****

# 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 5](#_Toc102040759)

[3. Deploy Cipher 6](#_Toc102040760)

[4. Secure Communications 6](#_Toc102040761)

[5. Secondary Testing 7](#_Toc102040762)

[6. Functional Testing 8](#_Toc102040763)

[7. Summary 9](#_Toc102040764)

[8. Industry Standard Best Practices 10](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/20/2024** | **Omer Cengiz** |  |

## Client



## Developer

Omer Cengiz

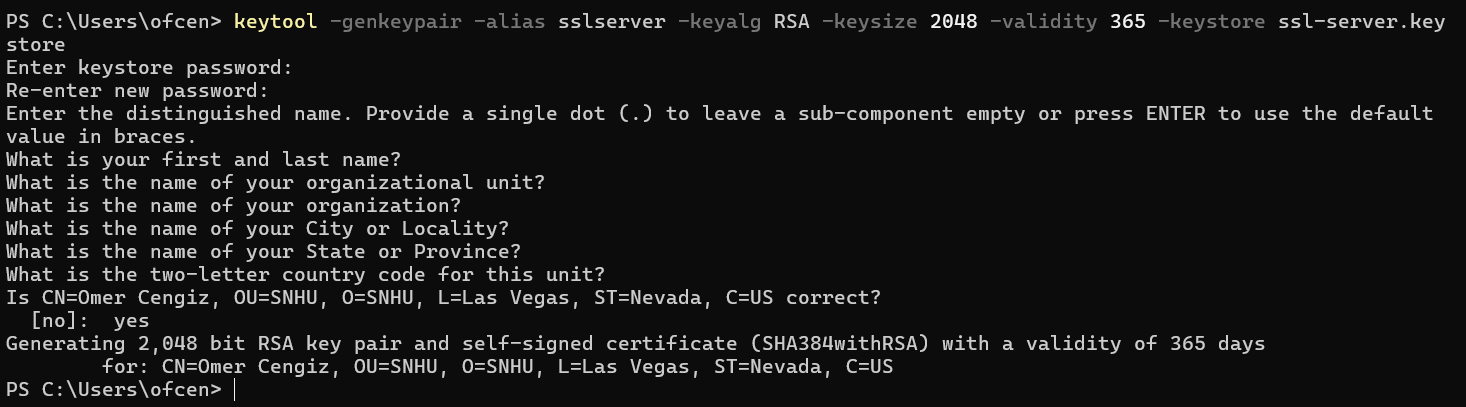
## Algorithm Cipher

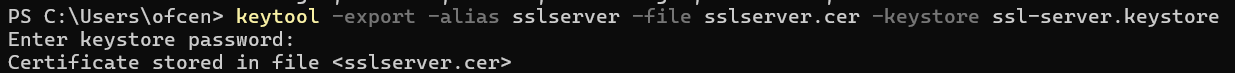
I recommend using AES (Advanced Encryption Standard), specifically AES-256, as the encryption algorithm. AES is a symmetric encryption method that is widely trusted for its security and efficiency. It encrypts data in blocks of 128 bits and supports key sizes of 128, 192, or 256 bits. AES-256 is the most secure option because of its longer key length, making it very difficult to break with brute force attacks. AES often works alongside hash functions like SHA-256, which help verify data integrity by generating a fixed size summary of the data. AES-256 uses a key length of 256 bits, providing an extremely large number of possible key combinations.

