****

# 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** | **10/15/2025** | **Nedim Mulahusic** | **Secure Application** |

## 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

Nedim Mulahusic

## Algorithm Cipher

To ensure secure communication, Artemis Financial, a global provider of financial services, wishes to incorporate a file verification step into its web application. Upon reviewing Artemis Financials' requirements, I will suggest the SHA--256 cipher.

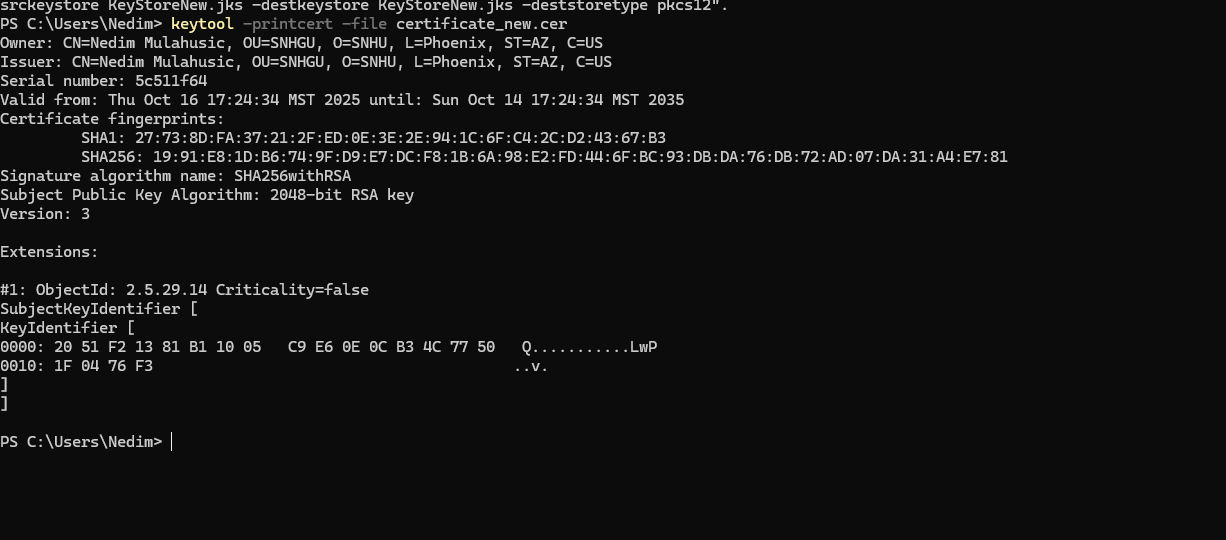
The SHA-256 cryptographic hash function is a dependable and trusted hash function that was created by the US National Security Agency and standardized by NIST. This hash function is extremely difficult to crack because it generates a 256-bit message digest from input. Data integrity is ensured because the entire hash result changes if even one bit of input changes. For any given input, the hash algorithm SHA-256 produces a 256-bit output. The main reason I choose this cipher is that it prevents collisions, which occur when two distinct inputs result in the same hash output. Because a hacker can alter the content while still producing the same checksum, this is risky. To make sure the data hasn't been altered and to stop any form of forgeries, collisions should be avoided at all costs.

When it comes to cryptography, randomness is crucial since it guarantees that keys are unexpected, making it impossible for attackers to replicate them. To prevent predictable keys, symmetric and asymmetric keys are produced at random using a secure random generator.

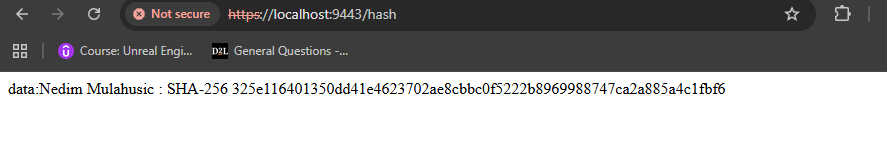
The use of encryption has predated the invention of computers by many years. The Caesar cipher was among the most well-known and early instances. This method involves selecting a key between 1 and 25 and shifting each letter in your message by that number. You can then decode it by shifting it backwards by that number. In just a few seconds, modern computers can decipher this cryptography. The threat posed by quantum computing demands a steady evolution of modern encryption systems. To combat the threat posed by quantum computing, new projects are being developed, and developers must stay current on emerging trends and practices to ensure the security of their applications.

## Certificate Generation

Insert a screenshot below of the CER file.

