****

# 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** | **12/15/2024** | **Patrick Quinn** |  |

## 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

Patrick Quinn

## Algorithm Cipher

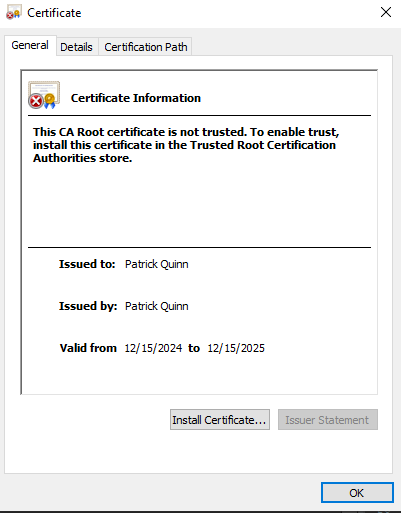
We will be using SHA-256. SHA-256 functions as a cryptographic hash function that generates a fixed-size 256-bit (32-byte) hash value for any input data. The algorithm processes data in 512-bit blocks through a series of sophisticated mathematical operations, including bitwise operations, modular additions, and compression functions. These operations work together to ensure robust collision resistance through multiple mechanisms.

The algorithm's collision avoidance capabilities are achieved through several key features. First, its 256-bit output space provides 2^256 possible hash values, making accidental collisions statistically improbable. Second, it demonstrates a strong avalanche effect, where even minimal changes in input produce significantly different hash outputs. Finally, while SHA-256 is deterministic (the same input always produces the same output), it is computationally infeasible to find two different inputs that generate identical hashes.

Understanding the importance of collision avoidance is crucial for appreciating SHA-256's value. In cryptographic applications, collisions could potentially enable attackers to substitute malicious data while maintaining the same hash value, compromising security. For digital signatures, collision resistance prevents the creation of fraudulent documents that share a hash with legitimate ones. In password storage scenarios, collision resistance adds an essential layer of security by preventing attackers from finding alternative inputs that would produce the same hash as a valid password.

## Certificate Generation

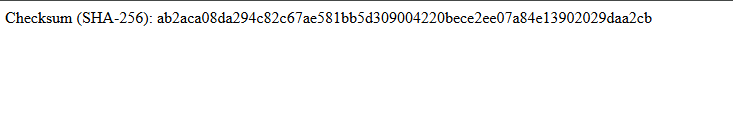
Insert a screenshot below of the CER file.