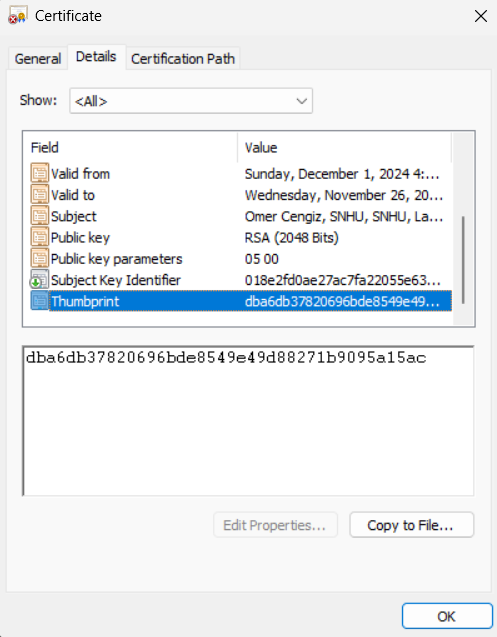
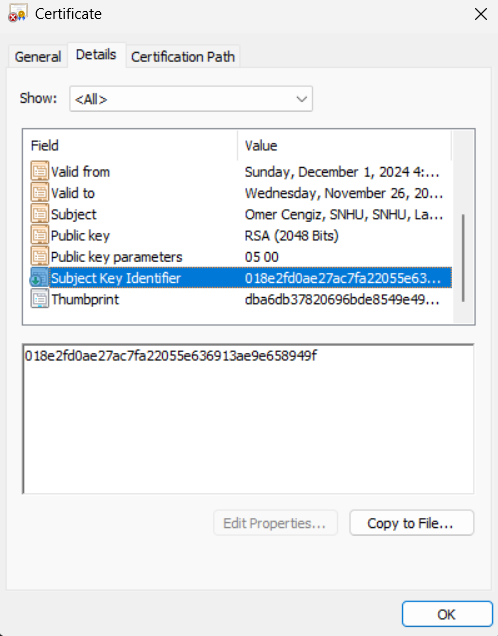
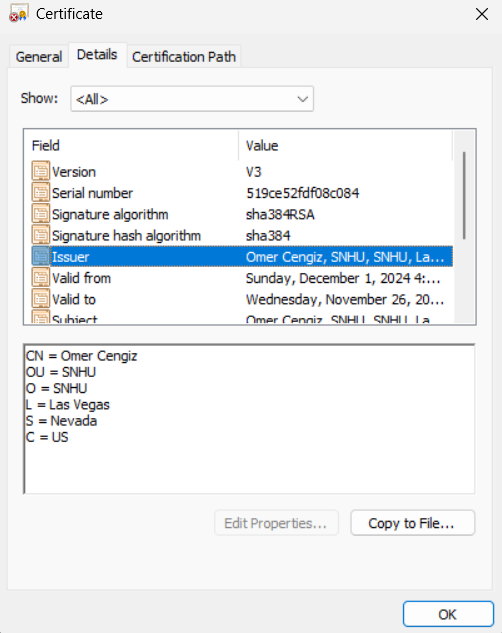
AES encryption uses random numbers to generate initialization vectors, which ensure that even the same data produces different encrypted outputs. It is a symmetric algorithm, meaning the same key is used for both encryption and decryption, so the key must be shared securely between parties. In some cases AES is combined with asymmetric algorithms like RSA to secure the key exchange while AES handles the data encryption.

AES was introduced to replace older algorithms like DES, which became too weak as computers advanced. Since its adoption in 2001, AES has been the standard for encryption and is widely used in secure web communications, data protection, and file encryption. While research is ongoing to address future threats like quantum computing, AES-256 is still considered one of the most secure encryption options available today.

