

# Practices for Secure Software Report

Table of Contents

Practices for Secure Software Report 1

Document Revision History 3

Client 3

Instructions 3

Developer 4

1. Algorithm Cipher 4
2. Certificate Generation 4
3. Deploy Cipher 5
4. Secure Communications 5
5. Secondary Testing 6
6. Functional Testing 7
7. Summary 8

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **02-23-2025** | **Connor Bailey** | **Initial Commit** |

## 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.

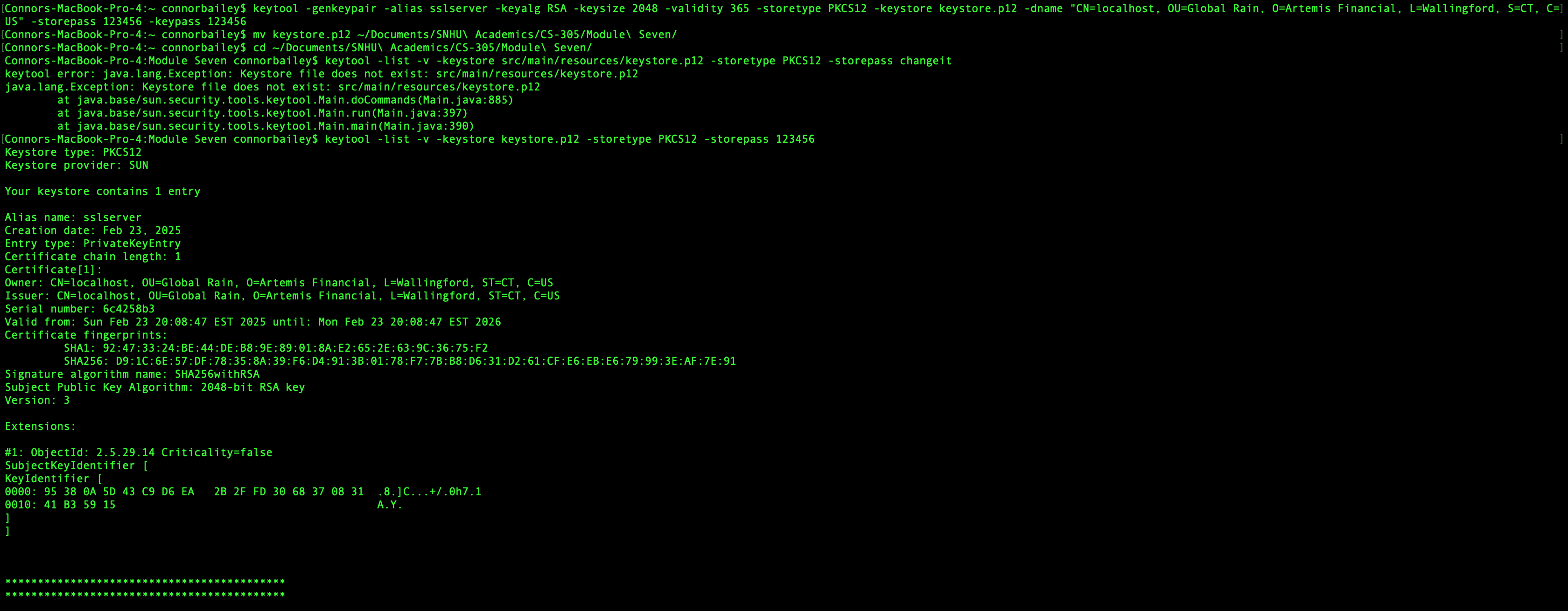
## Developer

Connor Bailey

## Algorithm Cipher

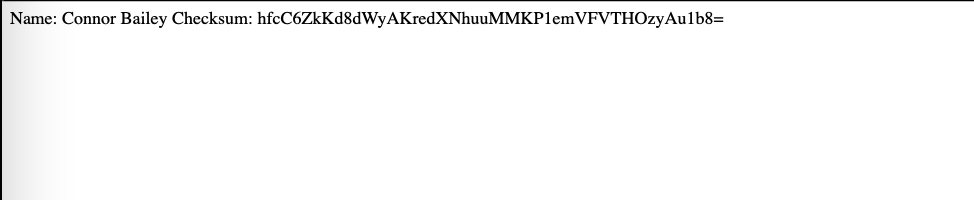
Artemis Financial is looking for a secure way to verify the integrity of transmitted data by implementing a checksum verification step within its web application. The most appropriate approach for this is to use a hashing algorithm like SHA-256 (Secure Hashing Algorithm 256-bit). SHA-256 is a widely used cryptographic function known for its strong collision resistance. Unlike algorithms like RSA or AES, SHA is designed for data integrity verification by generating a fixed length 256-bit hash from any input. When a file is transmitted, its SHA-256 hash can be computed before and after transmission. A hash function is a pseudo-random one way function that takes an input and generates a 256-bit length representation of the data. The data generated from a hash function (called the "hash") is unique to the input. If even a single character is changed in the input, a completely different output will be returned. If the hash taken on the client matches the hash taken on the server, then it can be verified that the message has not been tampered with. Historically, cryptographic hash functions have evolved along side the computational power of computers. As computers became more powerful, the need for stronger hashing algorithms resistant to attacks from these computers also grew. Older algorithms like MD5 and SHA-1 used smaller key lengths (MD5 uses 128-bit keys and SHA-1 uses 160-bit keys), and have also been proven prone to collision attacks. SHA-256 uses a key length that is double the size, and brute forcing collisions on this hashing algorithm is computationally inefficient. While a malicious attacker could theoretically produce a collision against this algorithm, it would take longer than the age of the universe to successfully brute force this using a classical computational computer. This means that there is no practical application with the current computing power available where an attacker could brute force a collision attack against SHA-256, making it incredibly secure and an industry standard choice for verifying the integrity of transmitted data.

