****

# 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** | **6/21/2024** | **Jason Barry** |  |

## Client



## Instructions

Submit these completed practices for a 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

Jason Barry

## Algorithm Cipher

*Artemis Financial* seeks to utilize a cryptographic hash function to ensure secure communications. This function should transform data into a hash value known as a checksum which should allow the company to download a public key from a website and verify the key with the aforementioned checksum. Once implemented, the checksum would prevent attacks originating from disguised malicious data. There are a variety of cryptographic hash functions that the company can utilize to achieve its objectives. Unfortunately, many suffer from collisions which leave the program vulnerable. Luckily the SHA-256 algorithm avoids this issue by verifying the content of transmitted data without exposing the data. When used, this one-way algorithm makes it nigh-impossible to reconstruct the original input based on the provided output. Although the SHA-256 algorithm can be attacked through brute force, this is unlikely. After all, there are “…more combinations than the number of atoms in the universe” (What is the Most Secure Hashing Algorithm?). The sheer number of combinations means that the SHA-256 algorithm has never been successfully reverse-engineered. This trait makes the SHA-256 algorithm the most secure software algorithm the company may utilize

## Certificate Generation

A screenshot of a computer program

Description automatically generated

## Deploy Cipher

A screenshot of a computer

Description automatically generated

## Secure Communications

## Because a self-signed keystore was used, the connection technically isn’t secure. Going forward, *Artemis Financial* must not utilize a self-signed keystore.

A screenshot of a computer

Description automatically generated

Once again, a self-signed keystore cannot create an authenticated connection because it does not appear in the Trusted Root Certification Authorities store.

A screenshot of a certificate

Description automatically generated

## Secondary Testing

## Refactored Code Executed Without Errors

## A screenshot of a computer program Description automatically generated

**Pre-Refactorization Code – Dependency Check**

A screenshot of a computer

Description automatically generated

**Post-Checksum Verification – Dependency Check**

As you can see, no vulnerabilities were added due to the refactored code. For clarification, this screenshot was taken before the one taken for the *Pre-Refactorization Code* because I forgot to take that screenshot when I began the assignment.

**A screenshot of a computer

Description automatically generated**

**Post Suppressions – Dependency Check**

## Suppressions were necessary to ensure none of the remaining vulnerabilities represent false positives.

**A screenshot of a computer

Description automatically generated**

## 

## Functional Testing

## Refactored Code, Executed Without Errors

## A screenshot of a computer program Description automatically generated

## Added Suppression File Without Errors

## A suppression file was necessary to ensure none of the vulnerabilities remaining represent false positives.

## A computer screen shot of a program Description automatically generated

## Summary

The refactored code complies with security testing protocols through its use of cryptography and the elimination of errors in the code. As a cryptographic hash function used to encrypt data, SHA-256 helps prevent vulnerabilities. This is accomplished because the algorithm transforms data into a hash value that is nigh impossible to reconstruct. The use of this algorithm ensures all communications adhere to the guidelines set by the Federal Trade Commission Act (FTCA). Another avenue used to ensure the refactored code complies with security testing protocols was through the elimination of errors in the code. Faulty code can leave the program vulnerable to external threats. Therefore, continuous testing was performed to ensure the refactored code produced no errors. The lack of errors helped ensure no new vulnerabilities were added once the code was successfully refactored. When combined, these two security methods help ensure the refactored code complies with security testing protocols.

## Industry Standard Best Practices

During refactorization, I utilized the industry-standard best practices for mitigating known security vulnerabilities. Many of these practices were utilized including the aforementioned use of cryptography and the elimination of errors in the code. However, the primary practice I applied during development was that I kept security in mind from the very beginning. No line of code was added without understanding the potential security risks that such an addition could spawn. Utilizing this thought process helped maintain the software application’s existing security. The use of these practices also helps ensure the overall well-being of *Artemis Financial.* As mentioned before, there are various guidelines set by the FTCA to ensure secure communications. Failure to abide by these rules would result in fines that would cut into the company’s profits. Furthermore, if users discover that the system was compromised, they will lose trust in the company. This lack of trust will convince customers to abandon *Artemis Financial* in favor of a more secure consulting company. When combined, these potential issues would cripple the company’s overall well-being. Luckily, these problems are easily avoided by following the aforementioned industry-standard best practices.

Works Cited

Brands, M. (2023, February 15). Cybersecurity Laws and Legislation (2024 update). ConnectWise. <https://www.connectwise.com/blog/cybersecurity/cybersecurity-laws-and-legislation#:~:text=The%20primary%20law%20governing%20cybersecurity,those%20related%20to%20data%20security.>

Java Security Standard Algorithm Names. (n.d.). <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names>

What is the Most Secure Hashing Algorithm? Code Signing Store. (2022, February 14). <https://codesigningstore.com/what-is-the-most-secure-hashing-algorithm#:~:text=To%20the%20time%20of%20writing,secure%20hashing%20algorithm%20out%20there.>