****

# 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** | **06/15/21** | **Ryan Mackenzie** |  |

## 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 Mackenzie

## 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.

For Artemis Financial we should utilize ECDHE\_ECDSA\_WITH\_AES. It will allow for certificate verification, which prevents impersonation attacks. We can verify the hostnames on the server and the certificate match and are not expired. It also utilizes public and private keys, which is another way of establishing a trusted connection between two devices. The federal government also uses AES as a Data Encryption Standards. It provides protection called forward secrecy, which will prevent your data from being decrypted even if the private keys are compromised later (Manico, Detlefsen, 2014).

The Cipher will use a hash function to generate a hash of the key. This hash is then used to tell whether the connection is trusted using a trust manager. The bit levels of the cipher indicate the size of the keys, which are typically 128 or 256 bits.

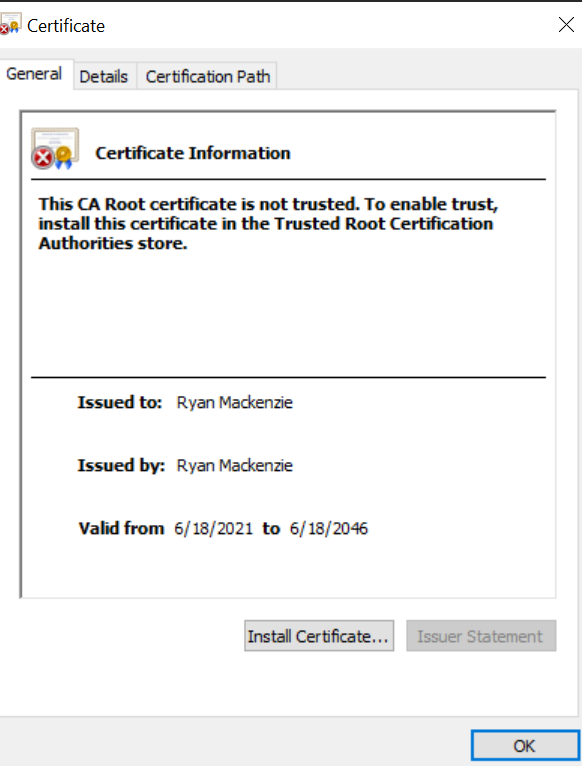
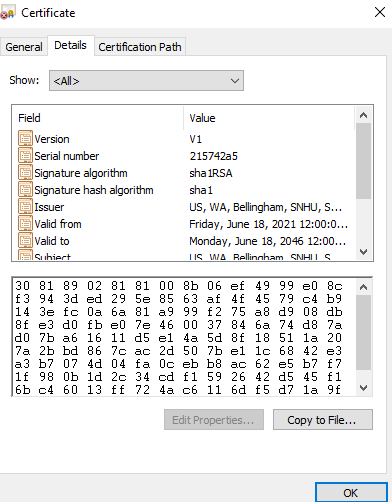
Cryptography uses a pseudorandom number generator to create keys in a way that cannot be predicted. Symmetric cryptography is used for encryption whereas asymmetric cryptography is used more for signing. Signing only hides the sender of the data but does not necessarily protect the data itself from being looked at. In symmetric cryptography a key is created which makes the data impossible to read without. The same key is used which is what makes it symmetric.

The first cryptographic protocol widely adopted is the Secure Socket Layer in 1995 by a company called Netscape. (Manico, Detlefsen, 2014) SSL had several weaknesses that began to appear over time, so a new protocol was developed in 1999 called Transport Layer Security. TLS is still widely used, and the current version is only 1.2. These protocols established the ability to include keys in machine communication that allowed for additional layers of security using cryptography and ciphers.

