****

# 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** | **6/18/2024** | **Alexander Feeney** |  |

## 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

Alexander Feeney

## Algorithm Cipher

The chosen algorithm cipher I have implemented into the system is the Secure Hash Algorithm. This algorithm is an enhanced and standardized method used to generate a unique hash value from input data. This unique value that is generated verifies data integrity. This algorithm was developed by the National Security Agency and is a standardized function used by the National Institute of Standards and Technology. The “256” value represents the bits in the context of our implementation representing the length of the key generated and gives us the option to specify the length to prevent duplicate values. The process of the hash function involves the usage of certain arithmetic and logic to compute the data to give us this unique value. The way the function operates in this deterministic manner reinforces its resistance to many different cyber related attacks. Encryption algorithms have evolved since primitive techniques. This continues as we consistently achieve further milestones in data integrity and security by continuously improving these techniques.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer program

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A black background with white numbers

Description automatically generated

## Secure Communications

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

A screenshot of a certificate

Description automatically generated

## Secondary Testing

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

Below is the initial dependency check on the codebase after the initial successful launch of the program along with its primary functionality’s basic logic:  
  
A screenshot of a computer program

Description automatically generated

After manually reviewing our code, and identifying all vulnerabilities as well as false positives, we implemented code to edit our report, suppressing all false positives and addressing vulnerabilities:  
  
A screenshot of a computer program

Description automatically generated

And finally, the report generated our suppressed vulnerabilities:

A screenshot of a computer

Description automatically generated

## Functional Testing

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

A screen shot of a computer

Description automatically generated

## Summary

While refactoring our codebase and design, we addressed multiple security vulnerabilities in the system. After the initial assessment we concluded there were multiple features that needed to be implemented to meet the requirements of this application. This included thew implementation of a hash function to promote data integrity, generating a sum value from input data. We then generated a self-signed certificate and implemented secure communications in which the system was lacking. After the initial configuration of the server, we moved onto implementation of the logic, taking an input and converting it to a unique value. The code was subjected to a dependency test, ensuring all our addons and plugins were functioning correctly without any vulnerabilities. After manual review of these errors, we then suppressed the false positives, resolving any errors that arise with the dependency check. Overall, we took multiple steps like implementing a secure hash function and securing our communications to make sure our system was safe and secure.

## Industry Standard Best Practices

Following best practices was critical when refactoring and designing the new codebase. When we were in our assessment and planning phase, analyzing the system helped us find areas that need improvement. We were manually able to find flaws in the system, giving us a preemptive edge by finding the vulnerability early. We utilized an industry standard hash function, the SHA-256. By utilizing this industry standard function, we can stay in compliance with regulations as well as maintain data security. By ensuring the security of our communication protocols, we can encrypt data between the client and server. Static dependency testing of our codebase further contributed to the identification of system vulnerabilities, further enhancing our security and code quality. In conclusion, following best practices fortifies our applications security by following a structured approach designed to find and mitigate vulnerabilities within the application.