**Project Two**

Southern New Hampshire University

CS 305: Software security

Jonathan Vest

12/11/2022

****

# 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** | **12/5/2022** | **Jonathan Vest** |  |

## Client



## Developer

Jonathan Vest

## Algorithm Cipher

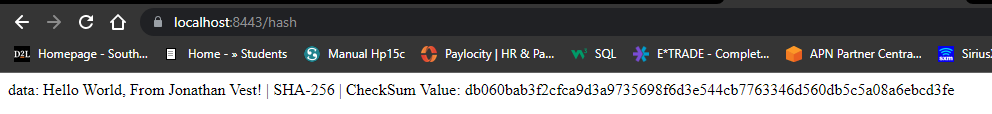
A hash function generates a nearly unique, pseudorandom value based upon the functions input. A hash function is not encryption and therefor does not need to be reversed, and is in fact not reversable. Collision occurs when two unique inputs to the hash function generate the same hashed output. To avoid collision, we will be implementing the algorithm cipher SHA-256 for this project. SHA- 256 can generate 2256 possible hash sequences consisting of upper and lowercase letters as well as the numbers 0-9. The larger the number of possible hashes the smaller the possibility of a collision. 2256 is an extremely large number and makes the possibility of a collision nearly impossible. SHA-256 hash values are computed with eight 32-bit words.

## Certificate Generation

Text

Description automatically generated

## Deploy Cipher



## Secure Communications



## Secondary Testing

Graphical user interface

Description automatically generated with medium confidence

Text, letter

Description automatically generated

## Functional Testing

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

Graphical user interface

Description automatically generated

A screenshot of a computer

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

While performing the functional testing and manual review of the code the primary vulnerability found was the hard coded unencrypted password being used within the application.properties file.

## Summary

The areas of security that were addressed in accordance with the Vulnerability Assessment Diagram are as follows:

* 1. APIs: I addressed the use of third party APIs with the implementation of HTTPS
  2. Cryptography: Cryptography was addressed with the use of algorithm syphers, hash functions and the use of check sum verification.
  3. Client/Server: Client/Server was addressed using certificates for server authentication.
  4. Code Error: Code error was addressed by the proper handling of exceptions. One example is located in the ServerController class where our methods will throw NoSuchAlgorithmExceptions.
  5. Code Quality: Code quality was address through static testing using a vulnerability and dependency check as well as through functional testing and manual code review.

We added layers of security to the application by first generating certificates for the application using the Java Keytool and by enabling the use of HTTPS. This allows for secure communication as well as providing authentication to our users so they can be sure they are using the authentic site an not an imposter. We then implemented a hashing function and ensured it was working correctly with the use of a check sum. The hash function will allow us to hash information and with the use of SHA-256 the possibility of collision is nearly impossible.

Finally we insured that we introduced no new vulnerabilities into the system when we refactored our code. By using the most recent version of the OWASP maven dependency check we were able to confirm that after our code was refactored no new vulnerabilities were introduced. Ensuring that all dependencies and vulnerabilities are addressed will help provide security to the application.

## Industry Standard Best Practices

By performing dependencies and vulnerabilities checks on our application and addressing those vulnerabilities helps to ensure we are following the industry standard and best practices. The dependency check tool looks for vulnerabilities that have been identified reported and often resolved by other developers. The tool identifies these vulnerabilities found within our application and can offer solutions to help mitigate or resolve those vulnerabilities.

We also ensured we were following industry standard and best practices through the proper handling of exceptions. By handling exceptions we mitigate the chance of a runtime error crashing our application or of a malicious actor gaining access to our system utilizing unhandled exceptions.