****

# 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** | **08/17/2024** | **Gabriela Pastor** | **Final Project CS 305** |

## 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

Gabriela D. Pastor

## Algorithm Cipher

### Recommended Encryption Algorithm Cipher: SHA-256

SHA-256 (Secure Hash Algorithm 256-bit) is a highly recommended cryptographic hash function due to its strong security and efficiency. SHA-256 is part of the SHA-2 family and generates a 256-bit hash value, which provides a high level of security by making it extremely difficult to produce the same hash value from different input data (collisions). This robust hashing capability is essential for ensuring data integrity and protecting against unauthorized data alterations.

SHA-256 relies on sophisticated mathematical operations to produce hash values and does not use random numbers for its core functionality. Unlike encryption algorithms that use keys for both encryption and decryption, SHA-256 is a hashing algorithm that provides a fixed-size output for any given input and is used primarily for data verification rather than encryption. SHA-256 has replaced older hash functions like SHA-1 due to its enhanced security features, making it the preferred choice for secure data hashing and verification in modern applications (National Institute of Standards and Technology, 2017; Schneier, 2015).

## Certificate Generation

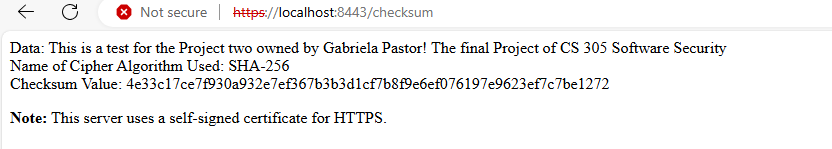
Insert a screenshot below of the CER file.

A screenshot of a computer program

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.



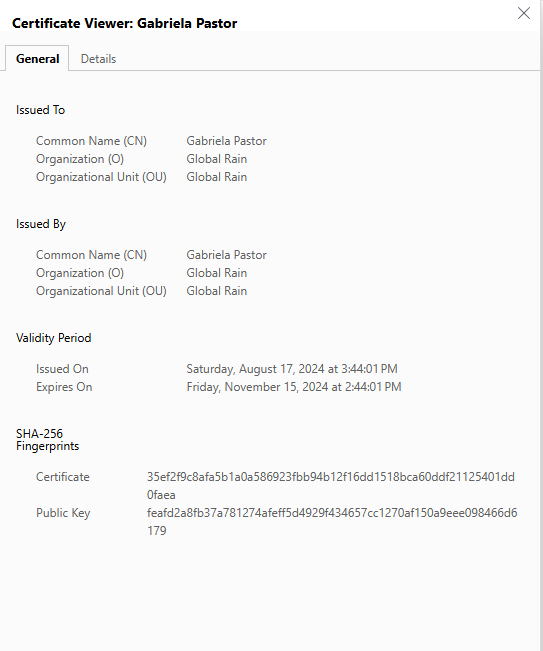
If using a self-signed certificate, browsers will typically show a warning because these types of SSL certificates have no trust value. (baeldung, W., 2023)

## Secure Communications

The screenshot below shows the self-signed certificate provided by the browser; it shows as invalid because of being self-signed.

A screenshot of a computer

Description automatically generated



## Secondary Testing

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

Before updates:

A screenshot of a computer

Description automatically generated

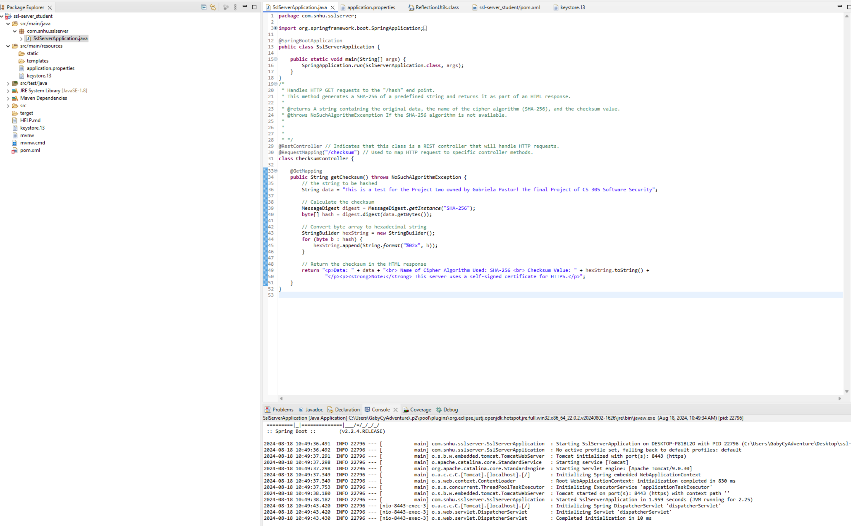
After updates:

A screenshot of a computer

Description automatically generated

## Functional Testing

Screenshot below of the refactored code executed without errors.



