****

# 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** | **2/23/2025** | **Scott Cain** |  |

## 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

Scott Cain

## Algorithm Cipher

1. The Advanced Encryption Standard (AES) would be a great choice for this project. It is a symmetric encryption algorithm used worldwide. It is recognized for its security and efficiency. It uses the same key for both encryption and decryption. It was established by the U.S. National Institute of Standards and Technology (NIST) in 2001.
2. AES uses key sizes of 128, 192, and 256 bits, each providing a different level of security. High security will would use 256 bits.
3. AES is a symmetric key algorithm, meaning the same key is used for both encryption and decryption. This key is generated using a secure random number generator to ensure it is unique and very hard to guess. Asymmetric key algorithms use a pair of keys: a public key for encryption and a private key for decryption. The public key can be shared openly but the private key will be confidential. Asymmetric keys will use random numbers for generating key pairs and allowing randomness of the encryption process. Using random numbers in both symmetric and asymmetric encryption is needed to keep the security of encrypted data.
4. Encryption algorithms have been used since ancient times to keep messages and information safe. They varied from Caesar Cipher to Enigma machine and now more modern methods of using AES and public-key cryptography. The future could hold advancements in quantum computing cryptography.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

AI-generated content may be incorrect.

## Secure Communications

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

A screenshot of a computer

AI-generated content may be incorrect.

## Secondary Testing

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

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

## A screenshot of a computer AI-generated content may be incorrect.

## Summary

I refactored the code to enhance readability, maintainability, and error handling. The changes included adding dedicated methods to handle exceptions and format the response string, making the code more modular and easier to understand.

For security, I improved error handling to manage exceptions and log errors, preventing potential security issues. Clean, modular code also simplifies static and dynamic analysis, making it easier to identify vulnerabilities.

The refactored code improves static analysis, dynamic analysis, and manual code reviews by making it more readable and improved structure. This makes it easier to spot and address security vulnerabilities.

Improved error handling and modular code structure allow better security of the application. This approach increases reliability, builds customer trust, and ensures compliance with industry regulations.

To maintain security, follow these best practices:

* Conduct regular code reviews.
* Implement automated testing tools.
* Provide ongoing security training for developers.
* Regularly update dependencies using tools like OWASP Dependency-Check.

## Industry Standard Best Practices

**Adopting a Secure Software Development Framework (SSDF):**

Follow the NIST's Secure Software Development Framework (SSDF). This framework includes practices for preparing the organization, protecting the software, producing well-secured software, and responding to vulnerabilities. It has security integrated into every phase of the software development lifecycle.

**Implementing Secure Coding Practices:**

The OWASP Secure Coding Practices Quick Reference Guide helps integrate a checklist of security coding practices into our development process. This proactive approach mitigates security risks.

**Conducting Regular Security Training:**

Ongoing security training for developers is important. It keeps the team updated on the latest threats and secure coding techniques as well as improving our overall security posture.

**Using Up-to-Date Frameworks and Libraries:**

Regularly updating all frameworks and libraries to their latest versions is essential to mitigate known vulnerabilities. Staying current with updates ensures that we are protected against the latest security threats.

**Performing Code Reviews and Static Analysis:**

Regular code reviews and the use of static analysis tools help identify and fix security vulnerabilities early in the development process. This practice ensures that code is secure and free from common vulnerabilities.

**Implementing Automated Testing:**

Automated testing tools continuously test security vulnerabilities throughout the development lifecycle. This approach helps detect and address security issues promptly, maintaining the integrity of my software.

**Securing Access to Databases:**

Proper authentication, authorization, and encryption are essential to secure access to databases. Ensuring that these measures are in place protects sensitive data and prevents unauthorized access.

**Managing Dependencies:**

Regularly updating and managing dependencies using tools like OWASP Dependency-Check helps identify and address vulnerabilities in third-party libraries. This proactive approach minimizes security risks from external sources.

**Defining Security Requirements:**

Clearly defining security requirements at the beginning of the project ensures that security is considered throughout the development process. This practice helps build secure software from the start.

**Responding to Vulnerabilities:**

Establishing a process for identifying, reporting, and responding to vulnerabilities in a timely manner is crucial. Prompt response to vulnerabilities helps address security issues before they can be exploited.

**Maintaining Existing Security:**

By following these best practices, I can make my software application secure against known vulnerabilities. Regular updates and ongoing security training help maintain a high level of security awareness and preparedness.

**Value to the Company:**

Applying industry-standard best practices for secure coding enhances our company's overall security posture. It builds customer trust, ensures compliance with industry regulations, and reduces the risk of security breaches. This proactive approach to security can save the company from potential financial and reputational damage**.**