****

# 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** | **10/18/2022** | **Nicolas Blaisdell** | **Initial Draft** |

## 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

Nicolas Blaisdell

## Algorithm Cipher

The algorithm cipher that I would recommend would be the SHA-256 algorithm, since there is a low probability of a hashing collision, in which two distinct values return the same hash value and avoiding collisions will be very important for the encryption process.

The SHA (Secure Hash Algorithm) is a hashing function which takes some input and produces an output which is unique to the input value provided, but has no resemblance to the input value, and such that the input value cannot be derived from the output hash value. 256 is the number of bits that the output hash value will return from the function. As the number of bits in the SHA algorithm increases, the possibility of a collision (in which two distinct inputs produce the same output hash value) decreases. If there is a possibility of a collision, that means that a potential hacker could provide a fake hash, impersonating our application, and possibly gain access to our system or data. Since we’ll want to ensure as much as possible that we don’t have a hashing collision, we’ll want to implement at least SHA-256 or greater.

## Certificate Generation

Insert a screenshot below of the CER file.

Text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

Graphical user interface, text, website

Description automatically generated

## Secure Communications

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

I’m able to view the page through HTTPS based on the changes made to the application.properties file. However, because the certificate is self-signed, and not authorized by one of the trusted Certificate Authorities (CA) organizations, the site is still considered “not secure”.

Graphical user interface, text, application

Description automatically generated

## Secondary Testing

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

Text

Description automatically generated

Graphical user interface, text, email

Description automatically generated

## Functional Testing

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

Text

Description automatically generated

## Summary

The refactored code now uses a “ServerController” class, which creates a REST endpoint “/hash” which handles taking the input (currently, a static value containing my name) and, using the SHA-256 algorithm, converts that input into its encrypted form. While this is currently using static data, the Artemis Financial group will be able to utilize this functionality with whatever string input they might also want to encrypt as part of their application.

The areas of security covered by this refactoring of the code were APIs, Cryptography, Client/Server and Code Quality. The API was created in a standardized way, making use of the Spring framework and the @RestController and @RestMapping annotations to generate the endpoint. Cryptography was included, since it involved taking the static input and encrypting the value using a particular algorithm cipher. Since this REST endpoint can be accessed through a web browser by a user, the Client/Server area of security must also be considered. Lastly, Code Quality must always be considered for security purposes, and the OWASP dependency-check library was used to check for any vulnerabilities in the application.

## Industry Standard Best Practices

Before making any code changes to the existing codebase, I ran the dependency check report to see what vulnerabilities might already exist, so I could compare this to a report ran after I made my changes. After doing so, and reviewing both reports, they came up the same, letting me know that the changes I made to the codebase did not introduce any new vulnerabilities into the application.

While the vulnerabilities identified are known by Artemis Financial, and they are willing to accept the risk involved by suppressing them, they should continue to check for any security flaws or new vulnerabilities by running the dependency check every few months, to make sure no new vulnerabilities are found and would make their application susceptible to attack.