****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 3](#_Toc33111305)

[1. Algorithm Cipher 3](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 5](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 6](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 9](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **October 17, 2021** | **Mauricio Bautista** |  |

## 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**

Mauricio Bautista

## 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.

Advanced Encryption Standard-256 (AES-256) is the appropriate encryption algorithm cipher to deploy given the security vulnerabilities. AES-256 is a symmetric-key algorithm that transforms data to make it extremely difficult to view without possessing a secret key. The algorithm is called symmetric because the same key is used for both encrypting and decrypting the data. Similarly, the algorithm has a key length of 256 bits, supporting the largest bit size, and it is practically unbreakable by brute force, making it the most substantial encryption standard. AES-hash is a secure hash function, meaning it takes an arbitrary bit string as input and returns a fixed-length (in this case, 256 bit) string as output. We referrer 256-bit encryption as to the length of the encryption key used to encrypt a data file. In other words, a hacker will require different combinations to break a 256-bit encrypted message, which is virtually impossible to be broken by even the fastest computer.

As the technology advanced, some encryption methods considered strong in the past are no longer sufficient. For example, there have been some data breaches or attempts to do damage to important companies. However, the state of cryptography is currently strong enough to protect most of our data, like emails and online transactions. But, its future status isn’t sure because even stronger cryptographic systems might be created; this is also the prospect of new means to break them. Therefore, the fight between algorithms and attackers will always continue.

## 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

Graphical user interface, text, application, email

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

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

Text

Description automatically generated

**DEPENDENCY REPORT BEFORE UPDATES.**

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.

**REFACTORED CODE**

Graphical user interface, text, application

Description automatically generated

**DEPENDENCY TEST AFTER CODE WAS REFACTORED**

Graphical user interface, text, application, email

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.

Areas of security addressed by refactoring the code were Cryptography, APIs, and Code Quality. First, we used cryptography to create an encryption algorithm cipher. The cipher allowed us to hide the data, and the checksum result verified that the data in the file had not been maliciously changed or intercepted by a hacker. Second, we used APIs to secure interactions with the client. We generated a digital certificate that authenticates or, in other words, establishes trust between servers, individuals, and organizations. Although the result showed the connection was not secure when accessing the localhost, we were not worried because we accessed the port with the HTTPS protocol. Finally, code quality ensured we used the best coding practices and patterns to avoid compromising the system. We made sure that all the properties, dependencies, and plug-ins in our POM file were up-to-date. For example, we updated the tomcat version to 9.0.54 and the maven plug-in to 6.3.2.

Providing the algorithm cipher, securing APIs, and making sure we follow our best coding practices will give Artemis Financial a sense of security when using the system. For example, the communication with their clients will be secure, which is the organization’s most important value. To conclude, the best practices for maintaining the security up to date are checking for new vulnerabilities using the dependency-check tool and performing a code update by implementing new security layers.