****

# 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** |
| --- | --- | --- | --- |
| **1.1** | **February 24, 2024** | **Ryan Hatch** | **First Deployment:  pom.xml needed to be revised. to be ran via OWASP tools. SSL server configured via attributes.propterties** |

## 

## Client



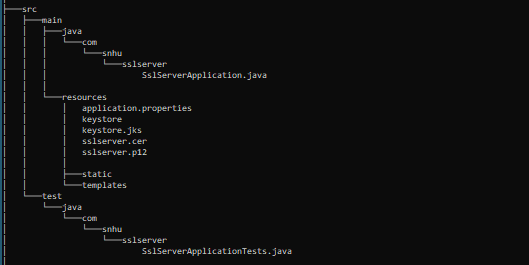
## Developer

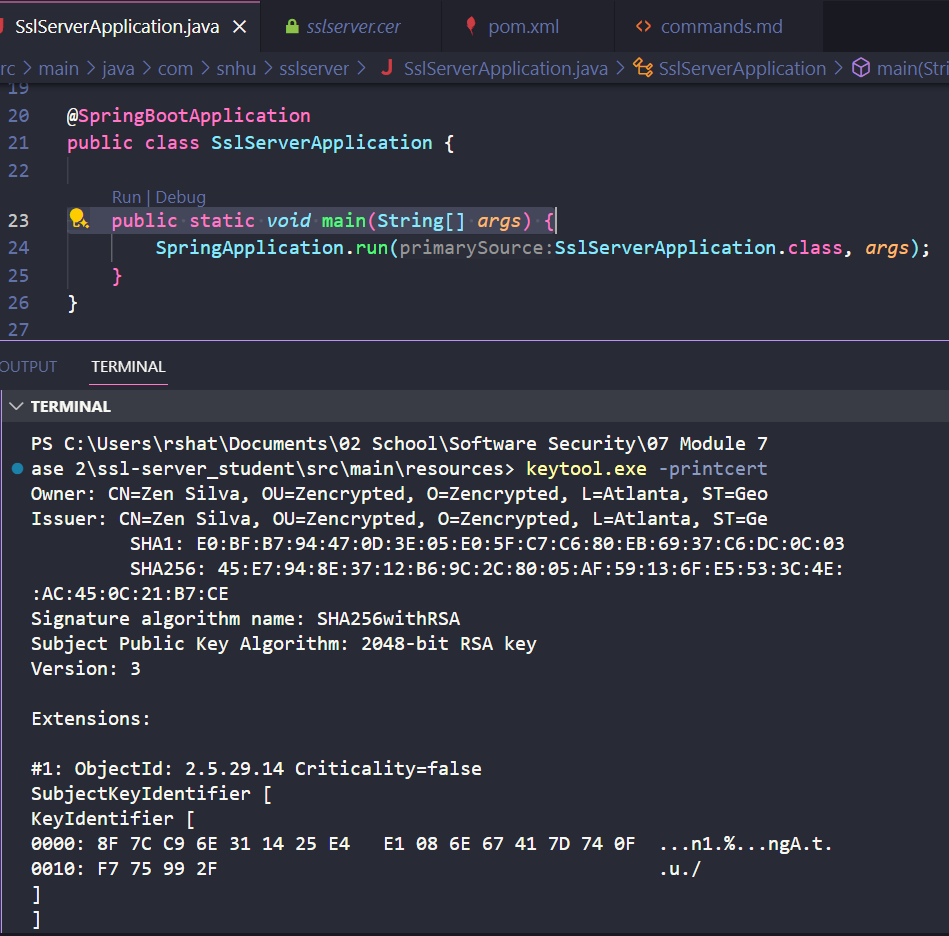
Ryan Hatch

## Algorithm Cipher

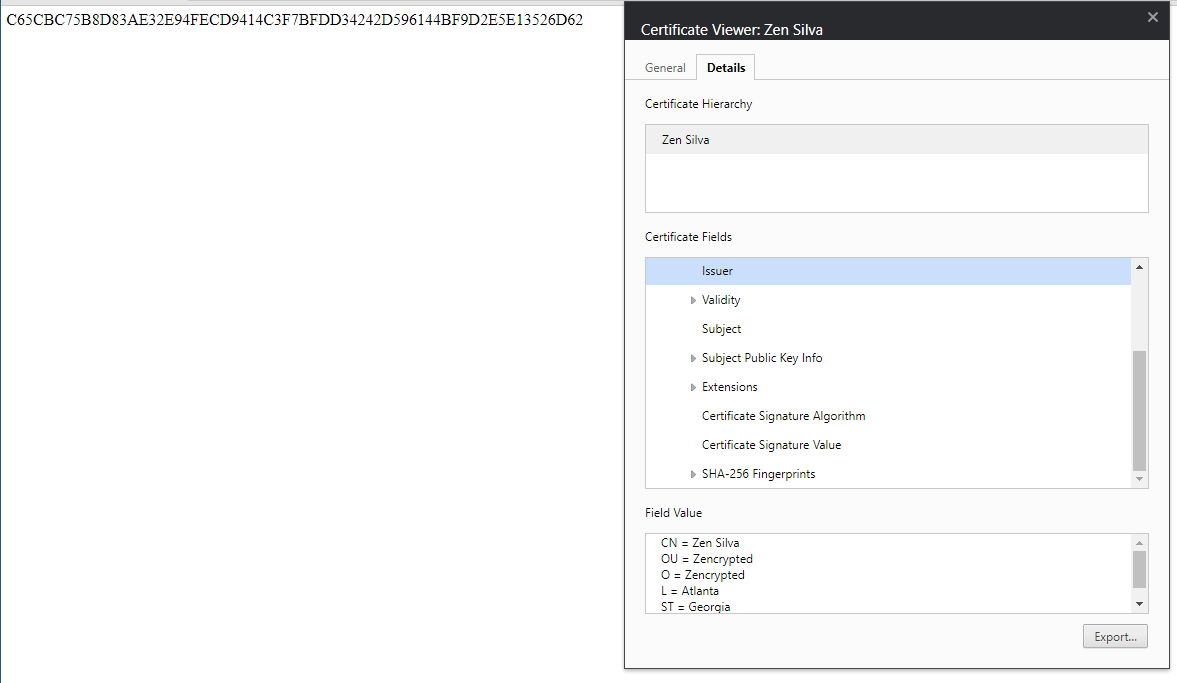
Within the algorithm for the cipher, there were two of the best algorithmic ciphers that have been recommended to be resilient to collision resistance attacks, and has also been selected to deploy within the program. The algorithms that were used to deploy the SSL -+server’s application were SHA-256 and SHA-3. SHA-256 forms part of the SHA-2 family of hashes, with it being in use in all security protocols. SHA-3 is the new standard and, although it is not a fix on SHA-2, it does provide a different approach to hashing, which in turn offers a better margin of security when the implementation of extra security layers and persuasions, for example, being proactive against any collision attacks coming into play.

