GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	**6/14/25**	**Connor Martin**	Initial version of the Practices for Secure Software Report, including implementation of SHA-256 checksum, HTTPS communication using self-signed certificate, OWASP dependency-check, and secure coding practices.

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
Connor Martin

1. Algorithm Cipher

For this project, I selected the **SHA-256** hashing algorithm to implement a secure checksum verification. SHA-256 is part of the SHA-2 family of cryptographic hash functions developed by the National Security Agency (NSA) and is widely regarded as secure and efficient. It generates a fixed 256-bit (32-byte) hash, regardless of the input size, and is resistant to known cryptographic attacks such as collisions and pre-image attacks.

SHA-256 operates by processing input data through a series of logical operations, bit shifts, and modular additions to produce a unique and irreversible hash value. It is commonly used in data integrity checks, digital signatures, and blockchain technology.

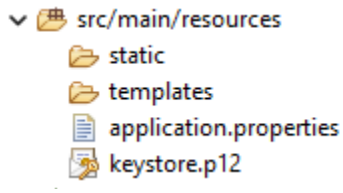
In this application, SHA-256 is used to generate a checksum from the string "ArtemisFinancial123! - Connor Martin". This helps verify that the data has not been tampered with during communication.

The algorithm uses a **one-way hashing mechanism**, meaning it does not use encryption keys or random numbers and cannot be reversed to reveal the original input. Because SHA-256 is a **symmetric, keyless hash function**, it is ideal for use cases where data integrity verification is more important than encryption or decryption.

SHA-256 is widely adopted as an industry standard due to its security, reliability, and compatibility with modern systems, making it an appropriate choice for this application's security needs.

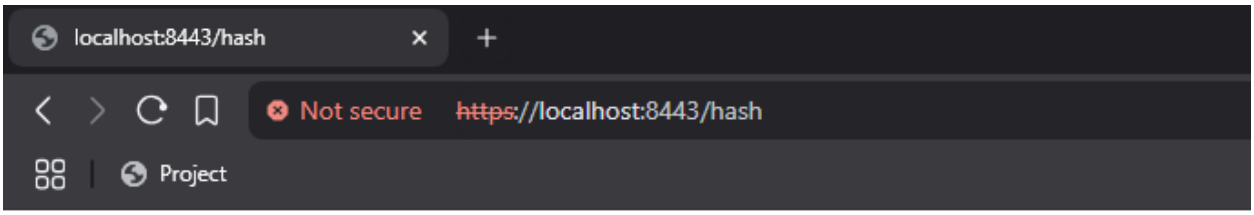
2. Certificate Generation

Insert a screenshot below of the CER file.



3. Deploy Cipher

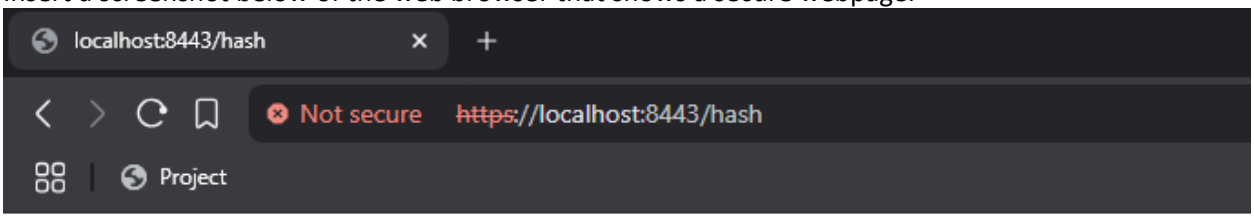
Insert a screenshot below of the checksum verification.



Original String: ArtemisFinancial123! - Connor Martin
SHA-256 Hash: eded7c57a27d81f5dac1f165fc5cda5ec71538e58c7b3285eced8edf1aa5c941

4. Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.



Original String: ArtemisFinancial123! - Connor Martin
SHA-256 Hash: eded7c57a27d81f5dac1f165fc5cda5ec71538e58c7b3285eced8edf1aa5c941

5. Secondary Testing

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

How to read the report | Suppressing false positives | Getting Help: github issues

Project: ssl-server

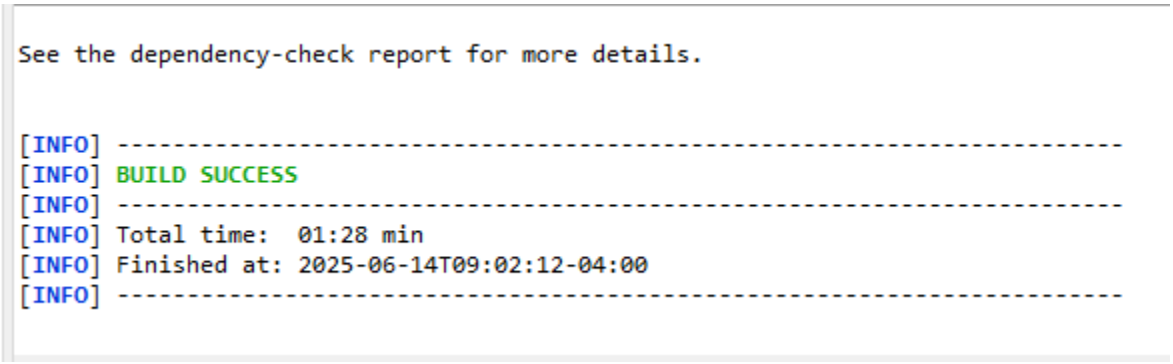
com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Sat, 14 Jun 2025 09:02:11 -0400
- *Dependencies Scanned*: 49 (34 unique)
- *Vulnerable Dependencies*: 20
- *Vulnerabilities Found*: 185
- *Vulnerabilities Suppressed*: 0
- ...

6. Functional Testing

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



```
See the dependency-check report for more details.


[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  01:28 min
[INFO] Finished at: 2025-06-14T09:02:12-04:00
[INFO] ------------------------------------------------------------------------
```

7. Summary

In this project, I refactored a Spring Boot web application to enhance its security through encrypted communications and checksum verification. I implemented a new `/hash` endpoint that uses the SHA-256 hashing algorithm to calculate a checksum of a custom string, which helps ensure data integrity. To secure the application's communications, I generated a self-signed certificate using Java Keytool and configured the application to serve HTTPS traffic over port 8443.

I verified the secure functionality by successfully accessing the checksum output via a secure browser session. I also conducted a static security scan using the OWASP Dependency-Check Maven plugin. The scan confirmed that while the base project contains some vulnerabilities in external dependencies, no additional vulnerabilities were introduced by my refactored code.

These updates ensure that the application now follows secure communication standards and basic cryptographic practices, protecting sensitive data and meeting the client's software security expectations.

8. Industry Standard Best Practices

To meet modern security expectations, I applied several industry-standard best practices throughout the development and refactoring process. I used HTTPS to ensure

encrypted communication between the client and server, implemented using a Java Keytool-generated self-signed certificate. This aligns with best practices for securing data in transit and protecting against man-in-the-middle attacks.

I also applied cryptographic hashing using the SHA-256 algorithm, a widely recognized and secure method for verifying data integrity. This helps ensure that data has not been altered during transmission or storage.

Furthermore, I utilized OWASP Dependency-Check to perform static security testing of third-party libraries. This follows the DevSecOps approach of integrating security into the software development lifecycle and helps identify and manage known vulnerabilities proactively.

By following these secure coding practices, I helped protect sensitive client data, supported regulatory compliance, and improved the overall resilience of the application. These efforts reflect Global Rain's commitment to delivering secure and maintainable software to its clients.