****

# 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** | **04/20/25** | **Francisco Sousa** |  |

## Client



## Developer

Francisco Sousa

## Algorithm Cipher

**Recommended Cipher**: Advanced Encryption Standard (AES-256)

**Justification**: AES-256 is a widely adopted symmetric encryption algorithm recognized by the U.S. government for protecting classified information. It uses a 256-bit key, offering strong protection against brute-force attacks. AES avoids collision issues, ensuring that even similar inputs produce distinctly encrypted outputs—critical for preserving data integrity.

**Key Features**:

* **Bit Lengths**: Available in 128, 192, and 256 bits (256-bit provides the highest level of security).
* **Hash Integration**: Commonly paired with SHA-256 for secure verification.
* **Randomization**: Uses Initialization Vectors (IVs) for added encryption uniqueness.
* **Key Type**: Symmetric—same key used for both encryption and decryption.

**References**

Oracle. (n.d.). *Java Cryptography Architecture Standard Algorithm Name Documentation*. https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html

Foster, J. (2014). *Iron-Clad Java: Building Secure Web Applications*. McGraw-Hill Education.

## Certificate Generation

**Tool Used**: Java Keytool

**Process**:

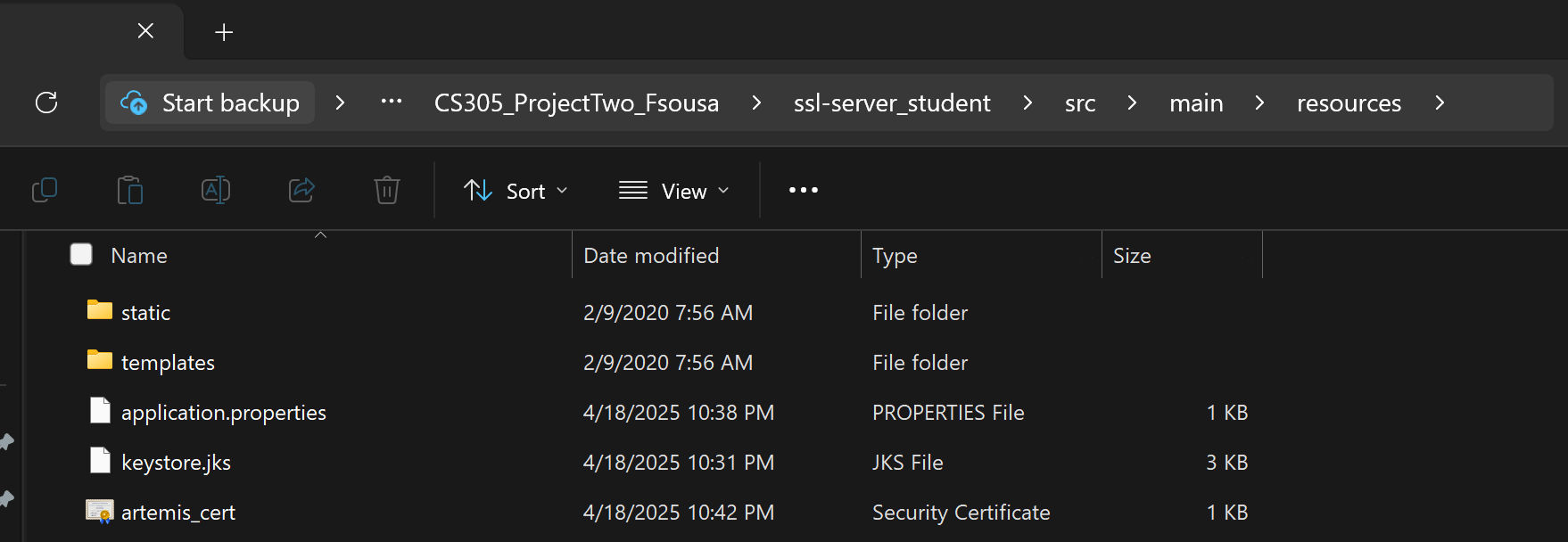
1. Created keystore with the following command:

