****

# 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** | **6/14/23** | **Cory Hurlbut** |  |

## 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

Cory Hurlbut

## Algorithm Cipher

I have chosen to use SHA-256 as it is fairly standard for security at this time. SHA-256 also has strong resistance to collisions and is a hashing function. This means it makes a hash, a complex generated string 64 digits long. SHA-256 is a hashing algorithm and does not use keys to decrypt. As it is one way, it is used for protecting the real contents but also being able to use a value. For example, when inputting passwords on a website, you would not send the password to the server over the request(even for https). You would hash the password then send it, and compare that hash with the user’s hashed password that is stored in the database. If the hashes match, then it is correct. There are symmetric and non-symmetric encryptions and SHA-256 is neither as it does not decrypt. Symmetric is when you have one key to both encrypt and decrypt. Non-symmetric is when you have one key to encrypt(usually a public key) and one to decrypt(usually a private key). Encryption uses random numbers to make the encryptions as unique as possible and limit collisions(when 2 different values have the same encrypted value). This is why the encryptions are so long, as this helps make them much harder to crack by bute force and much less likely to collide. Encrypting data has been around for a long time but likely was most notably used in world war 2 with the enigma machine and the Navajo code talkers to prevent enemies from being able to steal intelligence. As the internet grew, it became much more important and widespread. It is still growing and increasing in necessity to this day.

## Certificate Generation

Insert a screenshot below of the CER file.

A picture containing text, screenshot, software, multimedia software

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

The image above can be used to show that the self-signed certificate is in effect as it is https but is not fully trusted since it is not approved by a certificate authority. Below is the certificate information.

A screenshot of a certificate

Description automatically generated with medium confidence

## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

Below are the console showing no errors, the refactored code, and the dependency check from before refactoring.

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

## Functional Testing

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

Below is the dependency check from after refactoring, with no new vulnerabilities.

A screenshot of a computer program

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with low confidence

## Summary

I mostly focused on cryptography with the certificates, server/client with the localhost and https, code quality with making sure there were no vulnerabilities in the new code, and code errors with the exception handling. The biggest increase to security was the change from http to https. This has many benefits but most notably, it encrypts the data within the requests and allows for much safer internet operations.

## Best Practices

The best practice to follow in the future to maintain the security is to keep everything updated and running regular dependency checks to see if any new vulnerabilities are found. This would lead to refactoring when necessary to ensure the program is functioning as intended.