****

# 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 5](#_Toc102040760)

[4. Secure Communications 5](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 7](#_Toc102040763)

[7. Summary 8](#_Toc102040764)

[8. Industry Standard Best Practices 9](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/31/2023** | **Raphael Coloma** | **Outlines progress made to secure the Artemis Financial web application.** |

## 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

Raphael Coloma

## Algorithm Cipher

To ensure secure communication, a checksum was added to the application. The checksum verifies that the data has not been altered during transmission. I recommend a SHA-256 algorithm cipher to produce hash values for the data files transmitted. Hashing is a one-way encryption function that generates a value of fixed size after taking in data. Since the algorithm cipher will generate the same hash value given that the input is unaltered, the checksum received should match the checksum from the server. SHA-256 was suggested because of its low probability of producing a collision. A collision is when two or more distinct bit strings generate the same hash value. There are 2256 possible hash values when using SHA-256, so the likelihood of a collision is minimal. SHA-256 is the industry standard. It is safe and trusted by businesses and organizations worldwide, as well as the US government.

## Certificate Generation

Below is a screenshot of the Certificate Generation using the Java Keytool

Text

Description automatically generated

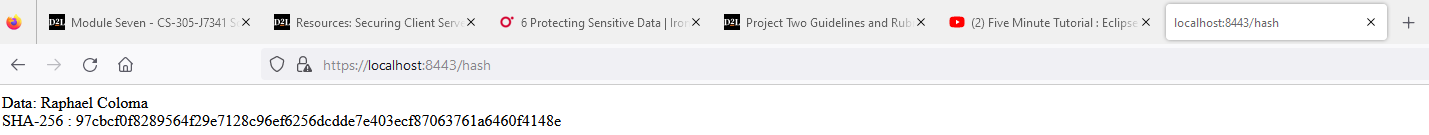
The Certificate was exported to a CER file and below is the server.cer print out.

Text

Description automatically generated

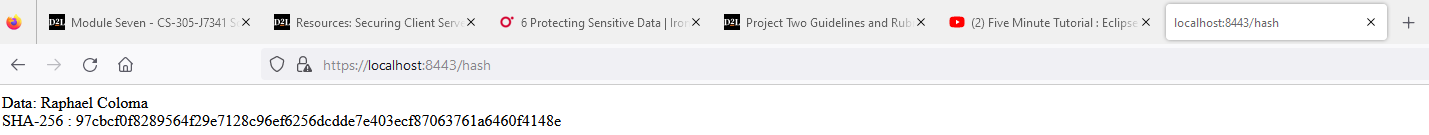
## Deploy Cipher

My name was hashed using the SHA-256 checksum in the screenshot below.



## Secure Communications

Because the certificate is self-signed, the connection is still considered unsecure as indicated by the padlock with the warning symbol icon in the URL bar.



Below is the certificate used showing that I do not own the website, but the connection is using a TLS\_AES\_128\_GCM\_SHA256 cipher suite.

Graphical user interface, text, application, email

Description automatically generated

## Secondary Testing

Below is the refactored code for the algorithm cipher.

Text

Description automatically generated

Dependency check report list the vulnerabilities below.

Graphical user interface, text

Description automatically generated

## Functional Testing

Below is the refactored code for the algorithm cipher.

Text

Description automatically generated

Updated the dependency check version to 8.0.2 in the pom.xml file.

Text

Description automatically generated with medium confidence

Corrections made to the application.properties file.

Graphical user interface, application

Description automatically generated

Console printout of Spring Boot operating without errors in the code

Text

Description automatically generated with medium confidence

## Summary

According to the vulnerability assessment process flow, I addressed the cryptography, client/server, code quality, and encapsulation categories in the Artemis Financial web application. Under cryptography, the checksum code added used a SHA-256 hash algorithm to encrypt the data being transmitted. The checksum ensures that the data was not corrupted or altered during transmission. Without this validation, users could potentially download/install malicious files onto their systems. The Client/Server category was satisfied by creating a security certificate and implementing the TLS protocol to ensure a secure connection to the web application. The security certificate validates the web application for the user to ensure that they are connected to Artemis Financial. Without a security certificate, users could be fooled into accessing bogus websites that could steal their information. The TLS protocol is responsible for encrypting the communication between the web application and the servers. TLS implementation converts the web application HTTP to the HTTPS protocol which encrypts the data transmitted to protect it despite leaks or interceptions. These implementations adhere to the Code quality and Encapsulation categories as well. Secure coding practices and patterns were utilized to prevent vulnerabilities. They were also contained within their respective functions/files to limit excessive access to code segments.

The first security layer added was the checksum. Because this was a requirement from Artemis Financial, I wanted to make sure that it was included and working. First, I created a RESTFul route using the @RequstMapping method to return the requested information to the browser. Then, I created and initialized a MessageDigest object using SHA-256 within the scope of the request. I then pass my data string to the MessageDigest object. With the passed data string, I generated a byte type hash value with the digest method. Using a bytesToHex fuction I created, I converted the hash value to hex. Once the hex was generated, I formatted it and returned it to the request.

The second security layer was creating a security certificate and implementing the TLS protocol. I created the certificate using CMD and the Java Keytool. I used the generate key command and set the key algorithm to RSA, the store type to JKS, the validity to 360 days, and the key size to 2048 bytes. I named the key “selfcert” and set the password for the key and the store to my name. Once I answered the prompts, the certificate was generated and I exported it to a CER file. With the certificate and key prepared, I added the necessary information to the application.properties file to enable the HTTPS with TSL keystore. Lastly, I set the server.port to 8443 and tested the connection.

The last security layer implemented was to the pom.xml file. I updated the Dependency Check plugin to ensure that I could do static testing. I wanted to make sure that all vulnerabilities were known and addressed. Unfortunately, there will be issues that cannot be resolved until it is patched in the next version. This is why the libraries used need to be up to date and maintained. By understanding this, I can keep the web application protected from vulnerabilities that otherwise could have been avoided.

## Industry Standard Best Practices

First and foremost, I listened to the client. I prioritized their requirements and implemented their requests. I also offered my professional suggestions and worked with them directly to help safeguard their web application. I did my due diligence to research secure coding practices and patterns and implemented them into my work. I followed industry standards with encryption types and protocols. I did frequent code reviews and testing. I ran the Dependency Check report often and validated that all available libraries were up to date. The next step would be to advise Artemis Financial to purchase the domain name to promote a secure connection for the web application.

Industry standards are what is to be expected. To maintain a positive image among customers, shareholders, and the public, it is important to have a foundation they can trust in. Vulnerabilities can lead to exploits that can jeopardize a company’s future. Attacks can come in a multitude of ways: data breaches, information leaks, Denial of Service, etc. All these scenarios can and will lead to a loss of revenue and trust. Nothing is safe online nowadays. New vulnerabilities are being discovered each day. This is why it is important to apply industry standard best practices for secure coding. Run periodic static tests and have routine maintenance on the web application libraries. Review and audit the web application code often. Stay informed on industry standards in the tech world. Stay up to date with rules and regulations with the scope of the business. Doing these things will help to keep the web applications you are working on secure and relevant.