****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[\_Toc33111301](#_Toc33111301)

[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 5](#_Toc33111308)

[4. Secure Communications 5](#_Toc33111309)

[5. Secondary Testing 6](#_Toc33111310)

[6. Functional Testing 7](#_Toc33111311)

[7. Summary 7](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/12/23** | **Laynie Tierney** | **Algorithm Cipher, Certificate Generation** |
| **1.2** | **12/14/23** | **Laynie Tierney** | **Deploy Cipher, Secure Communications, Secondary Testing** |
| **1.3** | **12/15/23** | **Laynie Tierney** | **Functional Testing, Summary, Industry Standard Best Practices** |

## 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

Laynie Tierney

## 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.

The proposal for Artemis Financial involves securing their web traffic by implementing a cipher suite. This suite revolves around the TLS protocol, an industry standard that ensures secure communication. TLS leverages a certificate signing architecture, establishing trust between client and server. Within this certificate, a crucial element is the private key, ideally generated using RSA, offering the most up-to-date security for private keys.

For encrypting web payloads, AES (Advanced Encryption Standard) is strongly recommended. Acknowledged by the National Institute of Standards and Technology (NIST), AES serves as a block cipher, significantly improving encryption speeds. Among its advantages, AES supports various key sizes. While 256-bit keys are the largest supported, the resource-efficient AES-128 is preferred for web applications, ensuring robust security without being overly resource-intensive. AES also integrates key hashing capabilities via SHA-256, an algorithm currently considered highly resistant to collisions.

Given that the web application transmits data between systems, an asymmetric cryptographic cipher is advised. This approach demands both private and public keys for decryption. It's crucial to employ a secure key manager for this cipher to maintain the highest level of security.

These recommendations are based on the latest cryptography tools and insights available. The landscape of cryptography tools evolves rapidly, emphasizing the necessity of staying updated with the most recent recommendations

## 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

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.

A screenshot of a computer

Description automatically generated

data: Laynie Tierney CS305 – Hello World Check Sum!

hash: 5c90ae790d8141aa4328b9b286fc60c49547ea76af1aed969442e1b48d28304a

## 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.

A screenshot of a computer

Description automatically generated

data: Laynie Tierney CS305 – Hello World Check Sum!

hash: 5c90ae790d8141aa4328b9b286fc60c49547ea76af1aed969442e1b48d28304a

## 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

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

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.

A screenshot of a computer program

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:

During the vulnerability assessment process, several areas of the codebase were evaluated for security improvements:

* **Views:** No modifications were necessary within the view components.
* **Models:** No changes were required in the model layer.
* **Controllers:** Updates were made to the REST controller to explicitly define HTTP request types.
* **Data Access:** The application has hardcoded key-store credentials, hindering local compilation. These credentials should be updated in the production environment for enhanced security.
* **Services:** No modifications were required in the service layer.
* **Plug-Ins:** Dependency checks were performed to identify and address potential vulnerabilities.
* **APIs:** No APIs were subject to assessment.

8. Industry Standard Best Practices

Implementing industry-standard best practices involved leveraging the latest encryption technologies, refining code for improved security, and employing vulnerability scanning tools for dependency assessment. By adopting these strategies for secure coding from the outset, Artemis Financial stands to benefit significantly. Prioritizing security in development reduces the need for extensive code revisions and minimizes potential incidents in the future.