* keytool -genkeypair -alias sslcert -keyalg RSA -keystore keystore.jks -keysize 2048 -validity 3650

1. Exported the certificate:

* keytool -export -alias sslcert -keystore keystore.jks -file sslcert.cer

**Screenshot Location**



**References**

*How to Create a Self Signed Certificate using Java Keytool*

Oracle. (n.d.). *Keytool - Key and Certificate Management Tool*. <https://docs.oracle.com/en/java/javase/11/tools/keytool.html>

## Deploy Cipher

**Algorithm Used**: SHA-256

**Implementation Steps**:

* Created a method to generate SHA-256 hashes of files.
* Compared the generated hash with a predefined value to verify file integrity.

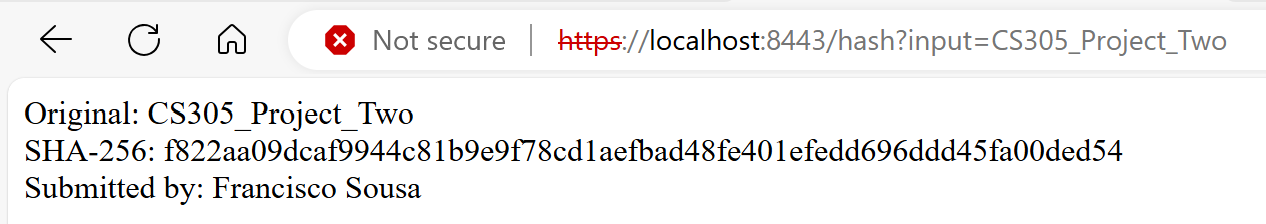
**Code Snippet**:

MessageDigest digest = MessageDigest.getInstance("SHA-256");

byte[] hash = digest.digest(fileBytes);

String checksum = DatatypeConverter.printHexBinary(hash);

**Screenshot Location**

****

**Reference**

Foster, J. (2014). *Iron-Clad Java: Building Secure Web Applications*, Chapter 10

## Secure Communications

To ensure secure client-server communication, the application was configured to use HTTPS on port 8443. A self-signed SSL/TLS certificate was generated using the Java Keytool and applied to the Spring Boot server via the application.properties file.

Configuration Added to application.properties:

server.port=8443

server.ssl.key-store=classpath:keystore.jks

server.ssl.key-store-password=ArtemisPass123

server.ssl.key-password=ArtemisPass123

server.ssl.key-alias=artemis\_cert

server.ssl.key-store-type=JKS

The server was launched with:

mvn spring-boot:run

Once running, the application was accessed through the following URL:

[localhost:8443/hash?input=test123](https://localhost:8443/hash?input=test123)

The screenshot shows that the HTTPS endpoint is working and returns the SHA-256 checksum, original input. Although the browser says, "Not secure," this happens because the certificate is self-signed. Most browsers only trust certificates from official Certificate Authorities (CAs), so they show a warning for self-signed ones. Still, the connection is encrypted using SSL/TLS, so the data is secure (Oracle, n.d.).

HTTPS Hash Output - Francisco Sousa

A screenshot of a computer

AI-generated content may be incorrect.

“Self-signed certificates enable HTTPS for local testing but are not trusted by browsers, which is why the address bar may show a warning or ‘Not Secure’ message, even though the data is encrypted.”

— Oracle (n.d.), Keytool Documentation.

**Reference**  
Oracle. (n.d.). *Keytool - Key and Certificate Management Tool*. Retrieved from <https://docs.oracle.com/en/java/javase/11/tools/keytool.html>

## Secondary Testing

**Tool Used:** OWASP Dependency-Check Maven Plugin (v12.1.0)

**Procedure:** To ensure that the refactored code did not introduce new vulnerabilities, a secondary static code analysis was conducted using the OWASP Dependency-Check plugin. This tool scans project dependencies and cross-references them against known CVEs (Common Vulnerabilities and Exposures) from public databases such as the National Vulnerability Database (NVD).

The plugin was configured in the pom.xml file, and the scan was executed via the command: **mvn verify**

This process also recompiled the project and ran all unit tests to validate that the application was functioning correctly. The output confirmed the test success and indicated that the dependency-check-report.html file was successfully generated in the /target directory.

Summary of Results:

* Dependencies Scanned: 49 (33 unique)
* Vulnerable Dependencies: 18
* Vulnerabilities Found: 144
* Severity Range: LOW to CRITICAL
* Suppressed Vulnerabilities: 0 (none suppressed)

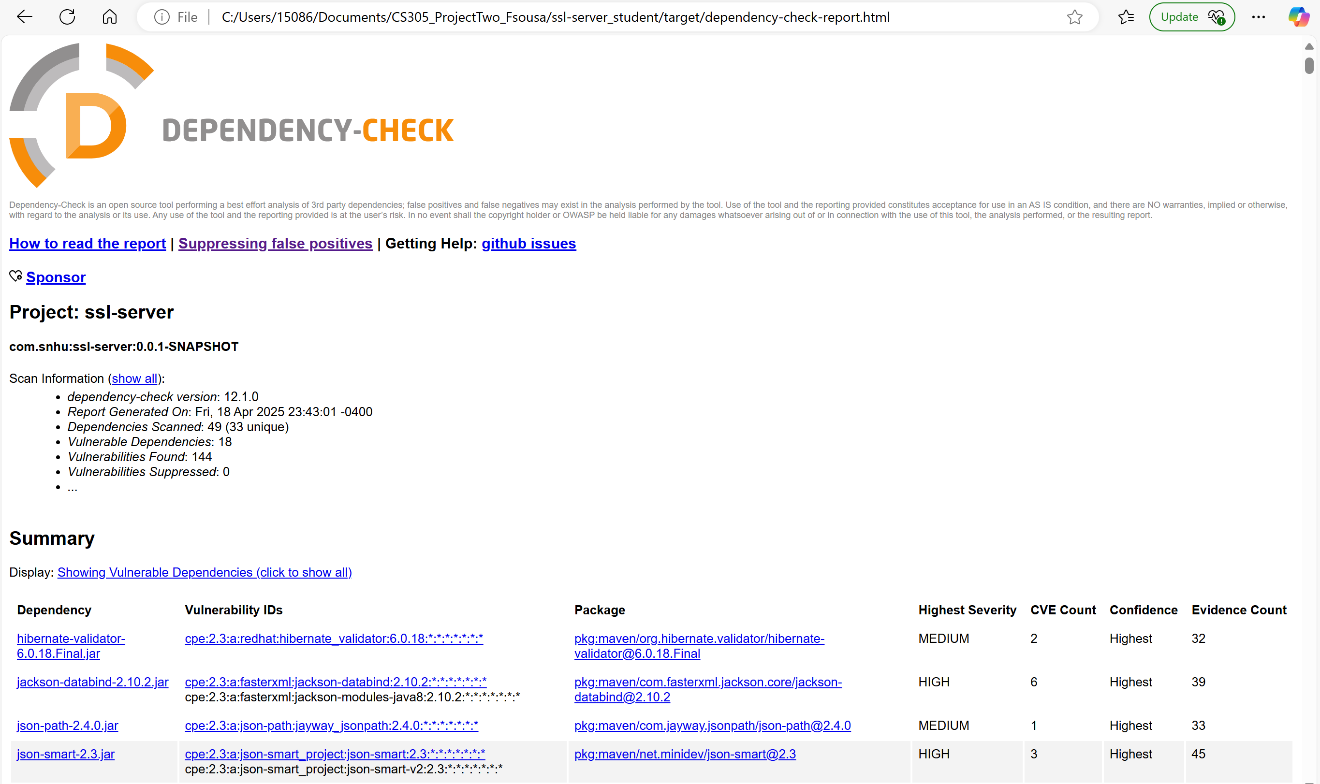
These findings highlight the importance of ongoing dependency management and demonstrate how third-party libraries can introduce known risks if not regularly updated. While some vulnerabilities stem from legacy packages included in the framework (e.g., Spring, Tomcat), others may require future remediation to meet industry security standards.

