Module 7-1: Project Two

Anthony Lee

SNHU

CS-230-T3674 – Software Security

Professor Vivian Lyon

February 18, 2022

****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2/18/22** | **Anthony Lee** | **Creation** |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Anthony Lee

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

The customer, Artemis Financial, is requesting additional security for their web application to ensure secure communication. As the most likely attack would be an unauthorized user, encryption would be the best course of action. Without a key, the data would be useless to attackers. The most secure option would be a non-symmetric key; a non-symmetric key, as opposed to a symmetric key, means there are two keys used, one for encryption and one for decryption. The best option for this would be SHA-256 cipher algorithm with a 256-bit key to encrypt.

SHA-256 takes any size data and maps it to a 256-bit string, useless until it is decrypted. There are three properties that make SHA-256 the best choice. First, a brute force attack is almost impossible due to the number of attempts required (2^256 attempts) to reconstruct the initial data from the hash value. Second, collisions are a non-zero small chance of happening because 2^256 possible hash values. A collision is when two different pieces of data get assigned the same hash value. Third, a small change in the original data alters the hash value so much it is not apparent the two hash values are from similar data (SHA-256 algorithm: N-able, 2021). While there are bigger bit lengths, such as SHA-512, the time to do more calculations come at a cost of speed and space. It is a tradeoff between security and convenience. SHA-256 security is strong while also being a manageable speed and space.

Encryption algorithms use cryptographic hash functions to encrypt data. Hash function are mathematical functions that transform, or “map”, a given set of data into a bit string of a fixed size, also known as “hash values”. Without the key to turn the hash value back into useable data, it is useless to attackers. This practice of encrypting data started back in WWII to keep classified information from falling into enemy hands. SHA-256 algorithm is one part of the SHA-2 family created by the NSA in 2001 to succeed the SHA-1 from 1993. As more technological advances come out, longer key lengths are going to be needed to ensure security.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

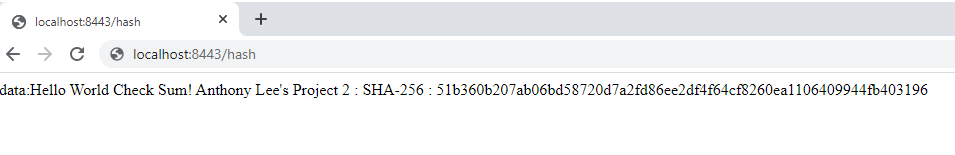
Text

Description automatically generated

## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

* 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

## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

The areas of security I addressed by refactoring the code were the APIs, cryptography, client/server, code error, and code quality. The APIs and cryptography were done by implementing a RestController with a SHA-256 encryption algorithm. The client/server were addressed using a self-signed certificate using the Java Keytool to generate the keys. To deploy and implement the encryption algorithm cipher, I used a checksum function to verify. In order to verify secure communications, I refactored the application.properties file to use the self-signed certificate Server.cer file I generated to change the HTTP to the HTTPS protocol, verified by using https://localhost:8443/hash after the server is running. I used best standard practices for the code to address the code error and code quality areas of security.

The process for adding layers of security to the software was to use the SHA-256 encryption algorithm cipher as it is very secure with a very small chance of collisions. The high security as well as the small chance of collisions allows the customer to have peace of mind when dealing with their users’ personal information.

To maintain the current security of the software applications, I suggest that a dependency check be run twice a month by running a Maven install configuration. Because there are going to be false positives thrown by the dependency check, the vulnerabilities need to be checked to see if there are any solutions for them. If there is no solution, add the suppression code to the suppression.xml file and rerun the dependency check. The reasoning behind twice a month checks is the constant threat of hackers discovering new vulnerabilities to exploit.

References

*SHA-256 algorithm: N-able*. N. (2021, April 1). Retrieved February 19, 2022, from https://www.n-able.com/blog/sha-256-encryption#:~:text=SHA%2D256%20is%20a%20patented,as%20long%20as%20when%20unencrypted.