****

# 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** | **6/30/2024** | **Madison Fleitas** |  |

## 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

Madison Fleitas

## Algorithm Cipher

AES (Advanced Encryption Standard) 256-bit encryption is widely regarded as the strongest and most secure encryption standard commercially available today. Although AES 256-bit encryption theoretically offers higher security than AES 128-bit encryption, AES 128-bit encryption has never been compromised.

Hash functions are essential in various information security applications, generating fixed-length outputs known as hash values or message digests from inputs of arbitrary length. These functions compress numerical input into consistent, fixed-length outputs, making them valuable for ensuring data integrity and security.

Secure hash keys require reliable random number generators, which produce random bits of data from minor system deviations. This randomness is crucial for forming unique hash keys and preventing reuse. In cryptography, symmetric keys are used for both encryption and decryption, requiring both the client and server to share the same private key, necessitating a secure initial exchange. Asymmetric encryption, in contrast, uses separate keys for encryption (public key) and decryption (private key), eliminating the need for both parties to share private information.

Historically, ciphers have been used for secure communication, especially in military contexts. Early ciphers were simple, often using letter pairings, which were easily deciphered due to predictable linguistic patterns. To enhance security, cryptographic shifting methods were developed, encoding messages with complex matrices to obscure these patterns. Today, RSA (Rivest–Shamir–Adleman) is a widely used encryption standard. Modern practices like two-factor authentication (2FA) combine multiple methods to enhance the security of private information exchange.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer program

Description automatically generated

A black background with white text

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

## Summary

The code has been thoroughly refactored to comply with security testing protocols, addressing critical areas highlighted in the Vulnerability Assessment Process Flow Diagram. Specifically, the review identified the need to focus on APIs, cryptography, code quality, and input validation. Refactoring efforts included enhancing API security to prevent unauthorized access, improving cryptographic practices to ensure data protection, and increasing overall code quality to reduce vulnerabilities. Additionally, robust input validation mechanisms were implemented to prevent injection attacks and other common exploits.

To add layers of security to the software application, a multi-faceted approach was employed. This involved integrating secure coding practices, conducting regular security audits, and employing advanced encryption techniques. By doing so, the application’s resilience against potential threats was significantly strengthened, adding immense value to the company’s overall wellbeing. This approach not only protects sensitive data but also boosts customer confidence and trust in the company’s software solutions.

Maintaining the current security of the software application requires adherence to best practices. These include continuously monitoring for new vulnerabilities, applying timely patches and updates, and conducting periodic security training for developers. Regularly updating security protocols and staying informed about the latest threats are essential practices to ensure the ongoing protection of the software application.

## Industry Standard Best Practices

Industry best practices in software security encompass a range of guidelines and methodologies designed to ensure robust protection against threats and vulnerabilities. Secure coding practices, such as adhering to OWASP Secure Coding Guidelines, regular code reviews, and static code analysis, are fundamental to eliminating potential security flaws. Regular security audits and testing, including penetration testing and automated security tools, help identify and mitigate vulnerabilities. Data encryption using strong standards like AES-256 for data at rest and in transit, along with proper key management, ensures the protection of sensitive information.

Authentication and access control measures, including multi-factor authentication (MFA) and the principle of least privilege, are crucial for restricting access to sensitive data. Input validation and sanitization prevent injection attacks by validating and sanitizing all user inputs, using parameterized queries, and implementing strong validation rules. Secure API practices involve authenticating and authorizing API requests, implementing rate limiting and throttling, and using HTTPS to secure communication.

Patch management and updates are essential for maintaining security by regularly applying patches, monitoring for new vulnerabilities, and automating the patch management process. Incident response planning, with a clear and regularly tested plan, ensures preparedness for security breaches. Security training and awareness programs for developers, IT staff, and end-users help in identifying and preventing social engineering attacks, fostering a culture of security within the organization.

Comprehensive monitoring and logging, using intrusion detection systems (IDS) and security information and event management (SIEM) tools, enable the detection and response to security incidents. Adhering to compliance and regulatory requirements, such as GDPR, HIPAA, and PCI-DSS, and conducting regular audits ensures that the organization meets industry-specific standards. Following these best practices helps organizations maintain a robust security posture, protect sensitive data, and mitigate the risk of security breaches.