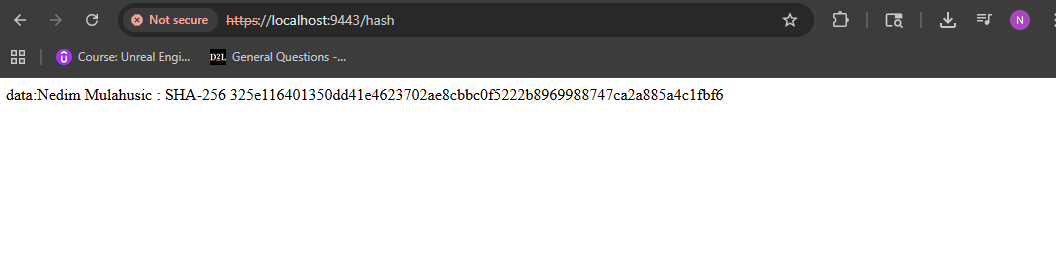


## Deploy Cipher



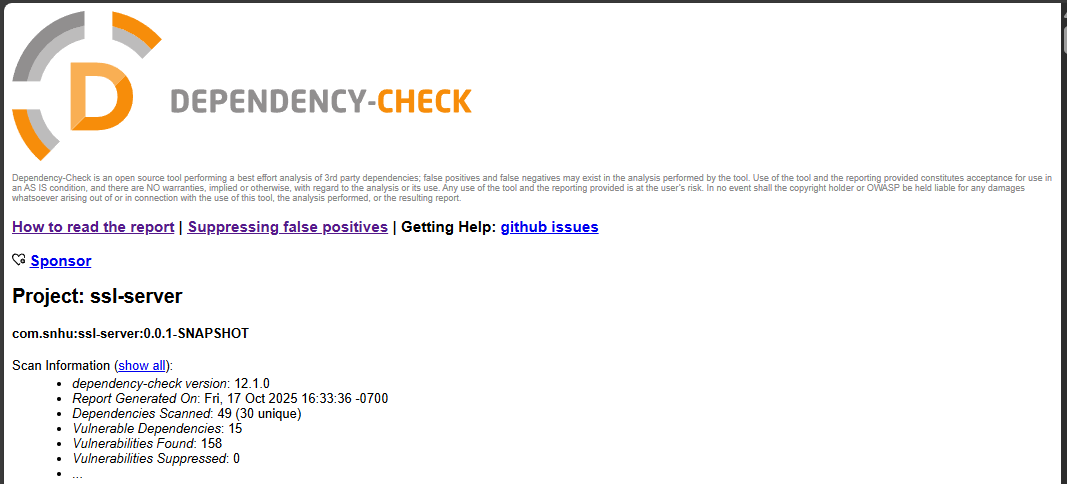
## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.



