

# 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** | **8/16/25** | **Christopher Gauthier** |  |

## 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.

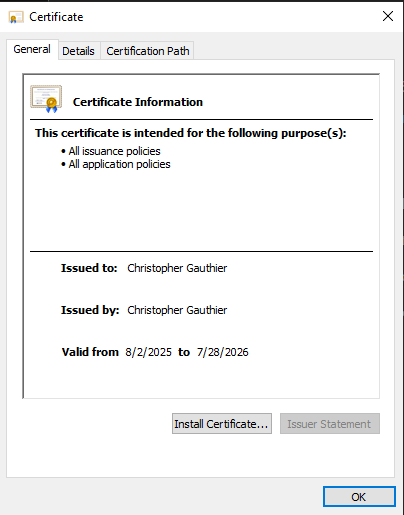
## Developer

Christopher Gauthier

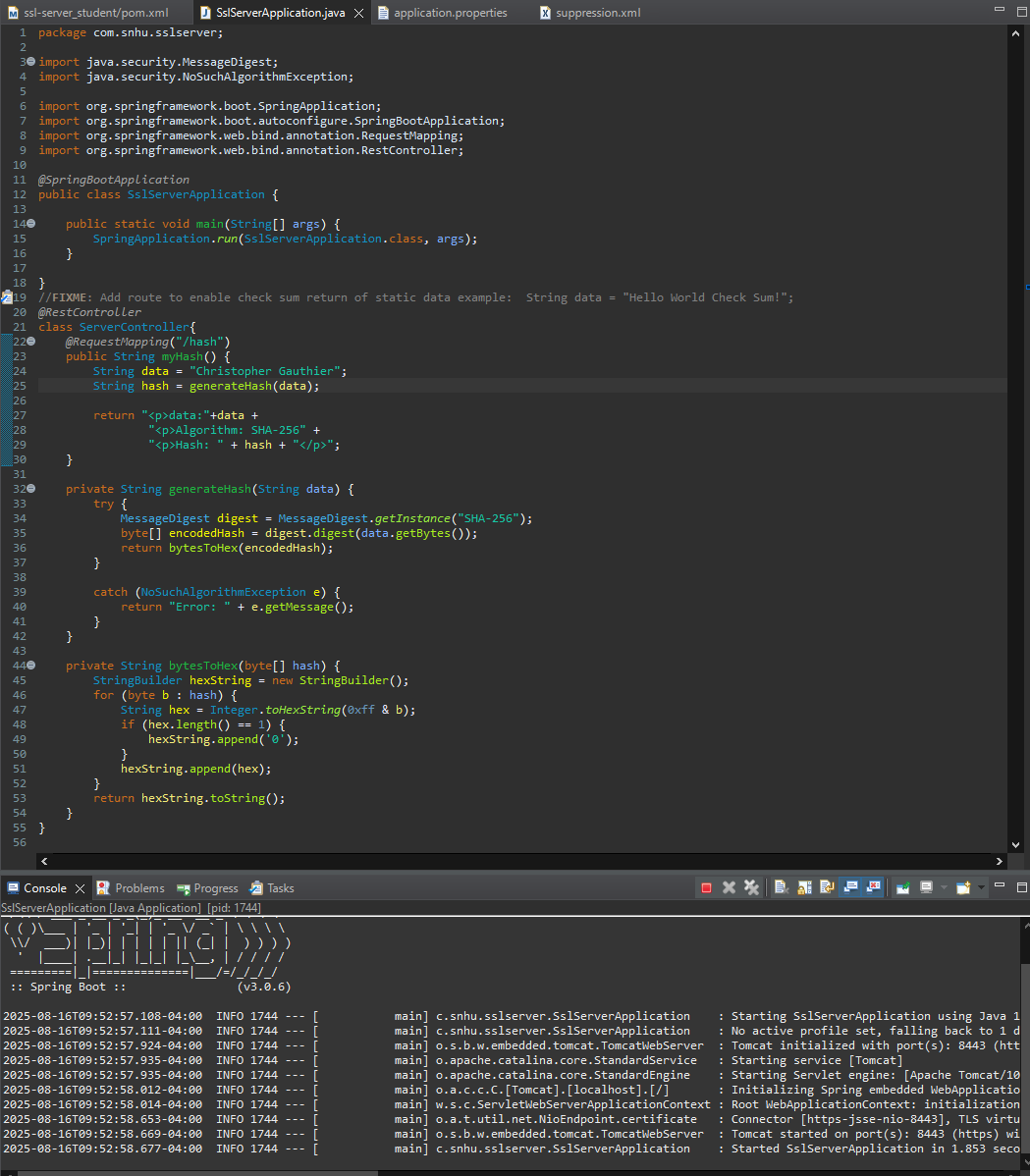
## Algorithm Cipher

Based on the Java Security Standard Algorithms, I recommend using the SHA-256 hash function. SHA-256 is an industry standard hashing algorithm, that generates a 256-bit fixed-length hash based on any size input. SHA-256 is designed to avoid collisions, which is critical for ensuring tougher security.

