****

# 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** | **6/18/2023** | **Hatcher Blair** |  |

## 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

Hatcher Blair

## Algorithm Cipher

For this project we are going to implement SHA-256. It can take a message of any length as input and will output a fixed size output, in this case 256 bits or 32 bytes. When the message is input, it goes through a process of padding and expansion to ensure that the message meets the format requirements for the algorithm and that no errors occur. Once the formatting requirements are met, the message is split into 512-bit blocks that are to be input into the hash algorithm. The algorithm takes the first block and uses a set of predefined hash values to hash the first block. Then it moves across all the blocks using the previous blocks hash value to generate new hashes. Finally, a 256-bit hash value is output that is unique to the input message.

Random numbers are key to encryption algorithms. Random numbers are used for key generation. All encryption algorithms require strong and unpredictable keys and random numbers are used to create these keys. You can think of random numbers as the foundation of the keys, and randomness helps prevent brute force and pattern recognition attacks. There are two types of keys that can be generated, symmetric and asymmetric. In a system that uses symmetric keys, the same key is used to encrypt and decrypt the data. Some of the benefits of symmetric keys are speed and efficiency, but the downside is that both parties need to have this key and so you need to be able to transmit this key safely or else anyone could decrypt the data being sent. In an asymmetric system there is a public key and a private key. The public key is used to encrypt the data to be sent and the private key is used to decrypt the data. These two keys are mathematically linked together and so the algorithm can encrypt the data in a way that the private key can understand. One of the downsides of asymmetric keys is that it is more computationally expensive compared to symmetric keys. Typically, both types of keys are used in conjunction with each other. An asymmetric key is used to establish a secure connection between the two parties and then once that connection is secured you can pass the symmetric key to the other party without the fear of transferring unencrypted data. Then you can use a symmetric key to encrypt and transmit the data being sent between the two parties. This is how HTTPS works and is a very common method for transmitting data securely across the internet.

Encryption algorithms have been in use for centuries and have evolved over time as our society has become more complex. Originally substitution ciphers were used such as the Caesar cipher, this is where you replace or rearrange letters to form an encrypted method. However, over time these methods became easily beaten. In the mid-20th century symmetric key ciphers were developed such as DES. Over time DES was replaced by triple DES and Blowfish because they offered longer key lengths. In the 1970’s asymmetric key algorithms were developed such as RSA which was based off the mathematical properties of prime numbers. However, asymmetric key algorithms fell off in popularity when AES emerged in the late 20th century. This is because AES was more lightweight than asymmetric keys and is widely supported. Looking into the future there is a looming threat that is quantum computing. Quantum computing could break any encryption algorithm that is currently used today in a very short time. To combat this people are starting to develop encryption methods that would be immune to this threat, but we aren’t able to test them in the real world until quantum computing becomes available.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a certificate

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated with medium confidence

## Secure Communications

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

A screenshot of a computer

Description automatically generated with medium confidence

The website is connected through https, however it still shows that the site is not secure because of the self signed certificate. As seen in the screenshot of the CER file, the CA is not trusted because it is self signed. Because of this the website displays as not secure even though it is connected over https

## Secondary Testing

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

A screen shot of a computer program

Description automatically generated with low 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.

A picture containing text, screenshot, font

Description automatically generated

## Summary

The code has been refactored to address several security concerns. First it was refactored to use a keystore to allow for communication over the SSL protocol. Second the server was configured to redirect all http requests to https to ensure that all connections were established using the more secure https method. The major areas of the vulnerability assessment process flow diagram that were addressed are cryptography and client/server. The code was refactored to use a SHA-256 encryption method to secure data that is being sent to the client from the server which addresses encryption and secure distribution from server to client. When adding layers of security to the application it is crucial that each layer is verified independently. To do this I first added the SSL configuration and ensured that it was working properly and then added the SHA-256 encryption and ensured that was working properly.

## Industry Standard Best Practices

I applied industry standard best practices by ensuring that I was using modern standards first and foremost. It is important to make sure that you are using a modern standard that is in its main release and not a beta or developer release to ensure that there are as few vulnerabilities as possible. I also made sure that all the known vulnerabilities with the dependencies used were not affecting the code that was written. There weren’t any vulnerabilities that affected my code and so there was no need to do additional refactoring to prevent attacks from those vulnerabilities. It is important to always use industry standard best practices so that there are as few methods for an attack as possible. It also allows the company to easily address new vulnerabilities that are found in the future.