****

# 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** | **[Date]** | **[Your Name]** |  |

## 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

Christopher Merced

## Algorithm Cipher

I recommend the use of SHA-256 as the encryption algorithm cipher. SHA-256 is a widely recognized cryptographic hash function that produces a fixed-size 256-bit hash value. This makes it a robust choice for ensuring data integrity, as even the slightest change in the input will result in a significantly different hash. SHA-256 is part of the SHA-2 family, which is considered secure and is widely adopted in various security protocols, including TLS and digital signatures. The use of SHA-256 allows for effective verification of files and data, helping to protect against data tampering and ensuring that the information remains unaltered during storage or transmission.

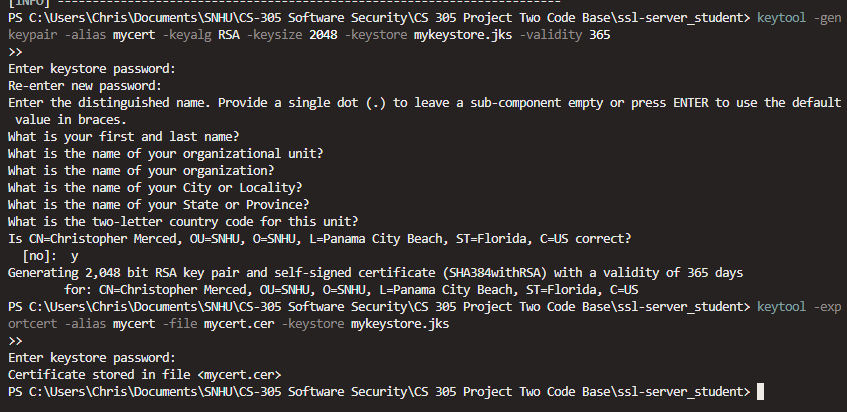
When discussing hash functions and bit levels, it is essential to note that SHA-256 provides a 256-bit output, making it resistant to brute-force attacks and collisions. Its design incorporates various mathematical properties that enhance its security, making it a reliable choice for cryptographic applications. Furthermore, SHA-256 utilizes random numbers primarily in its key generation processes, particularly when combined with symmetric encryption algorithms.

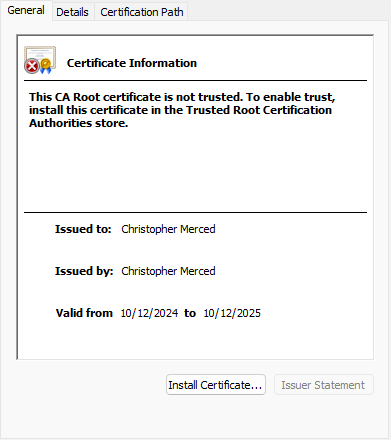
Random numbers play a crucial role in cryptography, particularly in the generation of keys used in both symmetric and asymmetric encryption. In symmetric encryption, where the same key is used for both encryption and decryption, random numbers ensure that the keys are unique and unpredictable, thereby preventing attacks that exploit predictable keys. Conversely, asymmetric encryption relies on a pair of keys: a public key, which can be shared openly for encrypting data, and a private key, which is kept secret for decryption. The generation of these key pairs also involves random numbers to ensure uniqueness and security. The effective use of random numbers is vital for enhancing the strength of cryptographic systems, making it more difficult for attackers to compromise the keys and, consequently, the encrypted data.

Data Encryption Standard (DES) was introduced in the 1970s although it became vulnerable as computational power increased, leading to the adoption of the Advanced Encryption Standard (AES) in 2001, which is widely regarded as the standard for secure encryption today. Currently, encryption algorithms are essential for securing data in various applications, including online banking and secure communications. With the potential rise of quantum computing, ongoing research focuses on developing post-quantum cryptography to ensure data protection against future threats.

## Certificate Generation

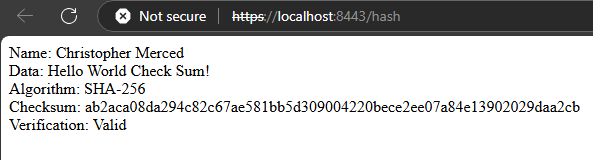
Insert a screenshot below of the CER file.





