****

# 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** |
| --- | --- | --- | --- |
| **2.0** | **12/12/2021** | **Ryan McFarland** | **Updated** |

## 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

Ryan McFarland

## 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.

I recommend using SHA-256 for encryption tasks related to this task of providing a checksum. SHA-256 is a collision free algorithm that is an improvement to SHA 1 algorithms. 256 bits are output regardless of the size of the data processed. The algorithm only works one way, so hashed values are not susceptible to attacks with the goal of revealing the data that was hashed. In this case it is being used as a checksum to verify the integrity of data. A piece of data is processed according to the SHA-256 algorithm and a hashed value is output based on the input. When a receiver of data wonders about whether their data has been altered, they can process the information through the SHA-256 algorithm and check whether their hashed value matches the senders hashed value.

A cipher is essentially a group of ordered steps to obfuscate data. Data may come under attack while being stored or in transport. If the data is of a sensitive nature and has not been encrypted or is encrypted with an inferior cipher, then a successful attack can lead to an attacker possessing sensitive data such as passwords or other similar information. Making a recommendation for using a particular cipher is not a trivial decision and the decision needs to be based on what is currently available for use. Unknown weakness may exist for a particular cipher, or the cipher may be weakened in the future as computers become more capable.

## 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

## 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, email

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, email

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

## 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.

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.

One thing I did that added value was to force the use of log4j2 to a secure version given the RCE possibility for certain versions and uses. I provided two reports, one that notes a log4j2 weakness, and the other after forcing the use of 2.15.0 in the pom file and no longer showing a log4j2 issue. The code has been refactored to include a hash function to provide a checksum. The value in this addition is that the company will be able to secure data in transit and see that it has not been altered. Areas of security that were addressed with the code refactor include client/server, cryptography, and code quality. The process for adding layers of security to the software application involved creating a keystore to be used in the server and assessing whether code I included introduced any new problems into the application using the OWASP Dependency checker. Bests practices for maintaining the security of the software application would include a regular assessment of the project and its dependencies. Products in use will inevitably be updated and the most secure and stable versions of any product should be in use. Algorithms used for encryption should regularly be assessed for any known weakness.