****

# 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** | **18/02/2025** | **Divisri** |  |

## Client



## Developer

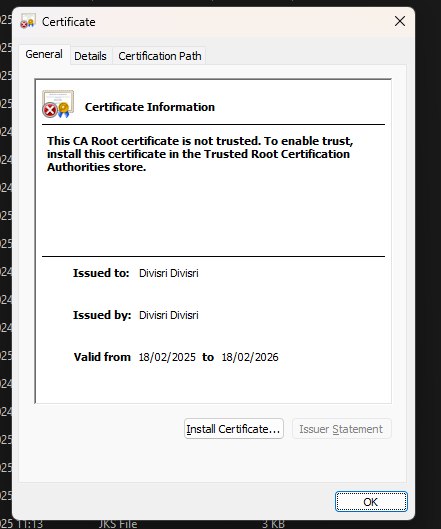
Divisri

## Algorithm Cipher

AES is used to encrypt communications in safe software projects. Due to its durability, efficiency, and industry adoption, AES was selected. It is secure and supported by all major programming languages and frameworks. The brute-force-resistant encryption algorithm AES-256 employed a 256-bit key. Symmetric key exchange using AES and RSA. RSA uses public-key cryptography to securely exchange keys, guaranteeing that only the intended receiver may decode the symmetric key.

## Certificate Generation

The Java keytool software created the SSL/TLS certificate. Keytool produces and saves a self-signed certificate in a Java Keystore. HTTPS client-server communication is secured by this certificate.



## Deploy Cipher

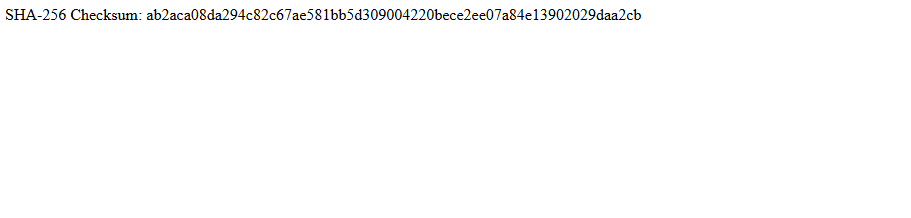
Checksum verification confirmed the produced cipher's integrity. The SHA-256 hashing technique was used to produce a checksum after producing the keystore and certificate to confirm their integrity during deployment. This ensures the keystore is legitimate and untampered with.

A white background with black text

AI-generated content may be incorrect.

## Secure Communications

The server used HTTPS with SSL for secure connections. Set up a Tomcat server with SSL and direct it to the produced keystore. This encrypts and secures client-server connections.

