****

# 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]** | **[Your Name]** |  |

## 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

Joseph Dumke

## Algorithm Cipher

I will recommend SHA-256 because it has the lowest chance of collision. In addition, it is a secure cryptographic hash function designed to be collision-resistant. SHA-256 produces a fixed-size 256-bit (32-byte) hash, ensuring that even a small change in input results in a vastly different output. This makes it ideal for verifying file integrity and detecting tampering. It is widely supported in Java through the MessageDigest class and is a standard choice for checksum verification due to its strength, efficiency, and widespread adoption.

SHA-256 is part of the SHA-2 family and generates a 256-bit hash. It is not reversible and is designed to be one-way, ensuring that attackers cannot derive the original input from the output. Its long bit length makes brute-force attacks computationally infeasible with current technology.

While SHA-256 itself does not use random numbers, it is often used in systems that incorporate salts or nonces — random values added to data before hashing — to further increase security, especially in password storage or digital signing. These random values prevent attackers from using precomputed tables (like rainbow tables) to guess the original input.

SHA-256 is not a key-based cipher, so it does not fall under symmetric or asymmetric encryption. However, it is often used alongside encryption algorithms in both symmetric (like AES) and asymmetric (like RSA) systems to ensure integrity. For example, in digital signatures, the data is hashed using SHA-256 and then encrypted with a private key.

SHA-256 was developed by the National Security Agency (NSA) and published by NIST in 2001 as part of the SHA-2 family to replace SHA-1, which showed vulnerabilities. SHA-256 is currently one of the most trusted hash algorithms in the industry, widely used in SSL/TLS, blockchain, password hashing, and software integrity checks. Despite advancements in computing, no practical collisions have been found, and it remains secure when used properly.

## Certificate Generation

Insert a screenshot below of the CER file.  
A screenshot of a computer

AI-generated content may be incorrect.

## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

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

To enable secure communication, the application.properties file was updated to configure SSL using a self-signed keystore. The application was then launched on port 8443 with HTTPS enabled. The screenshot below shows the browser securely connecting to the /hash endpoint, confirming encrypted communication using the SSL configuration.



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.  
  
1. **hibernate-validator-6.0.18.Final.jar**

**Description:** Hibernate Validator is the reference implementation of Java's Bean Validation API (JSR- 380). It allows developers to apply annotations to enforce validation rules on class fields and methods. Common in enterprise apps, it helps prevent invalid data from being processed, especially in forms or API payloads.

**CVE Numbers:** CVE-2020-10693, CVE-2023-1932

2. **jackson-databind-2.10.2.jar**

**Description:** General data-binding functionality for Jackson: works on core streaming API

**CVE Numbers:** CVE-2020-25649, CVE-2020-36518, CVE-2021-46877, CVE-2022-42003, CVE-2022-42004, CVE-2023-35116

3. **json-path-2.4.0.jar**

**Description:** Java port of Stefan Goessner JsonPath.

**CVE Numbers:** CVE-2023-51074 (OSSINDEX)

4. **json-smart-2.3.jar**

**Description:** JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language**.  
CVE Numbers:** CVE-2021-31684 (OSSINDEX), CVE-2023-1370, CVE-2021-27568

1. **log4j-api-2.12.1.jar**

**Description:** The Apache Log4j API

**CVE Numbers:** CVE-2020-9488

1. **logback-classic-1.2.3.jar  
   Description:** logback-classic module **CVE Numbers**: CVE-2023-6378 , CVE-2021-42550
2. **logback-core-1.2.3.jar  
   Description:** logback-core module **CVE Numbers:** CVE-2023-6378**,** CVE-2021-42550**,** CVE-2024-12798**,**  CVE-2024-12801
3. **snakeyaml-1.25.jar  
   Description:** YAML 1.1 parser and emitter for Java

**CVE Numbers:** CVE-2022-1471, CVE-2017-18640, CVE-2022-25857 , CVE-2022-38749, CVE-2022-38751, CVE-2022-38752, CVE-2022-41854, CVE-2022-38750

1. **spring-boot-2.2.4.RELEASE.jar**

**Description:** Spring Boot

**CVE Numbers:** CVE-2023-20873, CVE-2022-27772, CVE-2023-20883

1. **spring-boot-2.2.4.RELEASE.jar**   
   **Description:** Starter for building web, including RESTful, applications using Spring

MVC. Uses Tomcat as the default embedded container  
**CVE Numbers:** CVE-2023-20873, CVE-2022-27772, CVE-2023-20883

