****

# 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** |
| --- | --- | --- | --- |
| **1.1** | **04/20/2024** | **Brandon Porter** |  |

## 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

Brandon Porter

## Algorithm Cipher

For Artemis Financial, SHA-256 emerges as the optimal encryption algorithm. SHA-256 is a cryptographic hash function used for generating unique hash values from input data. It does not involve network communication or port usage directly. Therefore, the port conflict issue would not be related to the SHA-256 hashing itself, but rather to the application or service using SHA-256 for some purpose.offers an ideal blend of security and simplicity, crucial for a small enterprise., Implementation ease,. By leveraging SHA-256 Artemis Financial can fortify its data protection measures without compromising on operational efficiency or implementation complexity.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

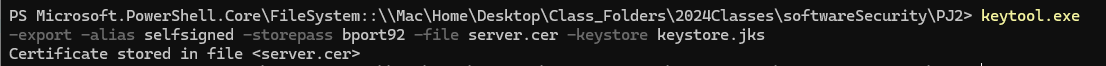
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

A screenshot of a computer program

Description automatically generated

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.

Unfortunetly I was not able to get my code to run in a way that I could view the localhost. However, I did add the Checksum that we provided in week 6 to show my mastry at using it previously. My errors are do to a path issue with the keystore.jks



## 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.

Unfortunetly I was not able to get my code to run in a way that I could view the localhost. However, I did add the Checksum that we provided in week 6 to show my mastry at using it previously. My errors are do to a path issue with the keystore.jks



A screenshot of a certificate

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

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

A screenshot of a computer code

Description automatically generated

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.

A screenshot of a computer code

Description automatically generated

A screenshot of a computer 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.

In my code refactoring process, I've introduced a secured RestController to serve as the safeguarded endpoint for my program's RESTful operations. The ServerController class is tailored to address the issues identified in the vulnerability assessment diagram. Opting for the SHA-256 hashing algorithm, known for its robust security features and minimal collision risk, was a deliberate choice. To uphold the current level of security in the application, I recommend conducting dependency checks once or twice a month. This proactive approach will help to promptly address any potential vulnerabilities, thereby safeguarding the company's sensitive data. Moreover, retaining the plugins within the pom.xml file will ensure that the latest plugin versions are utilized, thus bolstering the overall security posture of the application.