****

# 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** | **4/13/2023** | **Eric Farkas** | **Added cipher recommendation & certificate** |
| **1.1** | **4/14/2023** | **Eric Farkas** | **Added test data** |
| **1.2** | **4/15/2023** | **Eric Farkas** | **Added summary** |
| **1.3** | **4/16/2023** | **Eric Farkas** | **Added best practices & final revisions** |

## 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

Eric Farkas

## Algorithm Cipher

I’m recommending Artemis Financial use a secure hash algorithm (SHA) to protect their archived data. This method protects their data against unauthorized access and viewing. The cipher is commonly used and therefore inexpensive to implement. Since the cipher will protect archived data, there is no need to use asymmetric key exchanges. An archive also makes the speed of encryption of minimal consequence. Using the SHA-256 cipher algorithm with 256-bit keys to encrypt these files will match the application and provide adequate security. SHA-256 encryption is readily available and inexpensive to deploy. This key size will make brute-force attacks difficult and time consuming while reducing the frequency of collisions. SHA-256 uses symmetrical encryption keys, increasing security by assuring that only Artemis resources can access the archived data. The SHA-256 algorithm makes use of random number generation to ensure each encrypted file is as secure as possible. Using random numbers allows for the cipher to securely create a non-reversible checksum that verifies the authenticity of the file.

## Certificate Generation

Insert a screenshot below of the CER file.

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

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



This is working but because the certificate is self-signed, the browser won’t accept it.

Graphical user interface, text, application, email

Description automatically generated

## Secondary Testing

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

Text

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

## Functional Testing

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

Text

Description automatically generatedGraphical user interface, text

Description automatically generated

## Summary

During the code refactoring, I had to add secure RESTController to the application. I could not get it to work in any other way. The ServerController class addresses the secure coding concern that arose from the vulnerability assessment diagram-based review. I’m using SHA-256 as the hashing cipher for this function. I was surprised at how little code was required to implement it. This should improve security as there is little room for mistakes and little code to attack.

Both Maven and Spring had to be updated for functionality and to successfully eliminate all vulnerabilities. Even though snakeyaml was updated to 1.33, it retriggered due to a new vulnerability. Version 2 is not part of the Spring update so I had to force the latest through the dependency section of the POM file.

I was surprised to see additions to the dependency check results during the short amount of time I worked on this project. I assumed that there were new vulnerabilities regularly, but I assumed the impact was more targeted. I guess this is a by-product of the wide number of frameworks and open-source modules available to developers. It seems this type of check should happen at a high frequency.

## Industry Standard Best Practices

From a best practices perspective, I eliminated all of the legacy frameworks and vulnerabilities. This came mainly from the practice of running the dependency check as I would not have known about the older versions and vulnerabilities otherwise. I tested the application as much as possible throughout the update process to ensure dependencies and paths did not break or change. I used best practices for coding and code review during the refactoring to ensure I did not introduce vulnerability or inefficiency.