****

# 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** | **2/14/2022** | **Raechel McGuire** |  |

## 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

Raechel McGuire

## 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 recommended encryption algorithm cipher that avoids collisions is SHA-256.

SHA-256 is one of the most popular and secure hash function used today. Hashing is a type of encryption technique that encrypts data in unique and irreversible way- once data is hashed , the hashed text cant be used to determine what the original piece of data contained before being encrypted. Each piece of data that gets hashed has its own unique encrypted text generated, even the slightest change in a piece of data can drastically change the hashed text value.

A collision occurs when two different pieces of data produce the same hashed values, this can be dangerous because it can lead to improper values being validated. (ex. If “Password” and “password123” generate the same hashed value, the wrong password could be used to gain access to an account). Since each hashed value should be unique to each different piece of data, we want to avoid collisions at all cost. SHA-256 can avoid collisions by generating a large hash value of 256 bits. This means that hashed values can have possible different combinations, which is a large enough number to make it unfeasible that two pieces of data would have the same hashed value.

Since SHA-256 can ensure that collisions don’t occur, Artemis Financial can verify files by running them through the hash algorithm, if the hashed values match, they can be sure that data hasn’t been changed or corrupted in any way.

## 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.

[Insert screenshot(s) here.]

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.

[Insert screenshot(s) here.]Graphical user interface, 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.

[Insert screenshot(s) here.]

Graphical user interface, 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, application

    Description automatically generated

[Insert screenshots here.]

## 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.]

Graphical user interface, text, application

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.

I refactored the code to include a hashing function that takes the data and turns it into a hashed value using SHA-256. SHA-256 is one of the most secure hashing algorithms with no known breaches to date. This level of security allows the company to ensure that data hasn’t been corrupted in any way by running the data through a hashing algorithm. Any changes to the hashed value would indicate that something in the data was changed, which could be dangerous if implemented into the system.

Additionally the application is run through the https protocol using a self signed certificate, giving users assurance that theyre on the correct webpage.