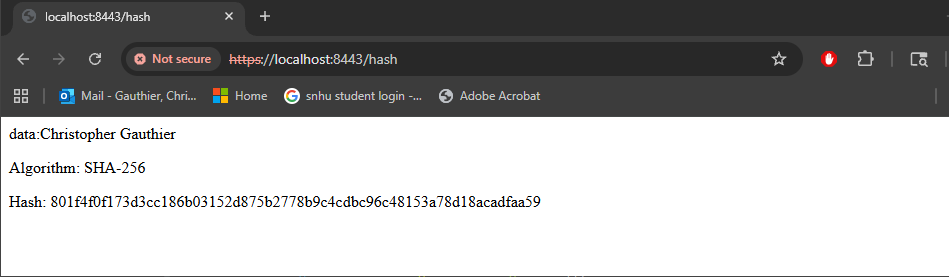
## Certificate Generation

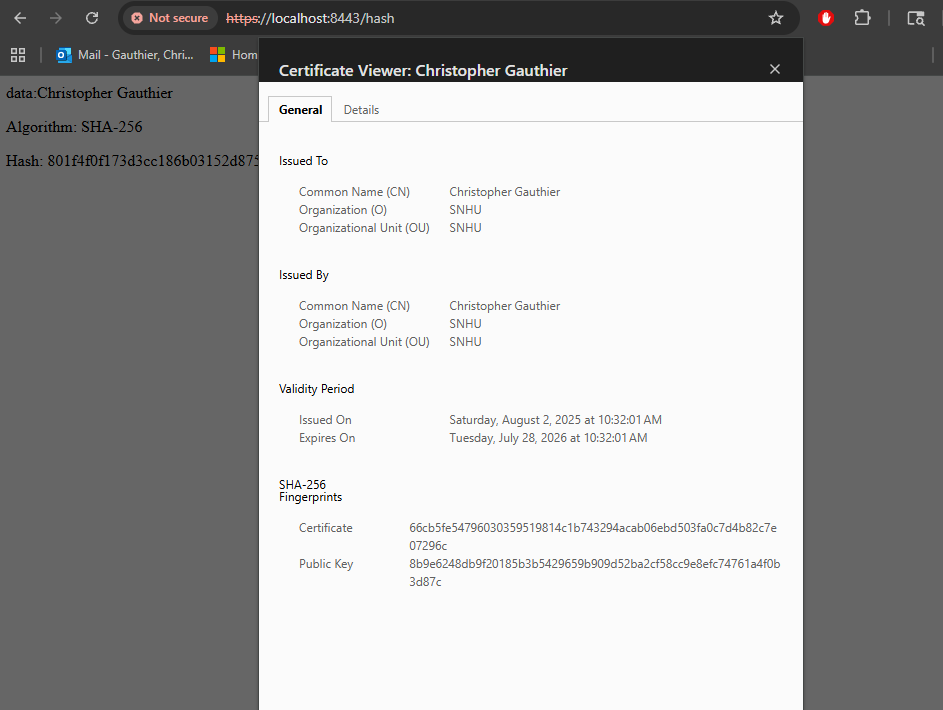


## Deploy Cipher



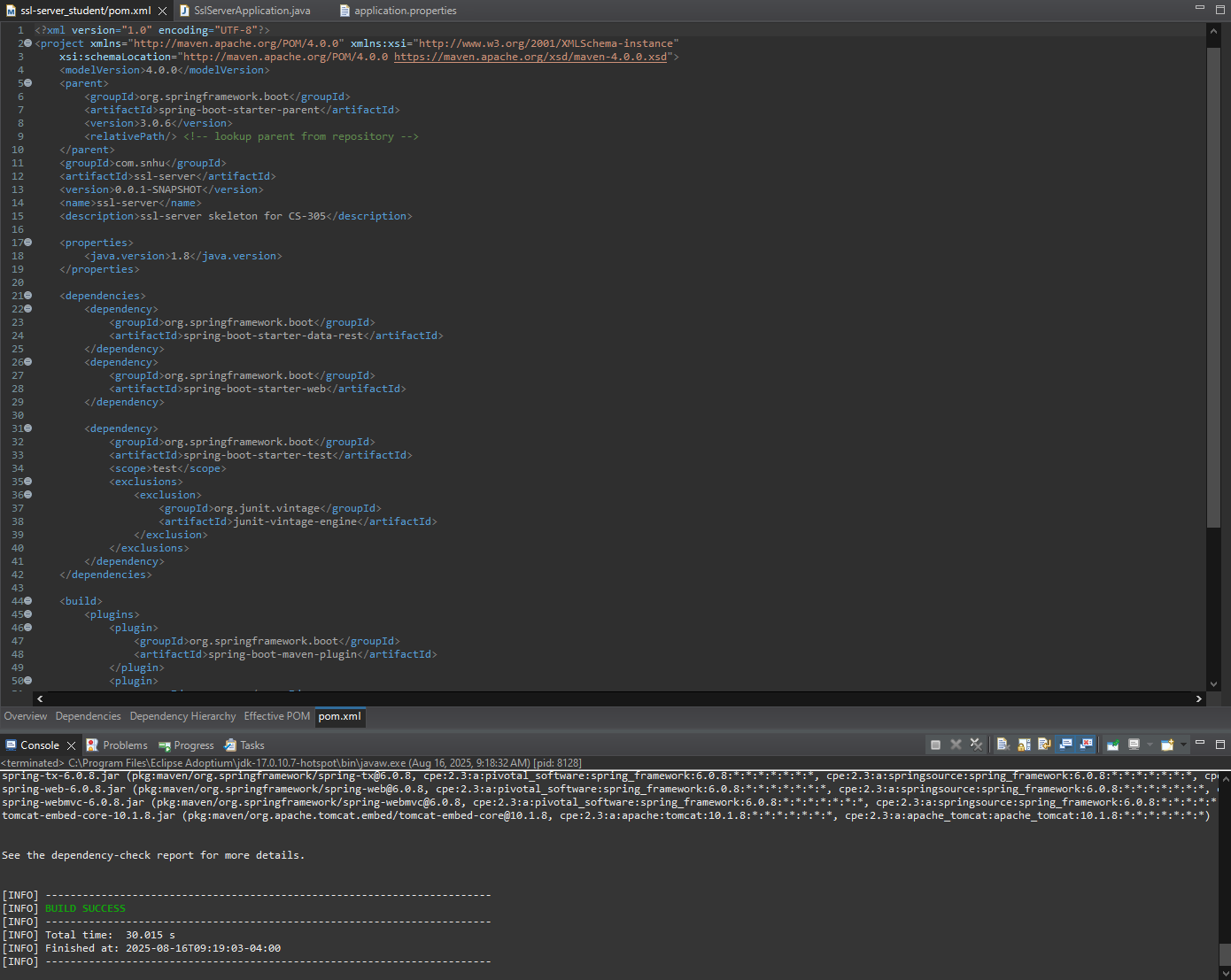
## Secure Communications

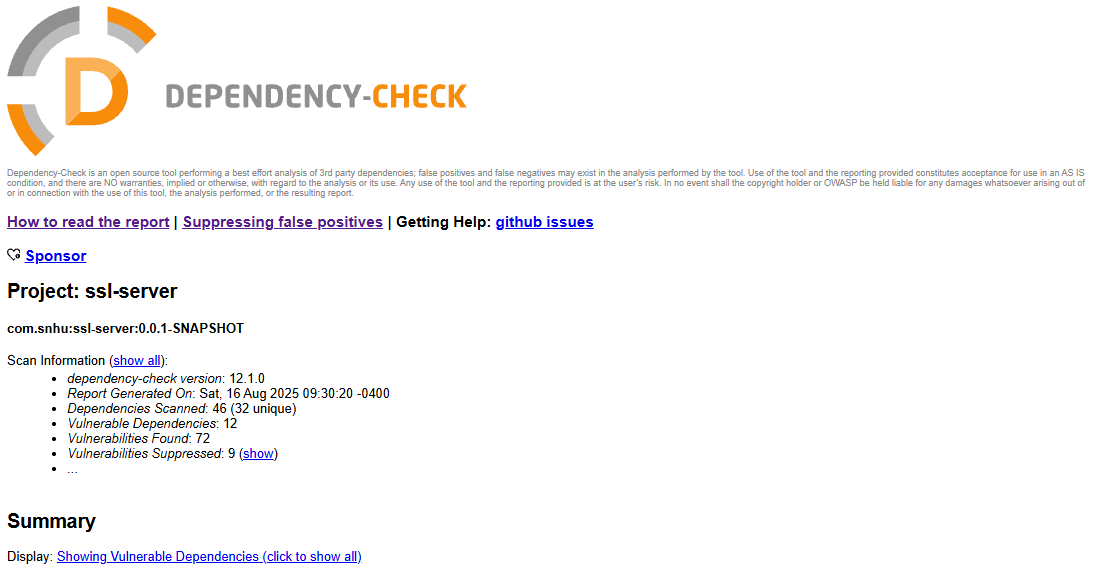




## Secondary Testing

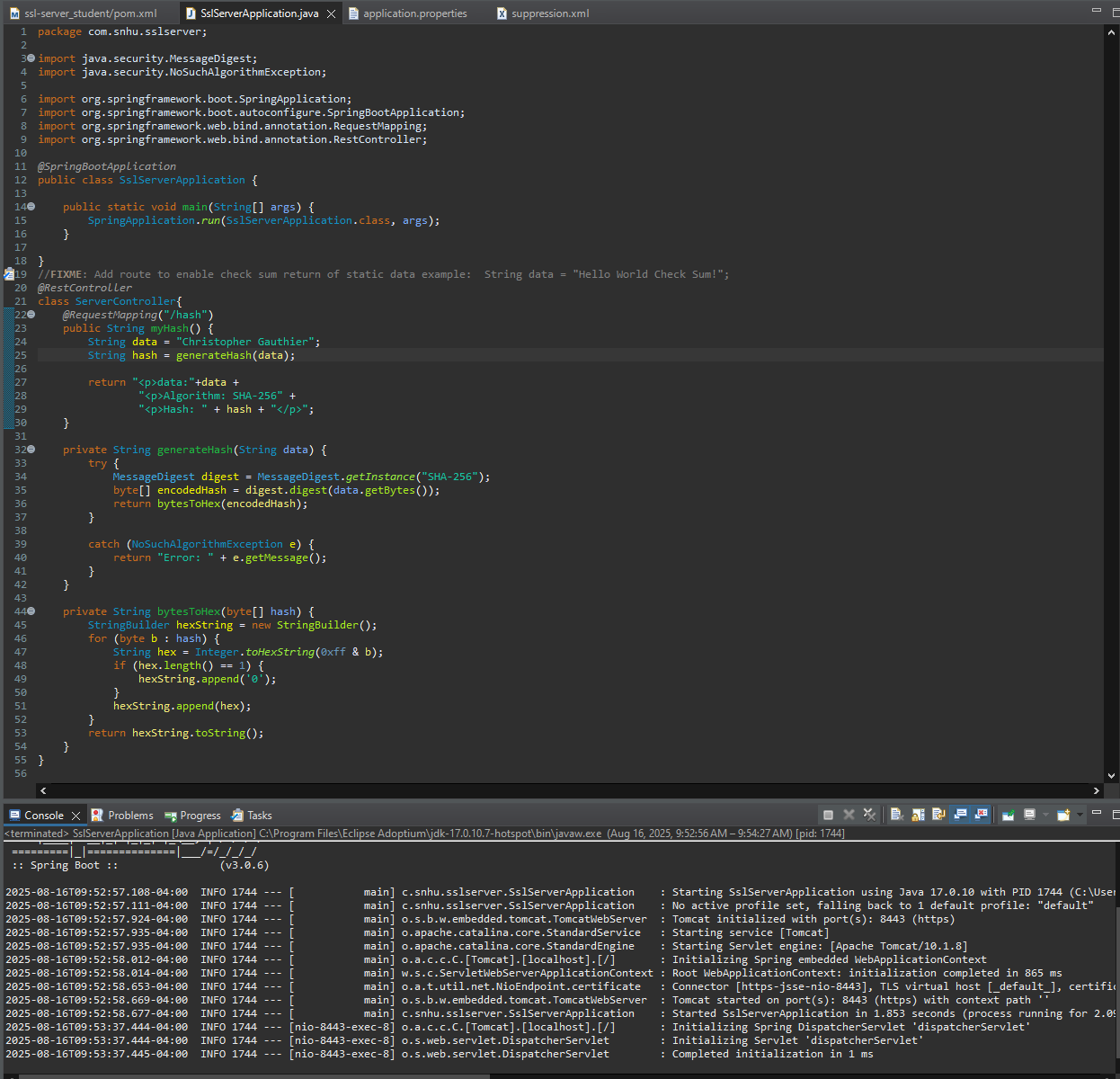
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

Working through the vulnerability assessment process flow, I secured APIs by adding trusted certificates, added cryptography by implementing a hash algorithm, and ensured best practices when reviewing and refactoring the code base, to include error handling and catching.

In the refactored code base, I implemented a hashing algorithm to protect string data and a functional example of how the hashed