****

# 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** | **Daniel Leon** |  |

## 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

Daniel Leon

## Algorithm Cipher

For Artemis Financial’s needs, I would recommend the SHA-256 hash algorithm. It is an irreversible function, meaning you should not get the original data if passed through the function again. It is one of the strongest hashing functions to exist today with one of the lowest probabilities of collisions occurring. A collision happens when any two pieces of data are hashed with the same exact value. SHA-256 has 2^256 possible hash values which yields an extremely low probability of collisions.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer screen

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.

I was not able to make a secure connection using a self-signed certificate. I searched for days and hours for an answer to this and all I’ve seemed to come across were unsecure workarounds. It seems that it is no longer possible to get any up-to-date browsers to trust a self-signed certificate. I’m not sure if I was expected to either download an older version of a browser or buy a CA certified certificate. The project 2 instructions did not seem to expect either of those actions from me.

## Secondary Testing

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

Before refactoring:

A screenshot of a computer

Description automatically generated

After refactoring:

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

Description automatically generated

A screenshot of a computer

Description automatically generated

## Summary

Creating the hashing algorithm for any data to be transferred was probably the most important aspect of securing Artemis Financial’s web application. Using the SHA-256 structure, we can definitely expect one of the most secure routes to be taken when dealing with sensitive client data. To add to that, generating a certificate using the RSA-2048 algorithm to ensure secure relationships with other websites and servers. This is a crucial step in the software security process. I was able to utilize this certificate by importing it into my resources folder and linking it through the application.properties file. From there, I imported the certificate into Chrome’s “Trusted Root Certification Authorities” certificate store in an attempt to get the browser to trust the ssl but was unsuccessful. For the purposes of this project, I was unable to demonstrate this secure connection by using a self-signed certificate. In a real-world environment I would’ve most definitely purchased an official certificate from a trusted CA.

## Industry Standard Best Practices

Reviewing the code over and again after refactoring was a huge best practice for me and my process. When it comes to coding, the smallest mistake can cause the biggest issues and sometimes take forever to find and correct. OWASP Dependency Check is another big best practice. By running the vulnerability check before and after refactoring the code, I was able to check for any issues that I myself may have introduced into the application. It is a great way to get a heads up into what’s going on with the code before, during, and after changes are made.