****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc929351281)

[Table of Contents 1](#_Toc992212673)

[Document Revision History 2](#_Toc1783407638)

[Client 3](#_Toc1672746572)

[Developer 3](#_Toc2004531781)

[1. Algorithm Cipher – SHA-256 4](#_Toc1678353142)

[2. Certificate Generation 4](#_Toc113678493)

[3. Deploy Cipher 4](#_Toc1001966519)

[4. Secure Communications 5](#_Toc397314391)

[5. Secondary Testing 5](#_Toc758607533)

[6. Functional Testing 7](#_Toc1508271044)

[7. Summary 9](#_Toc202294170)

[8. Industry Standard Best Practices 9](#_Toc142824636)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **2.23.25** | **Garrick Gross** | **Initial State** |

## Client



## Developer

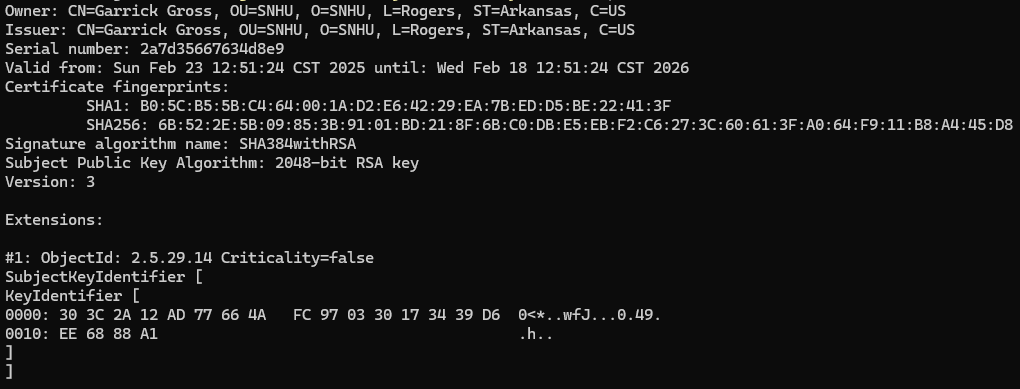
Garrick Gross

## Algorithm Cipher – SHA-256

1. Strong Security Margin. A 256-bit hash output is computationally infeasible to attack with modern hardware.
2. Widespread Adoption and Trust. Backed by NIST standards, extensively analyzed by the cryptographic community, and commonly supported in frameworks/libraries.
3. Versatility. Used in generating checksums for file integrity, digital signatures, certificates, and password hashing (with salts).
4. Compatibility with Regulatory Requirements. Meets or exceeds most compliance standards (e.g., FIPS 140-2/3).
5. Futureproofing. While quantum computing may eventually require further measures (e.g., post-quantum algorithms), SHA-256 still holds a robust security posture today.

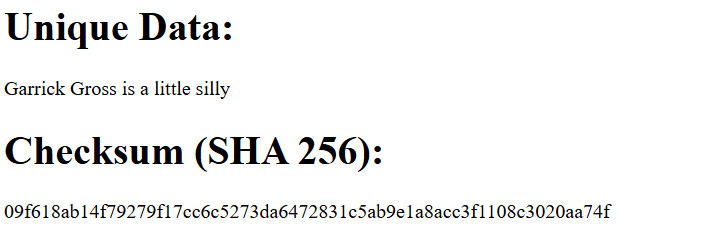
## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

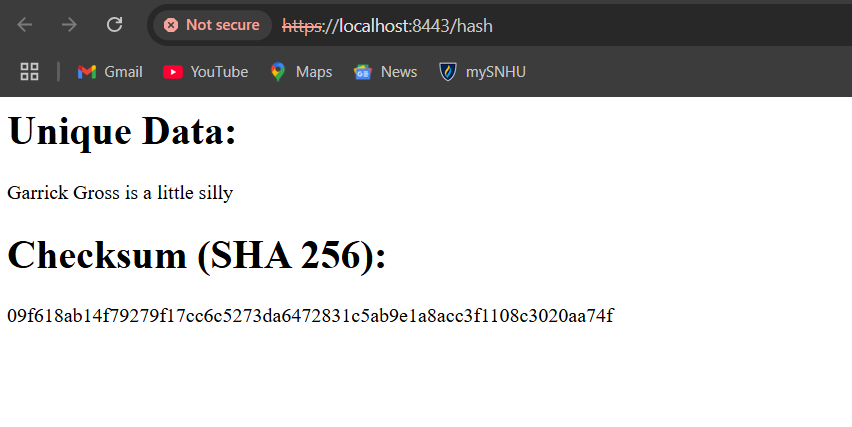
Insert a screenshot below of the checksum verification.

