****

# 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** | **04-21-24** | **Jasmine Bell** |  |

## 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

Ethan Daugherty

## 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.

Artemis Financial is requesting an additional security for their web application to ensure secure communications. Assuming the most likely attack vector to occur with a financial institution is a bad actor, attempting to gain financial gain by accessing the information stored, encryption would be the best recommendation. This will render the files useless without a key to any would be attacker. As the firm is looking to secure communication, I would recommend Asymmetric communication. Meaning that the key to encrypt is public and the key to decrypt is private. To use the highest level of security as this information could be being sent externally, I would suggest using the SHA-256 cipher algorithm with 256-bit keys to encrypt. SHA-256 encryption provides excellent high level of bit encryption with a great many of different possible key combinations with the key of 256 bits in length. Additionally, the SHA-256 algorithm uses Java’s random number generator ensuring that the encryption is very secure by creating a non-reversible checksum that verifies the validity of the file. The hash function will use the SHA-256 cipher to create a checksum of the provided message.

## 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. A screenshot of a computer

  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.

A screenshot of a phone

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.

* Showing the HTTPS is working but that my Cert isn’t official because it’s self signed.

A screenshot of a phone

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

**package** com.snhu.server;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

**class** ServerApplicationTests {

@Test

**void** contextLoads() {

}

}

@SpringBootApplication

**public** **class** ServerApplication {

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

**byte**[] hash = md.digest(name.getBytes(StandardCharsets.UTF\_8));

BigInteger number = **new** BigInteger(1, hash);

StringBuilder hexString = **new** StringBuilder(number.toString(16));

**while** (hexString.length() < 32)

{

hexString.insert(0, '0');

}

**return** hexString.toString();

@RestController

**class** ServerController{

//**FIXME**: Add hash function to return the checksum value for the data string that should contain your name.

String data = "Jasmine Bell - CS Major at SNHU!";

String hash = calculateHash(data);

**return** "<p>data:"+data+" : SHA-256 "+" : "+hash;

}

A screenshot of a computer

Description automatically generated

## 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.

A computer screen shot of a code

Description automatically generated

## 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.

Refactoring in my code I have added a secured RestController to work as the secure controller for my programs hash RESTful stop. The ServerController class works to match the problems presented by the vulnerability assessment diagram. I additionally chose to work with the SHA-256 hashing cipher as it’s very secure and runs a very small chance at collisions. To best maintain the current security of the application I would suggest once or twice monthly dependency checks of the application to keep the most up to date on potential vulnerabilities this will help to protect the company and their sensitive data. Keeping the plugins within the pom.xml additionally would do well to keep the latest iterations of the plugins running ensuring the highest security.