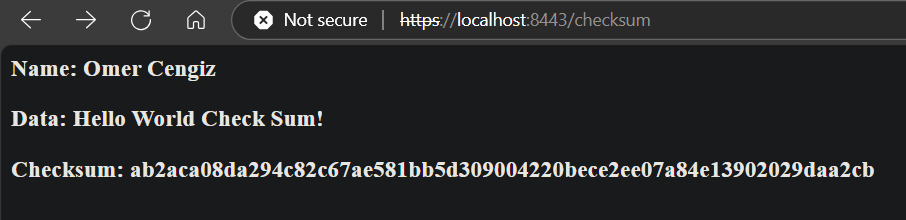
## Certificate Generation



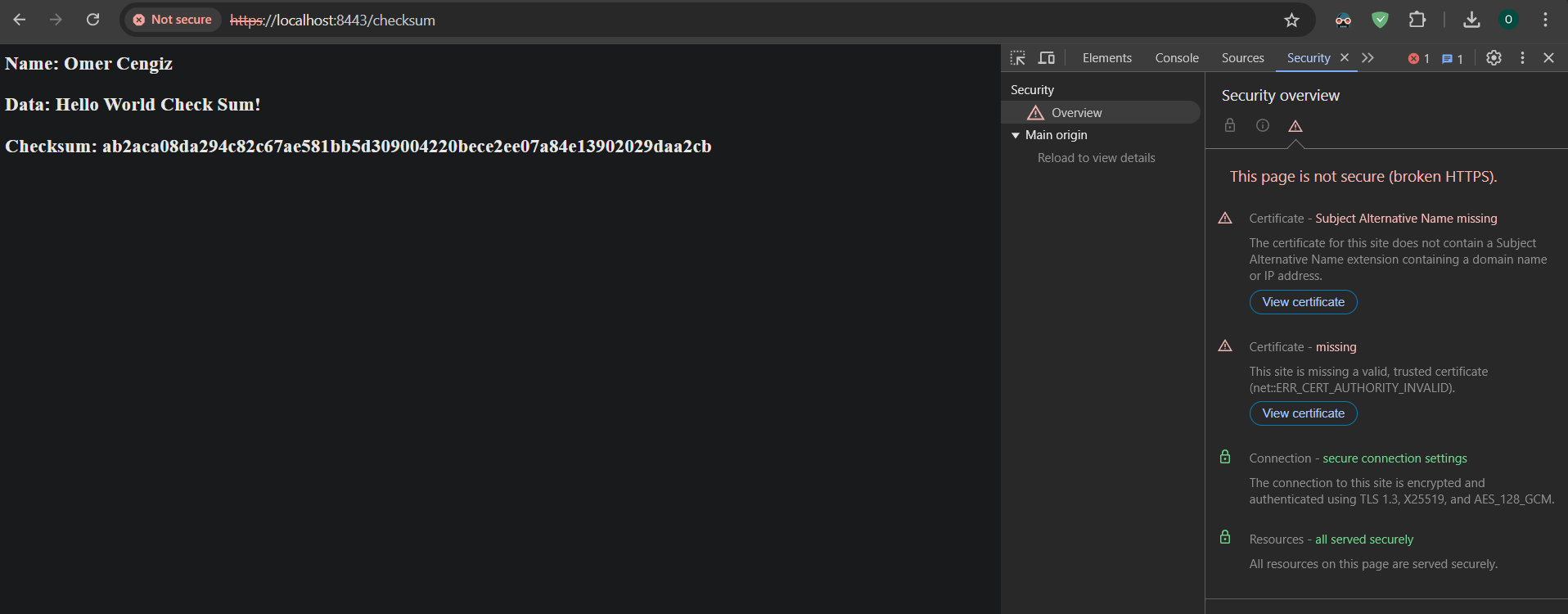




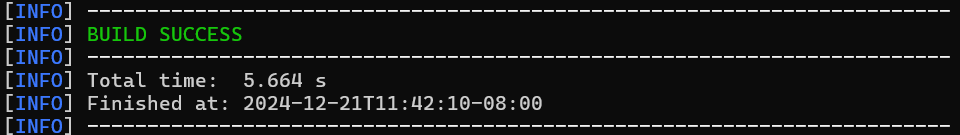
## Deploy Cipher

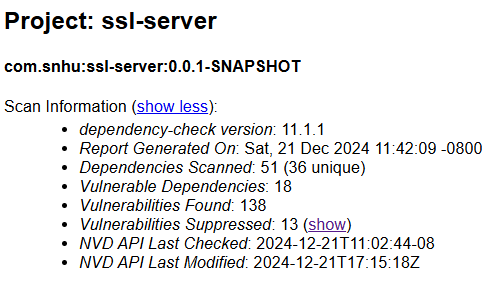


## Secure Communications

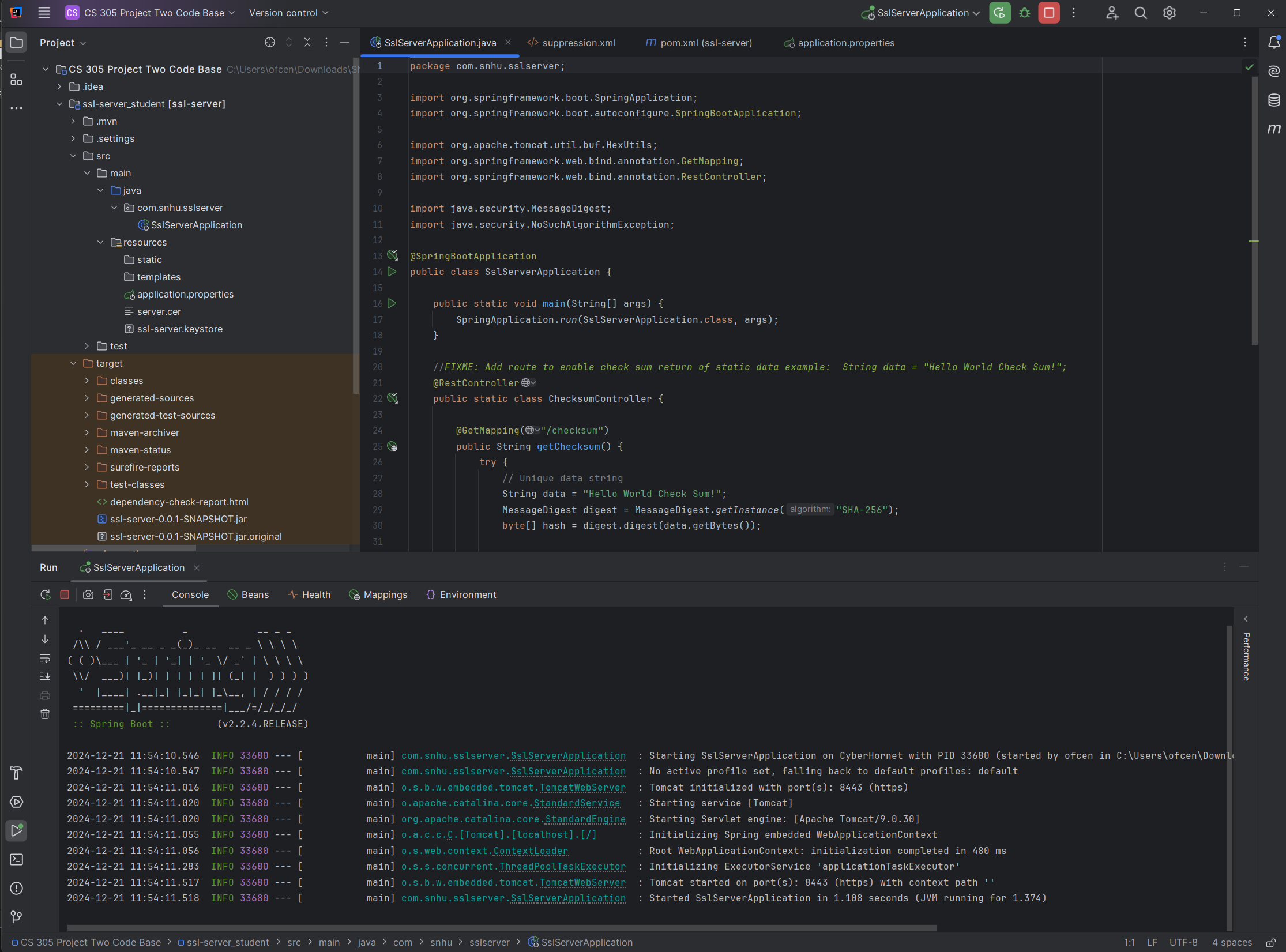


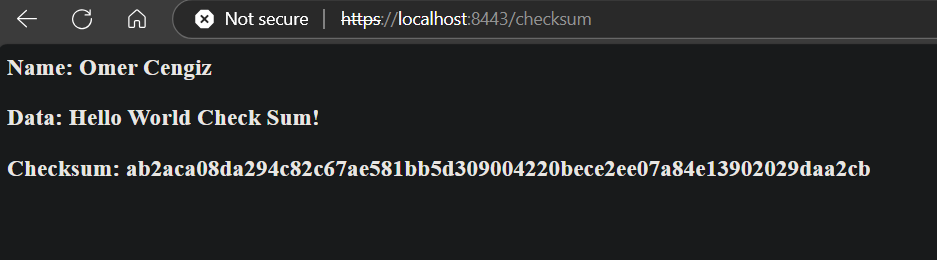
## Secondary Testing





## Functional Testing





As part of the functional testing process, I reviewed the refactored code to identify and resolve any syntactical, logical, or security vulnerabilities. I carefully inspected the application to ensure that all implemented changes worked as intended and adhered to secure coding practices. The code was manually reviewed to confirm that it followed best practices and maintained logical flow without introducing new issues.

To validate the application’s functionality, I executed the refactored code, and it ran successfully without any errors. Screenshots of the functional tests demonstrate that the code was executed properly, with the expected results being displayed. By completing these tests, I ensured that the system’s functionality aligned with its intended purpose while addressing vulnerabilities.

The testing process reinforced the importance of verifying all changes manually and through execution. This step provided confidence that the application was secure and ready for production without exposing users to any risks.