1. **spring-core-5.2.3.RELEASE.jar**

**Description:** Spring Core

**CVE Numbers:** CVE-2022-22965, CVE-2021-22118, CVE-2020-5421, CVE-2022-22950, CVE-2022-22971 s, CVE-2023-20861, CVE-2023-20863, CVE-2022-22968, CVE-2022-22970, CVE-2021-22060 , CVE-2021-22096

1. **spring-data-rest-webmvc-3.2.4.RELEASE.jar**

**Description:** Spring Data REST – WebMVC

**CVE Numbers:** CVE-2021-22047 (OSSINDEX), CVE-2022-31679 (OSSINDEX)

1. **spring-expression-5.2.3.RELEASE.jar**

**Description:** Spring Expression Language (SpEL)

**CVE Numbers:** CVE-2022-22965, CVE-2021-22118 , CVE-2020-5421, CVE-2022-22950, CVE-2022-22971, CVE-2023-20861, CVE-2023-20863, CVE-2022-22968, CVE-2022-22970, CVE-2024-38808 (OSSINDEX), CVE-2021-22060 , CVE-2021-22096

1. **spring-hateoas-1.0.3.RELEASE.jar**

**Description:** Library to support implementing representations for hyper-text driven REST web services.

**CVE Numbers:** CVE-2023-34036

1. **spring-web-5.2.3.RELEASE.jar**

**Description:** Spring Web

**CVE Numbers:** CVE-2016-1000027, CVE-2022-22965, CVE-2024-38809 (OSSINDEX), CVE-2024-22243 (OSSINDEX) , CVE-2024-22262 (OSSINDEX), CVE-2021-22118, CVE-2024-38828 (OSSINDEX), CVE-2020-5421, CVE-2022-22950 s, CVE-2022-22971 , CVE-2023-20861, CVE-2023-20863, CVE-2022-22968, CVE-2022-22970, CVE-2021-22060, CVE-2021-22096

1. **spring-webmvc-5.2.3.RELEASE.jar**

**Description:** Spring Web MVC

**CVE Numbers:** CVE-2022-22965 , CVE-2024-38816 (OSSINDEX), CVE-2021-22118, CVE-2020-5421, CVE-2022-22971, CVE-2023-20861, CVE-2023-20863 , CVE-2022-22968, CVE-2022-22970, CVE-2021-22060, CVE-2021-22096

1. **tomcat-embed-core-9.0.30.jar**

**Description:** Core Tomcat implementation  
**CVE Numbers:** CVE-2020-1938, CVE-2025-24813, CVE-2020-11996, CVE-2020-13934 , CVE-2020-13935, CVE-2020-17527, CVE-2021-25122, CVE-2021-41079, CVE-2022-29885, CVE-2022-42252, CVE-2023-44487, CVE-2023-46589, CVE-2024-38286, CVE-2020-9484 , CVE-2021-25329, CVE-2021-30640, CVE-2022-34305, CVE-2023-41080, CVE-2021-24122, CVE-2021-33037, CVE-2021-33037, CVE-2023-45648, CVE-2024-21733, CVE-2019-17569, CVE-2020-1935, CVE-2020-13943, CVE-2023-28708, CVE-2021-43980

1. A screenshot of the refactored code executed without errors  
     
   A computer screen shot of a program code

   AI-generated content may be incorrect.
2. A screenshot of the report of the output from the dependency-check static tester  
   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 computer screen shot of a program code

AI-generated content may be incorrect.

## Summary

To make sure the programs code has been refactored and complies with security testing protocol I added a RestController, to work as the secure controller for my programs hash RESTful stop. In addition, I chose SHA-256, because it is a secure cryptographic hash function designed to be collision-resistant. This makes it ideal for verifying file integrity and detecting tampering. Futhermore, to maintain the current security I would recommend once or twice a month dependency checks to check for vulnerabilities. Also would check for updates on the pom.xml file.

## Industry Standard Best Practices

## The industry standard best practices for security I applied are Switching from HTTP to HTTPS aligns with OWASP's recommendation to encrypt all data in transit. In addition, certificate management such as the use of a keystore and alias allows proper tracking and handling of digital credentials.

## Moreover, static code analysis, by running dependency-check, I ensured all libraries and dependencies used in the project were vetted for known CVEs (Common Vulnerabilities and Exposures). Furthermore, to maintain existing security, I did not alter any business logic or authentication mechanisms already in place, ensuring application functionality was preserved.