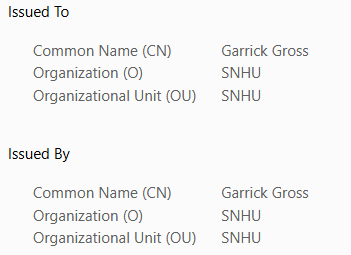


## Secure Communications

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

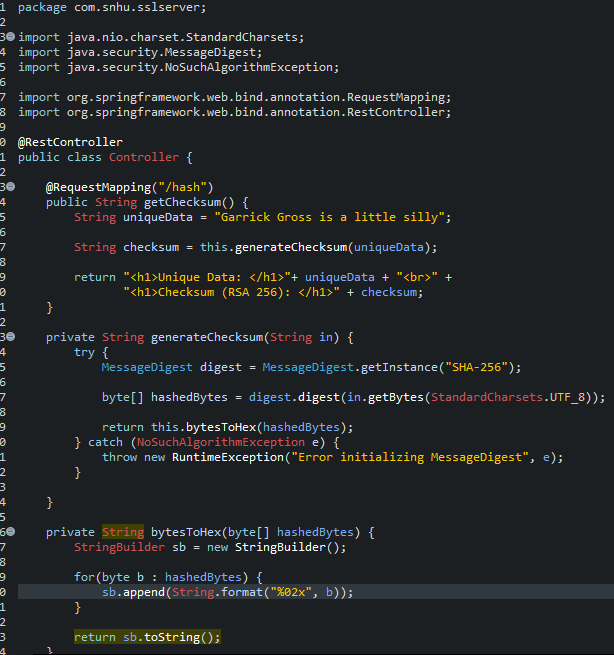
It states not secure, but I can view self signed certificate in browser, so assuming this is due to the weaker nature of the certificate.