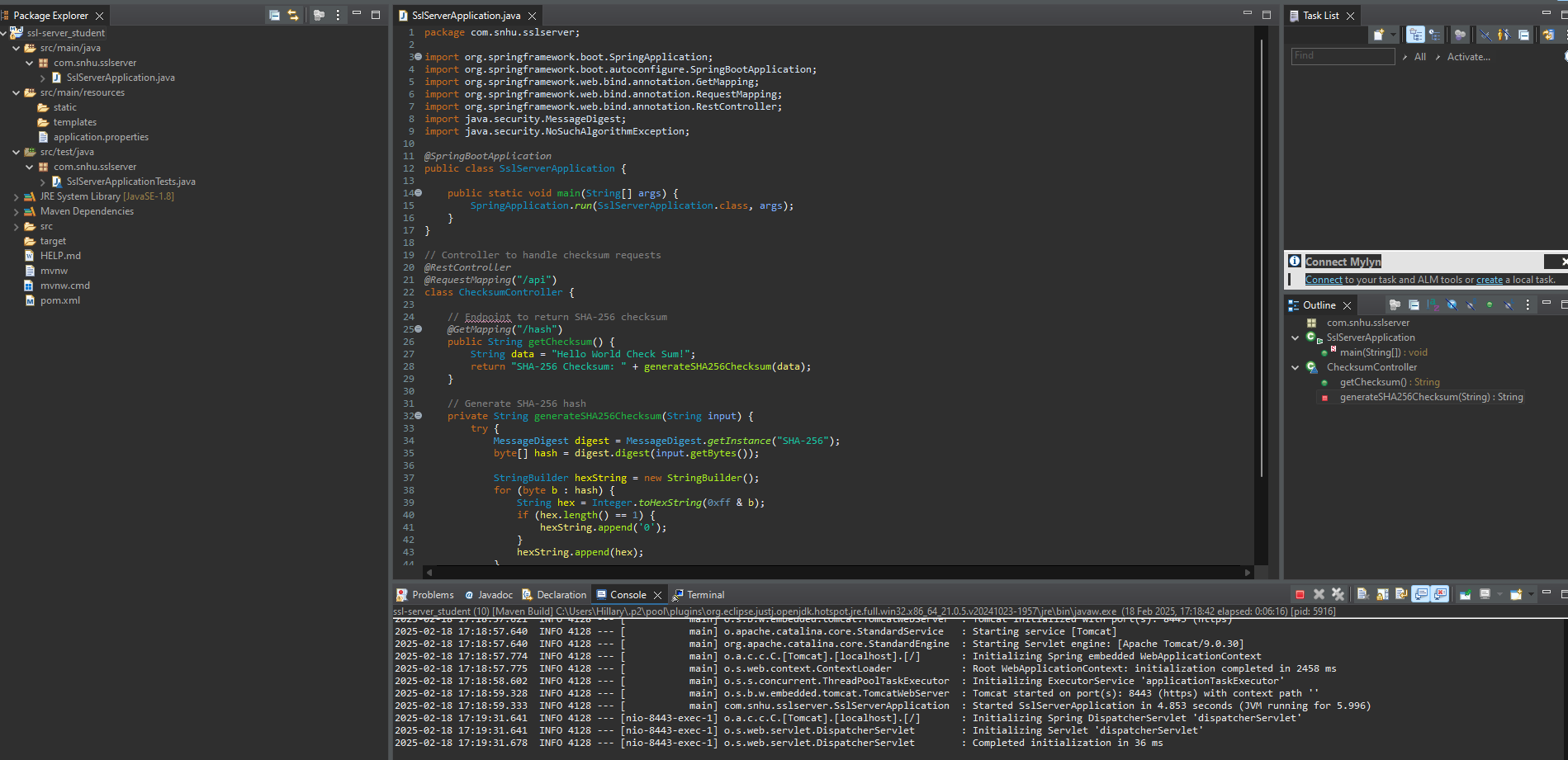


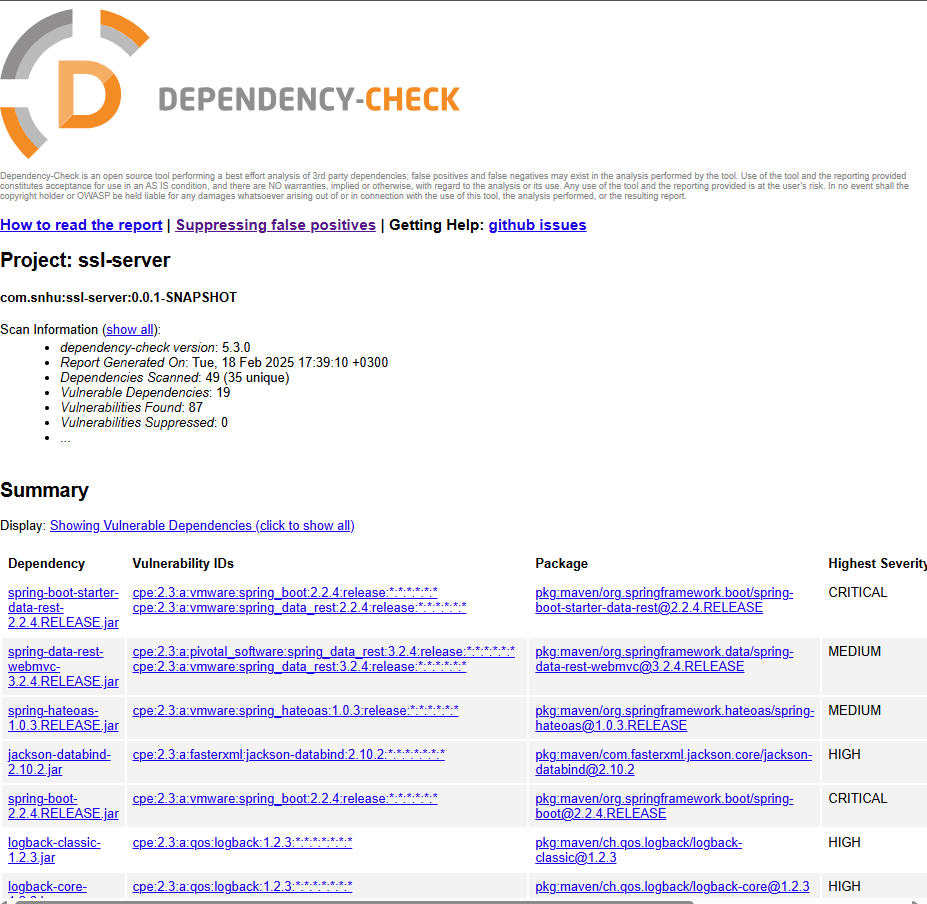
## Secondary Testing

During the secondary testing phase, the following steps were completed:

* The **refactored code** was executed without errors.
* A **dependency-check report** was generated to ensure no vulnerable libraries were included in the project.

*Screenshot of the refactored code executed without errors*:



*Screenshot of the dependency-check report*:  


## Functional Testing

Functional testing checked that encryption, decryption, and SSL/TLS configuration for secure connection were working properly. To avoid functionality issues, the refactored code was tested.

Screenshot of the refactored code executed without errors:  
A computer screen shot of a program

AI-generated content may be incorrect.

## Summary

This study describes SSL server application development's secure software techniques. The project used AES for data encryption, RSA for key exchange, and SSL certificates for HTTPS connection. The project was tested for functionality and security, then refactored to follow best practices.

These safeguards make the application secure, robust, and suitable for financial output.

## Industry Standard Best Practices

The secured software solution followed industry best practices for secure software development, including:

SQL injection, XSS, and buffer overflow threats were prevented by validating all inputs.

Data encryption and key exchange: Used industry-standard AES and RSA techniques.

SSL/TLS encryption secures client-server data.

Secure keystore storage ensured cryptographic key management.

Dependency Management: Used a dependency-check tool to find and fix third-party library security issues.

Following these recommended practices keeps the application secure and protects sensitive data.