****

# CS 305 Project Two

**Practices for Secure Software Report**

Sarah C Jodrey

Southern New Hampshire University

CS-305-T3674: Software Security

Professor Dr. Vivian Lyon

February 20, 2022

**Table of Contents**

[Document Revision History 3](#_heading=h.30j0zll)

[Client 3](#_heading=h.1fob9te)

[Instructions 3](#_heading=h.3znysh7)

[Developer 4](#_heading=h.2et92p0)

[1. Algorithm Cipher 4](#_heading=h.tyjcwt)

[2. Certificate Generation 4](#_heading=h.3dy6vkm)

[3. Deploy Cipher 4](#_heading=h.1t3h5sf)

[4. Secure Communications 4](#_heading=h.4d34og8)

[5. Secondary Testing 4](#_heading=h.2s8eyo1)

[6. Functional Testing 5](#_heading=h.17dp8vu)

[7. Summary 5](#_heading=h.3rdcrjn)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **February 20, 2022** | **Sarah C Jodrey** | **Discussed recommended and deployed algorithm cipher, generation of SSL certificate, refactored code for secure web application and communication, dependency check testing, functionality testing, and summary of vulnerabilities and security layers addressed.** |

## 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

Sarah C Jodrey

## 1. Algorithm Cipher

Artemis Financial is a financial consulting company looking to secure their clients data and financial information on their web application. As the web application is being used to transmit this sensitive data, they must implement data verification via checksums and encryption. This is to help prevent attacks by common culprits like bad actors/impersonators and brute force attacks. The company is best served utilizing the AES-256-CBC encryption algorithm, and the SHA-256 algorithm cipher, as I have implemented.

The AES-256-CBC encryption algorithm is a symmetric 128-bit block cipher that supports the private key sizes of 128, 192, or 256-bits to encrypt and decrypt data (Oracle, 2017). It is the current standard, and the algorithm trusted by the US government and Federal Information Processing Standards (FIPS) (Dworkin et al., 2021). As data is transmitted, the AES encrypts the information so that only an individual with the key can view its contents once transmitted.

As the AES-256-CBC scrambles data in transit, the SHA-256 encryption cipher is a way for the data received to be confirmed as authentic. This is done via hashing. Hashing takes data of any size and transforms it into an unrecognizable format which cannot be reversed (N-able, 2019). The fixed length for the SHA-256 cipher is 256-bits. The bit levels are indicative of the security strength. Put simply – the more bits, the more potential combinations. For example, a brute force attack would need to make 2^256 attempts before it could gain the necessary data.

Random numbers are important when it comes to secure encryption. You never want the numbers generated for encryption to be predictable as it makes it easier for attackers to break the code. Java 8 offers the getInstanceStrong() method, in the SecureRandom class, which returns the cryptographically strongest random number generator available (Manico & Detlefsen, 2015). The SHA-256 uses a random number generator with its 2^256 combinations. This makes it valuable in avoiding collisions which happens when a hash value is the same for two separate pieces of data.

The cryptography used in encryption algorithms can be symmetric or asymmetric. Symmetric cryptography, as used in the AES algorithm, has one key designated for both encryption and decryption of the data. Here it is essential that the key remain secret and only be available to those with rights to the data. Asymmetric cryptography has two keys – one that is public for encryption and signing, and one that is private for decryption and validation. No matter which approach is chosen, the security of those keys will determine the success of their implementation.

With roots in Roman and Egyptian civilizations, cryptography has been around for quite some time (TutorialsPoint, n.d.). However, implementations have come a long way as security in the digital world becomes a necessity. Specifically, Data Encryption Algorithms (DES) were introduced in the early 1970’s and approved by the U.S. government for use (Loshin & Cobb, 2021). Advanced Encryption Algorithms (AES) are now the standard for encryption, with SHA-256 being required by U.S. government agencies to protect sensitive information (N-able, 2019). As computer processing becomes more powerful, the bar for bit levels must be raised to avoid brute force attacks. This makes it especially important to stay informed when it comes to encryption algorithm use.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

Text

Description automatically generated

CER file screenshot.

Text

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.

