****

# 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** | **2/20/22** | **Shaun Ryan** |  |

## 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

Shaun Ryan

## 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 will be processing and storing client’s financial and personal information. Keeping that data safe is not only a best practice, but also mandated by law for financial institutions. There are many flavors of encryption algorithms, each with their own strengths and weaknesses. In this application, I would recommend AES-256, the Advanced Encryption Standard with a 256-bit key length. The National Institute of Standards and Technology (NIST) announced in 2001 that for the AES it had chosen the Rijndael algorithm, “a symmetric block cypher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits.” (2001) Used by the US Government for the encryption for classified information, AES-256 would provide more than adequate protection for Artemis’ data.

Hash functions are one-way cryptographic functions used to verify the integrity of a block of information. Using a hash algorithm cypher, such as SHA-256 from the Secure Hash Algorithm family, a file can be run through the function which will produce a hash digest 256 bits in length. It is not possible to derive the original contents of the file from the hash, thus the classification as a one-way function. If even a single bit of data is changed in the original file, it will produce a different hash when run through the function. This makes it an efficient and secure way to verify the integrity of a file without having a known good copy to compare it to.

In order for encryption algorithms to function, they rely on the generation of random numbers for keys. Random numbers can be generated in a number of ways, using values from the system that vary in an unpredictable way. Cryptographic keys come in two varieties, symmetric and non-symmetric. With symmetric encryption, the same key that is used to encrypt data is also used to decrypt it. This practical when one entity will be doing both the encrypting and decrypting, such as a business performing on-site backups. Where symmetric encryption becomes problematic is transmitting information, as the key must be transmitted as well. This can be solved with non-symmetric keys, where one key encrypts the data, and a separate key decrypts the data.

Encryption algorithms predate computers and were one of the driving forces behind the development of computing machines, both mechanical and electrical. “As far back as 700 BC, the Spartan military used a device known as a Scytale which relied on a wooden rod that had to be identical in diameter and length at both sender and receiver locations in order to be decrypted (read).” (Allen, 2016) In the age of the digital computers, encryption algorithms progressed rapidly. The RSA algorithm was developed in the 1970s and was the first to utilize the public key/private key paradigm which is widely adopted by many standards today. In 2001, the NIST chose the Rijndael algorithm to be the basis of the Advanced Encryption Standard (AES), which has become the de facto algorithm for high security applications.

## 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.

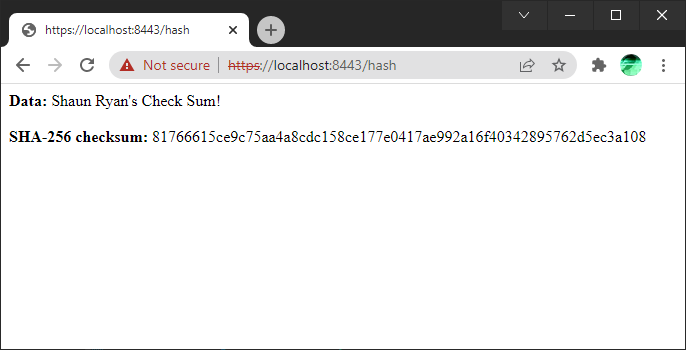
Text

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.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

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.

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

Text

Description automatically generated

Graphical user interface, text, application, email

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.

1. APIs: API issues were addressed by updating dependencies. The springframework-boot dependency was out of date and updated to the latest version.
2. Cryptography: Cryptography was address by the inclusion of the SHA-256 hash function and checksum verification.
3. Client/Server: The generation of a certificate and use of the SSL/HTTPS protocol addressed issues with insecure communication between the client and the server.
4. Code Quality: Code was manually reviewed for errors and security best practices.

The first step for increasing security was the generation of a self-signed certificate. This allows the client and the server to communicate via the SSL/HTTPS protocol. Using encrypted communications is paramount for a financial institution such as Artemis Financial. The next step before production will be to generate a certificate with a trusted CA.

In addition to utilizing HTTPS, a method for generating hashes was implemented. This will be useful for generating checksums to ensure the integrity of data transmitted between the server and the clients.

Utilizing a dependency check, 48 potential vulnerabilities were identified. By updating the springframework-boot package to the current version (from 2.2.4 to 2.6.3), the number of identified vulnerabilities dropped to 1. Dependencies are powerful and necessary tools for creating modern web applications, however they can compromise the security of your platform if not monitored and kept up to date. I would recommend frequent scheduled dependency checks and updating the production build as soon as it passes QA.

## 8. References

Allen, R. (2016, January 19). *The history of encryption*. MassTLC. Retrieved February 19, 2022, from https://www.masstlc.org/the-history-of-encryption/

U. S. National Institute of Standards and Technology. (2001, November 26*) Federal Information Processing Standards Publication 197: Announcing the ADVANCED ENCRYPTION STANDARD (AES)*