****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[\_Toc33111301](#_Toc33111301)

[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** | **6-18-2022** | **Jacob Muelelr** |  |

## 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

Jacob Mueller

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

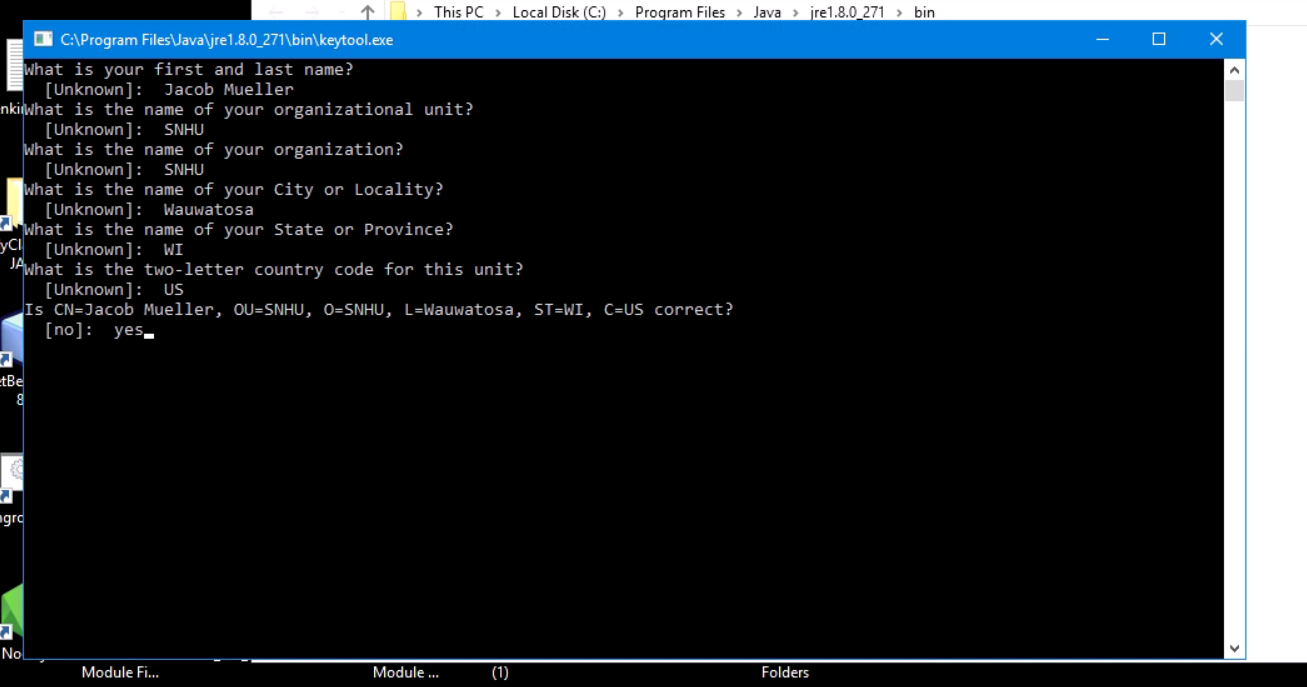
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

The customer, Artemis Financial, requires the increased security that can only be offered by the correct algorithm cipher. In my opinion, and the opinion of many others, SHA-256 is the correct option. It is asymmetric, meaning that the keys used to send information out is different from the keys used to send information back. SHA-256 provides a powerful level of protection with 256 bit long encryption and randomized numbers used in the encryption making it very nearly impossible to reverse the encryption.

## 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.



My keytool wasn’t staying open at this time, I was able to do the first part as normal from the module 5 directions and the resources provided for the project, but when I moved to export my certs and set a password it wouldn’t stay open to complete this section. So, at this time this is all that I’m able to provide as far a proof for my certificate generation.

