****

# 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** | **08/17/2025** | **Sean Born** |  |

## 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

Sean Born

## Algorithm Cipher

I recommend using the SHA-256 cryptographic hash for checksum verification. SHA-256 is currently considered secure against practical collision attacks (NIST, 2015). A collision occurs when two different inputs produce the same hash value. If an attacker could create a malicious file with the same checksum as a legitimate file, they could trick recipients into accepting compromised data (Stallings, 2017).  
Older algorithms like MD5 or SHA-1 are vulnerable to such attacks, making them unsuitable for security-sensitive applications (Oracle, n.d.). SHA-256 uses a 256-bit digest, making it systematically infeasible for an attacker to generate collisions or preimages with current technology (NIST, 2015).  
By using SHA-256, the business makes sure that when clients download its public key, they can verify the file’s integrity and authenticity by comparing the computed checksum to the published one. This prevents undetected modifications during transmission and provides a reliable way to detect both accidental corruption and deliberate tampering.

## 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 certificate

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A number on a white background

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 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 error

AI-generated content may be incorrect.

## Functional Testing

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

A screenshot of a computer

AI-generated content may be incorrect.

## Summary

For this projects I have added the hash functionality to generate a checksum verification on the SslServerApplication.java file. Generating this allows to add a “fingerprint” to the data. I used SHA-256 in this application. Using this helps make sure that data is not being corrupted, detects any tampering or unauthorized modifications, and makes sure that the sender and receiver is working with the exact same file or message (NIST, 2020). Then I updated the application.properities file to use the certificate that I generated for the application. This helps add authentication of the server. It also provides trust and encryption to create an environment for secure communication. For example, information being sent such as passwords, account numbers, or other personal information is scrambled to outsiders so it cannot be read(NIST, 2020). Lastly, On the pom.xml file I updates the dependency plug-in to make sure that it was using the most current version which is 12.1.0.

## Industry Standard Best Practices

During the development of this application industry standard best practices were used. I used proven cryptography of SHA-256 instead of creating custom code to ensure that attackers cannot just predict or brute-force the keys (NIST,2020). Next data was encrypted by using a server certificate allowing for the use of https rather than http. This prevents man-in-the-middle attacks between the browser and the server (OWASP, 2021). Lastly, I used dependency checks to scan the third-party libraries used for known vulnerabilities. These vulnerabilities can be addressed and suppressed where necessary and will help keep risk under control as the code changes (OWAPS, 2021). These practices reduce the likelihood and impact of breaches, protect customer trust, and help Artemis meet regulatory and contractual expectations without scrambling later. They also lower long-term costs: fixing security issues early is far cheaper than incident response or legal exposure after a breach (OWASP, 2021). One final note: aligning with recognized frameworks (OWASP Top 10, NIST SSDF) creates a consistent security baseline the whole team can follow. It allows new hires, auditors, and partners the ability to understand the standard, which speeds up collaboration and review processes. It is important to all be on the same page(NIST, 2022).

References

National Institute of Standards and Technology [NIST]. (2015). *Secure hash standard (SHS)* (FIPS PUB 180-4). U.S. Department of Commerce. <https://doi.org/10.6028/NIST.FIPS.180-4>

NIST. (2020). *Guidelines for the selection, configuration, and use of Transport Layer Security (TLS) implementations* (SP 800-52 Rev. 2). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-52r2>

NIST. (2022). *Secure Software Development Framework (SSDF) Version 1.1: Recommendations for mitigating the risk of software vulnerabilities* (SP 800-218). National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-218>

Oracle. (n.d.). *Java security programmer’s guide*. Oracle. <https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html>

OWASP. (2021). *OWASP Top 10: The ten most critical web application security risks*. Open Worldwide Application Security Project. <https://owasp.org/www-project-top-ten/>

Stallings, W. (2017). *Cryptography and network security: Principles and practice* (7th ed.). Pearson.