****

# 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** | **February 19, 2022** | **Benjamin Abbott** |  |

## 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

Benjamin Abbott

## 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.

Artemis financial requires an encryption algorithm cipher that must be secure enough to protect their client’s personal information and data. For this I believe the AES-256 encryption algorithm cipher would be the best option. AES is the most commonly used encryption protocol in the world and is the “industry-standard protocol that protects sensitive information from traditional brute-force attacks” (Hougen, 2021, para. 2).

The bit size of the key refers to the key length. The 256-bit key uses 10 transformation rounds to convert plaintext into ciphertext. Random number generators are used to generate the 256-bit symmetric cipher key that is then sent to the AES engine which encrypts the plain text into cipher text. These keys are crucial to decrypting the encrypted data and must be kept safe.

AES uses symmetric keys which means that the same key is used to encrypt and decrypt the data. An asymmetric key on the other hand would have different keys for encryption and decryption. While the use of asymmetric keys is a safer method, it does use more significantly more computational power. AES-256 has never been cracked and is still an extremely safe encryption cipher (Daniel, 2021).

Modern encryption algorithms started being used in the 1970s by IBM to protect their data, but the invention of the world wide web in 1989 and the rise of the personal computer really expanded the need for protection. The standard encryption used from the 70s to the 2000s was the DES or Data Encryption Standard. At that point the AES took over as the new standard and continues to be the leading algorithm for the foreseeable future (Thales, 2022).

## 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.

Graphical user interface, text, application, email

Description automatically generated Graphical user interface, text, application

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, 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, application

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, text, application

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.

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

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.

The areas of security that were addressed by refactoring the code were APIs, Cryptography, Client/Server, and Code Quality. I was able to create my own self-signed certificate using the Java Keytool which allowed for secure HTTPS connection. From there I deployed and implemented the encryption algorithm cipher to the software application. To verify this, I used a checksum which confirmed the HTTPS connection and the functionality of the cipher. I scanned the code for vulnerabilities and updated the Spring Boot Release version. From there I ran a test on the refactored code with the dependency check tool. This dependency check showed that no additional vulnerabilities were added from my refactored code, and many were fixed from the updated version of the Spring Boot Release. To ensure that the security stays maintained I recommend regular updates to all the software that is used. These updates will fix any new vulnerabilities that may be found and will make sure that Artemis and their clients’ data remains safe and secure.

Using these measures and added layers of security to ensure secure connection and secure encryptions is essential to most businesses, and even more so to a financial institution like Artemis. By doing this they can offer their customers supreme protections and give them a sense of comfort knowing that their data is safe with them.

References

Daniel, Brett. (2021, May 4). *Symmetric vs. asymmetric encryption: What’s the difference?* Trenton

Systems. <https://www.trentonsystems.com/blog/symmetric-vs-asymmetric-encryption>

Hougen, Aleksander. (2021, May 26). *What is AES encryption & how does it work in 2022? 256-bit vs 128-bit.* Cloudwards. https://www.cloudwards.net/what-is-aes/

Thales. (2022, Feb 8). *A brief history of encryption.* Thales Group.

<https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption>