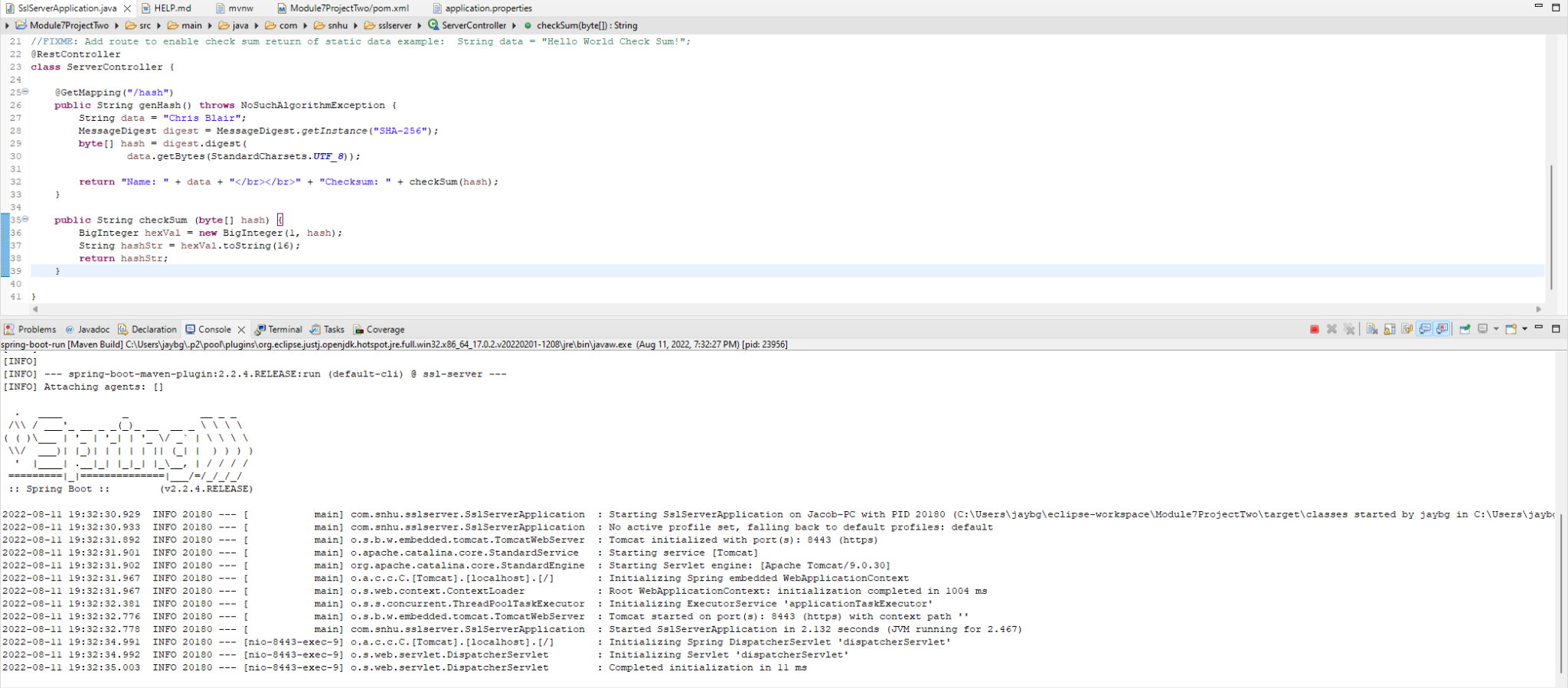
\* As you can see here, I have a certificate installed in my browser using sha256, this allows for secure communication between the client and the server.