## 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.

A screenshot of a computer

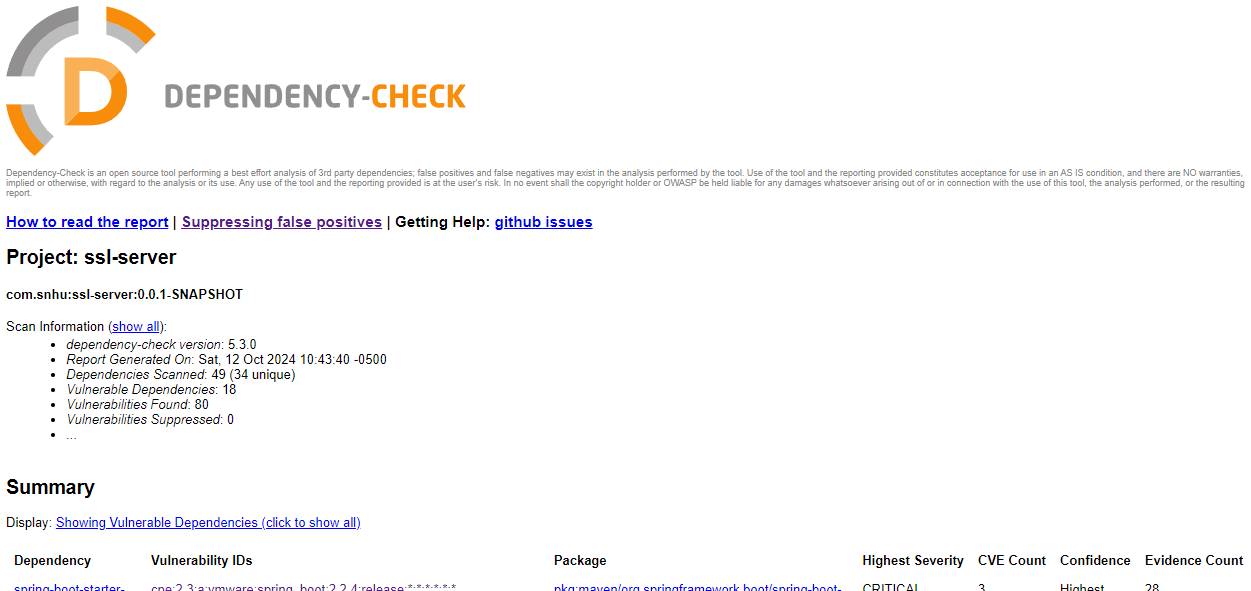
Description automatically generated

## Secondary Testing

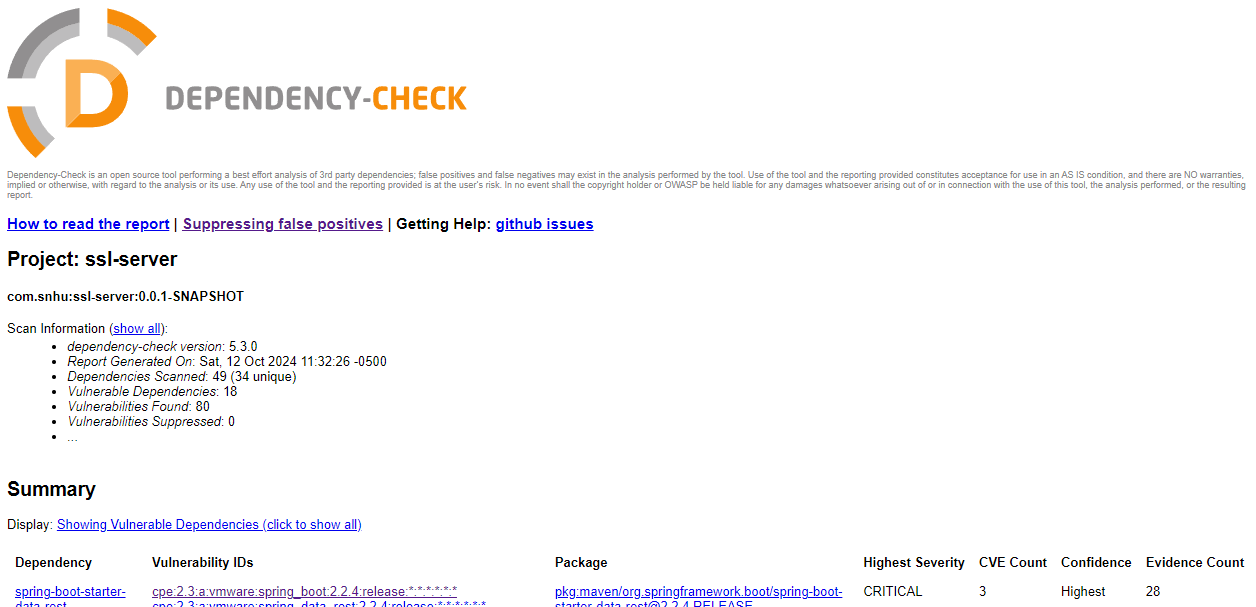
Insert screenshots below of the refactored code executed without errors and the dependency-check report.



