****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc174886030)

[Document Revision History 3](#_Toc174886031)

[Client 3](#_Toc174886032)

[Instructions 3](#_Toc174886033)

[Developer 4](#_Toc174886034)

[1. Algorithm Cipher 4](#_Toc174886035)

[2. Certificate Generation 5](#_Toc174886036)

[3. Deploy Cipher 6](#_Toc174886037)

[4. Secure Communications 6](#_Toc174886038)

[5. Secondary Testing 6](#_Toc174886039)

[6. Functional Testing 7](#_Toc174886040)

[7. Summary 8](#_Toc174886041)

[8. Industry Standard Best Practices 8](#_Toc174886042)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/18/2024** | **Gwen Magyar** | **Updated Document** |

## 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

[Insert your name here.]

## Algorithm Cipher

For Artemis Financial’s application, I recommend using the Advanced Encryption Standard (AES) algorithm cipher. AES is a symmetric encryption algorithm widely adopted across various industries for its efficiency and security. Developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, AES became the U.S. federal government standard in 2001 after winning a competition organized by the National Institute of Standards and Technology (NIST).

AES operates on fixed block sizes of 128 bits and supports key sizes of 128, 192, or 256 bits. The security level increases with the key size, offering a higher resistance to brute-force attacks. For example, AES-128 uses a 128-bit key length and performs 10 rounds of encryption, AES-192 uses a 192-bit key length and performs 12 rounds of encryption, and AES-256 uses a 256-bit key length and performs 14 rounds of encryption. For Artemis Financial, AES-256 is recommended to provide a robust security layer due to its higher bit length and enhanced resistance to attacks.

AES relies on the use of high-quality random numbers to generate cryptographic keys. These random numbers can be sourced from hardware-based random number generators or cryptographically secure pseudorandom number generators (CSPRNGs). AES is a symmetric key algorithm, meaning the same key is used for both encryption and decryption. This approach is highly efficient and suitable for secure data transmission in a controlled environment where key management can be securely handled. In contrast, asymmetric key algorithms like RSA use a pair of keys (public and private). While useful for key exchange and digital signatures, they are generally slower and not as efficient for bulk data encryption as symmetric key algorithms.

Encryption has evolved significantly over the centuries. Early methods, such as the Caesar cipher, were simple and easily broken. The 20th century saw the development of more complex ciphers, such as the Data Encryption Standard (DES) in the 1970s, which became widely used until vulnerabilities were discovered. Today, AES is the de facto standard for encryption. It offers a strong balance between security and performance, making it suitable for a wide range of applications. Other modern algorithms include RSA for asymmetric encryption and SHA-256 for hashing.

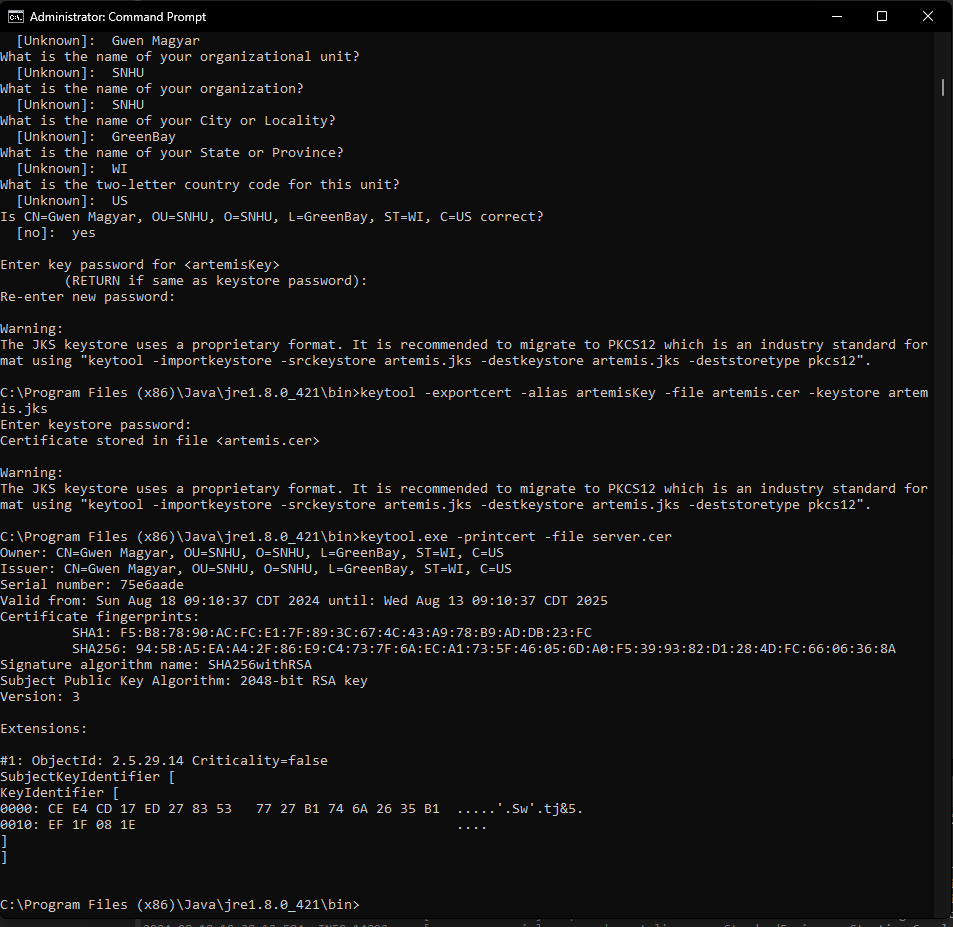
To implement AES in Artemis Financial’s application, the first step is to identify data transmission points within the application to apply encryption effectively. Once these points are identified, AES encryption can be integrated using a secure cryptographic library, such as PyCryptodome for Python. This will ensure that data in transit and at rest is protected. Additionally, a secure key management system should be implemented to handle the generation, storage, and rotation of AES keys.