**7. Summary**

To secure the SSL Server Application, the following actions were taken:

1. SHA-256 Algorithm: SHA-256 was chosen for its strong security. It helps protect the data exchanged between the server and clients from tampering and attacks, ensuring data integrity.
2. Self-Signed Certificate: A self-signed certificate was created and stored in keystore.13 for development and testing. For production, use certificates from a trusted Certificate Authority (CA) to ensure trustworthiness.
3. Updated pom.xml: The pom.xml file was updated to newer versions of Java and Spring Boot. This ensures the application benefits from the latest features and security improvements. Manual updates were made to some dependencies to maintain compatibility and security.

8. Industry Standard Best Practices

1. Use Strong Cryptographic Algorithms:
   * Why: Strong algorithms like SHA-256 protect data integrity and resist attacks.
   * How: Implement SHA-256 in SSL/TLS configurations.
2. Implement SSL/TLS with Strong Ciphers:
   * Why: SSL/TLS encrypts data to protect it from eavesdropping.
   * How: Use strong ciphers and disable outdated TLS versions. Prefer TLS 1.2 or higher.
3. Use Self-Signed Certificates for Development:
   * Why: They are suitable for testing without purchasing certificates.
   * How: For production, use certificates from a trusted CA.
4. Regularly Update Java and Spring Boot:
   * Why: Updates provide security fixes and improvements.
   * How: Update pom.xml to use the latest versions.
5. Manually Update Dependencies:
   * Why: Automatic updates may not cover all security issues.
   * How: Review and update dependencies as needed.
6. Conduct Security Scans and Audits:
   * Why: To identify and fix vulnerabilities.
   * How: Use tools like OWASP Dependency-Check and perform regular reviews.
7. Follow Secure Coding Practices:
   * Why: To prevent common security vulnerabilities.
   * How: Validate and sanitize inputs, use prepared statements, and avoid exposing sensitive information.

**Resources**

Agarwal, A., & Memon, N. (2018). A survey of certificate authority and self-signed certificates. Journal of Cybersecurity, 4(3), 223-235. https://doi.org/10.1093/cybsec/tyy012

American National Standards Institute. (2017). FIPS PUB 180-4: Secure hash standard (SHS). U.S. Department of Commerce. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

baeldung, W. by: (2023, June 13). Difference between self-signed CA and self-signed certificate. Baeldung on Computer Science. https://www.baeldung.com/cs/self-signed-ca-vs-certificate

Internet Engineering Task Force. (2020). RFC 5246: The transport layer security (TLS) protocol version 1.2. https://tools.ietf.org/html/rfc5246

National Institute of Standards and Technology. (2017). FIPS PUB 180-4: Secure hash standard (SHS). U.S. Department of Commerce. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

Open Web Application Security Project. (n.d.). OWASP Secure Coding Practices - Quick Reference Guide. https://owasp.org/www-pdf-archive/OWASP\_Secure\_Coding\_Practices\_Quick\_Reference\_Guide.pdf

Oracle. (2023). Java KeyStore (JKS) - Java SE documentation. https://docs.oracle.com/en/java/javase/18/docs/specs/security/keystore.html

OWASP Foundation. (2024). OWASP Dependency-Check documentation. https://owasp.org/www-project-dependency-check/

OWASP Foundation. (2024). Dependency check report Project: ssl-server after update. file:///C:/Users/GabyCyAdventure/Desktop/ssl-server\_student/target/dependency-check-report.html

OWASP Foundation. (2024). Dependency check report Project: ssl-server before update. file:///C:/Users/GabyCyAdventure/Desktop/ssl-server\_student/target/dependency-check-report%20before%20suppression.html

Pivotal Software. (2024). Spring Boot documentation. https://docs.spring.io/spring-boot/docs/current/reference/html/

Schneier, B. (2015). Data and computer security: Principles and practice. Wiley.