The unique data string has been encrypted using the SHA-256 cipher, and the value of the checksum is displayed.

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.

HTTPS enabled and secure communication successfully executed. There is warning that the certificate was self-signed.

Graphical user interface, text, application

Description automatically generated

This shows that the connection is encrypted and secure using TLS\_AES\_128\_GCM\_SHA256, TLS 1.3.

Text

Description automatically generated

An image of the self-signed certificate. This is provided in the attachment.

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.

A screenshot of the refactored code without any errors. This can be viewed in the attachment.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence

A screenshot of the dependency check report after refactoring the code shows that no new vulnerabilities were found. A copy of this html file is provided in the attachment’s target folder.

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.

Code has been refactored and executed without errors. File provided in attachment.

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence

The ssl-server\_student/pom.xml file in the attachment was revised to the current dependency check version so that all results are up to date. This executed without error.Graphical user interface, text, application

Description automatically generated

The application.properties file in the attachment was updated by adding a server and enabling HTTPS with the SSL keystore and self-signed certificate. This executed without error.

Graphical user interface, text, application, email

Description automatically generated

## 7. Summary

The areas of security that are addressed in the refactored code are client/server, cryptography, code error and code quality. Code error was handled by throwing an exception if the SHA-256 was not available for hashing in the SslServerApplication file. Cryptography was used to secure data transfer by enabling HTTPS in the application.properties, and by encrypting the data string with a SHA-256 cipher hashing algorithm in the SslServerApplication file. Client/Server was addressed by creating a self-signed certificate and enforcing the TLS connection to the web application. Code quality was implemented by addressing the areas of security that were lacking such as encryption, SSL certificate, TLS, HTTPS, and those identified by the dependency check.

The first layer of security addressed after reviewing the existing code for quality was the creation of an SSL certificate and the TLS connection. This is important not only for the security of Artemis Financial, but for the trust of their clients. It lets the clients know that the site is secure and that their communications are encrypted.

It’s clearly important that a client know their communications are secure, but it is equally important to Artemis Financial that their clients know their sensitive data is stored securely. This layer of security was added by implementing the SHA-256 encryption algorithm. This way if an attacker gained access to the data, they would be unable to decipher the contents.

Finally, HTTPS was enabled to ensure that the communications between the client and server are encrypted and secure. This layer of security is important because a client using an HTTP connection would be vulnerable to attacks as the data isn’t scrambled during transit. Forcing the user to use an HTTPS connection ensures their data is secure in transit. This adds precious value to Artemis Financial as they deal with primarily sensitive data from their clients.

Moving forward, I advise that the dependency check be run at least once a month. The results of the dependency check will inform the user as to what must be updated or revised. There is the possibility of newfound bugs and vulnerabilities in the program which would need to be addressed and resolved immediately. In the case that a newfound issue is without a current fix, it will help the user to remain vigilant until one is found. If any updates need to be made to the program code, practicing secure coding is essential. In summation, getting into the habit of running dependency checks monthly and practicing secure coding is truly a best practice for maintaining security for the life of the program.

**References**

Dworkin, M. J., Barker, E., Nechvatal, J., Foti, J., Bassham, L., Roback, E., & Dray Jr., J. (2021, March 1). *Advanced Encryption Standard (AES)*. NIST. https://www.nist.gov/publications/advanced-encryption-standard-aes

Loshin, P., & Cobb, M. (2021, August 20). *Data Encryption Standard (DES)*. SearchSecurity. https://www.techtarget.com/searchsecurity/definition/Data-Encryption-Standard

Manico, J., & Detlefsen, A. (2015). Iron-Clad Java: Building Secure Web Applications (Oracle Press) [E-book]. In *6 Protecting Sensitive Data* (1st ed., pp. 137–177). McGraw Hill.

N-able. (2019, September 12). *SHA-256 Algorithm Overview*. https://www.n-able.com/blog/sha-256-encryption

Oracle. (2017). *Java Security Standard Algorithm Names*. https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names

TutorialsPoint. (n.d.). *Origin of Cryptography*. https://www.tutorialspoint.com/cryptography/origin\_of\_cryptography.htm