****

# 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 5](#_Toc102040760)

[4. Secure Communications 5](#_Toc102040761)

[5. Secondary Testing 6](#_Toc102040762)

[6. Functional Testing 7](#_Toc102040763)

[7. Summary 8](#_Toc102040764)

[8. Industry Standard Best Practices 8](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/10/2025** | **Nicole Hutto** |  |

## 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

Nicole Hutto

## Algorithm Cipher

For this project, I constructed a system secure enough to generate checksums for input data. I opted for the SHA-256 hashing algorithm, a cryptographic hash function that produces a fixed 256-bit (32-byte) hash value with an almost non-existent probability of producing the same output for different inputs. That property, known as being collision-resistant, has to do with the kinds of mathematics that underline the workings of the SHA-256 algorithm.

The algorithm was incorporated into a Spring Boot HTTPS server. For a secure communication channel with the server, a self-signed SSL certificate was created and stored in a keystore.p12 file. The server was then instructed to use that keystore for HTTPS communication on port 8443. We specify the API endpoint we wish to secure: /hash. With the server secured, all user-to-server interactions should flow through it first and then get routed to /hash.

## 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 screenshot of a computer

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.

## 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 screen shot of a computer screen

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

AI-generated content may be incorrect.

## Summary

In this project, I took the existing, SSL-enabled Spring Boot application and completely restructured it. Along with that went the successful integration of the OWASP Dependency-Check Maven plugin into the project. What the plugin does is help identify potential vulnerabilities in the project's dependencies. After I executed the plugin, it served up a report that listed out the kinds of issues the project was dealing with. And I mean listed. A detailed list of everything it found — and then some. Including severity levels, CVE counts, and a whole bunch of affected packages. That served as a good overview of what needed to be acted on.

## Industry Standard Best Practices

Java web applications should be secured and maintained according to industry-standard best practices that include:

* Regular Dependency Scanning: Use tools like OWASP Dependency-Check to identify vulnerabilities early and keep libraries up to date.
* Version Management: Maintain dependencies at stable, supported versions to reduce exposure to known security flaws.
* SSL/TLS Implementation: Secure data in transit using valid SSL/TLS certificates, enforcing HTTPS by default.
* Configuration Management: Store sensitive credentials in secure locations (e.g., environment variables or encrypted vaults) rather than in plaintext within code.
* Fail-Safe Defaults: Configure systems to deny by default and only allow explicitly permitted actions.
* Continuous Integration Security Checks: Incorporate security scanning tools into the CI/CD pipeline to automate detection of vulnerabilities before deployment.