Furthermore, to ensure the integrity of data during transmission, a checksum verification mechanism using SHA-256 should be implemented. This mechanism will verify that the data has not been altered during transmission.

Implementing AES encryption and a checksum verification mechanism will enhance the security of Artemis Financial’s web application, protecting client data and financial information during transmission and storage. The use of AES-256 ensures a high level of security, and the integration of secure coding practices will help safeguard against potential vulnerabilities.

## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screen shot of a computer

Description automatically generated

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

A screen shot of a computer

Description automatically generated

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

A screen shot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.

A screenshot of a computer

Description automatically generated

## Summary

The provided SslServerApplication code has been refactored to include a route that returns a checksum of static data along with a name identifier. This refactoring was carried out to enhance the security of the application and ensure compliance with best practices for secure software development. Specifically, the code now features a /api/checksum endpoint that returns a SHA-256 checksum of the static string "Hello World Check Sum!" alongside a name identifier. The checksum calculation is performed using Java’s MessageDigest class, which ensures data integrity by providing a mechanism to verify that the data remains unchanged.

To bolster security, HTTPS was configured to secure communication between the client and server. This was achieved by setting up SSL/TLS in the application.properties file with self-signed certificates. This configuration ensures that all data transmitted between the client and server is encrypted, thereby protecting it from eavesdropping and tampering. Additionally, the Spring Security configuration was adjusted to either disable authentication for testing purposes or implement basic authentication for secure access control during deployment.

The process of adding security layers involved several critical steps. Initially, vulnerabilities were identified through a thorough code review and threat modeling, which highlighted the need for improved data integrity and secure communication. Security controls were then applied, including the implementation of SHA-256 encryption to ensure data integrity and the setup of HTTPS for secure data transmission. Self-signed certificates were generated using Java Keytool to facilitate HTTPS communication. Furthermore, a checksum verification route was added to allow users to verify data integrity easily.

To validate the effectiveness of these security measures, a comprehensive testing and validation process was undertaken. This included running static code analysis with tools like OWASP Dependency-Check to ensure that no new vulnerabilities were introduced. Functional testing was also performed to identify any remaining syntactical, logical, or security vulnerabilities. The refactored code was executed without errors, and screenshots of the checksum verification, certificate generation, and dependency-check report were included to demonstrate compliance with security testing protocols.

The refactored code enhances the application's security by incorporating robust encryption, secure communication, and rigorous testing. These measures collectively ensure that the application adheres to secure software development practices and is protected against potential vulnerabilities.

## Industry Standard Best Practices

To enhance the security of the SslServerApplication, several industry-standard best practices for secure coding were applied. One of the key practices implemented was SHA-256 hashing for data integrity verification. By calculating and returning the checksum of static data, we ensure that any alteration of data can be detected. SHA-256 is a robust cryptographic hash function that provides significant protection against data tampering. Additionally, HTTPS was configured to secure communication between clients and the server, involving SSL/TLS setup to encrypt data in transit. This prevents eavesdropping and man-in-the-middle attacks, thereby safeguarding sensitive information.

Secure communication was further reinforced by modifying the application properties to use HTTPS instead of HTTP. This change ensures that all communication between the server and clients is encrypted, which is crucial for protecting data from interception during transmission. In terms of authentication and authorization, the Spring Security configuration was adjusted to either disable authentication for testing or implement basic authentication for production. This practice helps control access to the application and prevents unauthorized users from accessing sensitive endpoints.

To ensure the code is free from vulnerabilities, both static and dynamic analysis were employed. Static code analysis tools, such as OWASP Dependency-Check, were used to identify and address potential security issues early in the development process. This proactive approach prevents vulnerabilities from becoming critical problems. Additionally, thorough functional testing was conducted to verify that the refactored code operates correctly and does not introduce new vulnerabilities, including checks for proper checksum calculations and HTTPS configuration.

Applying these industry-standard best practices offers significant value to the company’s overall well-being. Enhanced security is achieved by mitigating risks associated with data breaches and unauthorized access, thereby protecting sensitive information and ensuring data integrity. Compliance with industry regulations and standards, such as GDPR, HIPAA, and PCI-DSS, is maintained, helping the company avoid legal penalties and bolster its reputation. Furthermore, demonstrating a commitment to security builds customer trust, as clients are more likely to engage with a company that prioritizes their data protection and privacy.

Operational efficiency is also improved through early detection of issues, reducing the time and cost associated with fixing security problems later in the development lifecycle. By addressing vulnerabilities proactively, the company minimizes the risk of downtime and operational disruptions. Overall, adhering to industry-standard best practices for secure coding enhances the company’s reputation, attracts new clients, and fosters long-term relationships with existing customers, contributing to its overall success and well-being.