****

# 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** | **[8/10/2023]** | **[Joel Clark]** | **First edits** |

## 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

[Joel Clark]

## Algorithm Cipher

Given the scenario, I recommend using AES cipher for protecting our company’s data. AES is a symmetric encryption algorithm, which means the same key is used for both encryption and decryption. It operates on fixed-size blocks of data and supports key sizes of 128, 192, or 256 bits. AES has been adopted as the encryption standard by many organizations and is widely used in various applications, including financial and government sectors.

AES does not directly involve hash functions, as it's primarily an encryption algorithm. Hash functions are used for creating fixed-size "fingerprints" of data, while AES is used to encrypt and decrypt data. The bit levels of AES depend on the key size: AES-128 uses a 128-bit key, AES-192 uses a 192-bit key, and AES-256 uses a 256-bit key.

AES uses random numbers for generating encryption keys. These keys are then used for both encryption and decryption. AES employs a symmetric key approach, where the same key is used for both encryption and decryption operations. This key must be securely shared between the parties involved.

AES was established in 2001 by the U.S. National Institute of Standards and Technology (NIST) to replace the aging Data Encryption Standard (DES). It was chosen after an extensive evaluation process due to its strong security properties and performance. AES has since become the de facto standard for encryption and is widely used worldwide.

## Certificate Generation

Insert a screenshot below of the CER file.

A computer screen with white text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A computer screen with a white screen

Description automatically generated

A computer screen with a white screen

Description automatically generated

## Secure Communications

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

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

A computer screen with a white screen

Description automatically generatedA computer screen with text on it

Description automatically generated

## Functional Testing

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

A computer screen with text on it

Description automatically generated

## Summary

The refactored code has been designed to adhere to security testing protocols and enhance overall security by implementing multiple layers of security measures. A primary focus was placed on securely managing sensitive information, such as the keystore path and password, and addressing potential vulnerabilities related to cryptography and input validation.

Sensitive information handling was a critical aspect of the refactoring process. In the initial code, sensitive data was hardcoded directly into the code, which posed security risks. To mitigate this, the code was restructured to utilize Spring's property injection mechanism. The sensitive information is now stored externally in a dedicated configuration file named application.properties, and it is injected into the code using the @Value annotation. This separation of configuration from code improves security by preventing sensitive information from being embedded in the source code, making it easier to manage and update as needed.

The code's cryptographic operations were maintained while ensuring that cryptographic algorithms were implemented securely. A more structured approach to generating checksums was introduced to address potential vulnerabilities associated with data integrity. The refactored code also emphasizes proper input validation to sanitize user-provided data before using it. This validation helps prevent common attack vectors like SQL injection and Cross-Site Scripting (XSS), thereby enhancing the application's overall security posture.

The refactored code follows the principle of layered security, which divides the application into distinct layers, each handling specific concerns. By introducing a configuration class and a controller class while leveraging Spring's dependency injection, the application's architecture now supports well-defined layers. This separation not only enhances maintainability but also improves security by minimizing the risk of unintended interactions between different parts of the application.

Throughout the refactoring process, compliance with security testing protocols was a priority. By incorporating secure coding practices, adhering to established cryptographic standards, and implementing best practices for input validation, the code aligns with modern security requirements. The introduction of externalized configuration, secure cryptographic algorithms, and careful input validation contributes to a more robust and secure software application.

In summary, the code refactoring effort successfully enhanced the software's security posture by incorporating layered security, addressing sensitive information handling, and improving cryptographic operations and input validation. These security practices collectively contribute to the development of a more secure, resilient, and compliant application that aligns with current security standards and best practices.

## Industry Standard Best Practices

In the process of refactoring the software application, several industry-standard best practices for secure coding were applied to mitigate against known security vulnerabilities and maintain the software's current security.

One of the core practices adopted was the principle of input validation. By sanitizing and validating user-provided data before processing it, the application was safeguarded against common vulnerabilities like SQL injection and Cross-Site Scripting (XSS) attacks. This practice ensures that malicious input cannot be used to compromise the integrity or security of the application.

Another key practice involved secure data handling. Sensitive information, such as the keystore path and password, was removed from the source code and stored externally in a configuration file. This separation of configuration from code prevents inadvertent exposure of sensitive data and simplifies the process of updating or rotating credentials. By adhering to this practice, potential security breaches due to hardcoded sensitive information were mitigated.

Industry-standard cryptographic algorithms and protocols were employed to safeguard data integrity and confidentiality. The use of SHA-256 for checksum generation, a widely recognized and secure cryptographic hash algorithm, ensures the integrity of the data without compromising user privacy. Employing such trusted algorithms helps to prevent unauthorized tampering with data and maintains the application's security.

By following the Spring framework's dependency injection mechanism, the code was organized into distinct layers, enhancing maintainability and security. The introduction of dedicated controller and configuration classes enforces separation of concerns and minimizes the risk of unintended interactions, ultimately contributing to a more robust and secure architecture.

Applying industry-standard best practices for secure coding offers substantial benefits to the company's overall well-being. It ensures that the software application remains resilient against known security vulnerabilities and emerging threats. By mitigating security risks, the company safeguards its reputation, customer trust, and sensitive data from potential breaches, leading to reduced financial losses and legal liabilities. Additionally, adherence to secure coding practices aids in compliance with regulatory standards and industry certifications, which are increasingly vital in today's cybersecurity landscape.

In conclusion, the application of industry-standard best practices for secure coding in the software refactoring process significantly contributed to maintaining and enhancing the application's security. This approach not only fortifies the software against known vulnerabilities but also bolsters the company's overall well-being by protecting valuable assets, maintaining customer trust, and ensuring compliance with industry regulations.