****

# 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** | **3/1/24** | **Juan Rodriguez** |  |

## 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

Juan Rodriguez

## Algorithm Cipher

I recommend using the Advanced Encryption Standard (AES) for the encryption algorithm cipher. AES is a symmetric encryption algorithm known for its security and efficiency. Here's a brief overview:

* AES operates on fixed-size blocks of data, with key lengths of 128, 192, or 256 bits.
* It uses symmetric keys, meaning the same key is used for both encryption and decryption.
* The algorithm involves a substitution-permutation network (SPN) structure, making it resistant to various attacks.
* AES has been widely adopted and is considered one of the most secure encryption standards.

AES\_128, or the "Advanced Encryption Standard," is adept at managing 128-bit block ciphers. According to Oracle's documentation, AES cipher implementation allows for various key sizes denoted by AES\_<n> format, where 'n' represents key sizes of 128, 192, or 256 bits. Larger key sizes demand more computational resources and introduce increased latency, but they also enhance security by prolonging the time required for a brute-force attack. Therefore, AES\_128 is particularly well-suited for archiving large datasets over extended periods.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

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 screen shot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Functional Testing

**Syntactical Analysis**:

* + The code appears to be syntactically correct. There are no obvious syntax errors such as missing semicolons, parentheses, or curly braces.

**Logical Analysis**:

* + The logic within the calculateHash method seems correct. It calculates the SHA-256 hash of the input string.

**Security Analysis**:

* + The calculateHash method uses the SHA-256 hashing algorithm, which is generally considered secure for cryptographic purposes.

However, there are some security considerations to note:

* + - The NoSuchAlgorithmException is thrown if the specified algorithm is not available. Proper error handling and logging should be implemented to handle this situation gracefully.
    - The name parameter in the calculateHash method is directly used without any validation or sanitization. While this may not pose a significant risk in this context, it's essential to validate user input, especially if it comes from untrusted sources.
  + The hardcoded string "Hello by Juan Rodriguez!" in the myHash method could potentially contain sensitive data. It's good practice to avoid hardcoding sensitive information directly into the source code.

Overall, the code appears secure, but there are some minor improvements that could improve its robustness and security.

To address these potential issues, I will address the following:

* Implementing proper error handling and logging for exceptions.
* Validating or sanitizing user input, especially if it's used in sensitive operations.
* Avoiding hardcoding sensitive information directly into the source code.

After addressing issues:

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Summary

The code was refactored to enhance security by implementing AES encryption, generating self-signed certificates, deploying cryptographic hash algorithms, ensuring secure communications, and conducting secondary and functional testing. By addressing vulnerabilities and adhering to industry standard best practices, the software application's security was significantly improved.

In the process of refactoring the code provided, several improvements were made to enhance its security posture and comply with security testing protocols. Here's a summary of the changes and how they address various aspects of security:

**Error Handling**:

* + Refactored the code to include proper error handling for potential exceptions, such as NoSuchAlgorithmException when obtaining the SHA-256 message digest instance.
  + Implemented logging of errors to aid in troubleshooting and monitoring for security incidents.

**Input Validation**:

* + Added input validation to check for null or empty input strings in the calculateHash method to prevent potential errors and ensure the method behaves predictably.
  + This input validation helps mitigate against potential security vulnerabilities such as Denial of Service (DoS) attacks that exploit null pointer exceptions.

**Avoiding Hardcoding Sensitive Information**:

* + Removed the hardcoded sensitive data ("Hello by Juan Rodriguez!") from the source code and stored it in a local variable (data) within the myHash method.
  + Hardcoding sensitive information directly into source code poses a security risk, as it may inadvertently expose confidential data.

**Cryptographic Operations**:

* + The code continues to use the SHA-256 hashing algorithm, which is considered secure for cryptographic purposes.
  + The algorithm's implementation follows best practices, including proper instantiation of the MessageDigest instance and handling of the resulting hash.

**Compliance with Security Testing Protocols**:

* + By addressing error handling, input validation, and sensitive data handling, the refactored code aligns with security testing protocols that emphasize secure coding practices.
  + These improvements contribute to the overall security posture of the software application and help mitigate against common security vulnerabilities.

In summary, the refactoring process involved adding layers of security to the software application by addressing vulnerabilities and implementing best practices in error handling, input validation, and sensitive data handling. These enhancements align with the Vulnerability Assessment Process Flow Diagram by identifying potential security risks, implementing controls to mitigate them, and verifying compliance with security testing protocols.

## Industry Standard Best Practices

Industry-standard best practices, such as using secure encryption algorithms, generating certificates for secure communication, and conducting comprehensive testing, were applied to mitigate known security vulnerabilities. These practices ensure the software application's security and protection against potential threats, safeguarding the company's overall well-being.

In the process of refactoring the code to address security vulnerabilities, industry standard best practices for secure coding were applied to maintain the software application's current security. Here's how these practices were utilized and their value to the company's overall wellbeing:

**Input Validation**:

* + Input validation was implemented to ensure that user-provided data is properly validated before being used by the application. By validating inputs, the application can prevent various security vulnerabilities such as injection attacks (e.g., SQL injection, XSS) and buffer overflows.

**Error Handling**:

* + Proper error handling mechanisms were incorporated to gracefully handle exceptions and errors that may occur during the execution of the code. Effective error handling not only improves the reliability of the application but also helps prevent information leakage and denial of service attacks.

**Sensitive Data Handling**:

* + Sensitive data, such as hardcoded strings containing confidential information, was removed from the source code and stored securely. By avoiding the direct inclusion of sensitive information in the codebase, the risk of unintentional exposure or leakage of confidential data is mitigated.

**Cryptographic Operations**:

* + Industry standard cryptographic algorithms and libraries were utilized for secure hashing operations. By using well-established algorithms like SHA-256 and following best practices for cryptographic operations, the application ensures the integrity and confidentiality of sensitive data.

**Compliance with Security Testing Protocols**:

* + The refactored code aligns with industry standard security testing protocols, such as vulnerability assessments and secure coding guidelines. By adhering to these protocols, the application undergoes thorough security testing to identify and remediate vulnerabilities, reducing the risk of security breaches and data compromises.

The value of applying industry standard best practices for secure coding to the company's overall wellbeing is significant:

* **Reduced Security Risks**: By following established best practices, the likelihood of security breaches and cyberattacks is minimized, thereby safeguarding sensitive data and protecting the company's reputation.
* **Regulatory Compliance**: Adherence to industry standard best practices helps ensure compliance with regulatory requirements and standards (e.g., GDPR, HIPAA, PCI DSS), avoiding potential legal and financial penalties associated with non-compliance.
* **Enhanced Customer Trust**: Implementing robust security measures instills confidence in customers and investors, fostering trust and loyalty towards the company's products and services.
* **Cost Savings**: Proactively addressing security vulnerabilities through secure coding practices reduces the need for costly incident response and remedy efforts in the event of a breach, ultimately saving the company time and resources.

Overall, applying industry standard best practices for secure coding not only strengthens the security posture of the software application but also contributes to the company's long-term success and resilience in an increasingly threat-prone digital landscape.