### SNHU

### Project 2: Practices for Secure Software Report

Andreas Galatis

### Dr. Vivian Lyon

### CS 305 – Software Security

June 19, 2022

****

# 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** 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** | **06/19/22** | **Andreas Galatis** |  |

## 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

Andreas Galatis

## 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 recommended encryption algorithm cipher for Artemis Financial is the Advanced Encryption Standard (AES), specifically the AES-256 algorithm. Used by the government as a standard and trusted by many high-end businesses, the AES-256 cipher algorithm is one of the top encryption options available. This was not always the case, as the Data Encryption Standard (DES), which was what the United States government used as a standard for a long time, was found to not be as impenetrable as originally estimated. In 1999, researchers were able to crack the 56-bit key algorithm it uses, by a network of computers, and when that happened, the NSA came up with the AES algorithm (Bernstein & Cobb, 2021). Today the AES-256 algorithm stands out at the top of the list of bonified and trusted algorithms.

The most notable benefit of AES is its level of security with its three different key lengths used for encryption; a 128-bit key, 192-bit key, and the recommended 256-bit key length which has 1.1 x possible key combinations, the latter being the strongest level of encryption (Bernstein & Cobb, 2021). Given the nature of Artemis Financials’ need to secure highly personal and private customer data, AES-256 is the best recommended option. AES-256 has two key features to its algorithm.

First is that it uses a symmetric system for encryption and decryption. With a symmetric encryption system, both receiver and sender use the same key to decrypt and encrypt, as opposed to other algorithm ciphers that may use an asymmetric key system requiring two separate keys to do the same for each process (Manico & Detlefsen, 2015). Symmetric keys are used more often since they are secure and faster to execute with their shorter key lengths. Asymmetric keys, however, do have their advantages as there is no need for the distribution of keys.

Secondly, AES-256 uses a block cipher as a hashing function. What this does is, it breaks down the information that is to be encrypted into several segments of blocks. AES utilizes a 128-bit block size hashing function and separates the information into an array that holds 16 bytes, and each array is a four-by-four block. What ends up happening is that the size of the data that is encrypted always stays the same. Additionally, for the AES cryptographic algorithm to be effective, a random number generator must be used to rapidly come up with a complex sequence of numbers or characters. The randomness of so many characters and numbers make the algorithm nearly impossible to break.

Lastly, for the checksum verifications, the best algorithm to use is the SHA-256. Used by the NSA, and recognized as the industry standard, SHA-256 offers to lowest chance of a collision happening, and has an Avalanche effect as well, which means that if the original message is altered by just one data input, the algorithm will come up with a completely different set of hash characters (Soliman et al., 2017).

## 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.

Screenshot of form used to generate self-signed certificate using Java Keytool through command line

Graphical user interface, text

Description automatically generated

Screenshot of command to export certificate



Screenshot of printed certificate

Text, letter

Description automatically generated

Screenshot of CER file (file also included in attached Java application folder: ssl-server\_student)

Graphical user interface, application, table

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.

Screenshot of checksum verification coding (part of SlsServerApplication.java located in attached Java application folder: ssl-server\_student)

Text

Description automatically generated

Screenshot of checksum verification on web browser

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.

Screen shot of application.properties file from attached Java application folder: ssl-server\_student

Graphical user interface, application

Description automatically generated

Screenshot of web browser showing secure webpage

Graphical user interface, text, 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

Text

Description automatically generated

* + Dependency Report Before

Graphical user interface, text, application, email

Description automatically generated

* + Dependency Report After

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.

Upon manual review of the code one of the vulnerabilities identified was the outdated dependency check version which was set to 5.3.0 and needed updating to the current 7.1.0 version. This ensures that the vulnerability check will be fully effective. The spring-boot dependency was also outdated with a 2.2.4.RELEASE version. This version is vulnerable to temporary directory hijacking and needs updating. Also, the application was missing a secured HTTPS protocol, as a secure web interface needs.

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

Screenshot of refactored coed executed without errors

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.

Some of the areas of security addressed during the code refactoring were secure API interactions, encryption use and vulnerabilities, secured client/server communication, and secure coding practices. Clean and secure code was written in the RestfulController mapping to allow for HTTPS protocol to be implemented. And SHA-256 cipher algorithm was added with a checksum to ensure that a secure connection with the lowest possible chance for a collision on the algorithm. A self-signed certificate was added to the web API to authenticate connections and ensure that all connections to and from the browser were from trusted end points.

This added level of security will ensure that Artemis Financials’ private customer data, as well as company data will remain secure during ongoing web communications. The use of signed certificates will ensure that the company is remaining diligent in monitoring who is on their website and will give their customers the confidence that they are engaging in a secure web interaction. This will build trust and confidence for their clients, as just one breach could possibly cause the loss of not only much revenue, ut also the permanent loss of trust.

To maintain the current security level of the software application, regular maintenance, by running routine dependency checks should be conducted. This will ensure that all dependencies are up to date. This will also keep the system in balance with any new vulnerabilities that may arise by checking the NVD CVE’s. Those vulnerability checks should also be combed through for false positives to maintains a clean and reliable testing methodology that does not dispense undue effort on less risky vulnerabilities, but rather make sure that the most severe ones are addressed.

**References**

Bernstein C., & Cobb, M. (2021, September). *Advanced Encryption Standard (AES).* TechTarget. <https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard>

Manico, J., & Detlefsen, A. (2015). *Iron-Clad Java: Building secure web applications*. McGraw Hill Education

Soliman, J. N., Mageed, T. A., & El-Hennawy, H. M. (2017, December). Digital signature and authentication mechanisms using new customized hash function for cognitive radio networks. *2017 12th International Conference on Computer Engineering and Systems (ICCES)*. https://doi.org/10.1109/icces.2017.8275299