****

# 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** | **4/20/25** | **Jermaine Wiggins** | **Full report** |

## 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

Jermaine Wiggins

## Algorithm Cipher

To improve the security of Artemis financials web application and ensure a secure line of communication a file verification step will be added. To do this, a SHA-256 hash function will be used to generate a checksum of any file that gets transferred through the system. This checksum will verify the file hasn’t been tampered with during transit and provides a layer of protection by ensuring the integrity of the data. If the checksum generated on the receiving end matches the original, it confirms the file is unchanged and secure.

The SHA-256 is part of the SHA-2 family and generates a 256 bit has value making it resistant to collisions and brute force attacks. During the encryption process a fixed length hash value of 256bits is generated from the input data regardless of the inputs size. This hash value acts as an identifier as the same input will always produce the same value and even the slightest change will produce a completely different value.

Random numbers can be used in combination with the input to ensure if the file is hashed multiple times, it receives a different value each time. Symmetric keys provide one key used for encryption and decryption, while asymmetric keys from uses a pair of keys one to encrypt and one to decrypt.

In the 1970s the Data Encryption standard was the gold standard for security but was easily broken because if its 56-bit length it was replaced by the Advanced Encryption standard AES and is considered one of it not the most secure algorithm cipher today. On the hashing side SHA-1 was once widely used but later was found to be vulnerable to collisions, making it easy for a hacker to trick the system with matching hashes. The ShA-2 was introduces and has no known vulnerabilities to date.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A black text on a white background

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

## Secure Communications

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

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Note: my browser (Chrome) doesn't allow trusting of self-signed certs

## Secondary Testing

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

report.

Dependency report before refactoring

A screenshot of a computer

AI-generated content may be incorrect.

Dependency report after refactoring (no new vulnerabilities added)

A screenshot of a computer

AI-generated content may be incorrect.

Code executed with no errors

A screenshot of a computer program

AI-generated content may be incorrect.

## Functional Testing

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

A screenshot of a computer program

AI-generated content may be incorrect.

## Summary

To improve the security of the Artemis Financial application, the code was refactored to include a SHA-256 checksum feature, addressing vulnerabilities related to data integrity. This change aligns with key phases of the vulnerability assessment process, such as implementing and verifying security controls. A new hash endpoint was added to demonstrate secure hashing using Java’s Message Digest, with error handling to ensure application stability.

## Industry Standard Best Practices

In refactoring the Artemis Financial application, I implemented several industry-standard best practices to enhance security and ensure compliance with secure coding guidelines. This included using the SHA-256 cryptographic hash algorithm to securely verify data integrity, which aligns with NIST recommendations for strong hashing mechanisms. I also incorporated secure exception handling to prevent the exposure of sensitive internal information, and I avoided hardcoding credentials or exposing critical logic in client-facing code. By adhering to secure coding standards, such as those recommended by OWASP and NIST, the software becomes more resilient against common threats like injection attacks, broken authentication, and information disclosure.