****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[CS 305 Project Two 1](#_Toc111553495)

[Document Revision History 3](#_Toc111553496)

[Client 3](#_Toc111553497)

[Developer 4](#_Toc111553498)

[1. Algorithm Cipher 4](#_Toc111553499)

[2. Certificate Generation 4](#_Toc111553500)

[3. Deploy Cipher 5](#_Toc111553501)

[4. Secure Communications 6](#_Toc111553502)

[5. Secondary Testing 6](#_Toc111553503)

[6. Functional Testing 7](#_Toc111553504)

[7. Summary 8](#_Toc111553505)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/13/2022** | **Zachary Mohler** |  |

## Client



## Developer

Zachary Mohler

## 1. Algorithm Cipher

SHA-2 is the algorithm I suggest that Artemis Financial implement in order to secure communications on their network. SHA refers to the Standard Hash Algorithm as designated by the National Institute of Standards and Technology and is therefore a great algorithm to use as it is likely to be updated if any vulnerabilities are found, such as with SHA-1.

SHA-2 uses bit manipulation to mathematically ‘jumble’ values to an unrecognizable string of characters. More specifically, it jumbles the actual bits of information from the original data into unrecognizable bits of data—we can simply represent these bits as ASCII characters for representation on screen.

SHA-2 also is a great algorithm because it has no known collisions. A ‘collision’ is a term for a resultant encrypted set of data that can be produced by multiple original data sets. Collisions can break an algorithm to the point of a complete breach of encryption. This would allow malicious users to unencrypt data that shouldn’t be accessible to them. SHA-2’s collision avoidance is another great reason to use the algorithm.

## 2. Certificate Generation

Text

Description automatically generated

Above is the certificate I generated for *Artemis Financial* using java’s keytool. The certificate generated is valid for 360 days before it needs to be re-constructed.

## 3. Deploy Cipher

Below is my implementation of SHA-256 into the server controller. As you can see it uses the MessageDigest set to “SHA-256” to process the data. Once the data is stored in a byte array, we then use a StringBuilder to return the ‘digested’ byte data to a readable ASCII string, which is then returned.

In RequestMapping/hash, we simply designate our string data and call our generateChecksum method.

Text

Description automatically generated

Below is the result of the code when executed, hosted on localhost. As you can see, the string data is unrecognizable from the original string data.



## 4. Secure Communications

Another important aspect to protecting user data is to host the network on a secure HTTPS protocol. This prevents malicious users from intercepting data. The certificate generated handles this issue and our implementation from the previous section is successfully hosted with HTTPS protocol as shown here:

Graphical user interface, text, application

Description automatically generated

## 5. Secondary Testing

Here is the code compiled and executed without error, successfully hosting to localhost port 8443:

Text

Description automatically generated with medium confidence

Furthermore, I preformed maven’s dependency check before and after implementing my checksum to ensure I didn’t introduce any new dependencies into the project. Below are screenshots of the dependency check report before execution:

Graphical user interface, text, application

Description automatically generated

And the same scan after my changes were implemented:

Graphical user interface, text, application

Description automatically generated

As you can see, my implementation introduced no new vulnerabilities into the code base, all existing vulnerabilities are related to spring, tomcat, and other networking libraries.

## 6. Functional Testing

Upon inspection of the code base, there are no explicit syntactical errors present. One comment should be made about the error handling of my implementation of the checksum. Specifically, in the generateChecksum method, for simplicity I added “throws NoSuchAlgorithmException” this should not be considered the standard and more robust error handling should be implemented in a fully functional checksum generation method.

The current code base does not have any sort of input validation or input sanitation. This is a major security risk and *must* be addressed when developing the working product. Not implementing input validation/sanitation can lead to major security leaks not only as a result of malicious actors but also users who mistakenly input bad data.

## 7. Summary

I have already talked a bit about vulnerabilities *Artemis Financial* should consider moving forward such as input validation and cryptography. In summary there is a lot of moving parts to consider when developing a secure application for users.

Input validation and input sanitation should always be used when handling untrusted data, no matter the source.

As explained above, I have implemented an example of a secure hash algorithm to provide a level of cryptographic security over data transferred on the network. This example should be built upon and implemented project wide to handle any data that is transferred across the network in order to protect the data being transferred. This ensures users’ private data will not be compromised by man in the middle attacks.

Secure API interactions is another potential vulnerability that should be addressed. As you can see in the previous sections, the maven dependency check has revealed a number of potential vulnerabilities relating to the networking structure of the program. I have already sorted through these dependencies and suppressed a number of false positives in a previous release. The remaining vulnerabilities should be reviewed to address the problems they introduce.

In order to maintain this program and uphold its current level of security, the maven dependency check should be run if any new release of dependent libraries become available. Updates to libraries often address vulnerabilities discovered in previous releases—updating the libraries can sometimes be an easy fix for security vulnerabilities.