## Summary

After refactoring the code, I made sure that it complies with the necessary security testing protocols and addresses the vulnerabilities that were identified earlier. Here’s how I approached it:

A. Vulnerability Assessment Process

To address the vulnerabilities, I followed a clear process:

* Identifying Vulnerabilities: I used the OWASP Dependency Check tool to scan for vulnerabilities in the project’s dependencies, which flagged several issues in libraries like log4j and spring-core.
* Analyzing and Suppressing: I created a suppression.xml file to filter out false positives and low-risk vulnerabilities so that I could focus on critical issues.
* Fixing Issues: I updated dependencies to more secure versions wherever possible and ensured that cryptographic practices, like using SHA-256 for checksum generation, were implemented correctly.
* Re-testing: I ran the dependency check again after the changes to verify that the fixes worked and no new vulnerabilities were introduced.

B. Adding Layers of Security

I also added several layers of security to improve the overall safety of the application:

* Secure Hashing: I implemented SHA-256 for checksum generation, which helps ensure data integrity and secure handling of data.
* HTTPS Protocol: I enabled HTTPS by configuring the application to use an SSL certificate. This encrypts communication between the client and server, adding a layer of protection.
* Dependency Updates: I updated outdated libraries and managed unnecessary vulnerabilities using the suppression.xml file, which helps prevent exploitation of known weaknesses.
* Code Review: I carefully reviewed the code to ensure there were no logical or syntactical issues and followed secure coding practices.

## Industry Standard Best Practices

**A. Applying Industry Standards for Security**

To maintain and enhance the software application's existing security, I implemented several industry-standard best practices. I ensured the use of secure algorithms like SHA-256 for cryptographic operations, which is widely recognized for its strong security properties. I also configured the application to use HTTPS for all communications, protecting data in transit by encrypting it with SSL/TLS. Dependency vulnerabilities were managed by updating outdated libraries and suppressing false positives using OWASP’s Dependency Check tool and a well structured suppression.xml file.

**B. Value to the Company’s Well-Being**

Applying industry standard best practices for secure coding provides significant value to the company. It reduces the likelihood of security breaches, protecting sensitive customer data and maintaining the organization’s reputation. Secure practices also help comply with regulatory requirements, avoiding potential legal and financial penalties. By prioritizing secure coding, the company builds a strong foundation for its software, fostering customer trust and ensuring long-term sustainability in an increasingly security conscious market.