****

# 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** | **09 Dec 2021** | **Gunnar Dulle** |  |

## 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

Gunnar Dulle

## 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.

As a financial institution we know that Artemis Financial will need to comply with the Graham-Leach-Bliley Act and the PCI-DSS so ensuring that there is strong security of their communications is a key part of complying with these. To ensure that Artemis Financial was using the best cipher possible for their needs I selected SHA3-256. SHA-3 was added to the NIST standards list in 2015 after successful attacks on several legacy encryption standards led the NIST to investigate additional algorithms to have additional dissimilar algorithms to choose from. SHA-3 uses sponge construction and Keccak permutation this structure benefits from a non-symmetric key that is created by the sponge structure “absorbing” the original message. SHA3-256 has a 256 bit output size, the Keccak permutation will make a pseudo-random number that is then taken into the sponge construction which further adds to the security of SHA-3. SHA-3 is the most recent addition to the NIST Secure Hash Algorithm family of standards and as such has had the least amount of exposure in the world and less time for hackers to develop a way to crack it. This helps it to be one of the more secure algorithms available today and an excellent choice for Artemis Financial.

## 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.

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## 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.

Graphical user interface, text, application

Description automatically generated

## 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

## 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

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

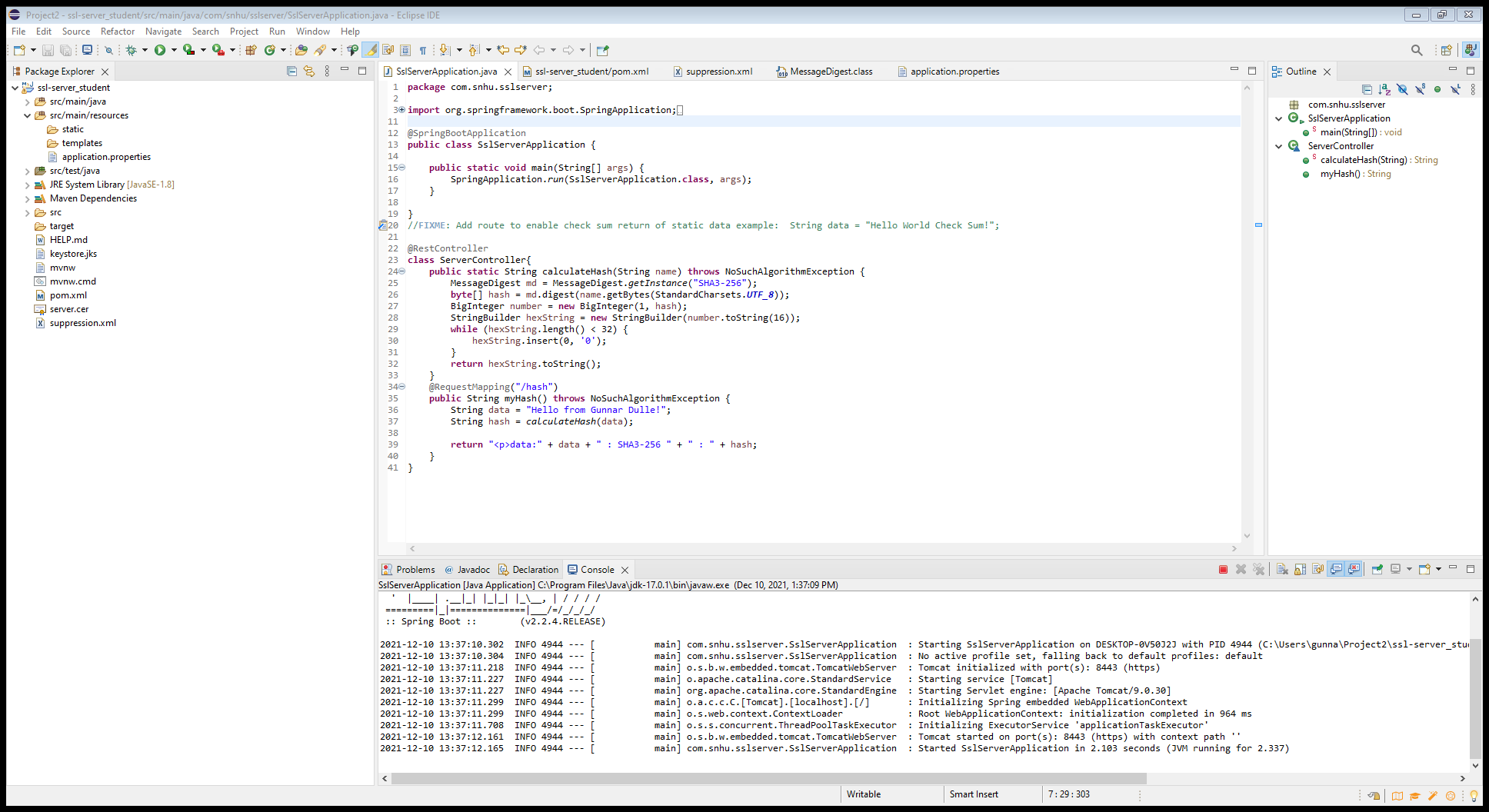
Description automatically generated

In order to ensure that I did not introduce any new vulnerabilities I suppressed the vulnerabilities that existed in the original code before refactoring the 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.



## 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.

In the process of refactoring the code for Artemis Financial I added a secure RestController to ensure that there was a secured controller for my RESTful API. I added the SHA3-256 cipher to ensure that the application had the most secure algorithm cipher for the application in order to secure communication to and from the API with a minimal chance of collision in encryption. I deployed a security certificate to ensure that the application can be accessed via https protocol adding an additional layer of security in communicating with the application. Best practices would be to run a dependency check at least once a month to see if new vulnerabilities from dependencies had been discovered and any updates to these dependencies can be added to ensure that the latest security measures are in place.