string connects with the Spring framework to communicate with an HTTPS port. By updating the program properties, I also include an SSL key and certificate to trust access to the secure port. Finally, in the dependency check, I identified multiple false-positives and suppressed each CVE to ensure going forward, any new dependencies or vulnerabilities are easily identifiable.

## Industry Standard Best Practices

Industry best practices applied in this code review and implementation include using SHA-256 hashing, secure key management, input validation and vulnerability assessments after refactoring. The SHA-256 encryption algorithm ensures strong brute-force resilience due to its output size and collision resistance. Secure key management is the standard for the principle of least privilege to ensure only authorized and vetted access is allowed. Finally, using the OWASP dependency check ensures that developers are aware of known vulnerabilities and threats, which allows them to mitigate or avoid known exploits.

Best practices are by definition, the most effective way of safeguarding sensitive data and building a resilient application. Layering security controls allows for sensitive data to be protected against known attacks. Additionally, by following industry best practices, most processes are well documented, which allows for easier troubleshooting, faster expansion of services and features, and flexible development options. A qualified developer can quickly be onboard and begin contributing without needing to learn a completely different development philosophy or practice.

**References:**

Kotowicz, J., & Berg, S. (2014). *Iron-clad Java: Building secure Web applications*. McGraw Hill Computing. <https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/>

Long, J. (n.d.). *Dependency-check Maven plugin*. OWASP. <https://jeremylong.github.io/DependencyCheck/dependency-check-maven/index.html>

Oracle. (2017). *Java platform, standard edition security developer’s guide: Standard algorithm names*. <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names>

SSL Shopper. (n.d.). *How to create a self-signed certificate using Java Keytool*. <https://www.sslshopper.com/article-how-to-create-a-self-signed-certificate-using-java-keytool.html>