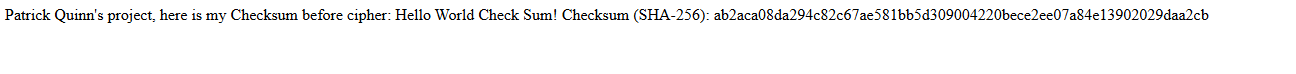




## Deploy Cipher

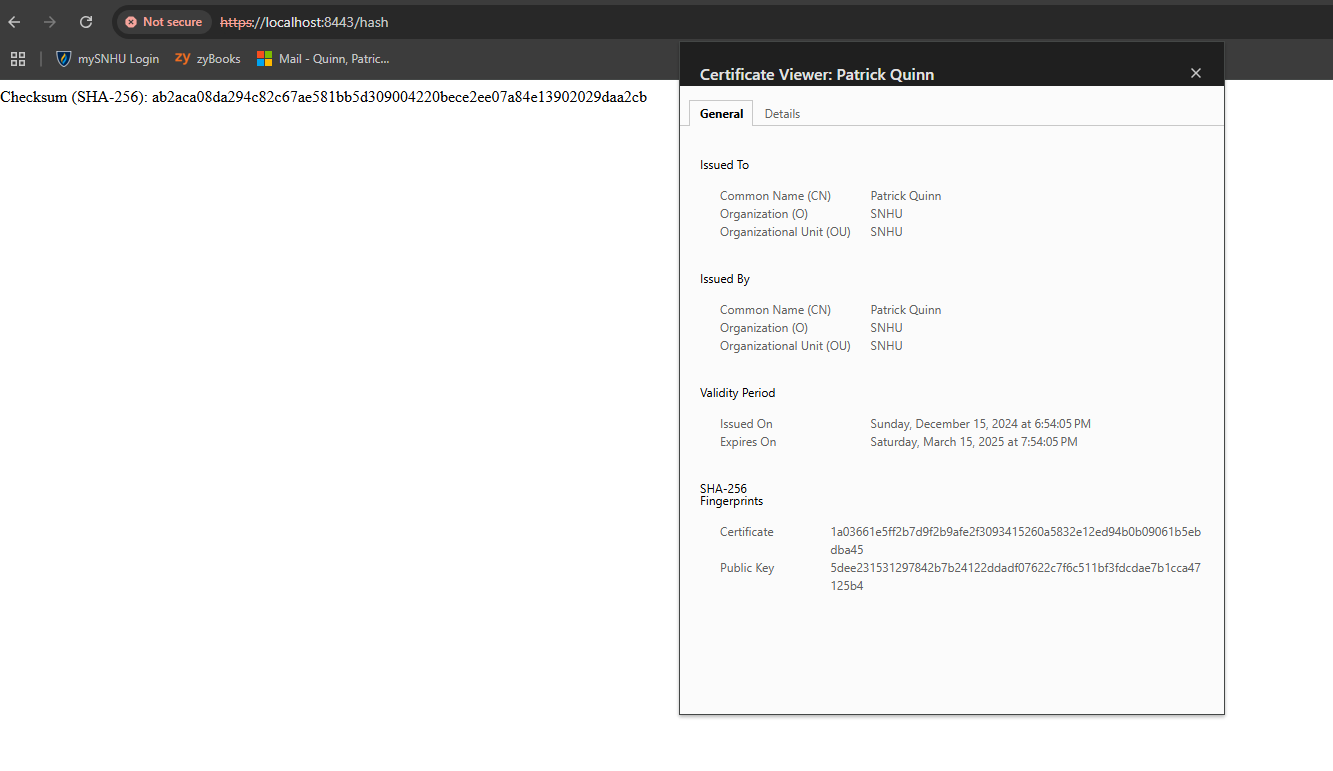
Insert a screenshot below of the checksum verification.