Before:



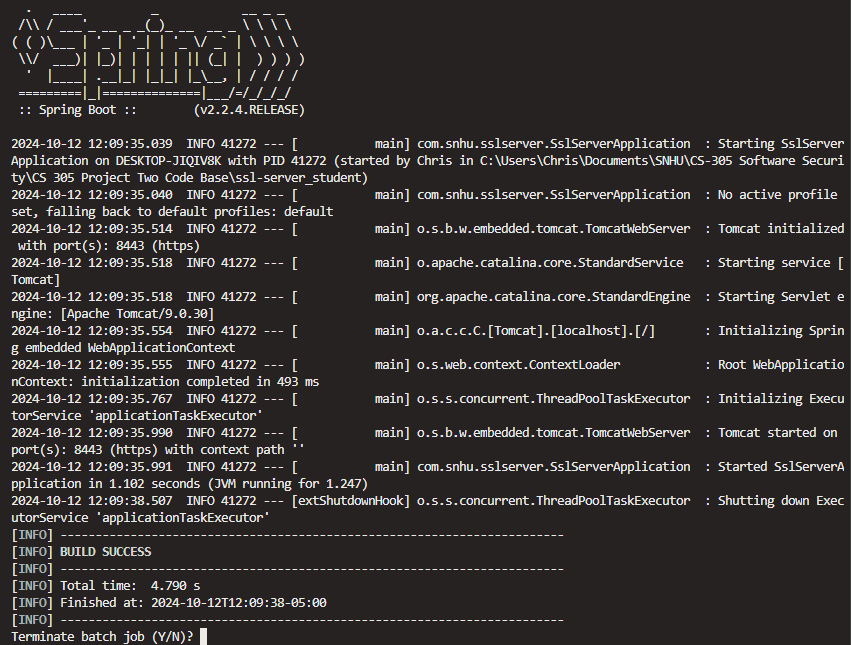
After:



## Functional Testing

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

A screenshot of a computer program

Description automatically generated

## Summary

The code has been refactored to enhance security and comply with security testing protocols, addressing key areas highlighted in the vulnerability assessment process flow diagram. The refactoring primarily focused on ensuring the integrity of data through the implementation of cryptography like SHA-256 for hashing, which allows for verification of the string "Hello World Check Sum!" and ensures that any alterations to the data can be detected. This measure is crucial for maintaining data integrity within the application.

The application has also been configured to operate over HTTPS to support secure client/server interactions, with the appropriate settings established in the application.properties file. This setup employs a self-signed certificate to secure data in transit, preventing interception and ensuring that sensitive information is transmitted securely. Forms of validation have also been introduced through error handling to ensure clarity in application operations.

By ensuring that sensitive data is securely handled and transmitted, the refactored code addresses critical areas identified in the vulnerability assessment, particularly in the realm of data integrity and secure communications. The emphasis on maintaining a clean codebase and applying secure coding practices lays a solid foundation for future development, ensuring that the application can be expanded securely.

## Industry Standard Best Practices

In applying industry standard best practices for secure coding, the focus has been on ensuring that the application maintains existing security while mitigating potential vulnerabilities. Best practices from frameworks such as OWASP were utilized to ensure secure coding principles were followed, including the proper handling of data hashing and secure transmission of information over HTTPS. The implementation of SHA-256 allows for effective integrity checks without compromising the security of the application.

Furthermore, maintaining existing security involves regular code reviews and adherence to secure coding guidelines, which ensure that the application remains resilient against known threats. The value of applying these industry standard best practices extends to enhancing the company’s overall well-being by fostering user trust and safeguarding sensitive data. Secure coding practices not only protect the organization from potential breaches but also help ensure compliance with legal and regulatory requirements related to data protection, thereby avoiding potential liabilities and contributing to the company’s reputation for delivering secure software solutions.