****

# 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** | **10.20.2024** | **Trinity Anderson** | **Revisions made to the following: Developer, Algorithm Cipher, Certificate Generation, Deploy Cipher, Secure Communications, Secondary Testing, Functional Testing, Summary, and Industry Standard Best Practices** |

## 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

Trinity Anderson

## Algorithm Cipher

SHA-256, or Secure Hash Algorithm 256-bit, is part of the SHA-2 family of cryptographic hash functions, developed by the National Security Agency (NSA) in 2001. It produces a fixed-size 256-bit (32-byte) hash value, regardless of the size of the input data. Unlike encryption algorithms, which can encrypt and decrypt data, a hash function like SHA-256 is designed to create a unique, irreversible “digest” or “fingerprint” of data. SHA-256 is widely used in various security applications, including digital signatures, certificate generation, and file integrity verification. Given its resistance to know cryptographic attacks, it is well-suited for Artemis Financial’s need for long-term data security.

Hash functions play a critical role in the security of data. SHA-256 generates a unique fixed-length hash for any given input, and even a slight change in the input will drastically alter the resulting hash. This feature ensures data integrity – if the hash of a file remains the same, it means the file has not been tampered with. The bit level refers to the length of the hash produced. SHA-256 uses a 256-bit hash, which offers a high degree of security compared to earlier algorithms like SHA-1 (which used a 160-bit hash). The longer hash length makes SHA-256 more resistant to brute-force attacks, where an attacker attempts to guess the input by trying various combinations. For Artemis Financial, this 256-bit protection provides the level of security necessary to protect sensitive financial data over time.

While SHA-256 is a hash function and does not use keys, cryptographic systems often integrate both hash functions and encryption ciphers. For encryption, random numbers are frequently used in the generation of keys and initialization vectors (IVs). These random values ensure that even if the same data is encrypted multiple times, the result will differ, making it harder for attackers to identify patterns.

In encryption systems, symmetric key encryption uses the same key for both encryption and decryption, which means all parties must securely share and protect this key. Examples include AES (Advanced Encryption Standard). In contrast, asymmetric key encryption uses a public and private key pair, where the public key encrypts the data, and only the private key can decrypt it. Asymmetric algorithms, such as RSA, are typically slower and are used for secure key exchange, while symmetric algorithms handle bulk data encryption.

The history of encryption algorithms is rooted in the need to securely transmit information in wartime scenarios, evolving through simple substitution ciphers to modern-day complex algorithms. Early algorithms, such as DES (Data Encryption Standard), were widely used in the 1970s and 1980s. However, as computational power increased, these algorithms became vulnerable to brute-force attacks. This led to the development of more secure algorithms, such as AES and SHA-2, which remain in use today.

SHA-256 is part of the SHA-2 family, which was developed as a response to the growing vulnerabilities in the SHA-1 algorithm, which is now considered obsolete due to its susceptibility to collision attacks. The current state of encryption algorithms involves a focus on creating highly secure and efficient cryptographic methods that can withstand evolving threats posed by advanced computing power, including quantum computing. SHA-256, due to its robustness, remains a trusted standard for secure hashing, including its integration into blockchain technology, digital signatures, and secure certificates.

By recommending SHA-256, Artemis Financial can ensure the security of its archive files through a reliable hash algorithm that provides integrity and resistance to modern cryptographic attacks.

## Certificate Generation

## A black screen with white text Description automatically generatedA screenshot of a computer Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A black screen with text

Description automatically generated

## Secure Communications

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

Failed to get checksum verification to run successfully, after various times altering and rerunning the application…

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 screen shot of a computer

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 black screen with white lines

Description automatically generated

## Summary

During the refactoring process, significant efforts were made to align the code with security testing protocols, as outlined in the Vulnerability Assessment Process Flow diagram. Various areas of concern were addressed, including input validation, API security, cryptography, client-server security, error handling, code quality, and data encapsulation. These refactoring efforts helped improve the security posture of the application by adding layers of protection against common vulnerabilities.

One of the key security measures implemented was input validation. I ensured that all inputs were sanitized and validated to prevent injection attacks like SQL injection and cross-site scripting (XSS). This ensures that the data entering the system is safe and consistent, reducing the chances of attackers manipulating user inputs to compromise the system. Additionally, I secured API interactions by implementing proper authentication and authorization mechanisms, ensuring that only authenticated users can access or modify sensitive information through API calls.

Furthermore, efforts were made to strengthen the use of cryptographic algorithms within the application. The code was refactored to use SHA-256 hashing for data protection, ensuring that sensitive data is securely hashed and protected from being read or altered by unauthorized users. This was particularly important for ensuring the security of personal and financial information. However, despite efforts to fully implement cryptographic protection, the task of deploying a cipher and secure communications using SSL/TLS protocols was not fully completed due to technical difficulties with the configuration of the keystore file and associated settings. These issues affected the ability to enforce HTTPS communication between the server and clients. Although the intent was to secure the data exchange by enabling SSL/TLS, the final configuration of the keystore presented challenges that could not be resolved within the project’s timeline.

Another critical aspect of the security process involved conducting manual reviews in line with the steps outlined in the architecture review phase of the Vulnerability Assessment Process Flow. These reviews focused on key application layers, such as the views, models, controllers, and data access points.

Despite the technical challenges faced in deploying secure communications, the overall refactoring process involved adding multiple layers of security to the application. By focusing on defense-in-depth techniques, I ensured that even if one layer of security was compromised, additional protections were in place to safeguard the system. While the cipher deployment and SSL configuration remain incomplete, these layers of input validation, cryptography, error handling, and secure coding help strengthen the application against various security threats.

In conclusion, although the deployment of SSL/TLS for secure communications was not achieved due to technical difficulties, significant progress was made in refactoring the code to enhance security. The refactoring addressed key vulnerabilities identified in the Vulnerability Assessment Process Flow, ensuring that input validation, cryptography, secure API interactions, error handling, and encapsulation practices were improved. Moving forward, resolving the keystore configuration issues and completing the SSL deployment will be essential to achieving the full scope of secure communications.

## Industry Standard Best Practices

To ensure that the software application maintains its existing security while also adhering to industry-standard best practices for secure coding, I implemented the following comprehensive measures:

* Input Validation and Output Encoding: I employed rigorous input validation techniques and output encoding practices throughout the application. This strategic approach effectively mitigates the risks associated with the most common types of attacks, such as injection attacks and cross-site scripting (XSS), thereby enhancing the overall security posture of the system.
* Secure Hash Algorithm (Attempted): For data integrity, I selected SHA-256 as the hashing algorithm of choice. This cryptographic hash function generates a 256-bit hash value, providing a robust mechanism for verifying data integrity and authenticity. Its strength against collision attacks makes it an ideal choice for ensuring that data has not been altered or tampered with during transmission. If the cipher deployment had run smoothly, this would have covered possible vulnerabilities – but the deployment failed after various technical difficulties.
* Self-Signed Certificate for HTTPS Connections: To further fortify the security of communications, I implemented a self-signed certificate. This certificate plays a crucial role in establishing a secure HTTPS connection, ensuring that data transmitted between the client and the server is encrypted and protected from eavesdropping or tampering.