****

# 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** | **Aug 10 2022** | **Jaime Rowland** |  |

## 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

[Jaime Rowland]

## 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.

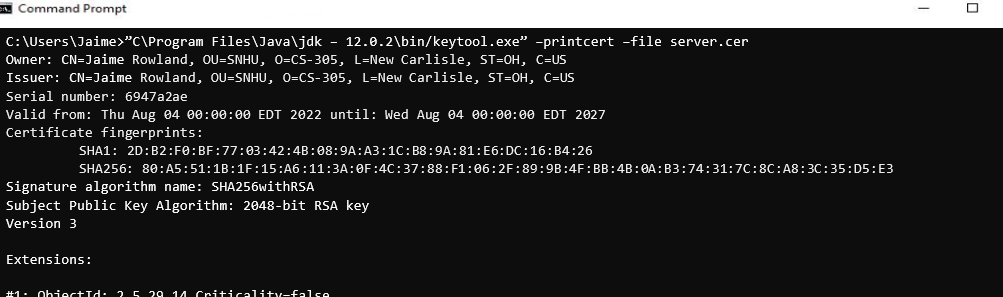
The primary objective of Artemis Financial is to offer financial programs to clients worldwide. I recommend choosing SHA-256 as the encryption algorithm cipher with this particular objective in mind. All information can be safeguarded with this cipher from outside access. It is regarded as one of the most secure algorithm ciphers and is practically hard to break or penetrate. It would take years of brute force to break the cipher. When communicating with financial institutions, SHA-256 is frequently the suggested cipher to employ. The SHA-256 hash function and bit levels are made out of randomness. When creating a hash function, the input value is compressed before being used. The hash value is the name for the compressed data. The length of the encryption is determined by the bit levels.

The quantity of possible encryption combinations determines a 256-bit encryption. It is more difficult for hackers to get unauthorized access to information when random numbers are used. Unpredictability is a result of randomness. Symmetric keys are regarded as one of the most basic encryption techniques. A significant advantage of having a symmetric key it takes less time to execute (Yedakula,K., 2019), it also only requires 1 key. In light of this, AES-256 frequently employs symmetric keys and encrypts plaintext using a key. Because asymmetric keys need two keys, they are thought to be more secure than symmetric keys. Internet communication often uses asymmetric keys. The history of encryption algorithms dates as far back as 600 BC (A brief history of encryption. Thales Group, 2016). With the development of encryption, it is now possible to safeguard data from illegal access. Humanity has always prioritized security and will continue to do so.

## 2. Certificate Generation

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

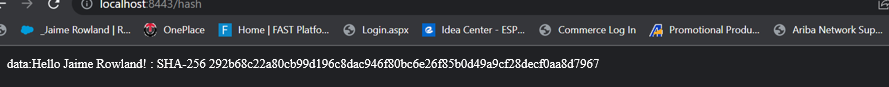
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

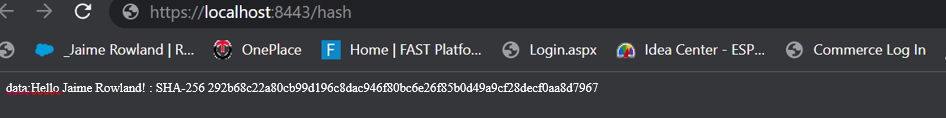
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

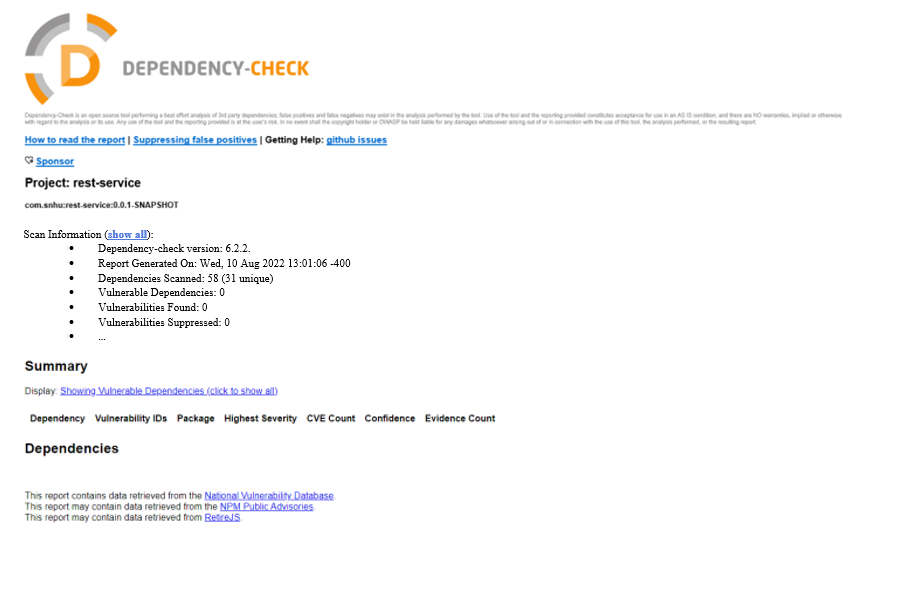
* Insert a screenshot below of the web browser that shows a secure webpage.