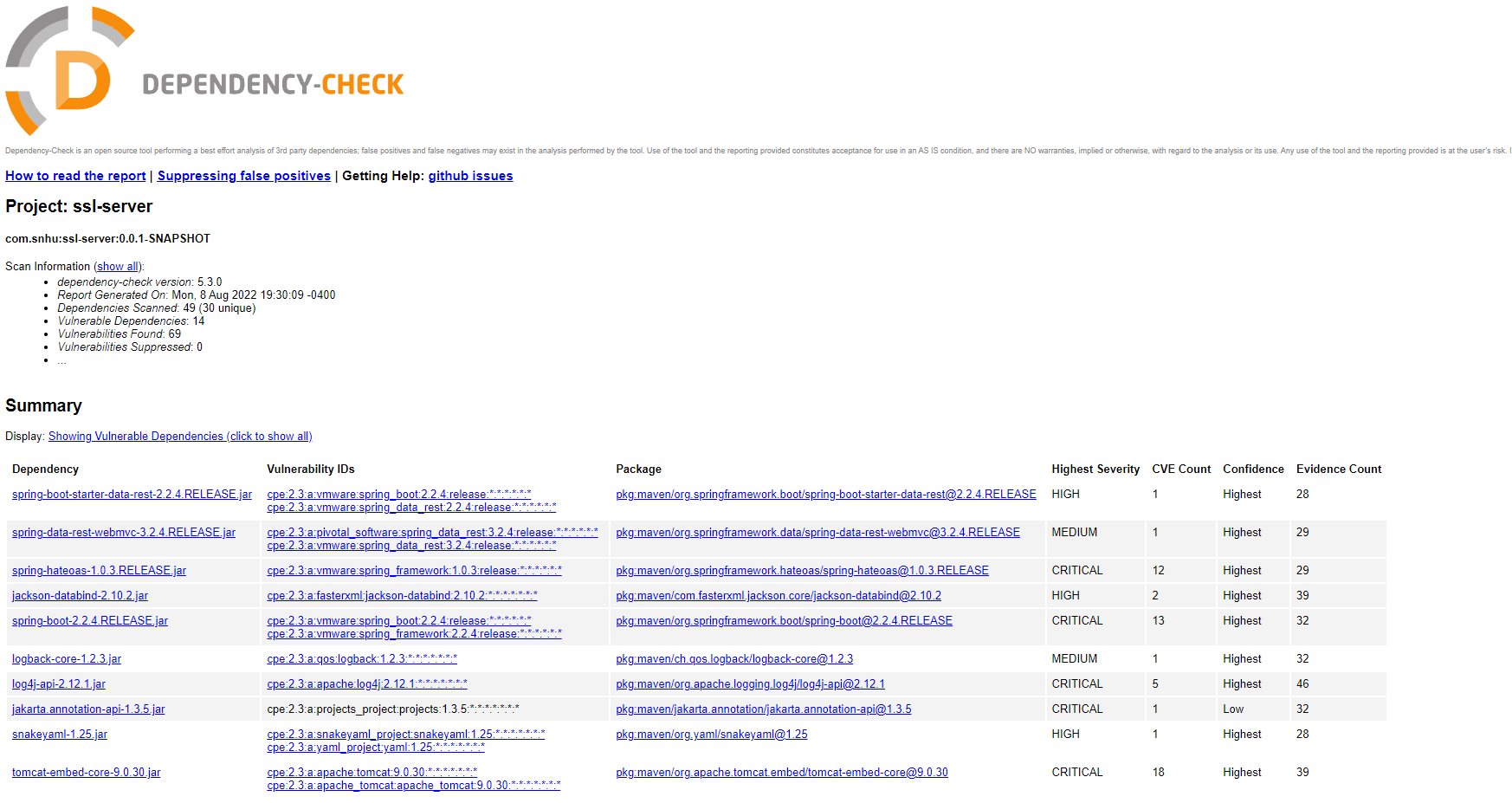
## 5. Secondary Testing

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

[Insert screenshots here.]



