****

# 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** | **[Date]** | **[Your Name]** |  |

## 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

[insert name here]

## 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.
  + For this project, I chose the SHA-256 algorithm. I chose SHA-256 because it is commonly used with very, very low chance of collisions. I could have opted for an even higher bit length such as SHA-512 but found that to be overkill for this use case.
* Discuss the hash functions and bit levels of the cipher.
  + SHA comes in many bit levels, with the most common being 256 and 512. There are several others as well, however.
  + These levels allow one to make a hash more or less secure, at an inverse cost of performance.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
  + There are many uses for these concepts in encryption. The idea behind encryption is obfuscation, and these are implementations of that. For example, in an asymmetric cipher, the same key used to encrypt a target could not be used to decrypt it. This is the opposite of a symmetric cipher.
* Describe the history and current state of encryption algorithms.
  + Methods of encryption have been around since ancient times. The Romans had the “Caesar cipher” which just shifted letters down a certain number of spaces. This evolved into methods such as the “Jefferson Disk” which involved plates arranged in a certain order that could be used to encrypt messages and only decrypted with a Disk in the same format as the one used to encrypt. Today we have even more such complex methods of encryption that are essentially unbreakable today without the proper keys. Any method can be brute forced, however the amount of computations this would take with current technology would make the timeframe to crack longer than any would deem feasible.

## 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.

See above screenshot

## 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

Running with no errors:  
Text

Description automatically generated

No new dependencies, so no new vulnerabilities:

Graphical user interface, text, application, Teams

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.

See above screenshot

## 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.
  + There wasn’t a lot to refactor here! I did add in the cipher component and this required security in the realm of APIs, cryptography, and client/server architectures. During code review I found it would be better to separate my work into a different file so that it wasn’t in the same file as the main method.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
  + The main point of security that I think deserves mentioning is within the cipher itself. There is a string of data that is converted to bytes and digested into SHA-256 format. This allows us to return an encoded string that can be related back to our original data for verification purposes.
* Point out best practices for maintaining the current security of the software application to your customer.
  + Security should be maintained throughout the development process. Even before development starts. It is important to *plan* with security in mind, to also perform through reviews of the system during and afterwards.
  + This includes everything in the Vulnerability Assessment Process Flowchart. Common vulnerabilities include Cross-site scripting attacks, SQL injections, and I/O attacks.
  + Damage can be further prevented by ensuring data is secure at all times, including while at rest.