## 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.

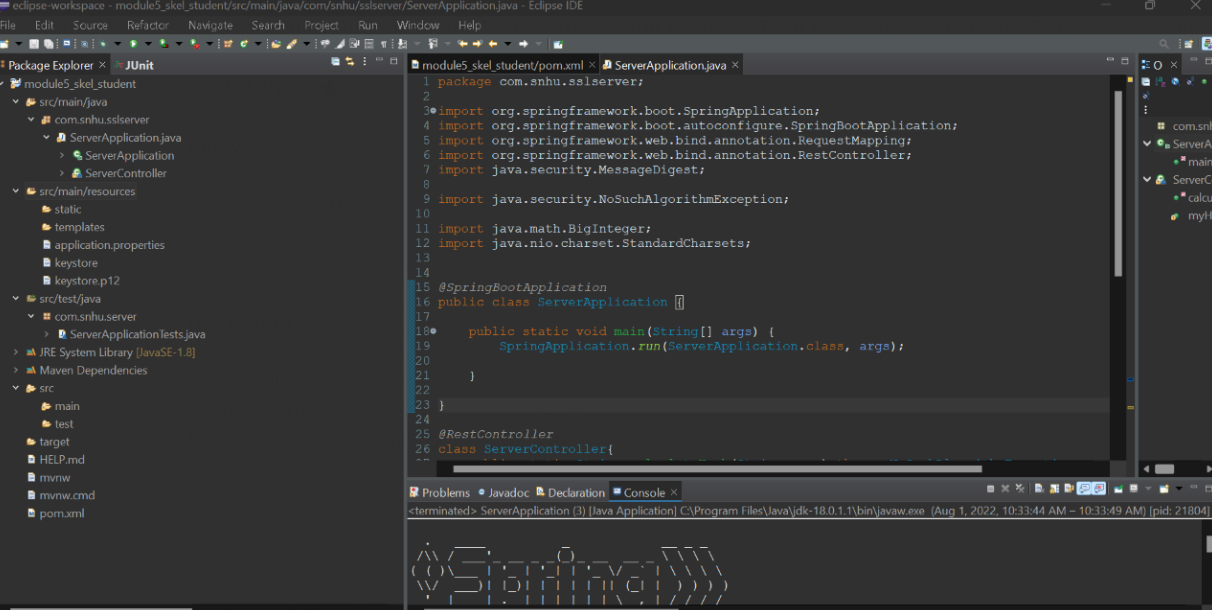
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



## 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.



## 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.

Our application's self-signed certificates, which enabled the usage of HTTPS, were the key security enhancement. Additionally, we refactored the pom.xml file to guarantee that all vulnerabilities found during the dependency check were fixed.

For us to be able to use HTTPS once our application was up and running, the first step in my process was making sure the certificates were created appropriately. By making sure that our website is secure and users can be sure they are dealing with us and not a fake, this security improves the health of our business.

The following step was ensuring that our hashing function operated correctly and checking this with the checksum. By giving us peace of mind that the data of our consumers is correctly scrambled and difficult to recover, this security promotes the health of our business.

The last stage involved making sure that all vulnerabilities were patched. Having such security guarantees that our business is prepared and that all of our application's internal workings are current and functioning as intended.

Patching our software and systems to keep them current is one best practice for preserving the security of our application. By doing this, outdated systems are protected from attack. It is also essential to enforce least privilege. The organization is protected from attacks from within the group by ensuring users only have access they need rather than granting everyone access to everything, even if it is not currently possible given the current state of our program.