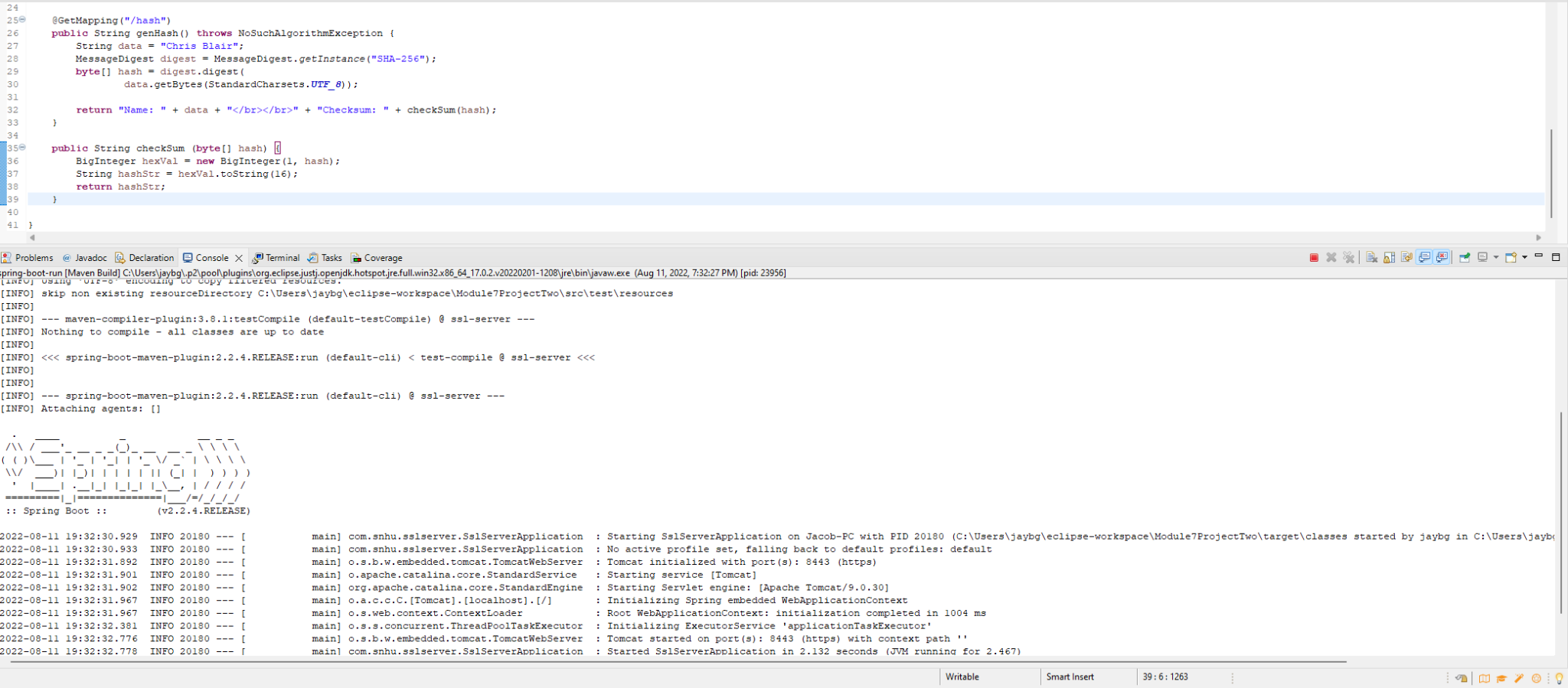


## 6. Functional Testing

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.

[Insert screenshot(s) here.]



## 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.

[Include your findings here.]

For my project, I have added security layers via SHA-256 hashing, which provides secure communication across client/server. Also I have added an SSL certificate so we can ensure a secure communication transfer between the client and the server from the client side. Together, these should provide sufficient security to be able to comfortably send data to and from the client and the server without worrying about brute-force attacks intercepting and stealing the data.

What we have not addressed however is the ability for anyone to access the system. If we have profile information setup for individual users, we will need to provide protection from gaining access to that user's profile to get their personal information. We can do this by setting up 2FA via an external authenticator application or via text message/phone call communications. This will make sure only those who need access to the application can get it.

As developers, we often overlook and underestimate the importance of security in the application we’re working on. I find myself sometimes only being concerned about the feature itself and not necessarily the impact on security or even sometimes the larger application as a whole. If every developer could take a few minutes and think about better ways to secure their code while they’re writing it, it will have a compounding effect on the entire application and make it vastly more secure as a result.

Maintaining the security from a customer perspective is one, making sure no unauthorized users have direct access with the application in the first place. As mentioned before, we can do this with 2FA. When setting up user access, we can also make sure that we create user tiers or roles to ensure access to certain areas are only given to the needed parties. Security is a never-ending battle so it needs to be continuously monitored and maintained/updated in order to ensure security at all levels. This means doing consistent dependency checks and audits of the software.