## Certificate Generation

Insert a screenshot below of the CER file.

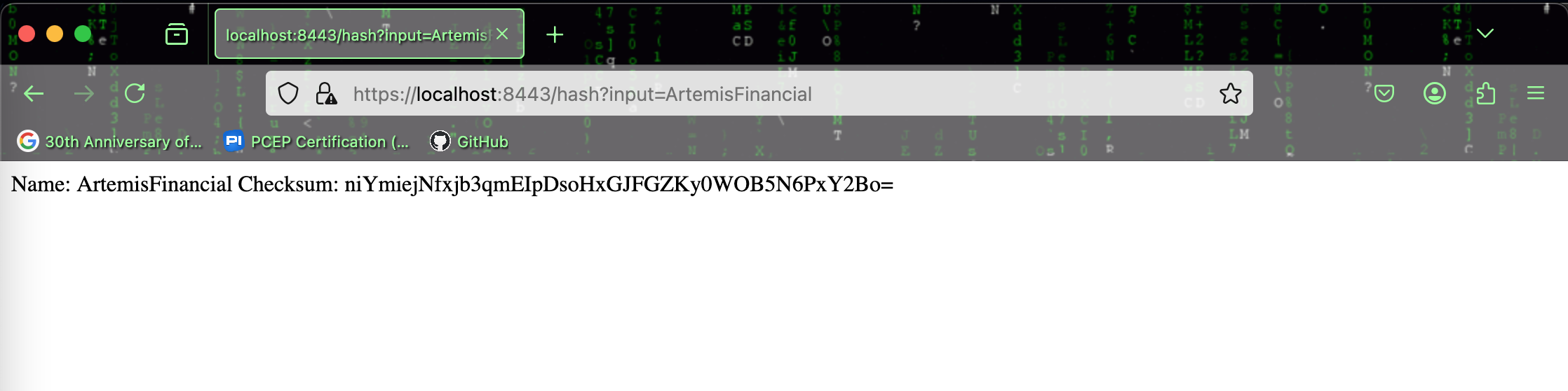
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

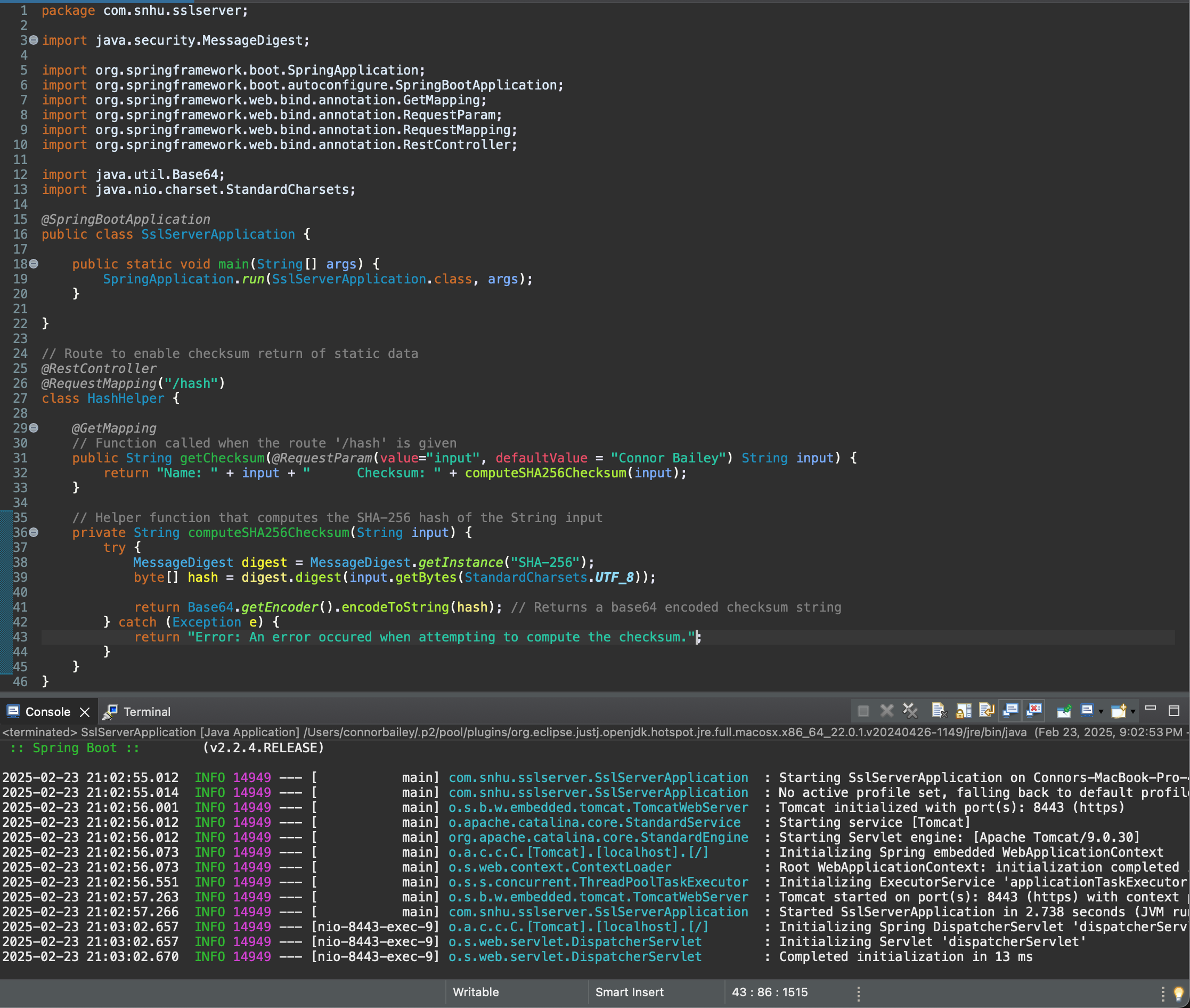
Insert a screenshot below of the web browser that shows a secure webpage.

