****

# 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** | **12/10/2022** | **Madison Lopert** |  |

## 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

Madison Lopert

## Algorithm Cipher

SHA-256 is one of the most popular hash functions. The purpose of a hash function is to take an input and create a digest, or a hash value, that represents that piece of text. Some priorities of a hash function are as follows, to be collision resistant, a small change to the input value should equate to a large change to the digest, and it should be computationally infeasible to derive the original message from the hash value. These are all traits of SHA-256. This function's output is 256 bits long meaning that the number of possible hash combinations is Since this number of possible values is so large, it would be nearly impossible for a collision to happen. This is important because if a hash function was not collision resistant, a user could force collisions and pass-through malicious files using a “valid” hash.

SHA-256 is a hash algorithm so it doesn’t use a key, but a block cipher algorithm like AES does. Block cipher algorithms are organized into two separate categories. It is symmetric and asymmetric. Symmetric algorithms use a key that works for both encryption and decryption while asymmetric uses a public key and a private key. There are several reasons such as data size, speed, security, etc. that can influence whether an organization would choose one or another. Encryption algorithms of different formats can also be utilized together to get the benefits of both processes.

The want to keep information private is not a new concept. Forms of cryptography date back all the way to 1900 BC inscriptions. Wars such as World War I and II both showcased to the United States and other nations around the world how important it is to prevent interference with sensitive information. With the rise of computer technology, cryptography became a crucial tool to develop. Starting with symmetric algorithms and then resulting in asymmetric, data could be hidden and protected from outside attackers. As it stands today, no algorithm is completely crack-proof. It is computationally infeasible to break through many, but as new technologies are developed, such as quantum computing, it is never safe to assume that they will forever be safe. That is a known issue and as threats continue to be identified, developers work to update and develop new algorithms.

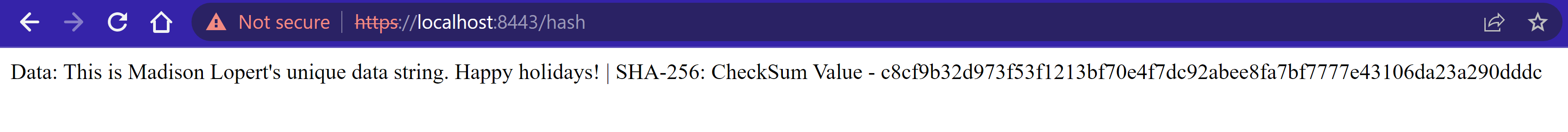
## 

## Certificate Generation

Graphical user interface, text, application, email

Description automatically generated

## Deploy Cipher



## Secure Communications

## The self-signed certificate was recognized but could not be validated in the browser. The keystore was exported into a certificate that was installed in the “Trusted Root Certificate Authorities”, yet it would not appear as valid through localhost.

Chart

Description automatically generated with low confidence

## Secondary Testing

## Dependency-Check Report:

Graphical user interface, text, application, email

Description automatically generated

**Refactored Code and Error Log:**

Text

Description automatically generated

## 

## Functional Testing

## Dependency-Check Report:

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

## Refactored Code and Error Log:

Graphical user interface, application

Description automatically generated

## Summary

## When the dependency report first ran for this program, there were 53 vulnerabilities. Looking through each I could see that many were referencing versioning issues. Besides updating the dependency check version, I also decided to upgrade the “Spring Boot Starter Parent”. I looked up its latest version, 3.0.0, and replaced the original 2.4.4 release that was being utilized. By doing this alone, the vulnerabilities went from 53 to 1. While I had no errors at this point, the program gave a warning about the execution environment being referenced in my build path. To resolve this issue, I replaced the JavaSE-1.8 JRE file that was being referenced with the jdk-19 file that I had installed in my system. By doing this I ensured that my build path is error-free and that each item can be applied successfully. I also added some comments for readability and understanding. The code I refactored most accurately aligns with code error, code quality, and cryptography. Maintaining the versions of dependencies and plugins to keep them up to date is a best practice to ensure that security patches and updates are installed and taken advantage of. By adding comments and resolving warnings, the code becomes more readable, and the risk of failure is decreased. The algorithm I used was SHA-256. Using this algorithm, I could conceal a data set by returning a 256-bit hash value that is collision-resistant. This prevents users from using malicious attacks to reveal sensitive information.

## Industry Standard Best Practices

There are multiple industry-standard best practices for secure coding. A few that I utilized in my code are cryptographic practices, error handling and logging, and system configuration. Cryptography is an important tool for any system that handles sensitive information. Encrypting this data can prevent users and attackers from accessing information that can be used maliciously. By resolving errors and vulnerabilities the system is in a better position to work as expected while maintaining a level of security. System configuration was demonstrated by updating the “Sprint Boot Starter Parent” version, dependency check version, and maven project. Patch management is a best practice to ensure that the system obtains vulnerability fixes that are already resolved during plugin updates. Each company should take advantage of industry best practices for secure coding otherwise they are putting themselves and their clients at risk. A system is a sensitive infrastructure that can fail if not maintained. Attackers are a prevalent issue in any system that carries sensitive data. If this information is leaked, an organization is likely to lose its client base's trust and they might put people in a dangerous or ill-fitting position.