****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[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** | **2/17/2023** | **Seth Hamrick** |  |

## Client



## Developer

Seth Hamrick

## Algorithm Cipher

Artemis Financial requires a verification step during data transfer in the form of a checksum.

Hashing algorithms are a one-way encryption that takes a key as input resulting in a unique and inimitable output. Selecting a hash algorithm for this verification step will allow the hash output to act as a fingerprint of the data to ensure integrity. The use of random numbers is also important for many encryption algorithms. Keys must be generated in an unpredictable manner and the use of random numbers greatly assists in this department. I feel that Artemis Financial has several hash algorithms that fit their current needs. However, this recommendation comes with an eye on the future, as the world of security and encryption is constantly evolving. Due to this growth, we have attempted to recommend the most secure and up to date hashing algorithm available. We recommend and have implemented the hashing algorithm known as SHA3-512. This algorithm is from the newest version of the SHA family SHA3 and was created in preparation of the inevitable fall of SHA2. Moreover, this algorithm is an approved fixed length hash function of the NIST. The “512” portion of the algorithm indicates that the output is of a fixed length 512 bits. This output length is greater than that of the other options and provides an output that is more difficult to guess, duplicate, and repeat. Hash algorithms are vulnerable to collision attacks and preimage attacks. Collision attacks may occur when two inputs result in the same hash output. Meanwhile preimage attacks are when attackers identify inputs that result in specific hash outputs. This algorithm provides a collision resistance strength of 256 bits and a 512 bit resistance strength to primage and second preimage attacks. These strengths are considerably higher than those of the comparing approved algorithms, and as such should provide Artemis Financial with the most strength for the longest duration.

## Certificate Generation

## A certificate of authority must be generated for our application. We will see this later used when addressing the application properties. The following is a walkthrough of the creation and storage of this certificate.

## Below we see a command to generate a key pair with a provided store password. The certificate is generated using the alias “Project2”. We also see that information is asked of us and the corresponding responses are provided to generate the certificate. In this case the key password was left the same was the store password.

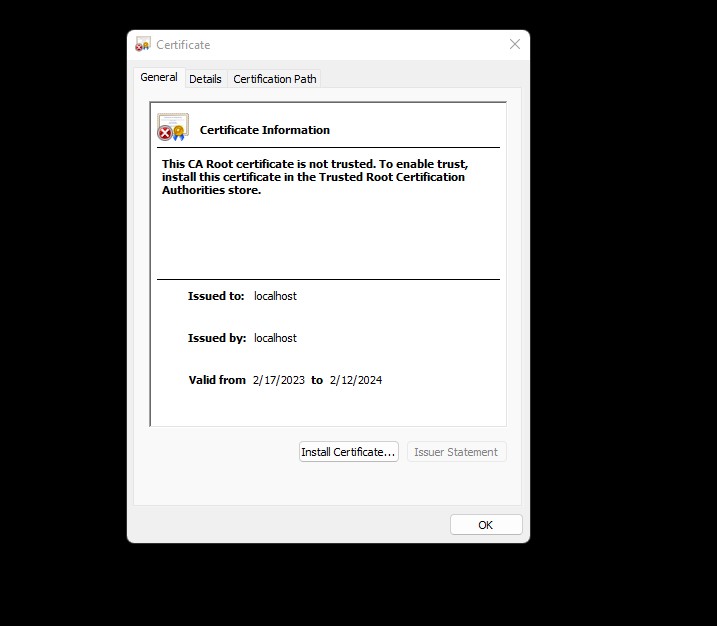
Text

Description automatically generated

Next, we use a command to export the certificate we created above to be stored in the specified file. This is followed by a command that prints the certificate so that we may verify that the certificate was correctly generated. In the final image we are able to see that the certificate is in fact stored in the desired file. This is the certificate accessed through the file directory.

Text

Description automatically generated



## Deploy Cipher

Insert a screenshot below of the checksum verification.

To illustrate the success of the checksum verification we have provided two examples below of the checksum verification in action. The first example is a simple bit of code that is used to illustrate the power of the algorithm cipher. This simple bit of code uses three inputs that each only differ by one character to generate hash values that we will see vary greatly from one another. The second example that follows is the application in action. We again use two separate inputs that differ only by one character followed by a viewing of the hash value results on the web page.

***Example 1***

Below is an image of the code used in the first example. We may notice the inputs are the following, “Seth Hamrick”, “SethHamrick”, and “Seth HamricK”.

Table

Description automatically generated with medium confidence

Next, we see the console displays the resulting three hash values. All of which differ vastly from one another and are of significant length.

Graphical user interface, text, application, email

Description automatically generated

***Example 2***

This example is similar to the previous except with three images and only two inputs. Below we see an image of the application code. The input has been altered from the original by one character. The two images that follow are the hash value results displayed on the webpage.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated with medium confidence

Thus, the two examples above display that we can generate unique hash values. Furthermore, if we were to compare the hash value results of the two examples with matching inputs we would notice that they also provide identical hash values. As a result, the two examples above display that this method may be used as a checksum for Artemis Financials data transfers.