## Certificate Generation

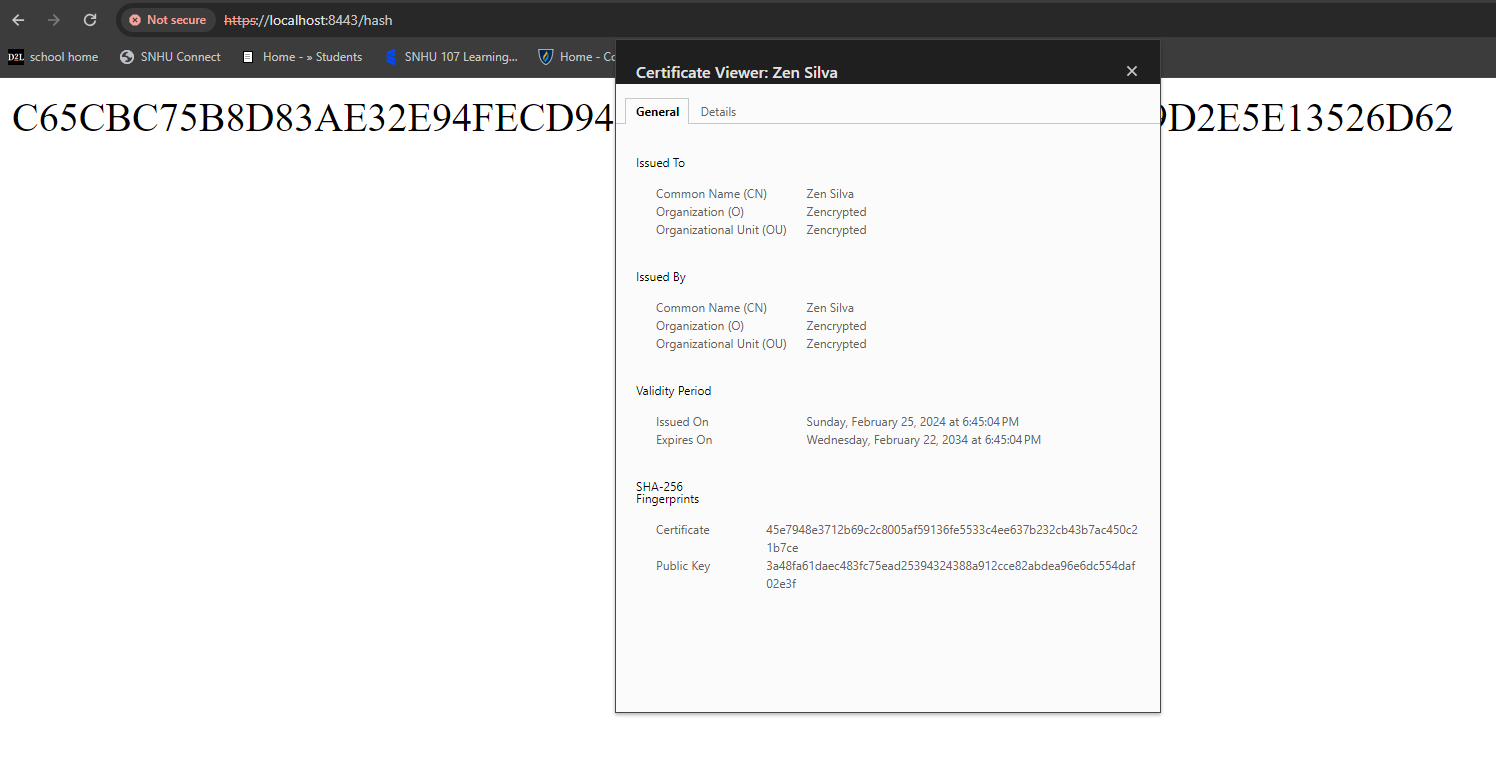




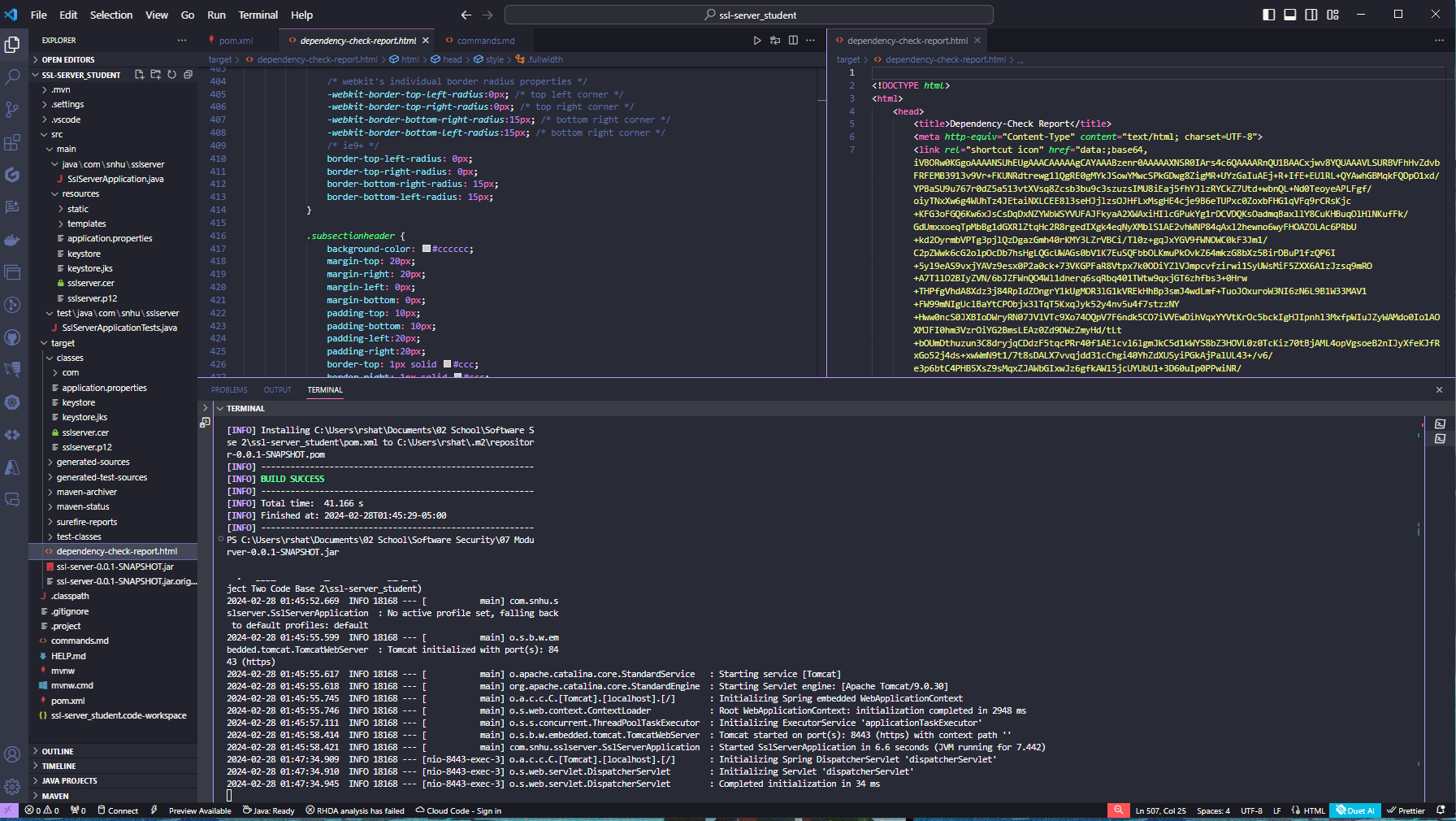
## Deploy Cipher

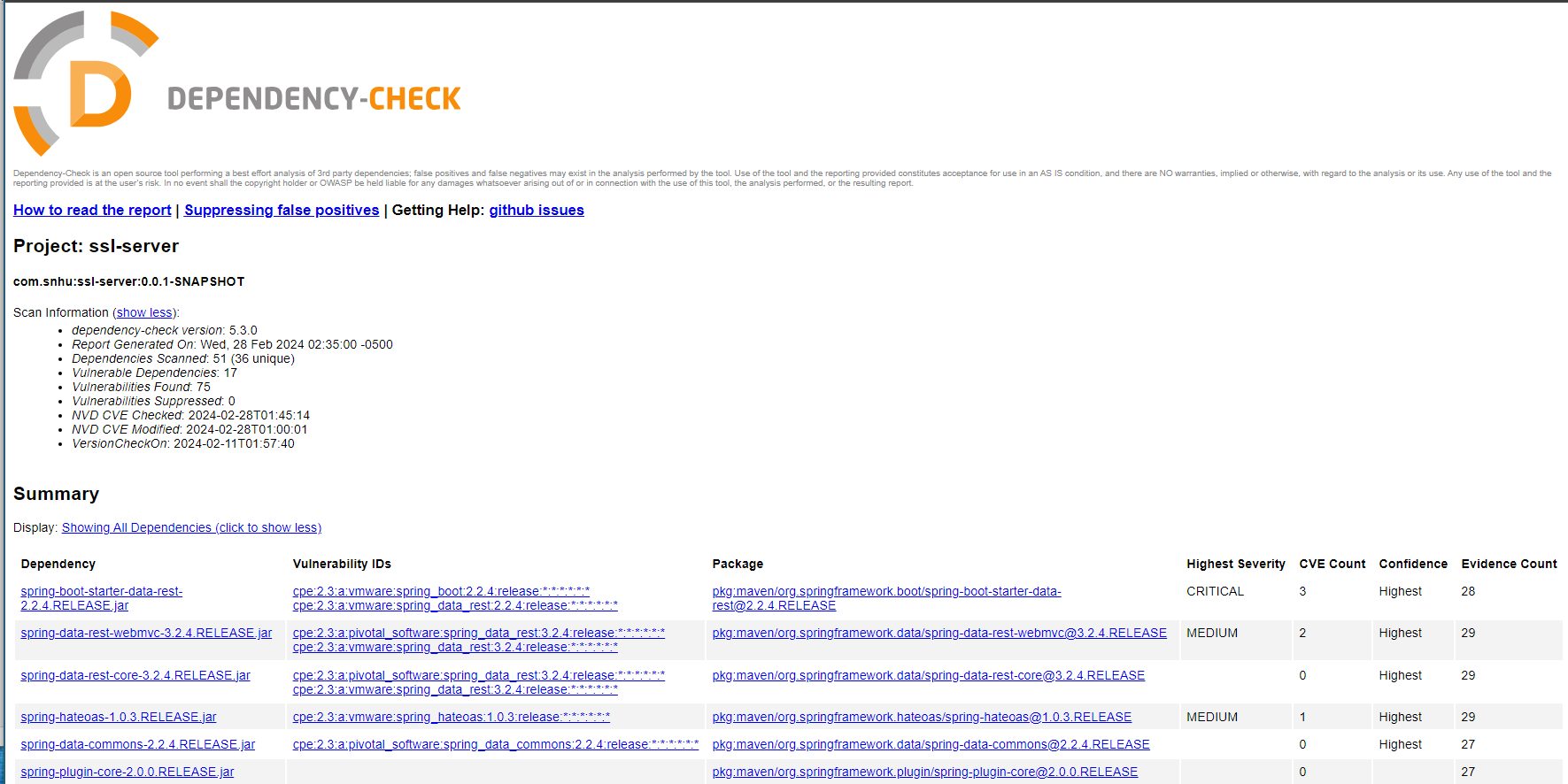


## Secure Communications

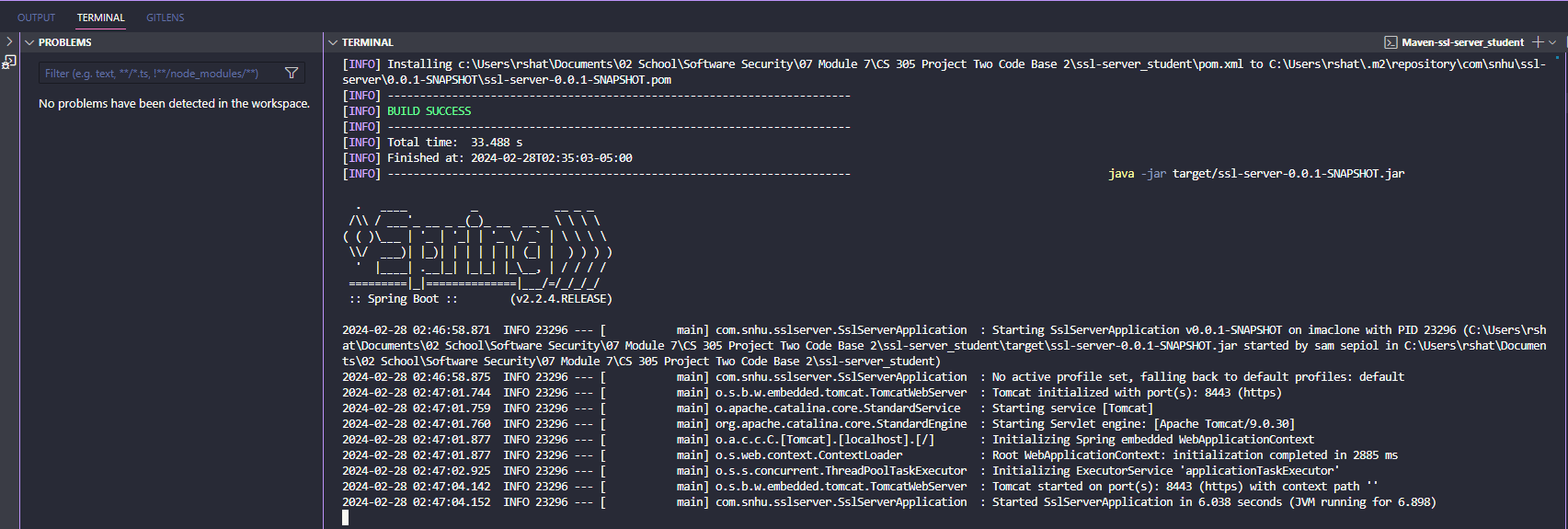


## Secondary Testing





## Functional Testing



## Summary

The refactoring of the SSL Server application was comprehensively security-focused, specifically by implementing SHA-256 hashing for data integrity verification. This cryptographic hashing function ensures that data has not been tampered with while in transit; using a checksum generated via SHA-256, we can ascertain that any alterations to the original data will be discovered and will thus avoid extraneous, unauthorized interference. Updating the code to handle user inputs gracefully while generating a secure checksum was final