## 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.

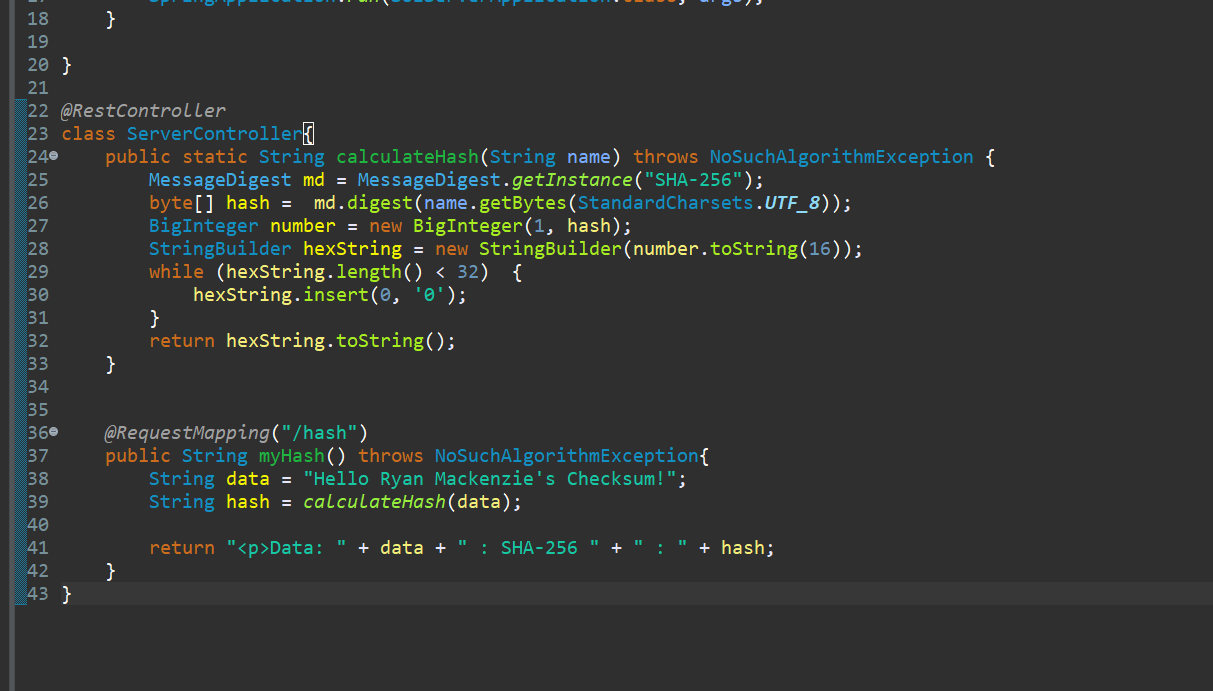
 

## 

## 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.

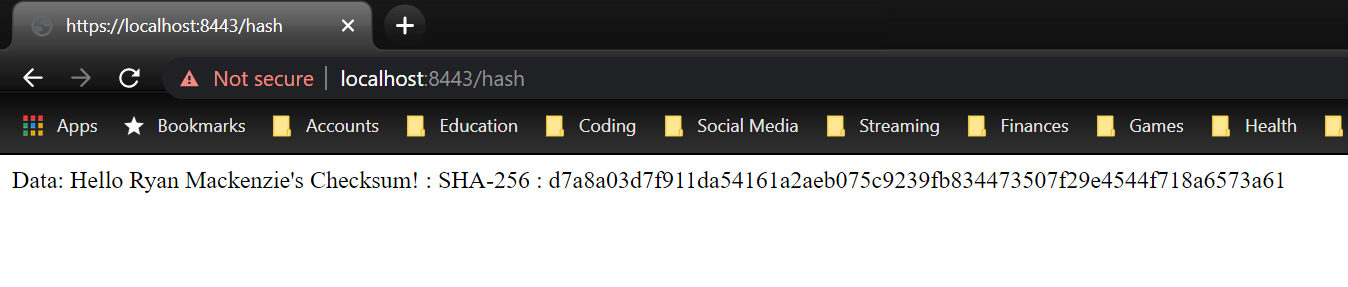


