****

# 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** | **10/16/21** | **Jason Marcil** |  |

## 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

Jason Marcil

## 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.

Because Artemis Financial is a client dealing with transactions and personal data that needs to be secured at the highest level, I recommend using the Advanced Encryption Standard (AES). I would specifically recommend using AES-256, as this is the highest level of this cipher. AES is a military grade encryption cipher that was established in 2001 by the National Institute of Standards and Technology (NIST) in 2001 (Raghavan, 2020).

Secure Hashing Algorithms (SHA) takes in plain text and converts it to ciphertext that is difficult and unlikely to be deciphered (Crane, 2021). SHA can be used alongside AES to enhance security because of this. AES-256 allows 128-bit, 192-bit, and 256-bit encryption. The higher the bits, the more difficult it would be for an attacker to crack the encryption. With AES-256, it can have 2^256 unique combinations, making it so strong that it is also resistant to attacks from a Supercomputer (ClickSSL.net, 2021).

The AES standard constitutes 3 block ciphers where each block cipher uses cryptographic keys to perform data encryption and decryption in a 128-bit block (Raghavan, R). Data is encrypted and decrypted in blocks of 128 bits and can do this using 128-bit, 192-bit, or 256-bit keys consisting of random numbers and characters. The randomness makes the keys even harder to crack because there is no logical meaning to why certain characters are used and where they are placed. Keys of these sizes are considered adequate for securing confidential or classified information (Crawford, 2019). With a minimum of 128-bits to work with, there are so many possible combinations of key characters that it makes this cipher virtually unbreakable when implemented properly. The only other way to decrypt encrypted data is to use a key that is provided by the sender to the receiver. This can be done using either asymmetric or symmetric encryption. Use of asymmetric encryption involves two keys, one for encryption and the other for decryption. Symmetric encryption on the other hand requires one single key to encrypt and decrypt (GeeksforGeeks, 2020). Asymmetric encryption is ideal for securing data in motion while symmetric encryption is better for data at rest. (Crawford, D). AES falls under the category of symmetric encryption, which works for this scenario because we are trying to secure archive files that will be at rest.

Encryption has been around for centuries. Some of the earliest civilizations that used encryption include the Ancient Egyptians, where standard hieroglyphs were swapped for ones with non-standard meanings that could only be understood by someone who knew their obscure definition. During both World Wars I and II, encryption was used for masking communications, and decryption methods were used to crack those codes. As technology continues to develop, we could see AES replaced by another algorithm that proves to be more secure. Hackers are always discovering new ways to exploit weaknesses that have not been thought of yet, so it is important to stay vigilant and use other simpler forms of security to protect data. Even with these threats in mind, Artemis Financial should feel confident in the AES cipher to protect this sensitive data.

## 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

Description automatically generated

Graphical user interface, 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, 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

Graphical user interface, text, application, email

Description automatically generated

Updating the spring-boot-starter-parent version to 2.3.12 RELEASE reduced the number of vulnerabilities reported from 41 to 2.

## 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, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

The application’s most significant vulnerabilities include the use of a non-complex password and the use of a hard-code input string in the hash function in SslServerApplication.java (i.e., “Name of Cipher Algorithm Used: AES Checksum Value”)

## 

## 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.

There were several areas of security from the Vulnerability Assessment Process Flow Diagram that were addressed by refactoring the code. APIs were addressed when HTTPS was implemented, and we could view the application in <https://localhost:8443/hash>. Cryptography was addressed when we implemented an encryption algorithm cipher and hash function. Client/Server was addressed when we tested the application by sending it from the client to the server to be displayed in a browser. The client and server were both the same in this instance, but we demonstrated this area security by testing. Code error was addressed by using a NoSuchAlgorithmException within the myHash function. Finally, Code Quality was addressed by reviewing the code for errors and ensuring that the code was functional and readable.

Adding layers of security and determining what vulnerabilities are still valid adds to the company’s overall wellbeing. The first layer of security that I created was an SSL certificate and keystore. I created a cipher and hash function that would serve as a checksum verification that the certificate was created correctly. Next, we refactored the application.properties file to convert HTTP to HTTPS and select the certificate that we created. Then I ran the application using the Maven goal spring-boot:run so that I could run the checksum verification that let me know that the certificate ran with the application, which it did. I then ran a dependency check of the application to check for vulnerable dependencies. This resulted in 41 vulnerabilities, however, the version of spring-boot-parent that I was using was out-of-date. I updated the spring-boot-starter-parent to the most recent RELEASE version and re-ran the dependency check, which resulted in only two medium severity level vulnerabilities, one of which had a low confidence level.

The best practices for maintaining the current security of the software application are to make sure everything is up to date. Dependencies with known vulnerabilities are typically updated when vulnerabilities are reported, and it is most often the case that the dependencies being used are not the most recent versions. It is important to note that some newer versions of java may be incompatible with JKS and PKC12 keystores. If the error “**java.io.IOException: Invalid keystore format” is displayed when trying to run the code,** rolling back the java version might be a solution to the problem.

References:

*256 Bit Encryption: Is AES-256 Bit Encryption Safe in Modern Times?* (2021, May 3). ClickSSL Blog - Information about SSL Certificates & Infosec. Retrieved October 15, 2021, from https://www.clickssl.net/blog/256-bit-encryption

Crane, C (2021, January 25). What is a hash function in cryptography? A beginner’s guide. Hashed Out by The SSL Store. https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a- beginners-guide/

Crawford, D. (2019, February 4). *AES Encryption | Everything you need to know about AES*. ProPrivacy.Com. https://proprivacy.com/guides/aes-encryption

GeeksforGeeks. (2020, June 28). *Difference Between Symmetric and Asymmetric Key Encryption*. <https://www.geeksforgeeks.org/difference-between-symmetric-and-asymmetric-key-> encryption/

Raghavan, R. (2020, April 19). *What Is Data Encryption? Which All Are The Top Encryption Algorithms?* Web Solutions Blog. <https://acodez.in/data-encryption-algorithms/>