****

# 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

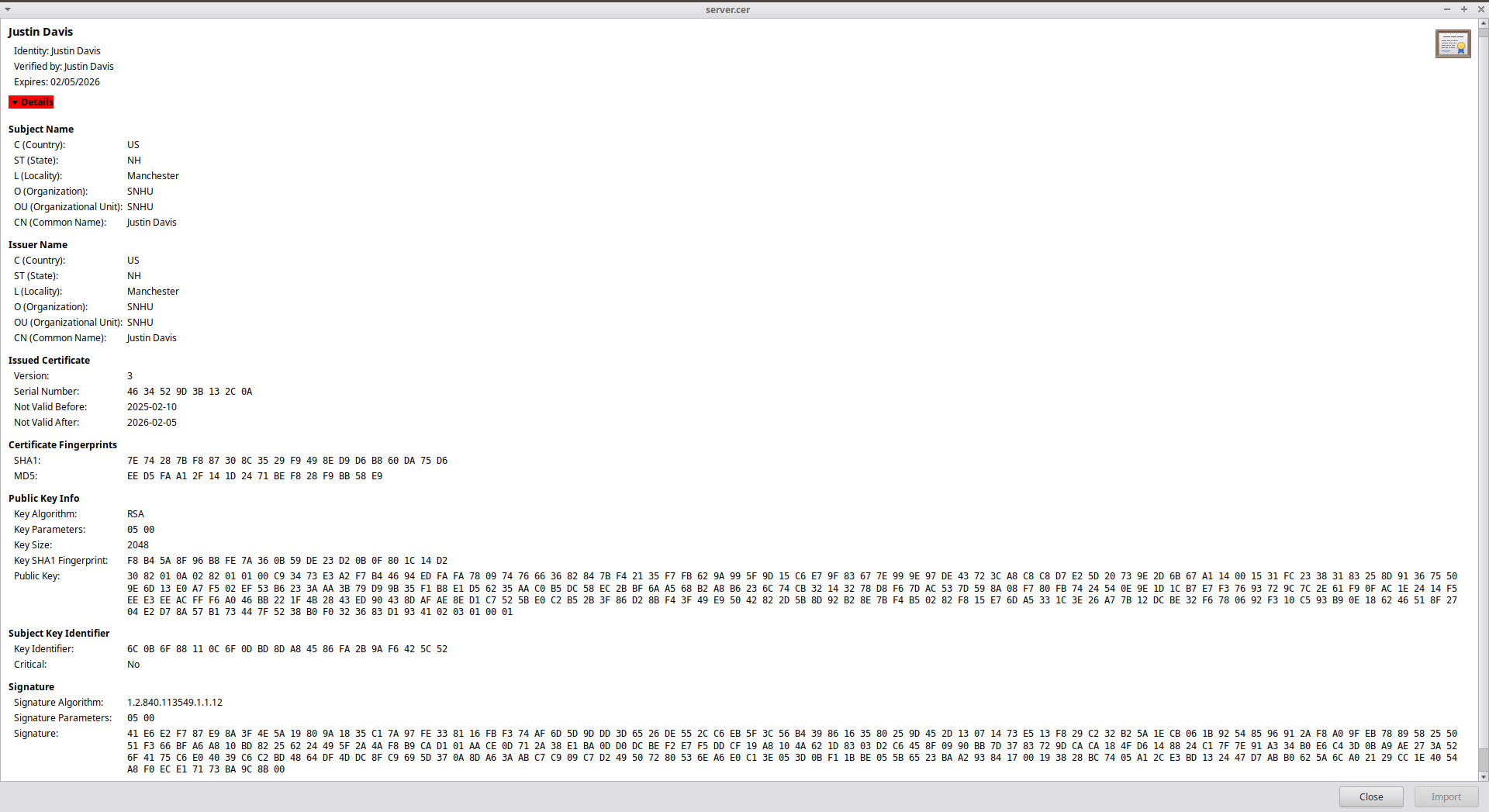
Justin Davis

## Algorithm Cipher

SHA-512 is a cryptographic hash function that belongs to the SHA-2 family and is widely used for data integrity verification and secure password hashing. Unlike encryption algorithms such as AES, which are used for encrypting and decrypting data, SHA-512 is a one-way hashing algorithm that generates a fixed 512-bit hash value from an input of any size. This makes it ideal for verifying data integrity, storing passwords securely, and ensuring that information has not been altered. SHA-512 is resistant to collision attacks due to its large output size, making it significantly more secure than its predecessor, SHA-1. The algorithm operates using a series of bitwise operations, modular additions, and compression functions to process data in 1024-bit blocks. It also incorporates randomization techniques through initialized hash values to enhance security. Originally developed by the National Security Agency (NSA) and standardized by NIST, SHA-512 remains a trusted hashing standard for cryptographic applications, including SSL/TLS, digital signatures, and blockchain security. Its strong resistance to cryptographic attacks and its widespread industry adoption make it an excellent choice for securing sensitive data in financial applications like those used by Artemis Financial.

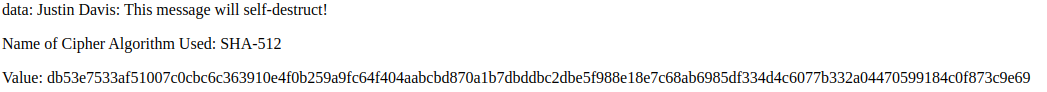
## Certificate Generation

Insert a screenshot below of the CER file.



