****

# 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** | **08/16/24** | **Aidan Farhi** |  |

## 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

Aidan Farhi

## Algorithm Cipher

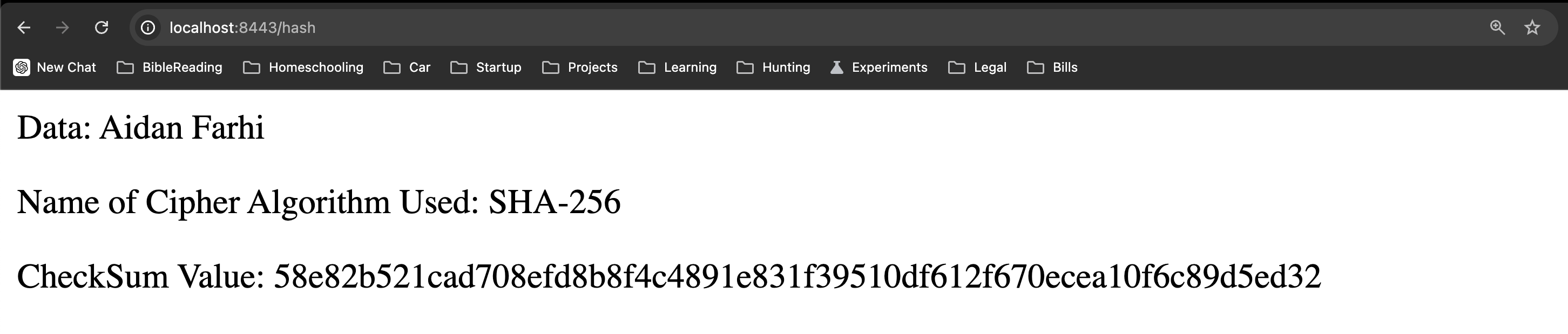
Hashing is the act of passing information into a mathematical function that scrambles the information in a deterministic way. For Artemis Financials' use case of data verification after a transfer, the SHA-256 cryptographic hash function should be used. Before the transfer is initiated, a hash will be calculated on the data. After the transfer is complete, the hash of the data on the receiving end will be compared to the original to ensure they are identical. SHA-256 is a cryptographic hash function that produces a 256-bit (32-byte) hash value from an input of any size. It uses a technique of breaking the data into chunks and shuffling the bits. This process is repeated 64 times before the final digest is produced.[[1]](#footnote-14080)

The Secure Hash Algorithm (SHA) family was developed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST). The original SHA-1 was released in 1993 but was found to be vulnerable to collision attacks, leading to the development of the more secure SHA-2 family, which includes SHA-256. Despite its strengths, the continuous advancement in quantum computing poses a potential threat to SHA-256. Quantum algorithms, like Shor's algorithm, could theoretically break the cryptographic strength of SHA-256, leading to the exploration of post-quantum cryptography as a future safeguard.

