****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/18/2024** | **John Miller** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

John Miller

## Algorithm Cipher

1. Provide a brief, high-level overview of the encryption algorithm cipher.
2. Discuss the hash functions and bit levels of the cipher.
3. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.
4. Describe the history and current state of encryption algorithms.

Artemis Financial must implement robust encryption measures, including various encryption patterns and algorithms, to ensure secure communication and data integrity. First, data must be securely stored. This is achieved with the Advanced Encryption Standard (AES), a symmetric encryption algorithm that excels at encrypting large amounts of data. Symmetric cryptography uses a secret key to both encrypt and decrypt data. Due to Artemis Financials dispersed nature, a secure way to establish communication and transmit keys is needed.

An asymmetric cryptographic pattern, such as RSA, is then used to sign the encrypted data. The signature is transmitted via certificates authenticated and signed by a trusted third-party certificate authority (CA), allowing trusted communication from a previously unknown server. Asymmetric cryptography is essential to the certificate process; the private key is used to sign the certificate, and the public key is used to verify the signature. The secret key from AES can then be encrypted from the client side using the public RSA key and only be decrypted from the private key stored in the server, allowing the encrypted files to be transmitted and decrypted.

A hashing algorithm, such as the Secure Hashing Algorithm (SHA), provides a hash signature to the encrypted data. This hash signature is used to provide confidence in the data’s integrity. SHA256 uses 256-bit encryption with no known collisions. This means that even the slightest change of a data’s character will result in a new hash signature. This means a user can be confident that the data has not been changed due to corruption or a bad actor.

Encryption algorithms have a long and rich history. The oldest encryption algorithms are pre-biblical and involved replacement algorithms that shifting characters a specific number to encode messages. Over the years, encryption has evolved into a multi-step process that aims to produce hashes and encryptions with little to no collisions.

Malicious users always make an effort to discover collisions within the algorithm. Once a collision is discovered, the algorithm is considered broken, and encryptions are at risk of being reverse-engineered. A strong random number generator is pivotal to preventing collisions. Systems should implement strong random number generators for encryption or risk using similar patterns among various files, raising the risk of the encryption being broken.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

A screenshot of a computer

Description automatically generated

## 

## Secondary Testing

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

I updated the pom.xml file to include the most recent version of the dependency test and the newest stable version of SpringBoot (v3.3). This reduced the number of vulnerable dependencies from seventeen to one. The only known vulnerability is in the SpringBoot dependency spring-web-6.1.8.jar, which introduces a regex vulnerability that may lead to a ReDoS attack.

## Functional Testing

## A screenshot of a computer program Description automatically generated

## Summary

**Input Validation**—The program does not currently take input; however, a robust input validator will need to be implemented in the future to prevent malicious characters or commands from being accepted. All input must be properly sanitized and all database commands will need to be parameterized to prevent injection attacks.

**API** – The server controller class can now interact with the Springboot RESTful API and direct traffic via mapping.

**Cryptography** – The SHA hashing algorithm concocts a hash from input and correctly displays it at the <https://localhost:8443/hash> URL.

**Code Error** – Accessing the SHA hashing algorithm is correctly encased in a try-catch block, preventing a crash if the algorithm is unavailable.

**Client/Server**—An SSL layer is implemented, presenting the web browser with a self-signed certificate. In the future, this should also include a signature from a trusted third-party CA to establish trusted communication with users. This will help prevent man-in-the-middle attacks.

**Code Quality**—As the program continues to develop, a secure software development lifecycle (SSDLC) should be implemented. The program shall be continuously tested and examined for vulnerabilities. Security shall not be an afterthought but a conscious decision as the program develops.

**Encapsulation** – The program shall be developed in a modular fashion. This allows functionality to be continuously integrated and examined for vulnerabilities. This also helps limit the amount of damage caused by a broken function. This is demonstrated by the myHash() function calling the helper function calculateHash(), which in turn calls the helper function bytesToHex().

## Industry Standard Best Practices

## The focus of security was once an afterthought during the SDLC. More recently, the industry has strived to be more conscious of security, and many organizations are implementing an SSDLC. This takes the traditional pipeline of productivity seen in an iterative approach to software development and filters it through security awareness and testing. Programs must be developed to understand the type of attacks they may be subject to. Federal/local regulations must always be followed, and the consumers' trust must be earned. As a worldwide organization, Artemis Financial must implement secure coding practices to handle user data and communication. The integrated SSL layer gives users confidence that the application represents who they say they are. The hashing algorithm helps prevent man-in-the-middle attacks where data is intercepted and altered. The hash changes drastically at any change in the original data compared to the original hash when received. Dependencies should be regularly updated and scanned so that any vulnerabilities are known and mitigated. I updated the SpringBoot dependency in the pom.xml file to version 3.3 (the most recent version) and had to update the Java library to version 17, and the program was still successfully compiled. This drastically cut down the number of vulnerabilities discovered during the OWASP scan. This should be a continual practice throughout the SSDLC and lifetime of the program, as new vulnerabilities are discovered and then addressed in newer versions