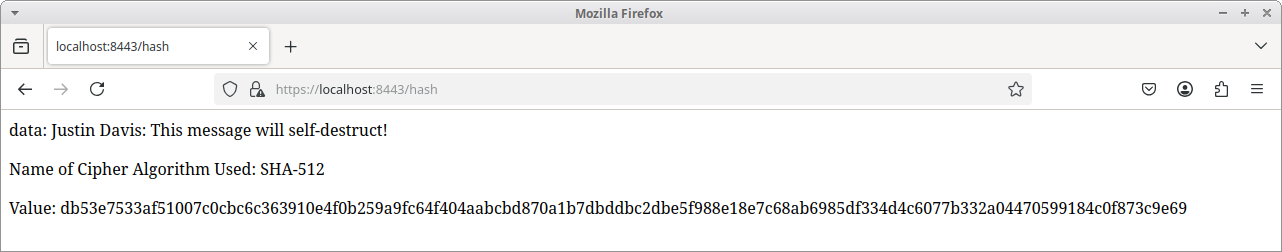
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

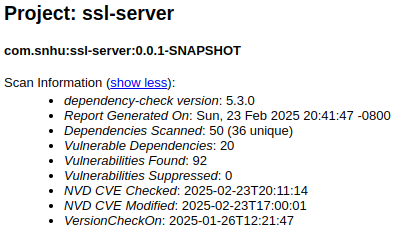
Insert a screenshot below of the web browser that shows a secure webpage.



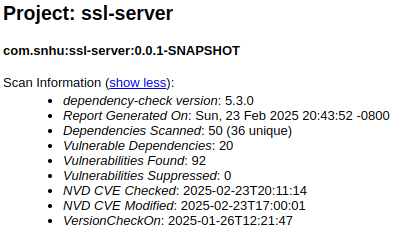
## Secondary Testing

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

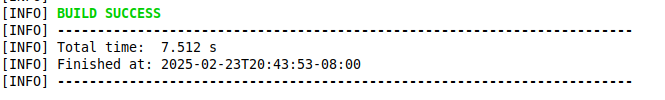
Before refactoring:



After refactoring:

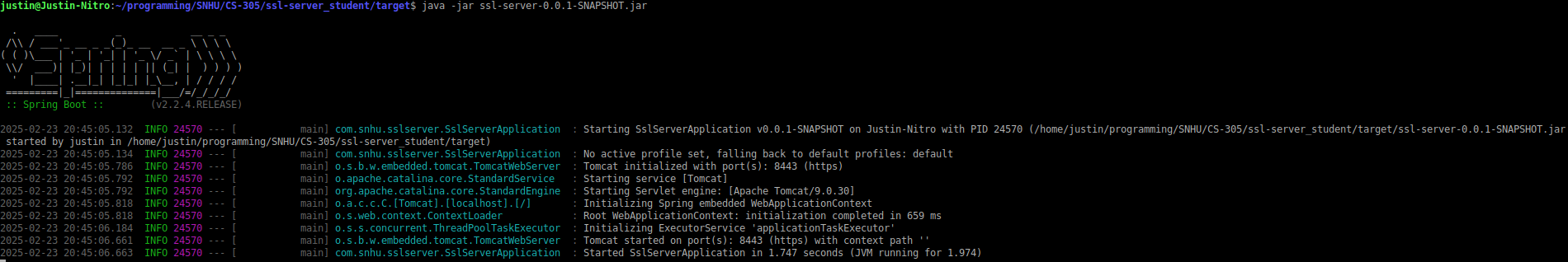


No new vulnerabilities were introduced when refactoring the code.



## Functional Testing

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



## Summary

The project involved a comprehensive security enhancement process for Artemis Financial’s application, focusing on encryption, secure communication, and vulnerability management. The first step was implementing a cryptographic hash function using SHA-512 to ensure data integrity and security. This involved refactoring the code to generate secure hashes and verifying their accuracy through checksum validation. Additionally, self-signed certificates were generated using Java Keytool, allowing the application to support HTTPS communication. The server configuration was modified to enforce HTTPS, ensuring encrypted data transmission and reducing the risk of interception. However, due to the nature of self-signed certificates, browsers flagged the connection as untrusted, requiring manual acceptance of the certificate to proceed.

A critical aspect of the project was running a dependency check using the OWASP Dependency-Check plugin. The initial scan revealed multiple vulnerabilities across various dependencies, some of which were false positives that could not be addressed through version upgrades. To handle this, a suppression file was created to filter out these false positives, ensuring that only actionable vulnerabilities remained in the report. After refactoring, a secondary dependency check was performed to confirm that no new vulnerabilities were introduced. The results showed that the suppression rules were applied correctly and that the changes did not negatively impact the application's security posture.

## Industry Standard Best Practices

Throughout this process, industry-standard best practices for secure coding were followed, ensuring that encryption, secure communication, and dependency management aligned with modern security requirements. By addressing security vulnerabilities through proactive measures, the software’s resilience against potential threats was strengthened. The implementation of encryption, certificate-based security, and static analysis techniques demonstrated a layered security approach, reinforcing the importance of secure software development practices.