## Certificate Generation

## 

## Deploy Cipher



## Secure Communications

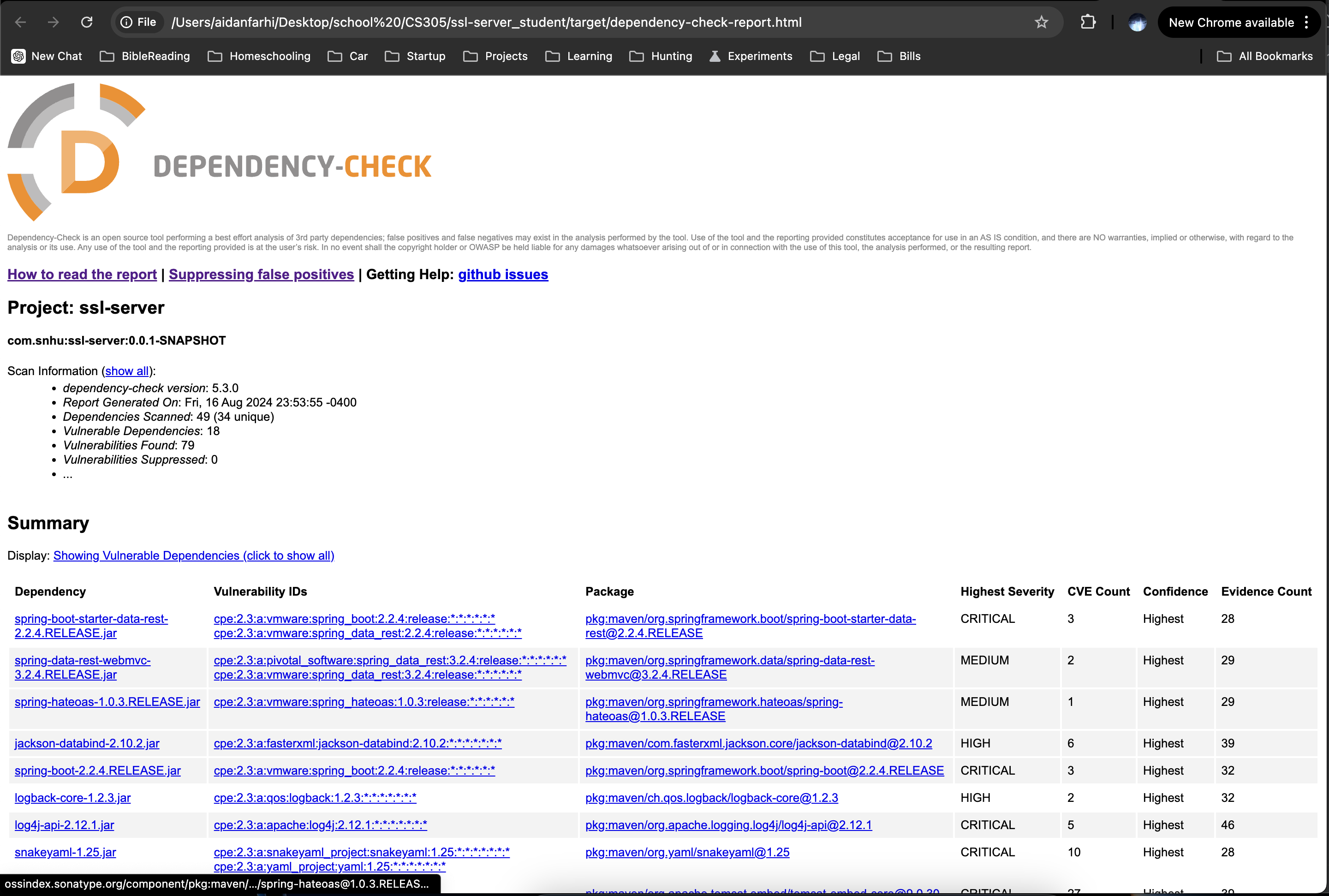
## 

## Secondary Testing

## Screenshot of maven verify being run and a successful build.

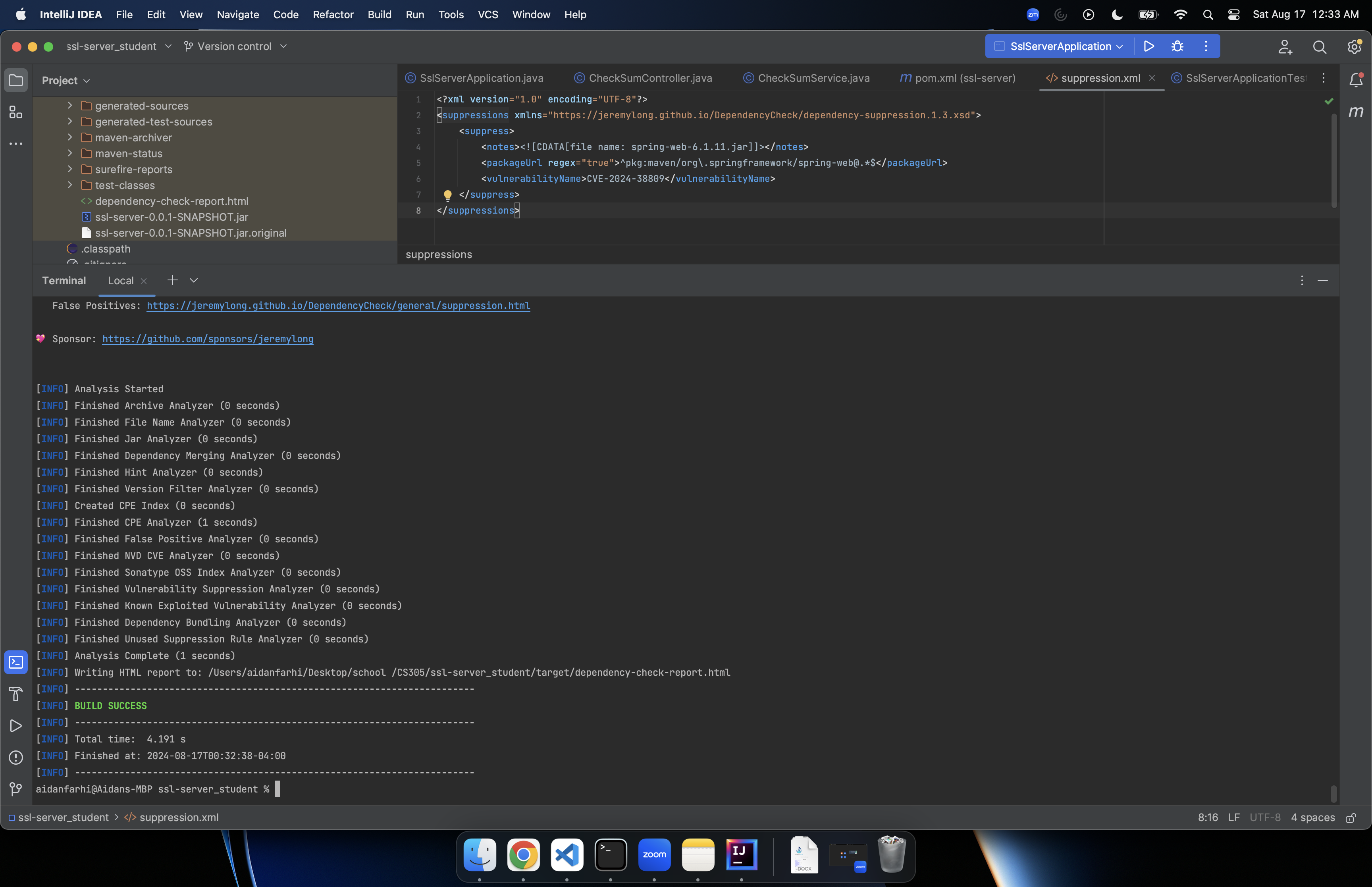
## 

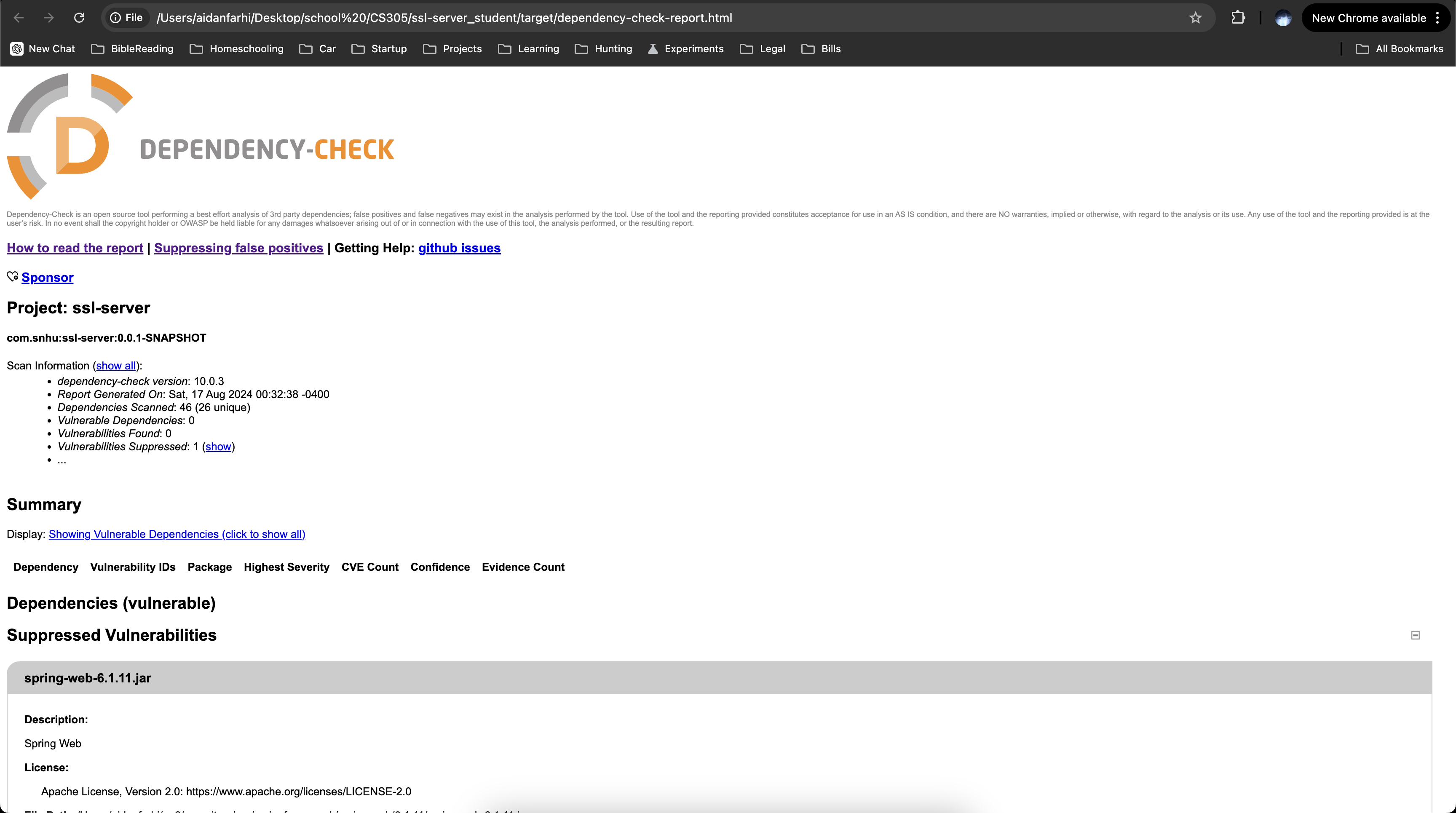
## Screenshot of the Dependency Check report after adding code changes.



## Functional Testing

## Screenshot of Maven verify being run after updating the pom.xml to use the latest spring-boot-starter-parent version. I also suppressed a CVE that was not relevant to the application code.

Screenshot of the Dependency Check report after updating the pom.xml and adding a suppression.



## Summary

The code has been refactored and the dependencies have been updated to meet the requirements and comply with security testing protocols. An endpoint has been added to accept a GET request which displays some data, the cipher algorithm being used on the data, and the check sum value of the data. After running the dependency check on the code, there were several vulnerabilities that needed to be addressed. The main culprit was the outdated version of the spring-boot-starter-parent artifact. This was updated to use the latest version which alleviated the vulnerabilities. After running the report again, there was only one CVE finding which was not relevant to the application code in its current state. The CVE was for applications parsing ETags from “If-Match” or “IF-None-Match" request headers. As the codebase is updated and changed, this suppressed CVE should be reviewed to ensure that the code is still compliant and secure.

**Vulnerability Assessment Process Flow:**

**Input Validation:**

The application is designed to avoid accepting any input data from clients, ensuring security by limiting exposure. The only available endpoint is a GET request to /hash, which serves to validate the checksum functionality.