## 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.

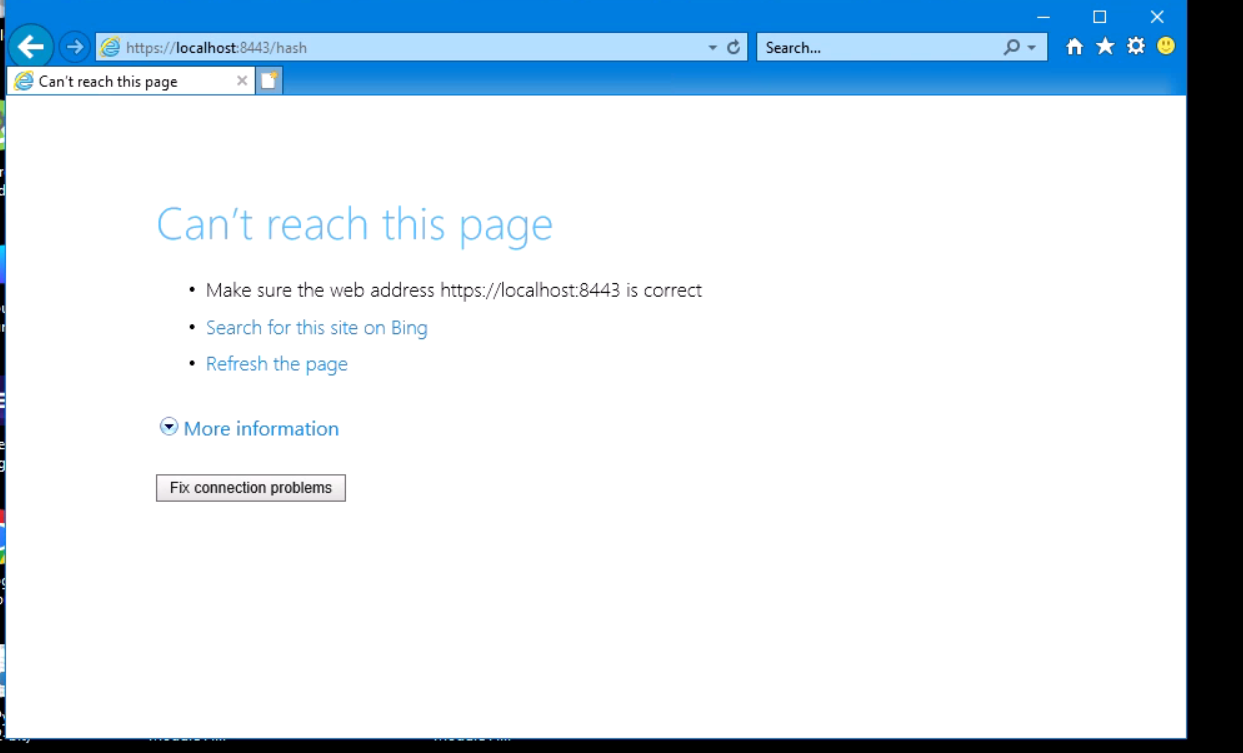
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

The controller wouldn’t work for this and I was unable to generate anything to screenshot.