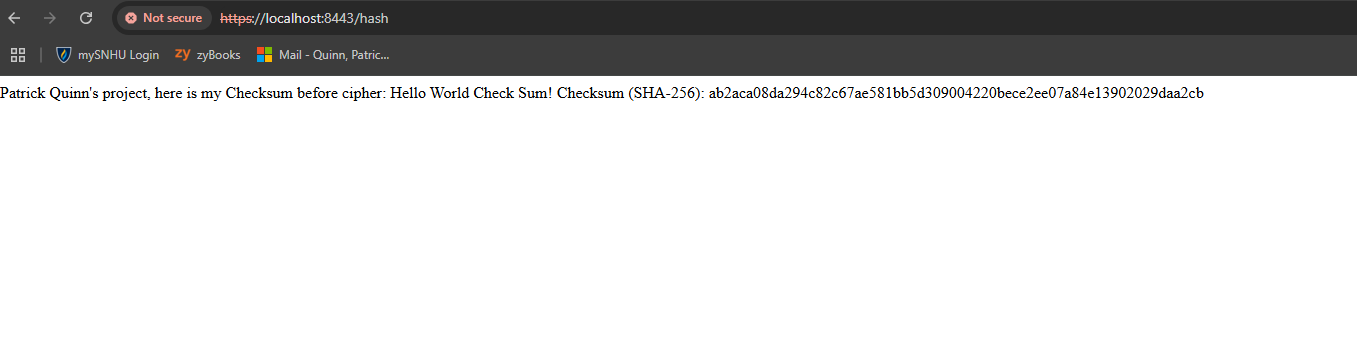




## Secure Communications

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