## 

## 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.

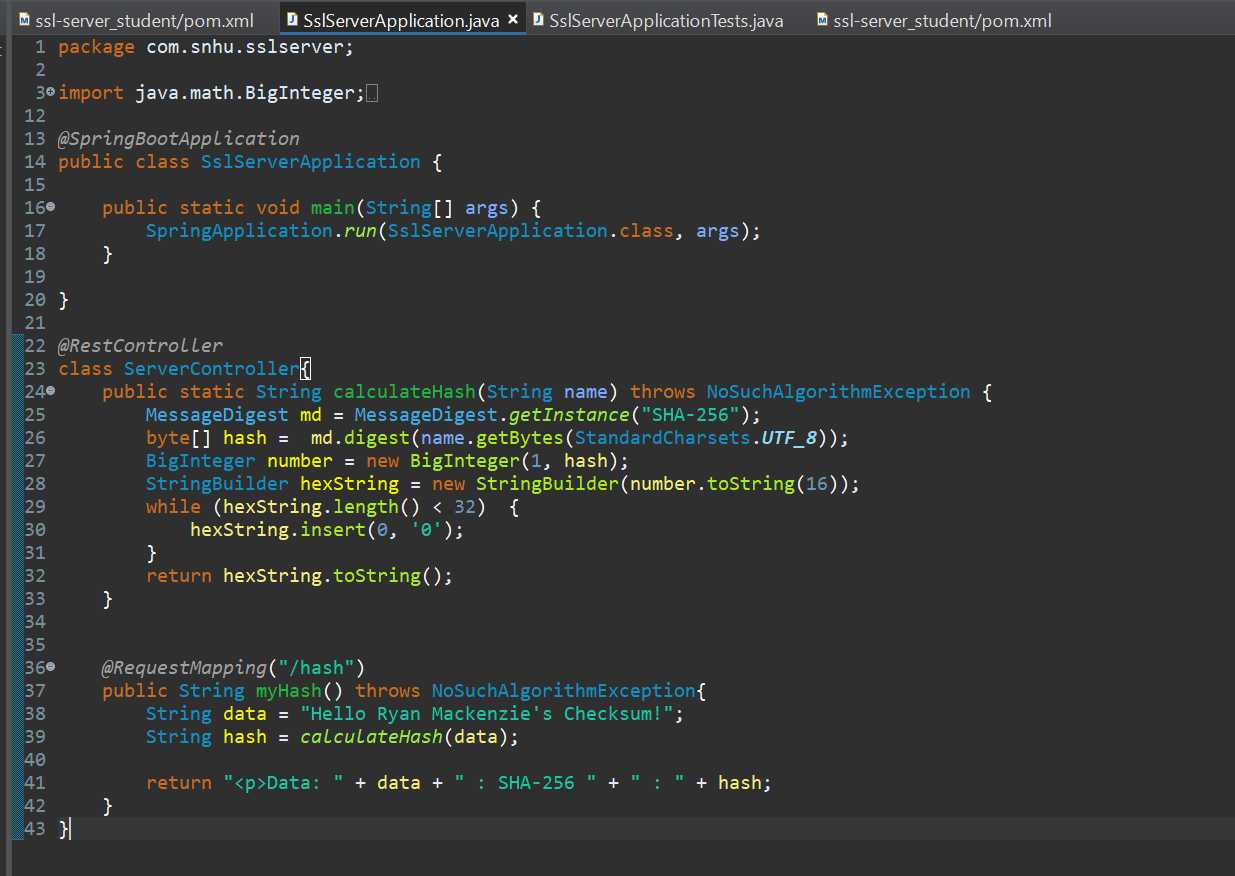


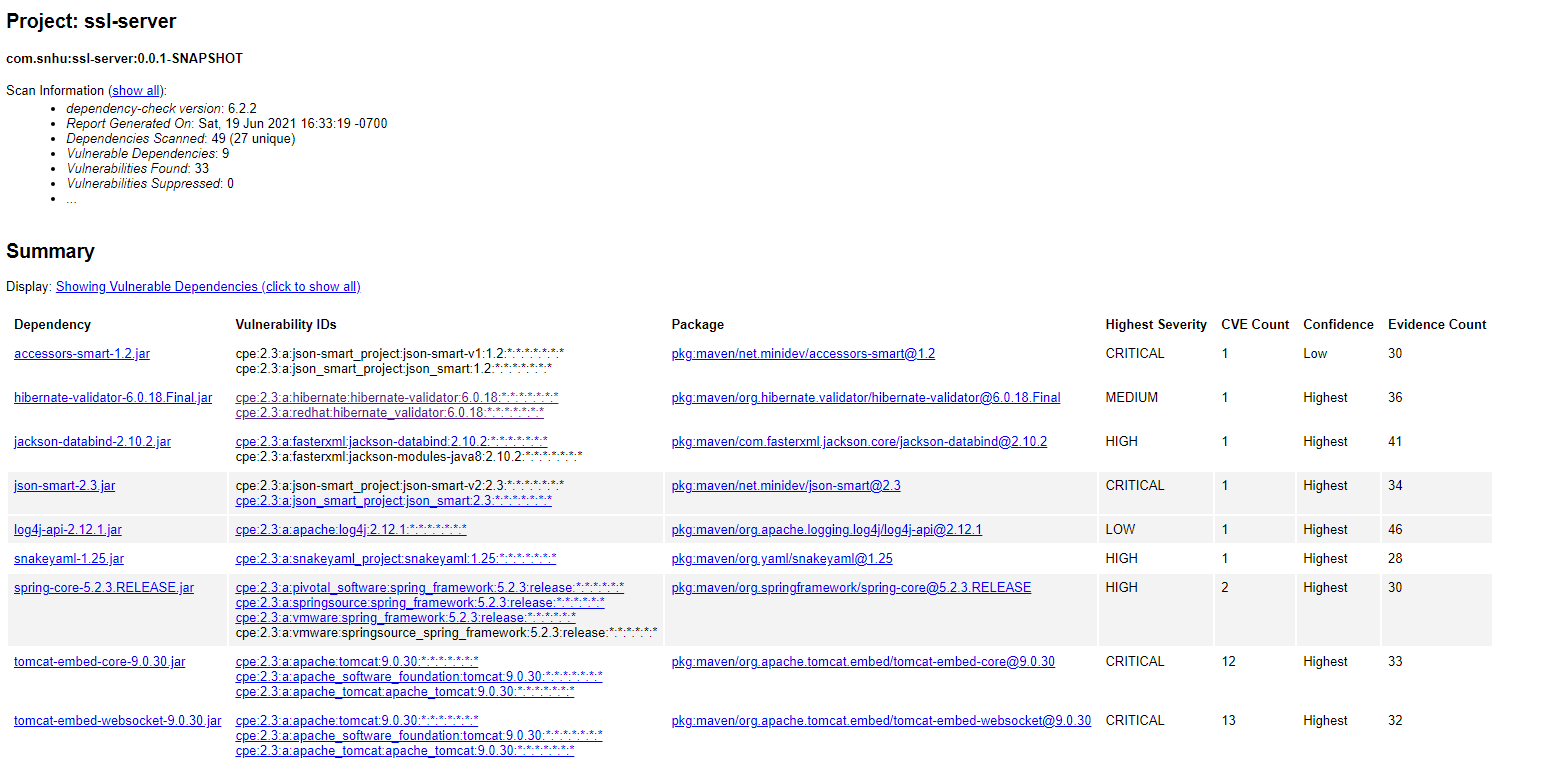
The “Not secure" is caused by using a self-signed certificate, which browsers will not be able to recognize (Kinsta, 2021). When building the actual application, we would go out and get a real certificate from a trusted authority.

