****

# 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** | **12 October 2021** | **Thomas Cogley** |  |

## 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

Thomas Cogley

## 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 chose to use SHA-256 as my encryption algorithm. SHA-256 is one of the most secure hashing functions. There are three properties that make SHA-256 one of the most secure. First, it is almost impossible to reconstruct the initial data from the hash value. A brute-force attack would need to make attempts to generate that initial data. Second, having two messages with the same hash value is called a collision. Since there is possible hash values the possibility of two messages having the same value is almost impossible. Finally, if the original data is altered by a little, the hash value changes substantially that it is not apparent that the data is similar. All these reasons are what make SHA-256 one of the most secure encryption algorithms and why I chose to use it.

SHA-256 generates a 256 bit or 32-byte hash function which helps tremendously with the security of the data being hashed. SHA-256 is also a symmetric encryption. The common encryption types are symmetric and asymmetric. Symmetric encryption uses one key for both encryption and decryptions, whereas asymmetric encryption uses a public key for encryption and a private key for decryption. Due to asymmetric encryption using two keys for encrypting it is considered more secure. However, symmetric encryption is less complex and executes much faster. Also, asymmetric encryption is much newer then symmetric therefore it is used much less. Due to symmetric encryption being quicker and older it is the preferred method for encryption.

The concept of encryption has been around for many years, in fact, since circa 600 BC. Since then, human civilization has constantly been improving on encryption techniques. Fast forward to the present our encryption algorithms is much more improved. Utilizing computers, we can now generate encryptions with trillions of possibilities to crack. However, as our encryption techniques improve, the hackers’ techniques improve. This results in each side pushing the other to constantly improve on their algorithms.

## 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

Graphical user interface, text, application

Description automatically generatedGraphical 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 generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

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.

Referring to the Vulnerability Assessment Process Flow Diagram (VAPFD) I covered a few areas of security by refactoring my code. The first area is cryptography. I used SHA-256 hash algorithm to convert my data strings to hex. I also used secure coding practices and error handling as shown with the throws no such algorithm exception.

I added layers of security to the application by using the maven dependency checker. I initially updated the maven check version. Next, I updated all the dependencies I could to the latest version by setting that as a goal in the maven build. Next, I ran the dependency check and noticed I still had 41 vulnerabilities. Finally, I reviewed my vulnerabilities for false positives and suppressed the ones I found. After running the dependency check again it resulted in eight vulnerabilities. Security is important to a company’s wellbeing. Securing your program help prevent many malicious attacks. Especially when your company works with sensitive information such as credit card information, names, phone number, etc. However, it is important to constantly update your security since hackers are constantly updating their forms of attack.

The best practices to maintain security for this program is to update frequently. At least once a month it is important to run the dependency check. This will show if any new vulnerabilities have presented themselves. Also, check for new versions of dependencies by running the maven goal to view version updates. If new versions are found that address current vulnerabilities, they can be installed to mitigate them.