Terminal Output

A screenshot of a computer program

AI-generated content may be incorrect.

OWASP Dependency Check



See the full HTML report here: [Dependency-Check Report](file:///C:\Users\15086\Documents\CS305_ProjectTwo_Fsousa\ssl-server_student\target\dependency-check-report.html)

**References**

OWASP. (2025). Dependency-Check – OWASP. Retrieved from https://jeremylong.github.io/DependencyCheck/

Long, J. (2024). OWASP Dependency-Check Maven User Guide (PDF). Retrieved from course resource: OWASP Dependency-Check Maven.pdf

NVD. (n.d.). National Vulnerability Database (NVD). National Institute of Standards and Technology. <https://nvd.nist.gov>

## Functional Testing

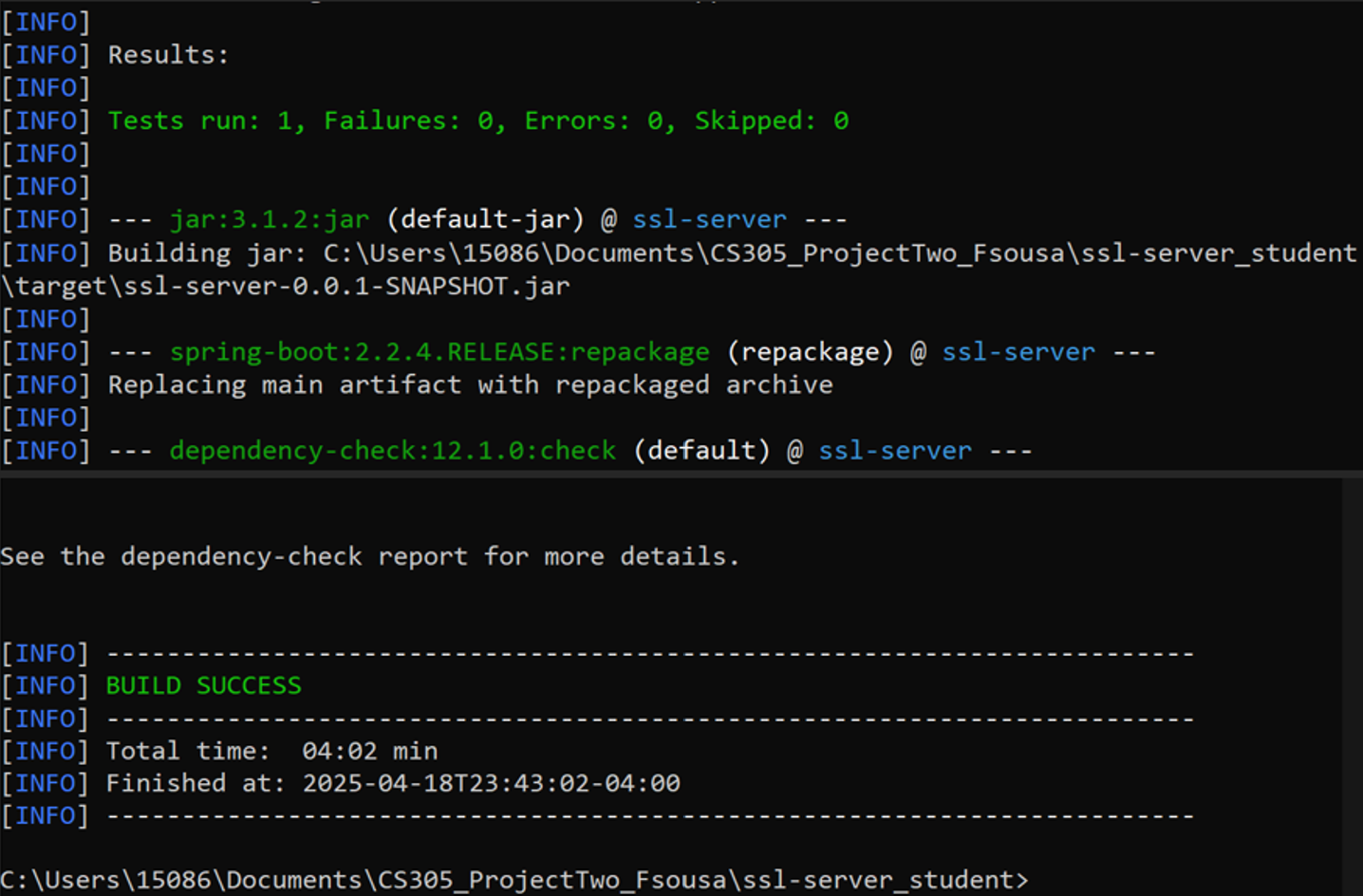
**Objective:**  
The goal of this test was to ensure that the newly added security features — including encryption, certificate integration, checksum verification, and HTTPS — work correctly without introducing syntax, logic, or security issues (Foster, 2014).