## 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

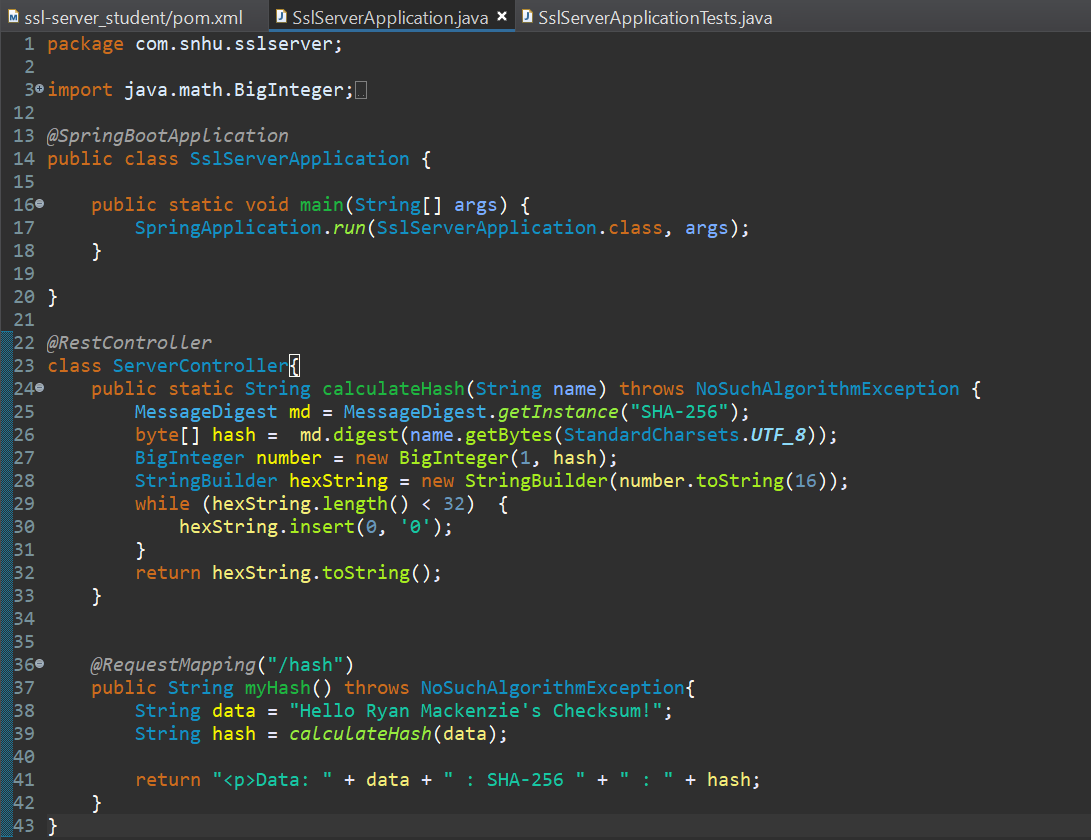


All vulnerabilities were present before code was refactored:  


## 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.



## 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.

There were multiple areas of security that we addressed with our program. We are utilizing cryptography to generate the SHA-256 key that is used by the checksum. We are also validating client server interactions by using a certificate to show that the clients have connected to the correct address. We validated that there were no code errors in our code. Code quality was ensured by our static testing where we validated that no new vulnerabilities were introduced by refactoring the code.

By refactoring the code, we were able to add the ability to check for certificates as well as a checksum. Using certificates allows for the browsers that customers will use to show that we are a trusted website that they do not need to worry about. If your service does not use valid certificates, it will show customers that your site is not secure. We also added a layer of security called a checksum that allows you to verify software if you need to distribute it to customers. It uses a SHA-256 hash function that minimizes any chances of a collision. A collision is when the hash function outputs the same value for two different pieces of data. The probability of an accidental collision with a SHA 256 is extremely unlikely. “The probability of just two hashes accidentally colliding is approximately: 4.3\*10-60“(Ramirez, 2021). SHA 256 is a very widely used algorithm for file hashing currently. A cipher that is susceptible to collisions can be victim of a collision attack. This type of attack specifically tries to cause a collision with the same hash value and can be prevented using SHA 256.

Going forward, the best practices for maintaining the security for Artemis Financial will be to continue to run the dependency checks on regular intervals. That way, if any new vulnerabilities are found with your software, you will be aware. This will allow you to also take action to correct the vulnerabilities if a solution has been found.

## 8. References

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Java*. O'Reilly Online Learning. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/>.

*How to Fix the NET::ERR\_CERT\_AUTHORITY\_INVALID Error*. Kinsta. (2021, June 18). https://kinsta.com/knowledgebase/neterr-cert-authority-invalid/.