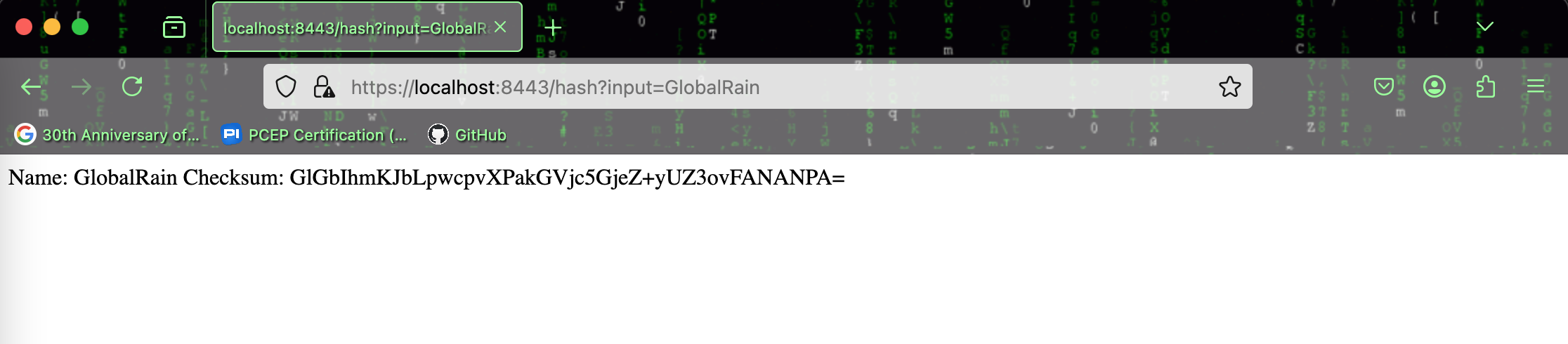


## Screen Shot 2025-02-23 at 8.45.55 PM.pngSecondary Testing

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

We refactored Artemis Financial’s web application to focus on enhancing security through encryption, secure communications, and integrity verification. From the vulnerability assessment diagram, we addressed important areas such as cryptography, error handling, and secure coding practices to refactor this into a secure application. To strengthen this security, we implemented SHA-256 hashing for checksum verification and enabled secure communications by configuring SSL/TLS with a self-signed certificate. This enables the use of HTTPS on port 8443, preventing man-in-the-middle attacks from data interception.

After making these changes, we performed a static dependency check to verify that no new security risks were introduced into the application. After performing this check, we were able to successfully verify that no new vulnerabilities were introduced, however, we would still recommend mitigating or resolving the other listed vulnerabilities when possible. We also perform functional and error-handling tests to ensure that responses behaved as expected while also ensuring we weren’t exposing any sensitive system data. In the first iteration of this code, we were returning the stack trace in our error message to the user. We modified these changes to prevent printing out the stack trace to prevent exposing information about how the system is constructed. By following these secure coding patterns, we have maintained compliance with industry standard security practices and have secured Artemis Financials’ sensitive financial data.