## 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.

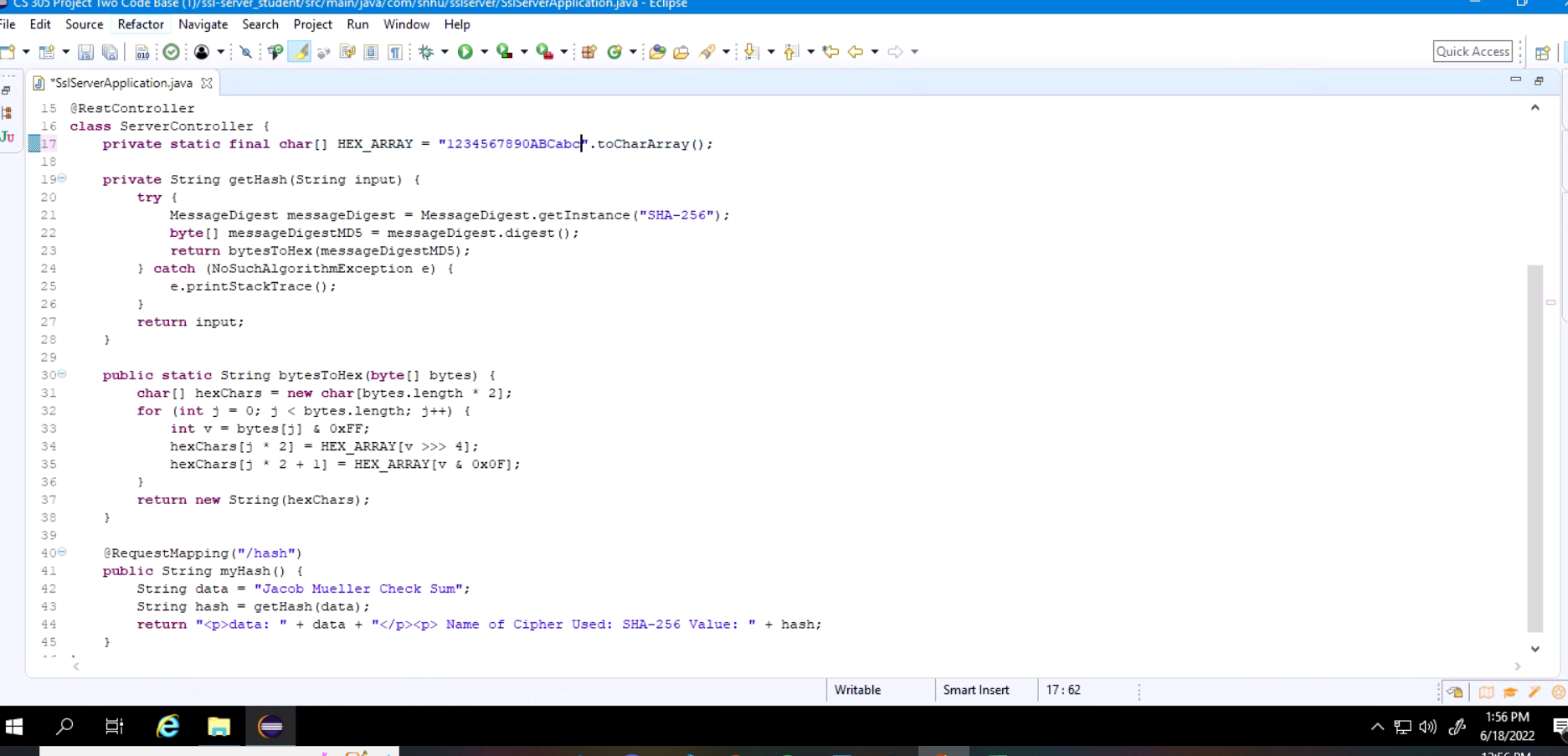


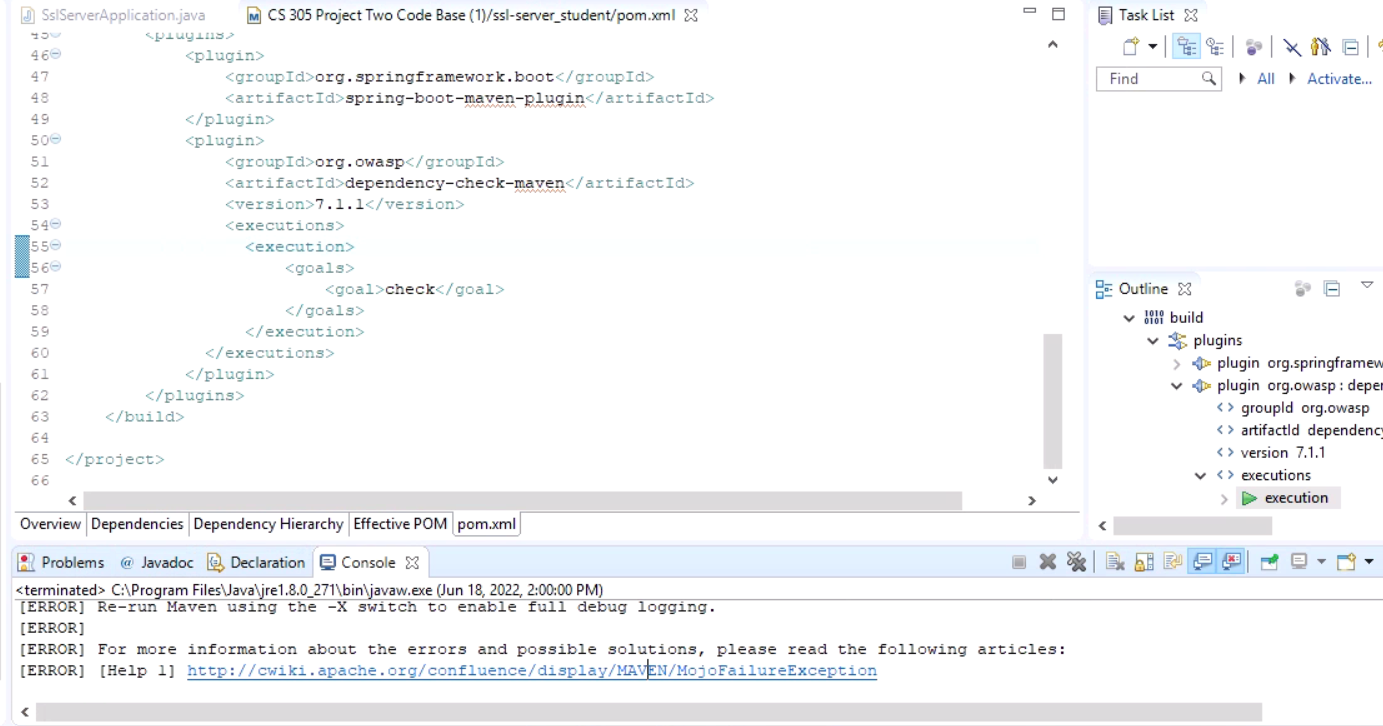
There were also issues here due to the persisting issue from earlier and the fact that it wasn’t able to be reached. I found some resources online at stack overflow that I will reference and use in a follow up attempt.

## 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





## 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.]

package com.snhu.sslserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SslServerApplication {

public static void main(String[] args) {

SpringApplication.run(SslServerApplication.class, args);

}

}

//FIXME: Add route to enable check sum return of static data example: String data = "Hello World Check Sum!";

@RestController

class ServerController {

private static final char[] HEX\_ARRAY = "1234567890ABCabc".toCharArray();

private String getHash(String input) {

try {

MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");

byte[] messageDigestMD5 = messageDigest.digest();

return bytesToHex(messageDigestMD5);

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

return input;

}

public static String bytesToHex(byte[] bytes) {

char[] hexChars = new char[bytes.length \* 2];

for (int j = 0; j < bytes.length; j++) {

int v = bytes[j] & 0xFF;

hexChars[j \* 2] = HEX\_ARRAY[v >>> 4];

hexChars[j \* 2 + 1] = HEX\_ARRAY[v & 0x0F];

}

return new String(hexChars);

}

@RequestMapping("/hash")

public String myHash() {

String data = "Jacob Mueller Check Sum";

String hash = getHash(data);

return "<p>data: " + data + "</p><p> Name of Cipher Used: SHA-256 Value: " + hash;

}

}

## 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.

Having a secure rest controller is important, so that was added to mitigate any concerns there. I chose the SHA-256 hashing cipher because it is nearly impossible to break by outside parties, who will be the main security concerns for the customer. Adding layers of security to the software is a long process and requires diligence and an eye for detail. Checking, rechecking, and addressing any issues is the best policy and that is what has been taught throughout the course. I would say that at the very least I have seen companies address security concerns every quarter, but with the way most financial institutions keep in contact with customers I would recommend revisiting security once a month for a quality checkup. Similarly some of the security companies or groups that oversee the factors I implemented update every three months, so it would also be prudent to check that all security measures are up to date so that there aren’t any glaring holes left open in the background.