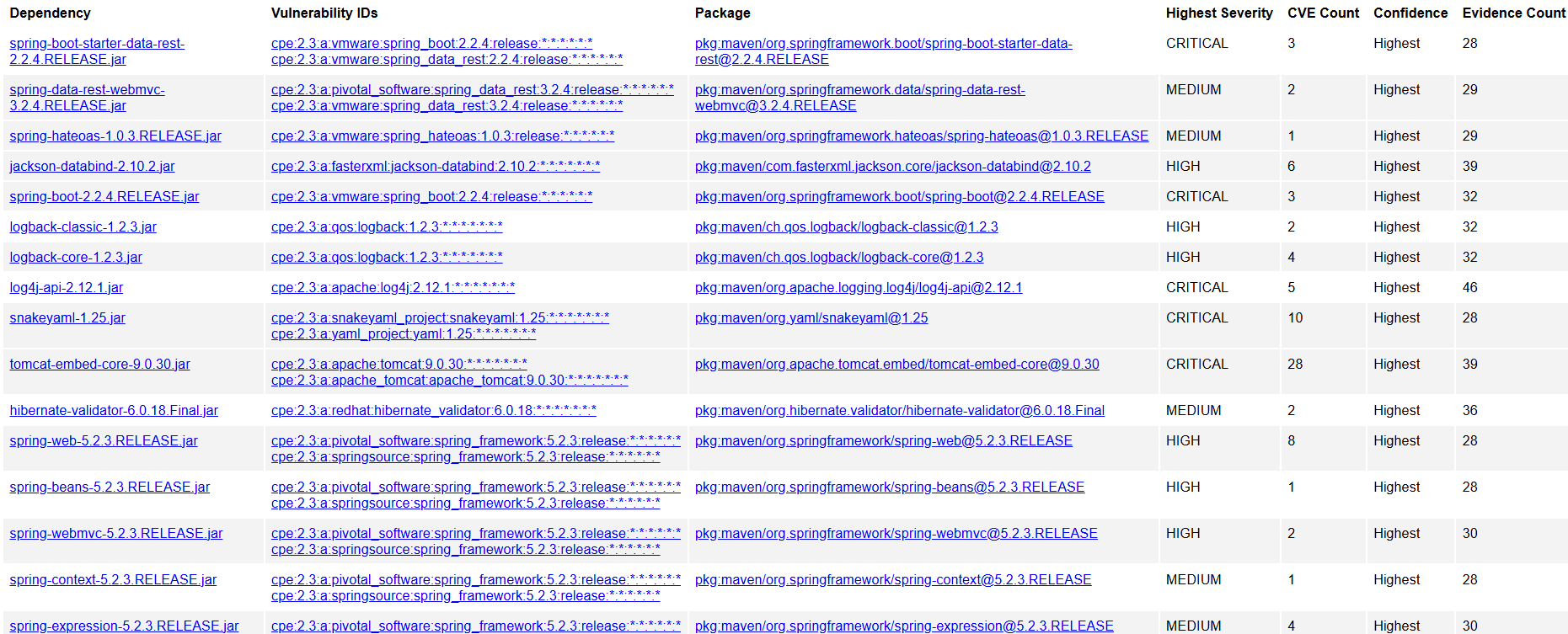




## Secondary Testing

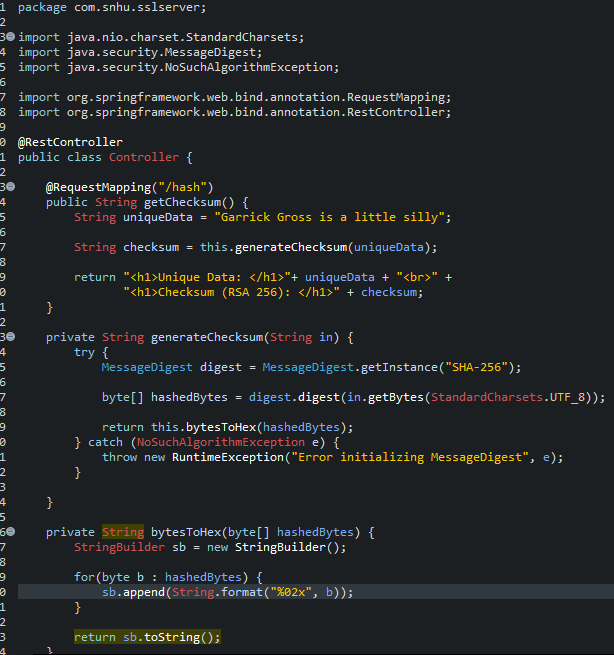
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



A thorough review of the refactored code was conducted to confirm there were no remaining syntactical errors, logical flaws, or unaddressed security vulnerabilities. In particular:

* Syntactical Issues: Verified that all imports, method declarations, class declarations, and annotations are valid and that the application compiles successfully without any IDE or compiler errors.
* Logical Flow: Confirmed that the methods for generating checksums, configuring SSL, and handling certificates align with the intended functionality. This includes verifying that the /hash endpoint correctly produces the SHA-256 hash of the unique data string.
* Security Concerns:
  + Ensured that no plaintext sensitive data remains in the refactored code.
  + Verified that the SSL configuration (in application.properties and the Java KeyStore) prevents insecure connections.
  + Checked that the code does not expose unnecessary endpoints or debugging logs that reveal implementation details.

## Summary

Following the **vulnerability assessment process flow** (as referenced in the Supporting Materials), the refactored code addresses security concerns in multiple areas:

1. Areas of Security Addressed:
   1. Data in Transit: Enforced HTTPS communications by generating and configuring a self-signed SSL certificate in the application.properties file. All traffic to port 8443 is encrypted.
   2. Data Integrity: Employed a secure cryptographic hash (SHA-256) to produce checksums for verifying data authenticity.
   3. Code Vulnerabilities: Analyzed code logic to remove any insecure practices, such as storing passwords in plaintext or exposing internal details of the hashing process.
2. Process for Adding Layers of Security:
   1. Certificate Generation: Created a project2store.jks and used a strong password to secure it. This ensures that the truststore is not publicly accessible.
   2. HTTPS Configuration: Updated application.properties to specify that the server should run over port 8443 using TLS/SSL with the generated certificate.
   3. Hashing Implementation: Ensured that the hashing function (SHA-256) is invoked properly and that no weaker algorithms remain in use.
   4. Static (Dependency) Testing: Re-ran the dependency-check plugin after refactoring to confirm that no new security vulnerabilities were introduced.

## Industry Standard Best Practices

Industry standard best practices for secure coding were incorporated throughout the refactoring process to mitigate known security vulnerabilities:

1. Maintaining Existing Security:
   1. Ensured that existing protective measures—such as not exposing stack traces or sensitive data—remained intact.
   2. Preserved the robust hashing method (SHA-256) and did not reintroduce insecure algorithms such as MD5 or SHA-1.
2. Value to Artemis Financial’s Overall Well-Being:
   1. Integrity & Trust: Secure communications (HTTPS) protect clients’ sensitive financial data, enhancing trust in Artemis Financial’s services.
   2. Regulatory Compliance: Adhering to industry best practices aids in meeting regulations like PCI-DSS or GDPR (depending on data type and region), reducing liability.
   3. Reduced Threat Surface: Systematically eliminating vulnerabilities through encryption, SSL, and code reviews decreases the risk of data breaches and reputation damage.