****

# 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** | **2/14/2020** | **Kyle Hake** |  |

## 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

Kyle Hake

## 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.

Since Artemis Financial is has a web-based interface, using a TLS protocol with AES-256 encryption algorithm would be recommended because it would prevent man-in-the-middle attacks as well as provide a secure line of communication via HTTP over TLS. TLS uses an asymmetric (a public key used to authorize identity) encrypted (by a secret key) handshake to establish a secure connection to share a symmetric (the same secret key is used on both ends) secret key (AES – 256). AES is a block cipher, which means it divides the data into block of 128 bits and then encrypted. The “256” of the AES description is the size in bits of the key used to encrypt and decrypt. AES is the gold standard for encryption and is used in federal agencies as a standard for encryption. There have been other encryption ciphers (DES, MD5, SHA-1) that have been outdated because once a cipher can be cracked, it is no longer useful.

## 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.

Text

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

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

Text

Description automatically generated

Graphical user interface, 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.

Graphical user interface, text

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.

During this security assessment of Artemis Financials new web application, a few areas were addressed to make the software more secure and protected. Using the Spring framework for the application, the company has access to input validation parameters when needed as the application is developed. This framework also has API capabilities built in so that they can be secured using Spring Security. In the code refactoring, cryptography and cipher algorithms are used to secure communications between users and the server. Using a SHA-256 cipher algorithm, a CheckSum was used to encrypt communications between the client and the server using an SSL protocol (HTTPS). Also, self-signed certificates were generated using Java’s keytool so they can be used to authentication. This gives the application an extra layer of security between the client and the server. A vulnerability check using Maven was ran to check for any big weaknesses in packages being used in the app’s development. The code has also been reviewed to make sure it runs properly, and secure coding practices are being followed.

Making sure an application is secure is one of the more important processes outside of the functionality of the application itself. Not addressing security can lead to a loss of money, time, resources, and the trust of the consumers of the product. Building something that is secure might take some time and money, but there really isn’t a dollar amount that can be put on the loss of trust in a product. Security is something that should be addressed from the beginning of the project to the launch of the software because as the project changes, so might its security requirements. Adding different layers of security (user input validation, encryption for at rest and communicated data, SSL/TLS protocols, etc.) can be used depending on the functionality of the application and the end goals of the client.

The process or evaluation of a software’s security doesn’t end at the release however, there is still the maintenance that needs to be done to keep sensitive data protected and unauthorized users at bay. The Maven vulnerability assessment tool is valuable if third party plug-ins or packages are used in the development process because it can alert of new security issues and how to mitigate them. Also, making sure that all plug-ins and frameworks used are updated frequently can help improve a products security because vulnerabilities are addressed in updates. Outside of software updates, making sure that cipher protocols used for encryption aren’t outdated as well as rotating decryption/encryption keys can help keep data safe as well as protect communications.