****

# 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** | **4/26/2024** | **Marc Aradillas** |  |

## Client



## Instructions

Submit these 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

Marc Aradillas

## Algorithm Cipher

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that generates a 256-bit (32-byte) hash value from an input data stream of arbitrary length. It belongs to the SHA-2 (Secure Hash Algorithm 2) family of cryptographic hash functions, designed by the National Security Agency (NSA) in the United States. SHA-256 is widely used for various security applications, including digital signatures, message authentication, and data integrity verification.

* **Hash Functions and Bit Levels:**
* SHA-256 operates by transforming the input data into a fixed-size output, irrespective of the input's size.
* It uses a series of logical operations, bitwise operations, and modular arithmetic to process the input data in blocks and produce the final hash value.
* The output hash value is 256 bits (32 bytes) long, providing a high level of cryptographic security due to the large output space.
* **Use of Random Numbers, Symmetric vs. Non-symmetric Keys:**
* SHA-256 is a cryptographic hash function and does not directly involve random numbers or keys. It is a one-way function, meaning it cannot be reversed to retrieve the original input from the hash output.
* While SHA-256 itself does not involve keys, it is commonly used in conjunction with symmetric or asymmetric encryption algorithms to ensure data integrity and authentication.
* In the context of Artemis Financials’ web application, SHA-256 can be used to generate checksums for files or data being transmitted, ensuring data integrity during communication.
* **History and Current State of Encryption Algorithms:**
* Encryption algorithms have evolved over time in response to advancements in computing power and cryptographic attacks.
* SHA-256 was introduced as part of the SHA-2 family in 2001, replacing the earlier SHA-1 algorithm due to vulnerabilities discovered in SHA-1's collision resistance.
* As of now, SHA-256 is widely regarded as secure and resistant to known cryptographic attacks, providing a high level of security for data integrity and authentication purposes.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generatedA computer screen with text on it

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

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

A screenshot of a computer

Description automatically generatedInsert screenshots below of the refactored code executed without errors and the dependency-check report.

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 program

Description automatically generated

A screen shot of a computer

Description automatically generated

## Summary

**Architecture Review:** The code has been refactored to ensure that it follows secure application architecture principles. This includes structuring the application in a secure manner to mitigate potential vulnerabilities.

**Input Validation:** The refactored code includes input validation to ensure that data received from external sources, such as user input through the API endpoint, is properly validated to prevent injection attacks and other security vulnerabilities.

**Cryptography:** The use of cryptography, specifically in generating the SHA-256 checksum, has been carefully implemented to mitigate vulnerabilities related to encryption and hashing algorithms. The code utilizes a well-established algorithm (SHA-256) and handles exceptions securely.

**Code Error Handling:** Secure error handling practices have been incorporated into the code. Error messages are logged using the SLF4J logging framework, ensuring that sensitive information is not leaked to potential attackers.

**Code Quality:** The refactored code adheres to secure coding practices and patterns. It follows industry-standard conventions and maintains readability, which is essential for identifying and mitigating security vulnerabilities during code reviews.

**Encapsulation:** Secure data structures are used throughout the code to encapsulate sensitive information. This helps prevent unauthorized access to data and reduces the risk of data breaches.

By systematically addressing these areas of security through refactoring and implementing additional security measures, the software application is better equipped to withstand potential security threats. The process involved careful consideration of security principles at each stage, from input validation to error handling, ensuring that the application is robust and resilient to attacks.  
  
The refactoring process has enhanced the security posture of the software application by incorporating layers of security measures, from architectural design to code implementation. The findings from the vulnerability assessment process have been considered and mitigated through a structured approach, resulting in a more secure and reliable software application.

## Industry Standard Best Practices

**Input Validation:** By validating input data received from external sources, such as user input through API endpoints, we mitigate risks associated with injection attacks, including SQL injection and cross-site scripting (XSS). Input validation ensures that only expected and properly formatted data is processed, reducing the likelihood of malicious input causing security vulnerabilities.

**Cryptography:** We utilized industry-standard cryptographic algorithms and best practices for hashing to generate the SHA-256 checksum. By using well-established algorithms and handling cryptographic operations securely, we mitigate vulnerabilities related to encryption use and potential cryptographic weaknesses.

**Secure Error Handling:** Secure error handling practices, such as logging error messages with the SLF4J logging framework, were implemented to prevent sensitive information leakage and provide adequate information for debugging and troubleshooting. Proper error handling ensures that potential attackers are not provided with useful information that could aid in exploiting vulnerabilities.

**Code Quality and Secure Coding Practices:** Adhering to industry-standard secure coding practices and patterns ensures that the codebase is maintainable, readable, and resilient to common security vulnerabilities. By following established conventions and patterns, such as avoiding hardcoded sensitive information and implementing proper access control mechanisms, we reduce the likelihood of introducing security flaws during development.

**Regular Security Audits and Reviews:** Continuous monitoring and periodic security audits and code reviews help identify and mitigate security vulnerabilities in the software application. By regularly reviewing the codebase for potential vulnerabilities and addressing them promptly, we maintain the application's security posture and reduce the risk of security breaches.

**Value of Industry Best Practices:**

**Protecting Customer Data:** By adhering to secure coding practices, we protect sensitive customer data from unauthorized access, manipulation, and theft. This helps maintain customer trust and loyalty, which are critical for the company's reputation and long-term success.

**Mitigating Legal and Regulatory Risks:** Compliance with industry standards and regulations, such as GDPR, HIPAA, or PCI DSS, is crucial for avoiding legal penalties and regulatory fines. Secure coding practices help ensure that the software application meets these requirements, reducing the company's exposure to legal and regulatory risks.

**Preventing Financial Losses:** Security breaches can result in significant financial losses due to data breaches, lawsuits, and damage to the company's reputation. By mitigating security vulnerabilities through secure coding practices, we minimize the risk of financial losses associated with security incidents.

**Maintaining Competitive Advantage:** In today's competitive landscape, customers expect software applications to be secure and trustworthy. By demonstrating a commitment to security through the implementation of industry-standard best practices, we differentiate ourselves from competitors and attract customers who prioritize security and reliability.