****

# 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** | **2/19/2023** | **Michael Medina** |  |

## 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

Michael Medina

## Algorithm Cipher

The client Artemis Financial wishes to enhance the security of their existing web application by adding a form of verification of transferred data by using a checksum. Checksums are an excellent way to ensure secure communication. They are basically a string composed of random letters and numbers that are generated using a cryptographic hash function (*What Is a Checksum? | Open Science Chain*, n.d.). The checksum is then used to verify the authenticity of the data being transferred. The Algorithm Cipher that I will choose is the SHA-256. This is one of the most robust cipher algorithms available. A 256-bit key will be virtually impossible to for even the best of hackers to decrypt. An astonishing 2^256 possible hash values can possibly be generated by the SHA-256, thus making it extremely unlikely for a collision to happen (N-able, 2021). The SHA-256 is an asymmetric type because it uses two keys (a public and a private) unlike symmetric encryption that only uses one private key to encrypt and decrypt (Manico, n.d.-b). Secure cipher algorithms use a pseudorandom number generator to constantly supply a large amount of random data that can be used in encryption keys (Manico, n.d.-b). In the past cipher algorithms generating long keys would cause performance issues, but today it is possible to use very advanced cipher algorithms without compromising performance thanks to the fast computer processors that exist today.

## Certificate Generation

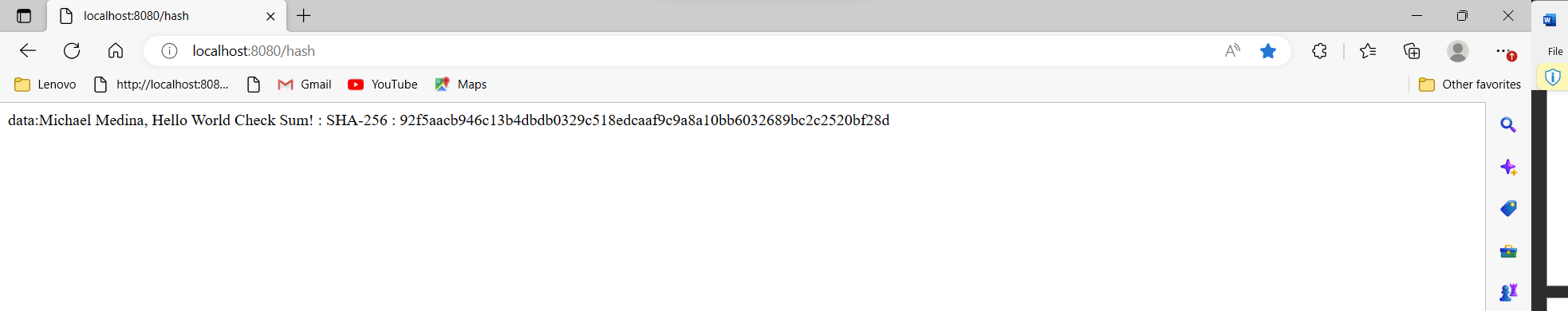
Insert a screenshot below of the CER file.

Text

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.

Graphical user interface, text, application

Description automatically generated

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.



Graphical user interface, text, application, email

Description automatically generated

## Functional Testing

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

Text

Description automatically generated

## Summary

In order to make the application secure the code base was refactored adding the ServerController class. This class contains the code to generate the message digest using the recommended SHA-256 cipher algorithm. Also, the route “/hash” was added by means of @RequestMapping. This is the route that will allow us to see the data along with its hash value in the web browser. Besides this code it was necessary to generate a certificate. The certificate was generated from the command line using Java keytool. Once generated the certificate was added to the web app by refactoring the code in the application.properties file. In this file I specified instructions such as server port, certificate directory, password, and key alias. With the code being refactored, I then ran the app and was able to view it locally on port 8443.

## Industry Standard Best Practices

I adhered to industry best standards by first selecting a robust cipher algorithm such as the SHA-256. The SHA-256 is one of the best ones available because it can generate 256-bit keys. These keys are almost impossible to decipher due to the extremely large number of possibilities. At this encryption level, files being transferred are assured to be secured and unchanged because the checksum will catch any alteration to the file. The use of the certificate made the app more secure upgrading it from http to https. Finally, I ran a dependency check which generated a report highlighting all the known vulnerabilities. This report can be analyzed to determine which dependencies could be exploited by attackers. Fixing these vulnerabilities will make the app even more secure. Applying all the latest industry best standards for secure coding will save the company from many headaches later on. Not taking care of security, leaves your app vulnerable to many attacks that may result in large financial losses, exposed sensitive data, and service interruption. The effects of cyber attacks are very detrimental to any company that has an online presence, therefore it is essential to keep up with the latest security features available.

**References**

*Java Security Standard Algorithm Names*. (n.d.). https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html

Manico, J. (n.d.-b). *Iron-Clad Java*. O’Reilly Online Learning. https://www.oreilly.com/library/view/iron-clad-java/9780071835886/ch06.html

*What is a checksum? | Open Science Chain*. (n.d.). https://www.opensciencechain.org/node/22