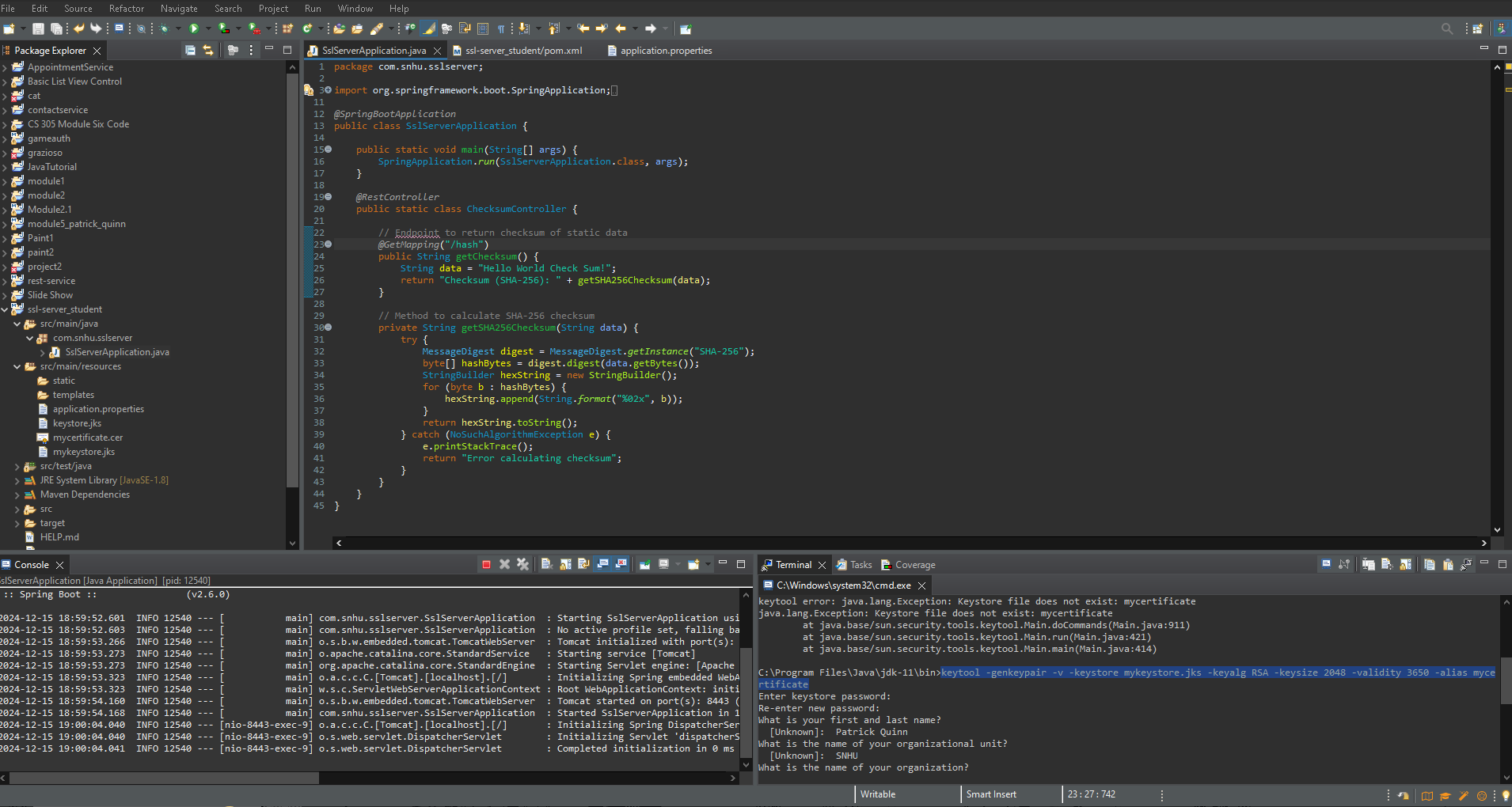


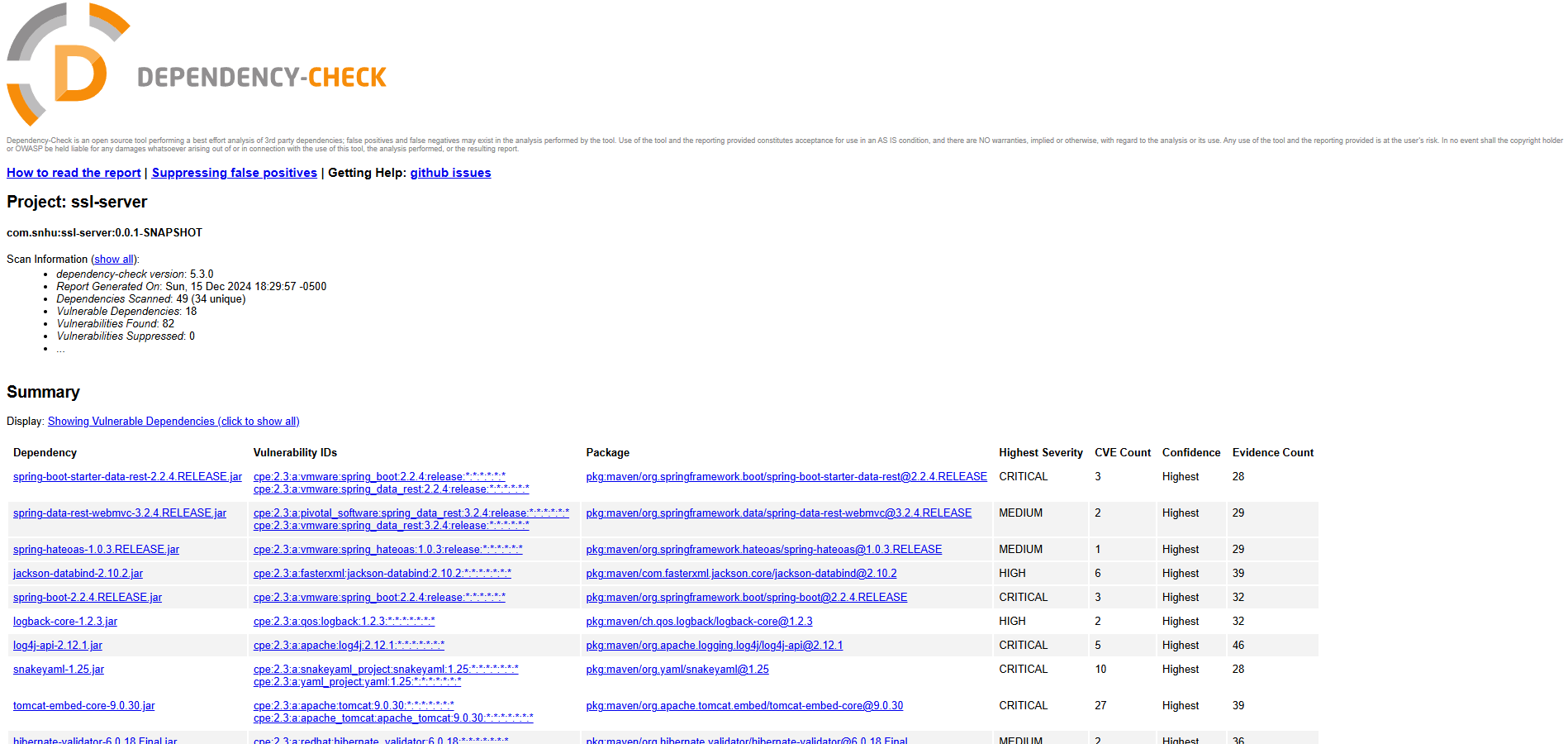


## Secondary Testing

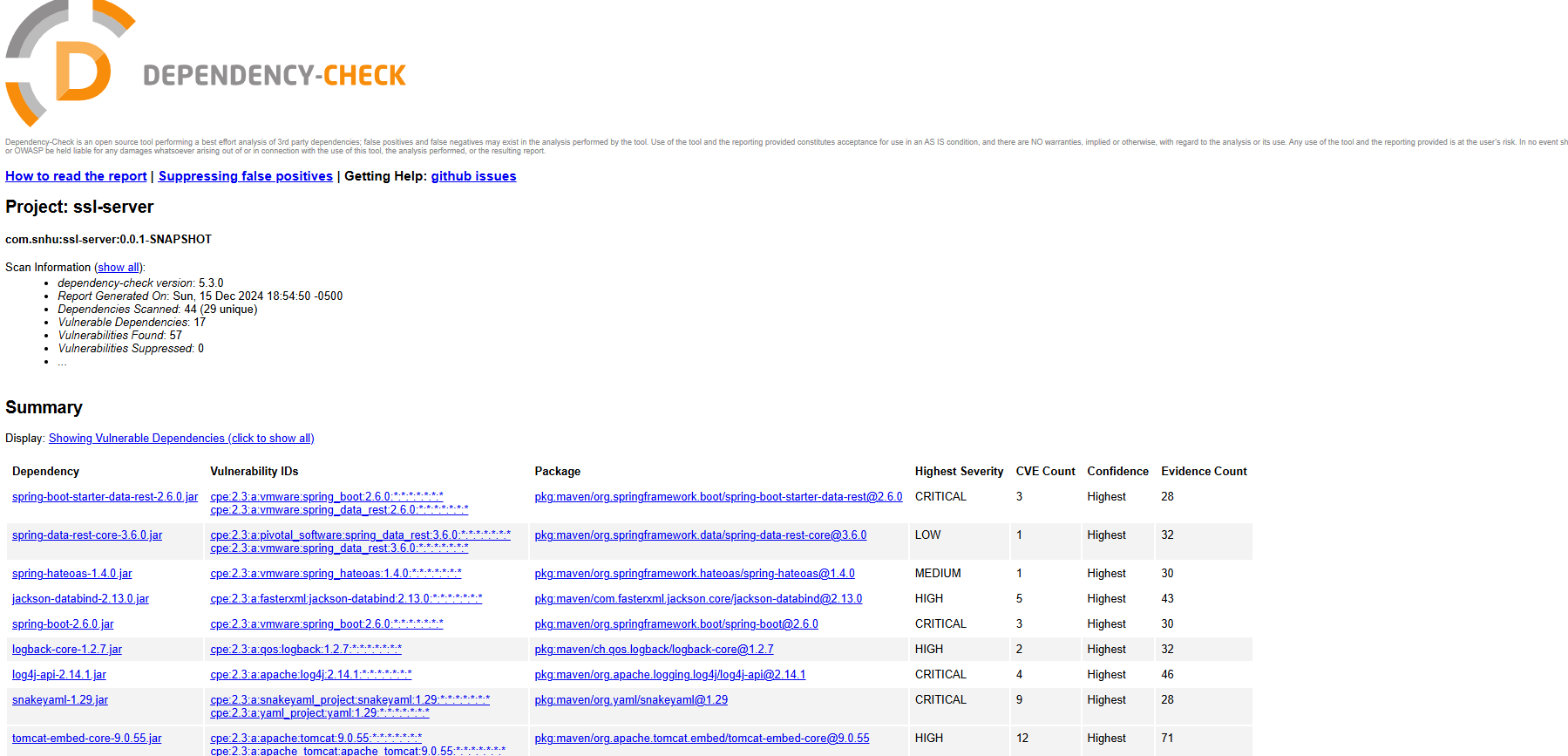
Insert screenshots below of the refactored code executed without errors and the dependency-check

report.