point in the process, entailing a methodology that reviewed and updated all dependencies to guard against vulnerabilities associated with outdated libraries. The application now adheres to the protocol of security testing practices that checks libraries for known security risks upon their implementation. In creating additional layers of security, the application was also set up to work over HTTPS, which entailed using a self-signed certificate to establish a secure channel; this protects the communication between the client and the server from being intercepted. All the application’s properties and sensitive configurations were also given secure treatment.

## Industry Standard Best Practices

A number of industry-standard best practices were applied to maintain the application’s security, like the principle of least privilege, which ensures that an application is running with only the permissions it needs, and input validation to prevent exploits like injection attacks, as well the implementing of secure coding practices that thwart common vulnerabilities like cross-site scripting (XSS) and cross-site request forgery (CSRF). In addition, implementations of dependency management tools that keep libraries up-to-date and scan them for vulnerabilities were continued; this is key in flagging security holes before they become a problem, and opposes the outdated, reactive model of using libraries as long it is possible, until they absolutely need to be updated due to a breach or bug. Code reviews and static analysis tools were implemented to seek out and remedy security issues before deployment. It can’t be overstated how much applying these industry-standard best practices to secure coding protects the company from data breaches and potential financial loss, as well as preserving the company's reputation and the trust it has with its clients. An application that embodies these adjustments is imperative on all levels. Not only does it safeguard against potentially massive financial losses by virtue of avertable breaches; it propagates the idea that you can and will secure private data in a responsible and proactive manner. Demonstrating that commitment to a client’s financial security is incredibly valuable to a company, beyond the immediate financial release of a commercial application. It’s also invaluable to the reputational health of a business to keep the client’s data secured by making sure that the applications security hardening is crucial and imperative for success.