## Secure Communications

## The following image is the application.properties file associated with establishing the secure communications of the application. We see the selected port is 8443. We may also see in the image that we have used the alias and store password of the certificate previously generated. Finally, the path of the certificate is copied so that the keystore may be accessed by the application and the key store type is provided which is JKS.

## Graphical user interface, text, application, email Description automatically generated

## With the propertied properly established we now run the application. As we see from the image below the connection is using HTTPS. This web browser views it as unsecure as it is unable to validate the certificate we have generated. This is expected as the credentials that have been provided for the certificate are fictional. Nonetheless, secure communications can and have been established using HTTPS.

## Graphical user interface, text, application, email Description automatically generated

## Secondary Testing

The first thing we will look at for this section of the report is the dependency report from the base code provided for the application. The dependency report shows 14 vulnerable dependencies with a total of 88 vulnerabilities. Table

Description automatically generated

The previous image is now followed by a dependency report of the application with the refactored code. As the report illustrates the recognized vulnerabilities are identical to the previous report. Thus, the refactored code has introduced no new vulnerabilities recognized by the dependency report.

Table

Description automatically generated

## Functional Testing

This section will first continue with some static testing as we will discuss vulnerabilities recognized from the code review. The primary concern recognized upon code review is the lack of error handling. A specific example of this is error handling for the mapping of the webpage to properly communicate with users. Otherwise the refactored code performs without error in a secure manner as all of the actions take place within the server and are simply then communicated to the user. Below we will again see an image of the code itself, followed by an image of the console when the application successfully executes without error.

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Summary

The refactored code and application provide layers of security for Artemis Financial not previously displayed. There are several areas of security which have been addressed and considered during development. These areas of security include cryptography, client/server, and code error. Cryptography regarding this project was heavily discussed in the first portion of this report. The importance of this cryptographic method being secure, however, must not be overlooked. The hashing algorithm deployed in the refactored code is one of the most up to date and secure hashing algorithms recommended by the NIST. Other algorithms may be used to generate a checksum for this application, but this algorithm is likely to serve Artemis Financial longer than some of the other options. The refactored code in which an instance is created for this encryption process displays another area of security that has been considered. The creation of an encryption instance through the method we have selected requires proper error handling for the application to compile and run. Proper error handling has been incorporated as the report displays. The final area of security the refactored code addresses is the client/server area. This is addressed by the generation of a certificate and the establishment of a secure connection using HTTPS. This helps to secure the communications to the application by providing encryption and a secure channel of communication for users. In conclusion, not all but many areas of security have been addressed in the refactored code and development took place with security at the forefront.

## Industry Standard Best Practices

Throughout the report have revolved much of our discussion around the refactoring of the code and justification as to why we have made such decisions. In the previous section we discussed how a security minded approach was used when developing the refactored code. Now we will discuss how we attempted to apply industry standard best practices throughout the development to achieve a sound and secure end result. As we have discussed the algorithm cipher at length, I believe it has been established that the selected algorithm was done so as it indeed meets industry standards. Next, the certificate we used during development was a self-signed certificate. This option was selected simply for demonstration as we highly suggest that Artemis Financial consider purchasing a certificate from a CA for third party validation. Nonetheless, we were able to use our certificate to establish secure communications using HTTPS. HTTPS is an industry standard for protecting data in transit through use of the transport layer security (TLS). Finally, we reach the actual development of the code itself. Throughout the writing of the code meticulous care was made not to introduce any logical, functional, or security errors that would introduce new vulnerabilities to the application. Coding standards were applied as conventional naming, minimal code, frequent testing, and proper readability were used. Static testing was performed throughout development including the use of an OWASP dependency -check report. The report ran both before and after the completion of refactoring and was compared to ensure the same results. Proper error handling was implemented where needed within the refactored code. Applying industry standard best practices is important towards our overall goal of meeting Artemis Financial’s needs. By applying these practices, we are able present our results with confidence. These practices provide us with a bar in which we must meet but should look to surpass. These standards also provide us with an understanding of what we truly can provide. Finally, these standards assist throughout development as a constant reminder of what we are seeking to achieve. Standard best practices were applied throughout this process helping to provide and in result in which we believe Artemis Financial may be proud of.

**References**

Marico, J. & Detlefsen, A. (2015). Iron-Clad Java: Building Secure Web Applications.

Mcgraw Hill Computing.

Kim, D. & Solomon, M.G. (2023). Fundamentals of Information Systems Security. Jones & Bartlett Learning LLC.

Oracle. (2017). Java Security Standard Algorithm Names. Retrieved on February 17, 2023, from <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names>

NIST. (2022). Hash Functions. Retrieved on February 17, 2023 from

<https://csrc.nist.gov/projects/hash-functions>