****

# 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** | **[Date]** | **Jose Medina** |  |

## 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

Jose Medina

## Algorithm Cipher

1. **Overview of the Cipher:**

I am using AES (Advanced Encryption Standard), which is a symmetric encryption algorithm widely adopted for securing data. It supports key sizes of 128, 192, and 256 bits, with 128-bit keys being common for many applications.

AES operates on blocks of 128 bits of data, applying multiple rounds of substitution, transposition, and mixing to produce the ciphertext.

2. **Hash Functions and Bit Levels:**

AES does not rely on hash functions; instead, it uses a key and processes data in blocks. For this project, AES with a 128-bit key means that the encryption key consists of 128 bits, providing a balance of security and performance.

3. **Symmetric vs. Non-Symmetric Keys:**

AES is a symmetric key algorithm, meaning that the same key is used for both encryption and decryption. This differs from asymmetric encryption (e.g., RSA), which uses a public/private key pair where one key encrypts and the other decrypts.

4. **History and Current State:**

AES was established as the encryption standard by the U.S. government in 2001. It replaced the older DES algorithm due to security concerns. Today, AES is widely used across different industries, including finance and healthcare, for securing sensitive data.

## Certificate GenerationA screenshot of a computer screen Description automatically generated A screenshot of a computer Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screen shot of a computer code

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

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 generated

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

As part of the refactoring process, several security vulnerabilities were addressed, particularly concerning outdated dependencies. The **OWASP Dependency Check** tool was utilized to identify and mitigate these vulnerabilities, as displayed in the report. A total of **17 vulnerabilities** were detected across several dependencies, including critical components such as **Jackson**, **Spring Boot**, **Logback**, and **SnakeYAML**.

To address these vulnerabilities:

1. **Jackson Libraries**:
   * **Jackson-core** and **Jackson-databind** were updated to **version 2.15.2**, which addressed vulnerabilities related to **data binding deserialization** attacks. These versions mitigate security flaws, such as arbitrary code execution through untrusted input data.
2. **Spring Framework**:
   * **Spring Boot** and its associated libraries were updated from older versions (e.g., **Spring Boot 2.2.4**) to **Spring Boot 3.1.5**, which contains numerous security patches and optimizations, improving the resilience of the web framework against attacks like **cross-site scripting (XSS)** and **request forgery**.
3. **Logback**:
   * **Logback-core** and **Logback-classic** were updated to **version 1.4.11**. These updates resolved vulnerabilities in logging, which could be exploited for logging injection attacks, where attackers craft log messages to execute unintended actions or gain unauthorized access.
4. **SnakeYAML**:
   * The **SnakeYAML** library was updated to **version 2.0** to fix vulnerabilities related to YAML parsing, which could allow attackers to exploit deserialization vulnerabilities when processing untrusted YAML inputs.
5. **OWASP Dependency Check Plugin**:
   * The **OWASP Dependency Check plugin** was also updated to **version 7.4.4**, ensuring that all future scans are executed with the latest vulnerability detection mechanisms. This update enhances the tool’s ability to detect the latest CVEs (Common Vulnerabilities and Exposures), reducing the risk of unpatched libraries being deployed.

By updating these dependencies and addressing vulnerabilities as identified in the **OWASP Dependency Check** report, the application now follows more secure practices in dependency management. The vulnerabilities related to deserialization, logging, and YAML parsing have been mitigated, ensuring that the system is more resilient to potential exploits.

## Industry Standard Best Practices

Throughout the refactoring process, we followed key industry standards for secure software development. Below are the best practices that were implemented:

1. **Encryption Standards**: The project employs **AES encryption** for data protection, adhering to the standards outlined by the **National Institute of Standards and Technology (NIST)**. AES is a widely recognized and trusted encryption algorithm used for safeguarding sensitive data.
2. **Secure Communication (SSL/TLS)**: The implementation of **HTTPS** using **SSL/TLS** ensures that data transmission between clients and the server is encrypted, protecting against **man-in-the-middle** attacks and eavesdropping. This is a standard practice for securing communication over the internet, as outlined by organizations like **OWASP** and **NIST**.
3. **Dependency Management**:
   * The **OWASP Dependency Check** tool was used to scan for known vulnerabilities in third-party libraries, addressing security issues related to outdated or vulnerable dependencies. This aligns with OWASP's recommendations for maintaining secure and up-to-date dependencies in software development.
   * The update of critical libraries such as **Jackson**, **Spring Boot**, and **Logback** to their latest secure versions ensures that the project adheres to best practices for maintaining dependency security. Regularly updating dependencies is a key industry recommendation to prevent known exploits from being leveraged in production systems.
4. **Code Quality**: We maintained **code encapsulation** and followed secure coding practices throughout the refactoring process. The updated versions of libraries not only improved security but also maintained the application's performance and functionality. These changes align with the **Code Quality** principles outlined in the **Vulnerability Assessment Process Flow Diagram**.

By incorporating these industry standards, the project ensures compliance with best practices for secure software development. This includes using robust encryption methods, securing communications with SSL/TLS, and actively managing dependencies to prevent known vulnerabilities from impacting the application's security posture.