**APIs:**

The application currently features a single API endpoint: the GET /hash. This endpoint is crafted to adhere to secure API interaction standards.

**Cryptography:**

For cryptographic operations, the application employs the SHA-256 algorithm from the java.security package. This algorithm is widely recognized as secure and is considered an industry standard.

**Client/Server:**

The application is configured to enforce secure communication exclusively via HTTPS using a self-signed SSL certificate. The server operates on port 8443, with SSL enabled. The key alias is selfsigned, and the keystore is of type JKS, securely stored within the application classpath as keystore.jks. The keystore is protected by a password, ensuring that all interactions between the client and server remain encrypted and secure.

**Code Error:**

The application includes comprehensive error handling, ensuring that all potential application errors or exceptions are properly managed.

**Code Quality:**

The application code has undergone thorough review for quality assurance. It follows a modular architecture, with each functional layer encapsulated within its own package. No security issues have been identified.

**Encapsulation:**

All data structures used in the application conform to industry standards and comply with security requirements.

When adding layers of security to applications, I employ an iterative approach that ensures both functionality and security. I begin by implementing the necessary code to meet the application's requirements. Once the code achieves the desired functionality, I focus on refactoring it to enhance modularity and organization, which improves maintainability and reduces potential vulnerabilities. After refactoring, I conduct a thorough dependency check, analyzing the results to identify and address any vulnerabilities. This includes updating dependencies, validating that the code continues to function correctly, and suppressing any false positives that may arise. Following this, I perform a detailed manual review to ensure that the code adheres to secure coding standards. In this particular project, I also generated a self-signed certificate and refactored the application to use SSL, securing all client-server communications. Each of these steps is meticulously executed to ensure that the application remains secure, compliant, and resilient against potential threats.

## Industry Standard Best Practices

In this project I leveraged a comprehensive approach to maintaining the software application's security by adhering to industry-standard best practices. This process includes meticulous updates to dependencies, such as upgrading the outdated spring-boot-starter-parent artifact to its latest version, thereby addressing critical vulnerabilities. Additionally, the application code was refactored to enhance modularity and organization, which not only improves maintainability but also reduces the attack surface by isolating functionality into distinct layers. The introduction of a self-signed SSL certificate and the enforcement of HTTPS communication ensure that all data exchanged between the client and server is encrypted, protecting against potential eavesdropping or man-in-the-middle attacks. The use of the SHA-256 algorithm for cryptography further aligns with industry standards, providing a strong layer of security for data integrity. By conducting thorough dependency checks and manually reviewing the code for compliance with secure coding practices, the report demonstrates a commitment to proactive security measures, ensuring that even suppressed CVEs are monitored for relevance as the code evolves.

Applying industry-standard best practices for secure coding is crucial for any organization’s overall well-being, as it not only protects the software from current threats but also ensures long-term resilience against emerging vulnerabilities. These practices build a strong security foundation, reducing the likelihood of breaches that could compromise sensitive data, damage the company’s reputation, and lead to financial losses. Moreover, by implementing rigorous security protocols and maintaining up-to-date dependencies, the company fosters a culture of security awareness, which is vital for staying ahead in a constantly evolving threat landscape. This proactive stance on security not only instills confidence in clients and stakeholders but also positions the company as a trustworthy entity that prioritizes the safety and integrity of its digital assets. In essence, adhering to these best practices is not just about maintaining compliance; it’s about securing the future of the company in an increasingly digital world.

**References:**

Thakkar, M. (2022, April 22). *SHA-256 algorithm explained by a cyber security consultant.* SectigoStore. https://sectigostore.com/blog/sha-256-algorithm-explained-by-a-cyber-security-consultant/

1. Megha Thakkar, "SHA-256 Algorithm Explained by a Cyber Security Consultant," *SectigoStore*, April 22, 2022, https://sectigostore.com/blog/sha-256-algorithm-explained-by-a-cyber-security-consultant/. [↑](#footnote-ref-14080)