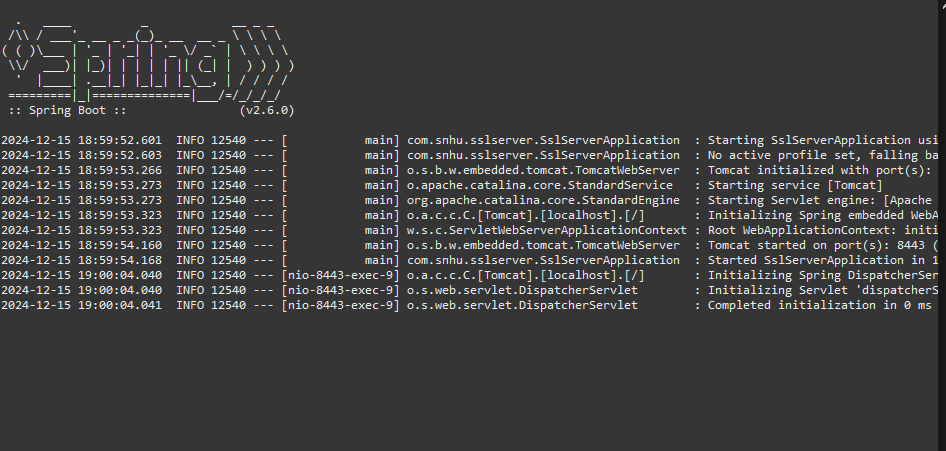


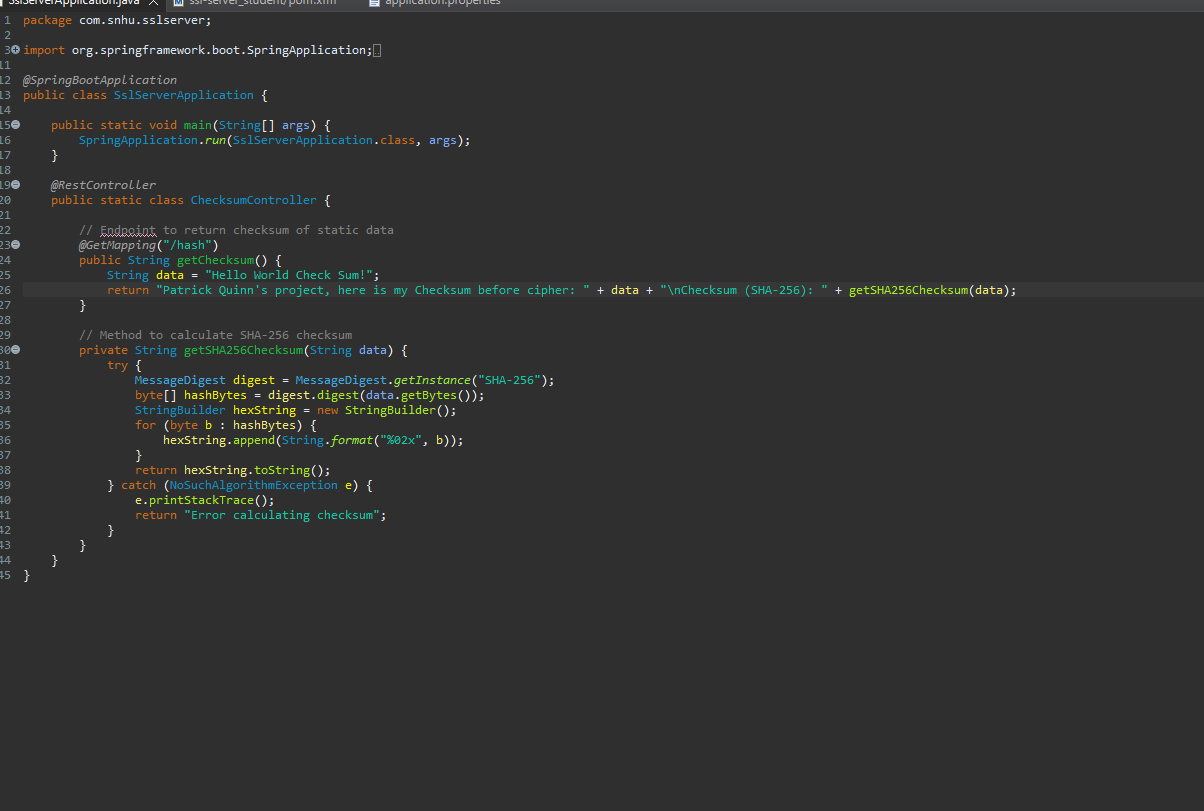
After refactoring, there were less vulnerabilities.



## Functional Testing

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





## Summary

In the process of refactoring the code for the Spring Boot application, the primary focus was on enhancing security by implementing HTTPS encryption, ensuring that sensitive data in transit is protected against eavesdropping and man-in-the-middle attacks. The original code was refactored to include SSL/TLS configuration, specifically by setting up an SSL certificate and key pair to be used by the embedded Tomcat server. This allows the application to serve secure connections over port 8443. Additionally, a checksum API endpoint was implemented to use SHA-256, a secure cryptographic hash function, to validate data integrity without exposing the original data. This refactor addresses key vulnerabilities such as insecure communication channels and the need for data integrity verification.

To comply with security testing protocols, the vulnerability assessment process involved identifying potential risks related to data transmission, data integrity, and encryption. The refactored code directly addresses vulnerabilities in the communication layer by enforcing HTTPS, which mitigates risks like interception and tampering of sensitive information. By utilizing a strong hash function like SHA-256, the application ensures that data integrity can be independently verified, addressing potential risks associated with data manipulation. Further layers of security were added by configuring the server to reject insecure HTTP connections and enforce the use of SSL/TLS certificates, ensuring that only secure communication is possible. These measures follow the principles of defense in depth, where multiple layers of security (encryption, hashing, and secure communication) are implemented to safeguard the system against various attack vectors. The process also involved ensuring that proper error handling is in place, and that key cryptographic elements, such as the private key and certificate, are securely managed. These steps are in line with industry best practices for secure software development, contributing to a more resilient and trustworthy application.

## Industry Standard Best Practices

This approach aligns with industry best practices for developing secure web applications. First, by using HTTPS (SSL/TLS), we ensure that all communication between the client and server is encrypted, protecting sensitive data from potential interception. The use of SHA-256, a widely accepted and secure cryptographic hash function, ensures that the data integrity can be verified without exposing the original data. Additionally, Spring Boot is a modern framework that simplifies application development while following the principles of modularity and maintainability. By providing a clean, RESTful API for the checksum functionality, the application adheres to the best practices of web service design, enabling ease of integration with other systems and scalability for future development.