****

# 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** | **6/29/2025** | **Jake DeMuesy** |  |

## 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

Jake DeMuesy

## Algorithm Cipher

I recommend using the Advanced Encryption Standard (AES) with 256-bit keys in Galois/Counter Mode (GCM) for the SSL server. AES-256-GCM offers a strong symmetric encryption that combines high-level security with efficient performance. It provides authenticated encryption, which not only ensures data confidentiality but also protects against tampering by verifying integrity. As widely adopted, NIST-approved standard, it’s trusted across the industry. In the Spring Boot SSL server, AES-256-GCM would be responsible for the symmetric encryption phase of TLS communications following the initial key exchange.

The main hash function used here is SHA-256, which outputs 256 bits and offers 128-bit strength against collision attacks. It’s commonly used for things like digital signatures, checking certificates, and HMACs. If you want stronger security, there’s also SHA-3, which has the same 256-bit output but works differently under the hood. SHA-3 is better at avoiding some specific attacks (ex. length-extension). This matches current NIST guidelines and should keep data protected well into the next decade or so.

When it comes to keys, generating random numbers securely is critical. That’s why cryptographically secure random number generators (like Java’s “SecureRandom”) are used to create things like initialization vectors and keys, pulling from hardware and system randomness. There are two main types of keys: symmetric and asymmetric. Symmetric keys, like AES-256, use the same key to lock and unlock data and are really fast, which is great for encrypting large amounts of info, but sharing the key safely can be tricky. Asymmetric keys, such as RSA with at least 3072 bits or elliptic curve options like ECDSA with P-384, use key pairs—one public and one private—and are slower, but they’re great for things like securely exchanging keys and signing data. Usually, protocols like SSL/TLS use both: asymmetric encryption to set up a secure connection and exchange symmetric keys, then symmetric encryption to handle the actual data.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer program

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A black screen with white text

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.

A black screen with white text

AI-generated content may be incorrect.

## Secondary Testing

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

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Functional Testing

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

A screen shot of a computer program

AI-generated content may be incorrect.A computer screen shot of text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

## Summary

The server application successfully transitioned from an insecure HTTP-based system to a secure HTTPS implementation, addressing multiple critical vulnerabilities. This upgrade addressed several key security vulnerabilities by integrating TLS 1.2 and 1.3, using modern cipher suites and adding strong certificate management. I also added multiple cryptographic hash algorithms (SHA-256, SHA-384, SHA-512) to ensure data integrity and authenticity. Additional improvements include better input validation, secure error handling, and reduced information disclosure, all of which contribute to a more secure and reliable application.

To make the application more secure, I implemented a layered security strategy. This included encrypting data during transmission, supporting multiple cryptographic hash algorithms, and following secure coding practices at the application level. I validated and sanitized all user inputs, used UTF-8 encoding to avoid inconsistencies, and added secure random number generation with 256-bit salts. On the configuration side, I disabled outdated protocols and only allowed strong cipher suites. I also set up logging and monitoring tools to catch issues early and help maintain accountability.

During development, I used both static and dynamic analysis tools to test the system against industry standards (like OWASP, NIST, and ISO 27001). The final version supports the core principles of cybersecurity - confidentiality, integrity, and availability - by encrypting all traffic, detecting tampering, and handling failures smoothly. This project gave me valuable security experience and helped me bridge academic concepts with real-world applications!

## Industry Standard Best Practices

In The refactoring of the SSL server application, I believe I have successfully integrated a wide range of industry-standard secure coding practices, guided by OWASP, NIST, SANS, and ISO 27001 frameworks. These practices address critical security areas including input validation, cryptographic implementation, access control, and secure configuration management. For instance, the project utilized OWASP's input validation and cryptographic cheat sheets to ensure user data is handled safely and securely.

The practical application of these standards has significant organizational benefits. By reducing the risk of security breaches and ensuring compliance with regulations like GDPR, the organization can avoid heavy penalties (and vulnerabilities) and maintain eligibility for contracts in regulated markets. I do believe it’s also important to note that these measures also decrease cybersecurity insurance premiums by demonstrating due diligence and proactive risk management, which I believe is not talked about much. From a financial perspective, secure coding lowers the total cost of ownership by minimizing technical debt and reducing the need for reactive / last minute fixes, ultimately preserving revenue.

In addition to meeting compliance requirements and reducing costs, these security measures directly enhance system performance and reliability. Secure error handling helps maintain consistent uptime, while updated configurations allow the system to scale efficiently and respond quickly under load. As a result, the organization can deliver a more dependable experience, building customer trust and reinforcing its reputation as a security forward company. Implementing secure coding practices also helps the organization stay ahead of future challenges. With built-in support for cryptographic flexibility and algorithm upgrades, the system is well-positioned for upcoming transitions the future holds.