****

# 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** | **10/20/2024** | **Mark Turner** |  |

## 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

Mark Turner

## Algorithm Cipher

I recommend using the Secure Hashing Algorithm (SHA) 256 for implementing a checksum verification program. SHA-256 is a widely used hashing algorithm that generates a unique 256-bit hash for any input size. It divides input data into 512-bit blocks, splits them into 16 words of 32 bits, and expands these into 64 words through mathematical operations. The algorithm processes each block over 64 rounds, combining current hash values with expanded words and constants. This ensures a unique and collision-resistant output, making SHA-256 suitable for digital signatures, integrity checks, and secure communications.

Random numbers are essential in cryptography for key generation and unpredictability. Symmetric algorithms use the same key for both encryption and decryption, while asymmetric algorithms use separate public and private key pairs, allowing secure communication without sharing the private key. Encryption has evolved far from the simple methods used during the Roman empire, like the Caesar cipher, to hashing algorithms like SHA (Secure Hash Algorithm), which are crucial for data integrity and authentication.

## Certificate Generation

Insert a screenshot below of the CER file.

A computer screen with text on it

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.

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

A screenshot of a computer

Description automatically generated

## Functional Testing

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

A screenshot of a computer screen

Description automatically generated

## Summary

**Areas of security:**

- **Cryptography**: SHA-256 cryptographic hashing to ensure data integrity and obfuscate sensitive data.

**- Exception handling**: Secure error handling to prevent sensitive information exposure through error messages.

- **Input validation**: Validating input data to guard against injection attacks.

- **API**: Ensure HTTPS is used to prevent unauthorized access to APIs.

To add layers of security, I begin with a risk assessment to identify potential threats. Secure algorithms like SHA-256 are implemented, along with enhanced exception handling and input validation mechanisms. Regular code reviews help identify vulnerabilities, and continuous monitoring ensures that dependencies are updated to address any newly discovered issues. By addressing these areas systematically, the application develops a more robust defense against potential threats.

## Industry Standard Best Practices

Implementing a checksum-verifier with a SHA-256 hashing controller helps securely obfuscate sensitive data from unintended recipients, lowering the risk of breaches. Proper exception handling and response codes help prevent sensitive info from showing up in error messages, further reducing the risk of an attack. I’ve also implemented input validation to guard against injection attacks. Additionally, encoding output in hexadecimal helps block cross-site scripting vulnerabilities.

To add layers of security, I start with a risk assessment to identify potential threats. I implement secure algorithms like SHA-256, along with enhanced exception handling and input validation mechanisms. Analyzing dependency reports and assessing package vulnerabilities helps reduce the risk of our applications being subject to exploits prevalent in 3rd part software. By addressing these areas, I can build our defense against potential threats.