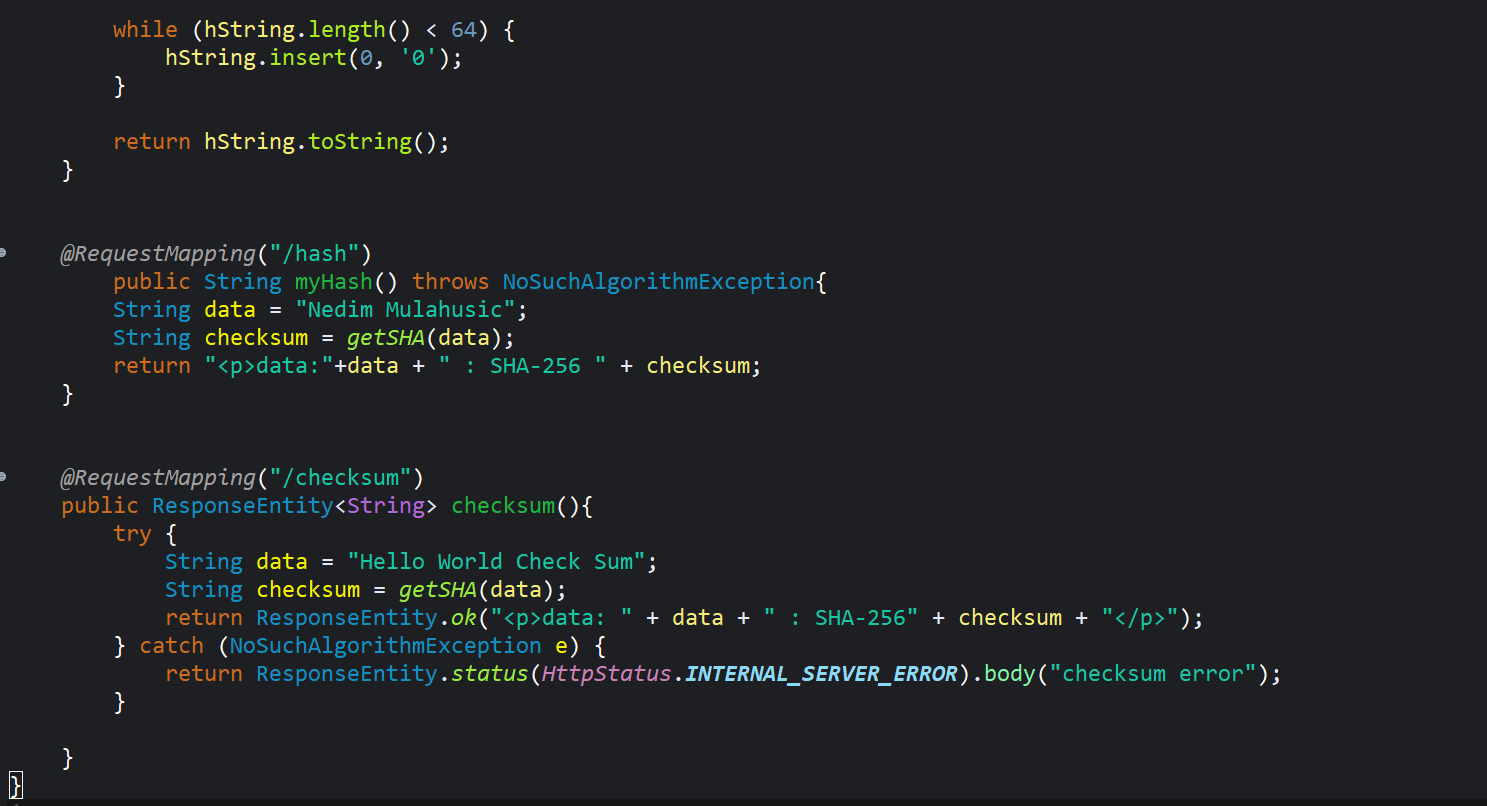
## Secondary Testing

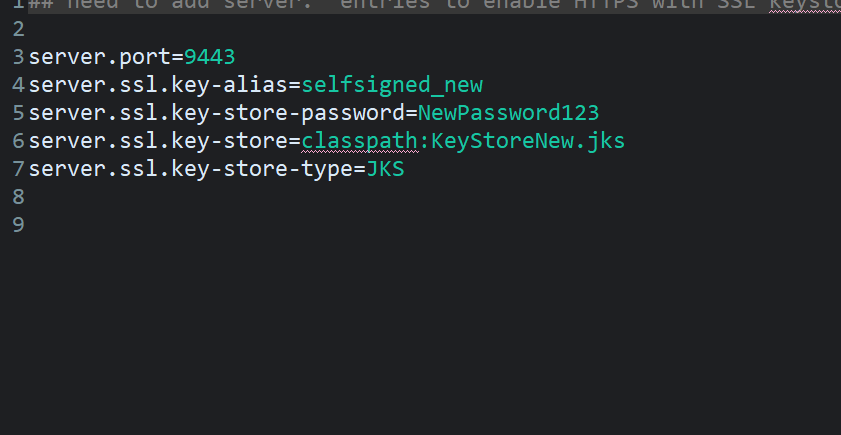
Insert screenshots below of the refactored code executed without errors and the dependency-check report.



## Functional Testing

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





## Summary

I made some security improvements to the SSL server program during its refactoring, following the flow diagram for the vulnerability assessment method found in the supporting materials. By integrating certificate creation for secure communication channels and creating the SHA-256 algorithm cipher for cryptographic hashing, I specifically addressed weaknesses pertaining to data integrity, secrecy, and authentication. By ensuring that data is encrypted and protected from manipulation, the SHA-256 hashing reduces the possibility of replaying attacks and integrity breaches found in previous evaluations. The refactoring improved the flow's data processing and encryption verification phases in line with the evaluation procedure.

To provide a nice cryptographic function for data verification, I included SHA-256 hashing into the getSHA() method, adding levels of protection. To encrypt client-server connections, I additionally generated SSL/TLS certificates. Finally, I employed secondary and functional testing to confirm both my encryption and cipher deployment.

## Industry Standard Best Practices

Throughout the reworking process, I used a number of industry-standard best practices for secure coding to make sure the software program maintains a high level of security and integrity. Frameworks and guidelines like the OWASP Top Ten, NIST SP 800-53, and ISO/IEC 27001 standards for application security are all in line with these principles. By employing cryptographic techniques such as the SHA-256 hashing method and applying certificate generation to secure HTTPS channels, I was able to retain security. I also carried out secondary security testing and code validation to make sure that no outdated or ineffective techniques were being utilized and that appropriate exception handling was in place to keep system secrets private. All things considered, adhering to industry best practices promotes a security-first culture throughout the development lifecycle, guaranteeing that the business and its clients gain from a stable, reliable, and legal software environment.