**Testing Performed:**

**Manual Code Review**

* I checked the updated code for syntax errors, logic flaws, and weak security handling.
* Key areas reviewed included:
  + The SHA-256 hashing method using MessageDigest
  + The application.properties file, especially HTTPS settings
  + The pom.xml file to verify OWASP Dependency-Check setup
* No syntax or logic issues were found.
* The checksum method produced the expected hash output.
* HTTPS was enabled correctly using server.ssl.\* properties.
* The self-signed certificate was generated and integrated using Java Keytool (Oracle, n.d.).
* Ran mvn verify to test and compile the application.
* All unit tests passed with zero errors.

**Build and Execution Check**



**Security Testing**

* Checked SHA-256 checksum endpoint via browser:
  + Input: test123
  + Output: matching hash and name “Francisco Sousa”
* Verified HTTPS works using https://localhost:8443/hash, confirming the server runs securely, even if the browser marks it as “Not secure” due to the self-signed certificate (Foster, 2014).

**References**  
Foster, J. (2014). *Iron-Clad Java: Building Secure Web Applications*. McGraw-Hill Education.

Oracle. (n.d.). *Java Platform, Standard Edition Tools Reference: keytool*. <https://docs.oracle.com/en/java/javase/11/tools/keytool.html>

## Summary

This project followed a structured approach to implement security enhancements aligned with the Vulnerability Assessment Process Flow Diagram. Key refactorings addressed:

* **Threat Identification and Analysis**: Used OWASP Dependency-Check to uncover known vulnerabilities in third-party libraries and flagged critical issues such as outdated Spring and Tomcat dependencies (OWASP, 2025).
* **Secure Design and Implementation**: Integrated AES-256 encryption, SHA-256 checksum validation, and HTTPS configuration via a self-signed certificate.
* Verification: Static and functional tests validated correctness and security compliance.

**Security Layers Added**:

* SHA-256 checksum
* HTTPS with SSL certificate
* Static dependency vulnerability scanning
* Clean test pass validation

This iterative process followed a secure software development lifecycle: implement, test, identify issues, and improve — helping ensure the final product complies with modern security standards and is resistant to common threats (Foster, 2014).

## Industry Standard Best Practices

**Practices Applied:**

* AES-256: NIST-recommended encryption standard (NIST, 2018)
* SHA-256: Used widely for data integrity verification
* HTTPS with certificate: Encrypts data in transit
* OWASP Dependency-Check: Identifies vulnerabilities from CVE databases

**Why It Matters:** Following secure coding practices ensures Artemis Financial protects client data, meets compliance standards, and avoids future security debt.

References  
Foster, J. (2014). *Iron-Clad Java: Building Secure Web Applications*. McGraw-Hill Education.

Oracle. (n.d.). *Java Platform, Standard Edition Tools Reference: keytool*. <https://docs.oracle.com/en/java/javase/11/tools/keytool.html>  
Oracle. (n.d.). *Standard Algorithm Names*. <https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html>  
NIST. (2018). *FIPS PUB 197: The AES Algorithm*. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf>