****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_30j0zll)

[**Client 3**](#_1fob9te)

[**Instructions 3**](#_3znysh7)

[**Developer 4**](#_2et92p0)

[**1. Algorithm Cipher 4**](#_tyjcwt)

[**2. Certificate Generation 4**](#_3dy6vkm)

[**3. Deploy Cipher 4**](#_1t3h5sf)

[**4. Secure Communications 4**](#_4d34og8)

[**5. Secondary Testing 4**](#_2s8eyo1)

[**6. Functional Testing 4**](#_17dp8vu)

[**7. Summary 4**](#_3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **02-25-2024** | **Peter F. Tumulty** |  |

## 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

Peter F. Tumulty

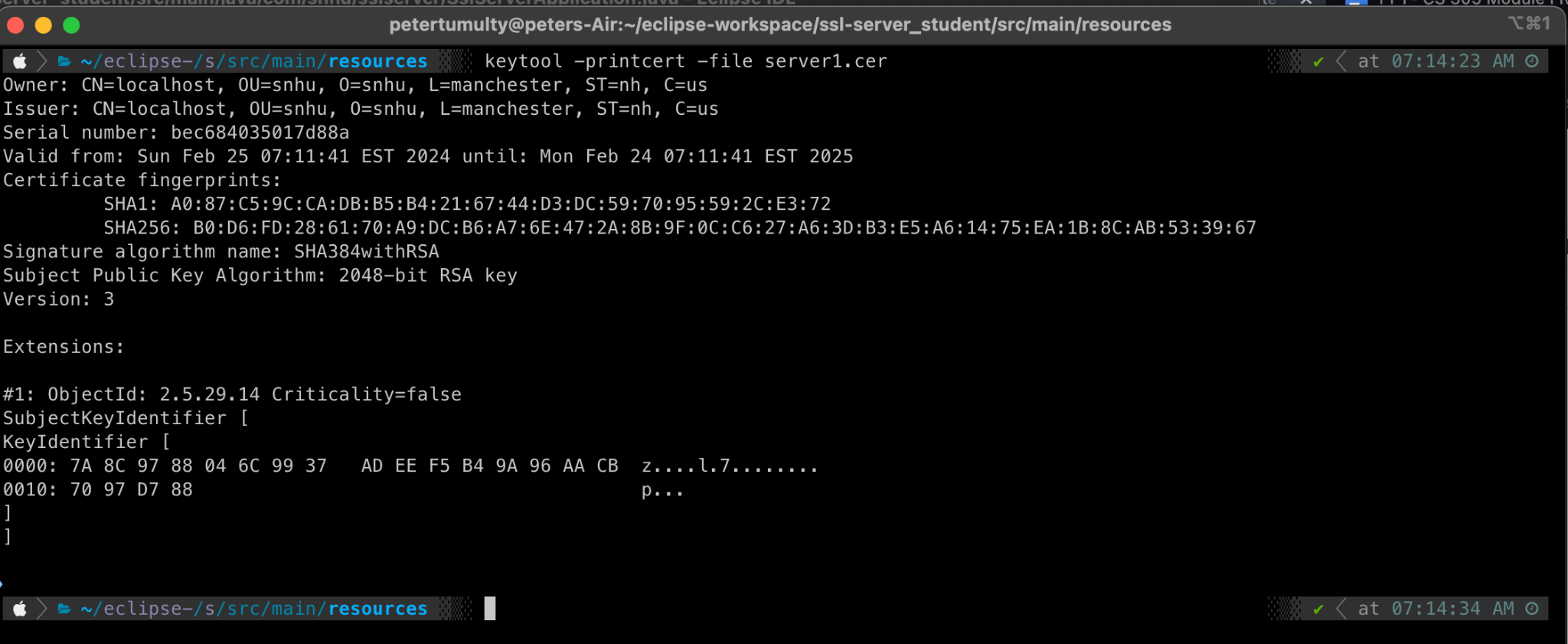
## Algorithm Cipher

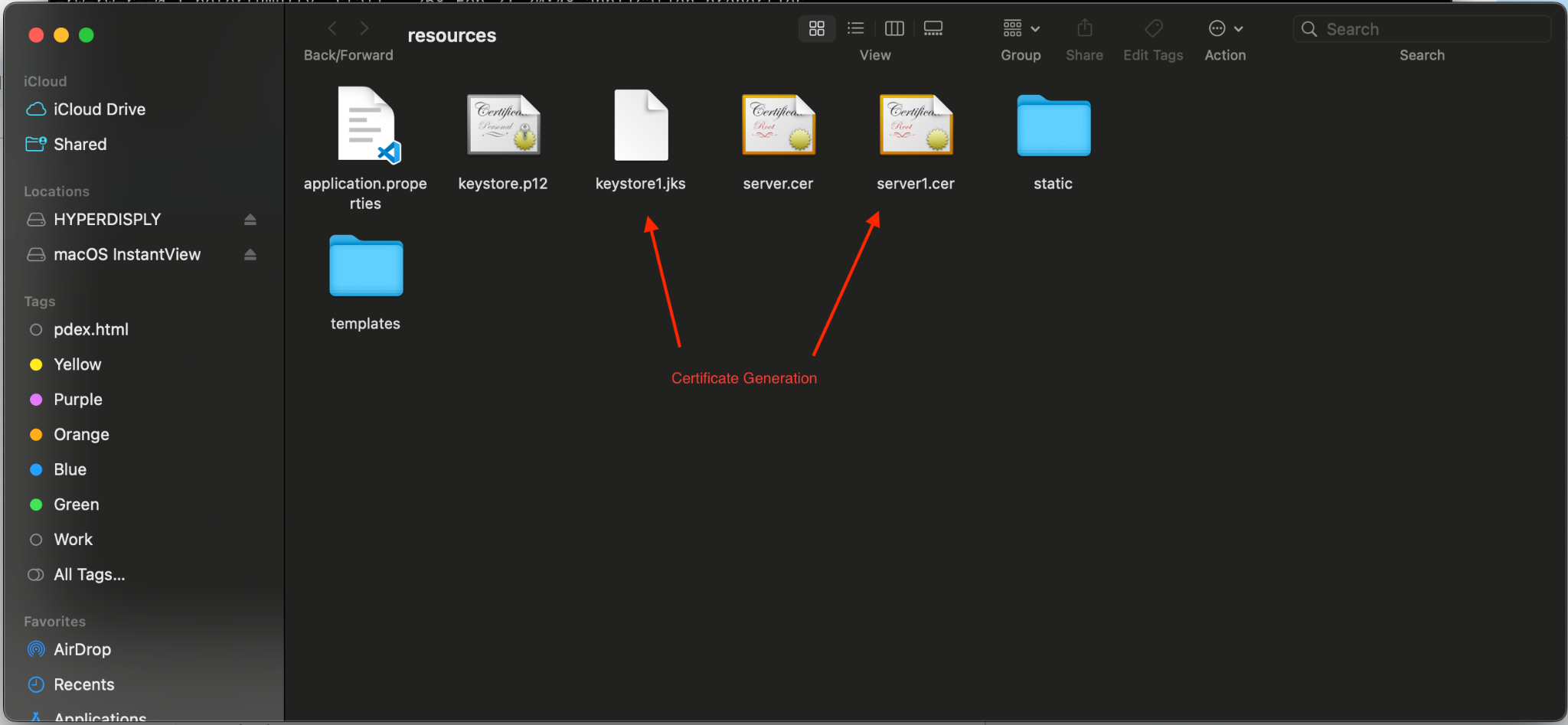
The algorithm chosen for this Java coding assignment is SHA-256, a variant of the Secure Hash Algorithm (SHA). According to Dang (2015), SHA encompasses a suite of cryptographic hash functions used to "generate message digests for ensuring the integrity of messages and detecting changes." Among its several versions, SHA-256 is the most widely used, owing to its straightforward implementation via java.security.DigestException, robust documentation support, and absence of known vulnerabilities. Since SHA-256 is a cryptographic hash function, the message digest appears as a series of random numbers and characters. The design of the hash function ensures that the output is one-way, making it impossible to deduce the input from the output. As outlined in the paper "Recent Contributions to Cryptographic Hash Functions," the older versions of SHA-1 and MD5 have been compromised. Cryptographers Andre Joux and Xiaoyun Wang devised an attack that undermines the collision-resistance property of these widely deployed hash functions (Walker et al., 2009). The development of SHA-256 is of critical importance when selecting a secure and modern cryptographic hash function to safeguard data in Artemis Financial's application.

**Reference**

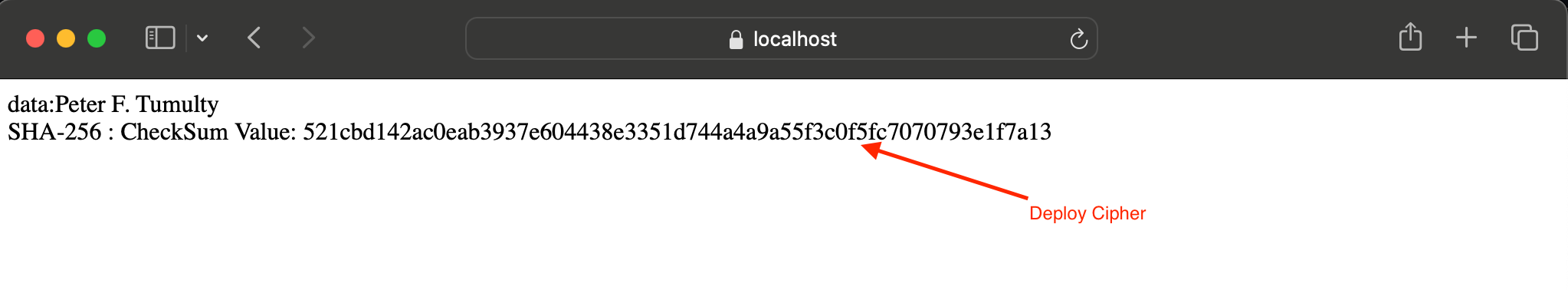
Dang, Q. H. (2015, August 4). *Secure Hash Standard (SHS)* (NIST FIPS 180-4). National Institute of Standards and Technology. <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>  
  
Walker, J., Kounavis, M., Gueron, S., & Graunke, G. (2009). Recent contributions to cryptographic hash functions. *Intel Technology Journal, 13*(2). <https://www.techrepublic.com/resource-library/whitepapers/recent-contributions-to-cryptographic-hash-functions/>

## Certificate Generation

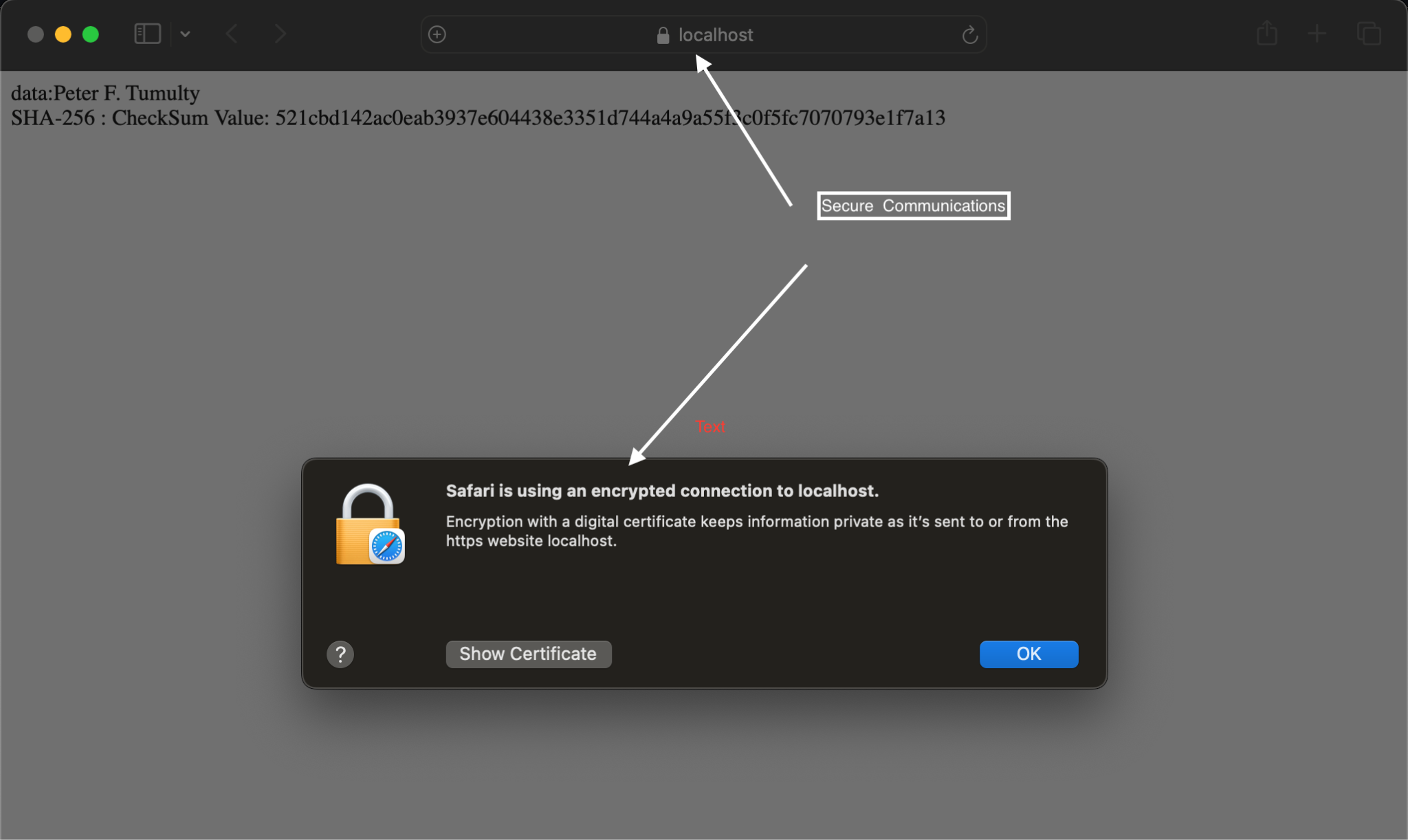




## Deploy Cipher

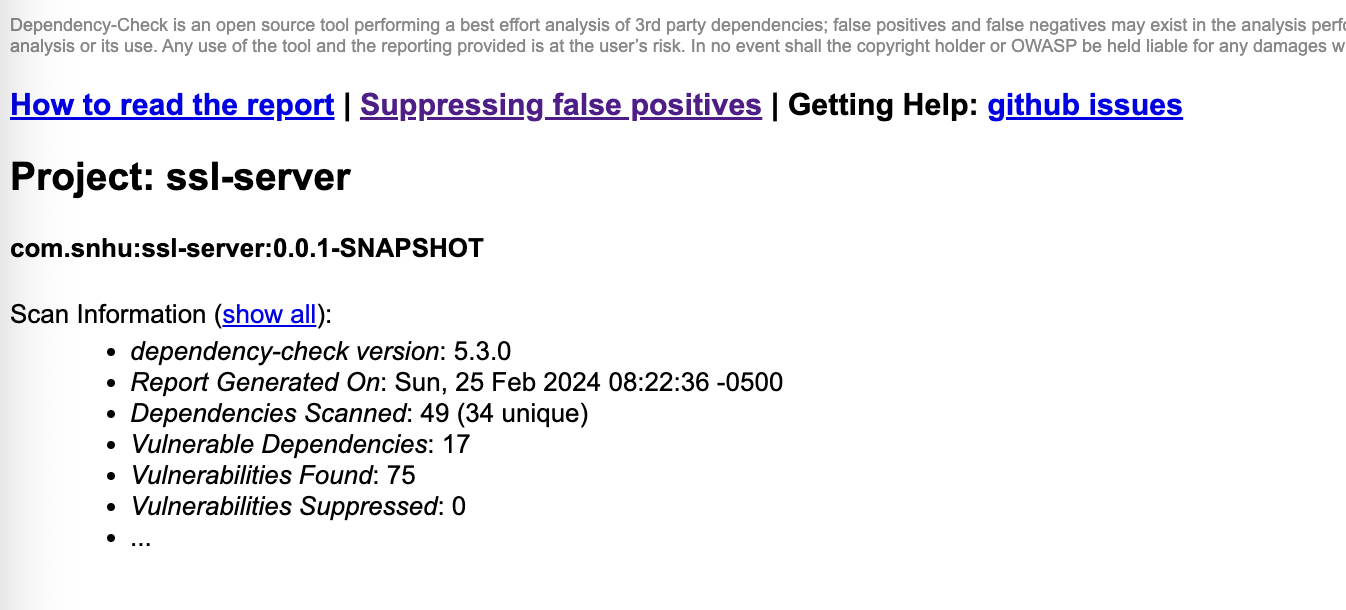


## Secure Communications

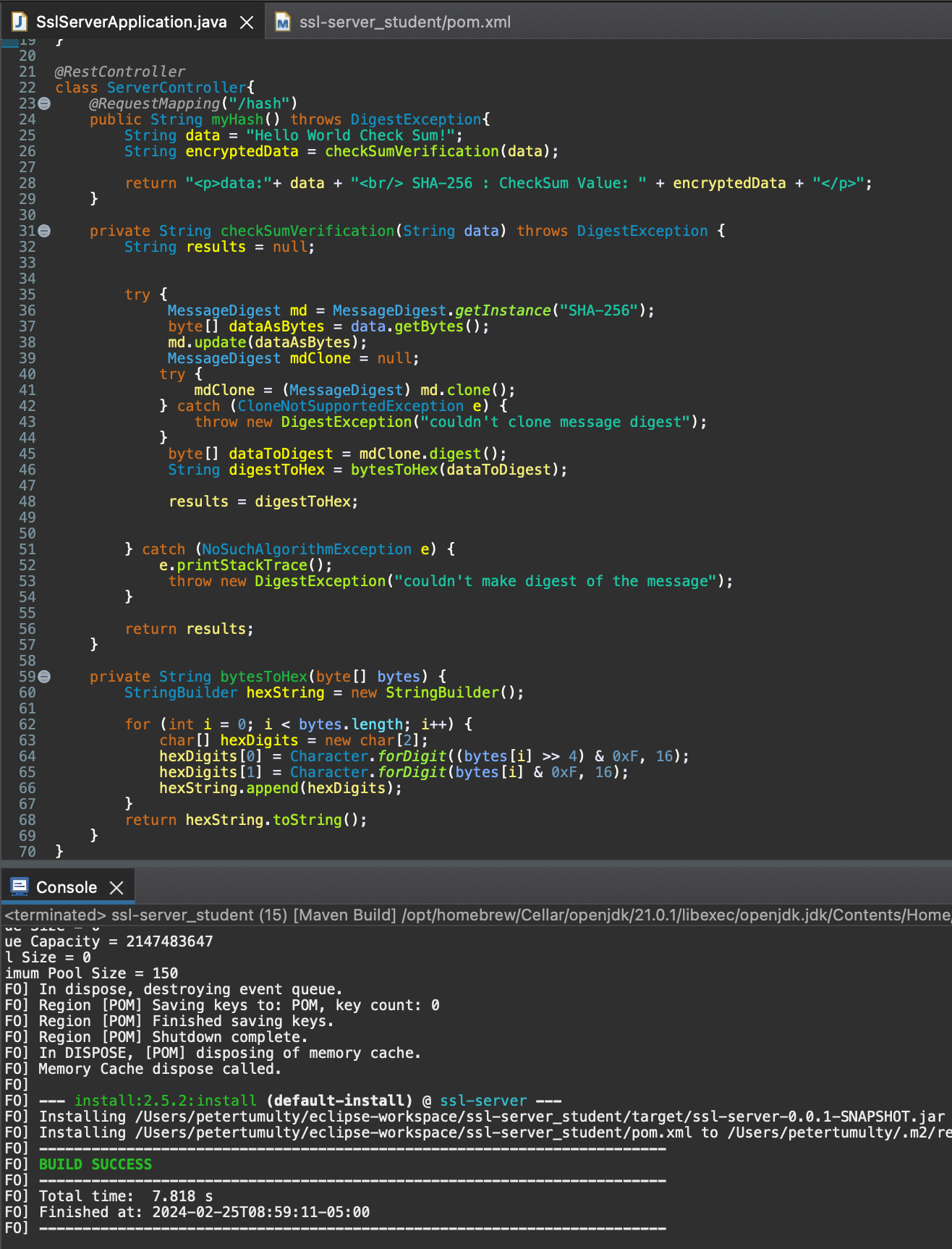


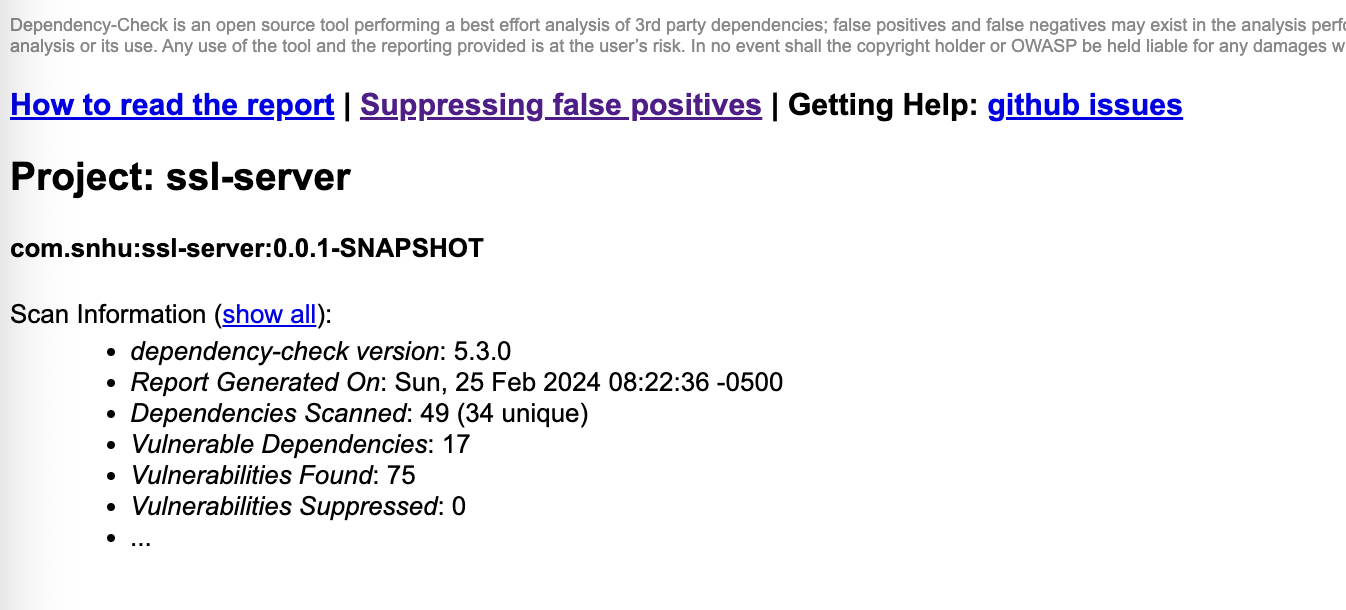
## Secondary Testing

**The results of dependency vulnerability report before refactoring the SslServerApplicaiton.java file**

****

**The results of the dependency vulnerability report after refactoring the SslServerApplicaiton.java file**

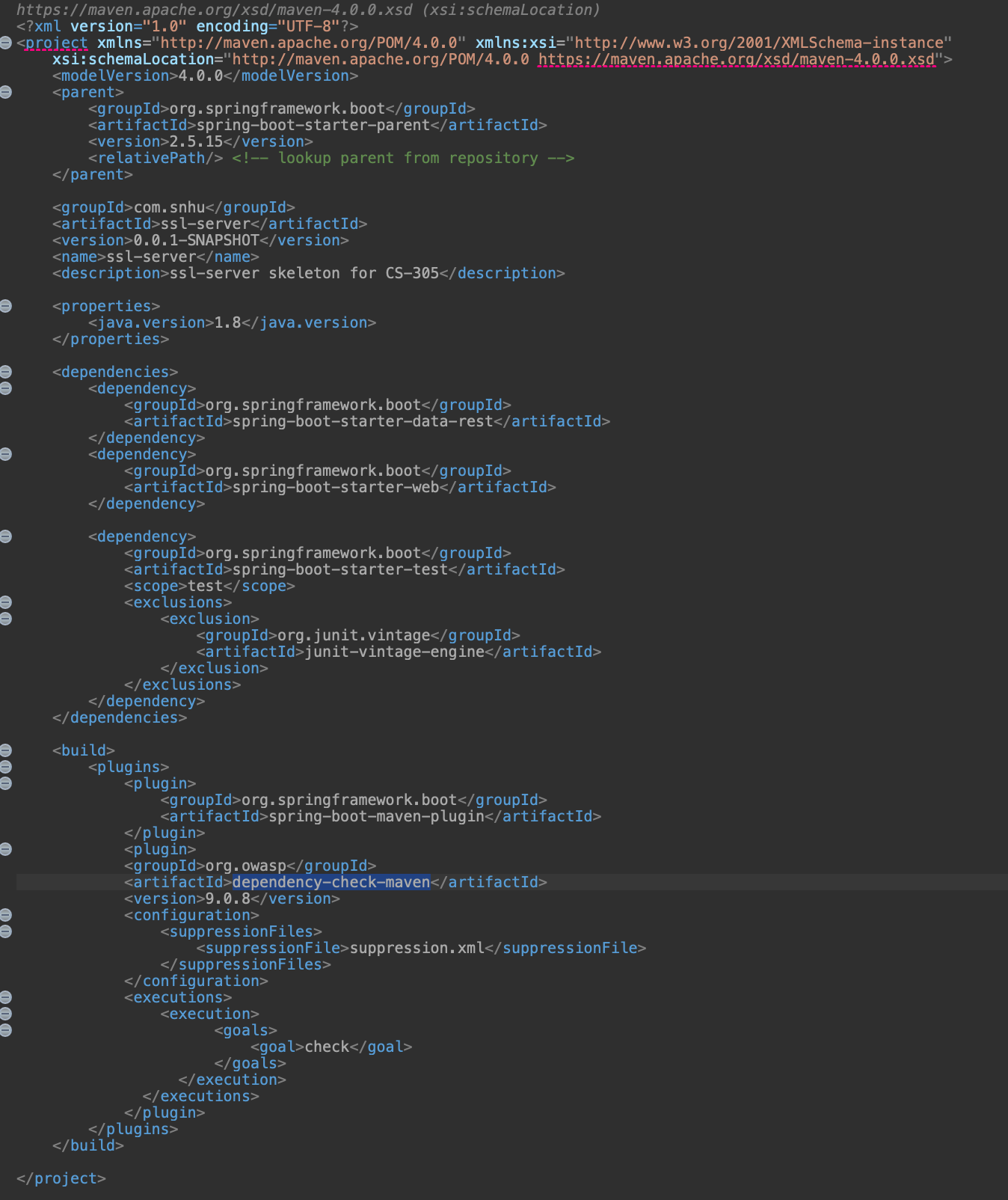


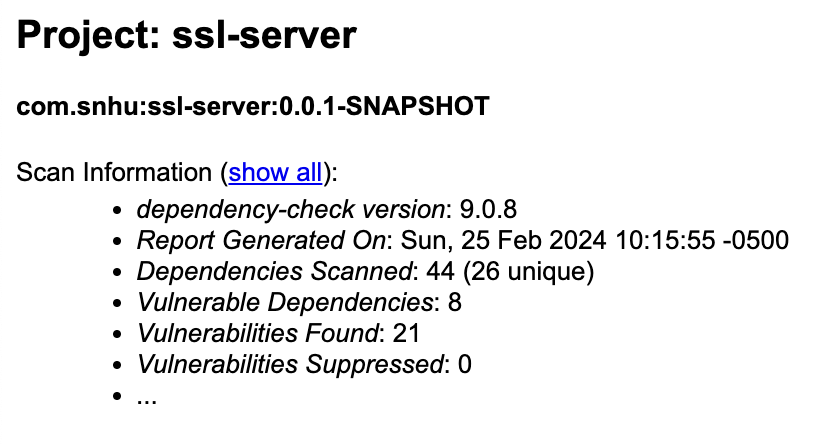
****

As you can see the dependencies scanned, vulnerable dependencies, and vulnerabilities did not change after adding the refactored code, so the code did not cause any additional errors.

To correct these vulnerability issues it is important to follow best practices and ensure you are using the latest versions of the applications dependencies. In this case both Spring Boot and Maven Dependency Checker were both outdated.

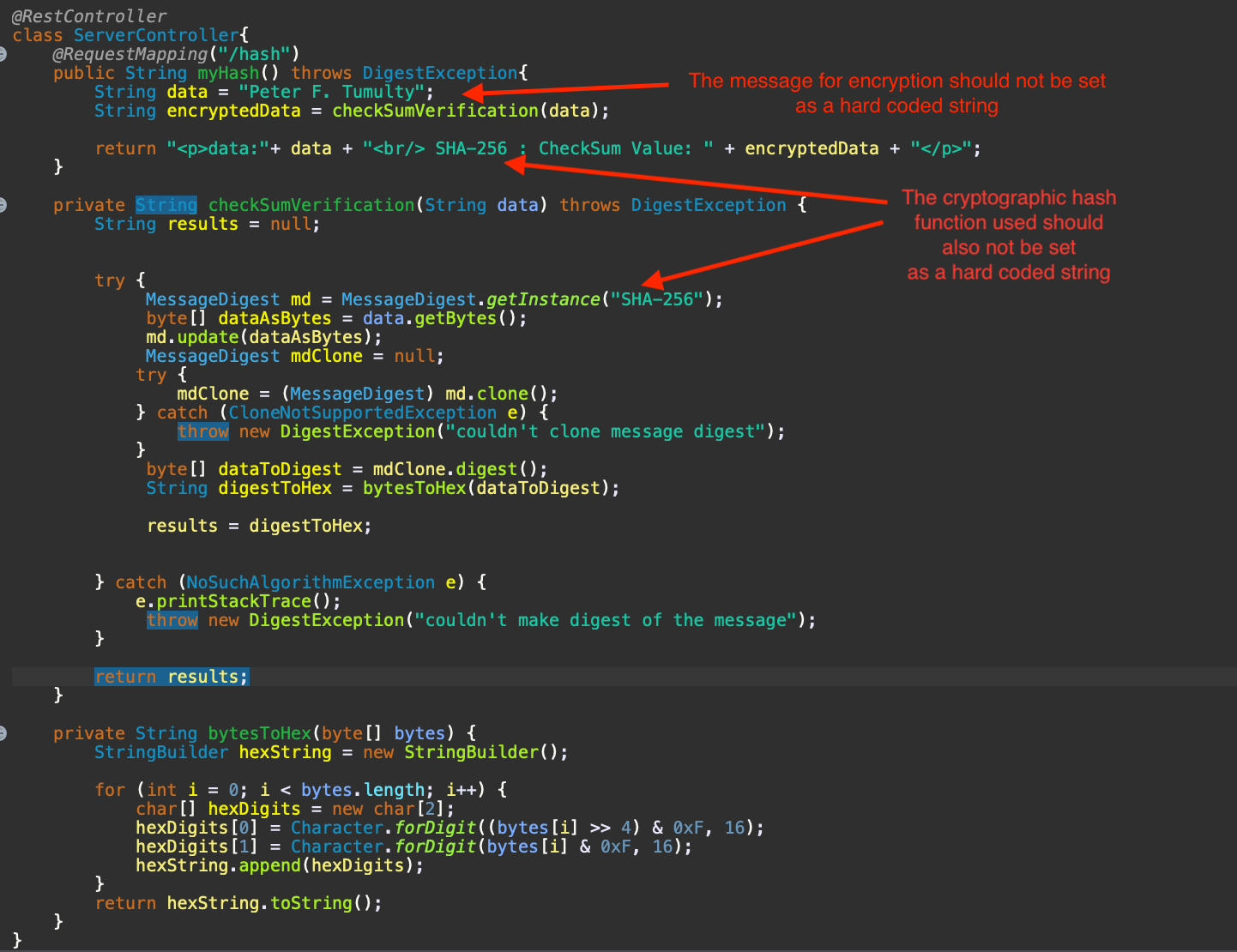
**The results of the dependency vulnerability report after updating the version of Spring Boot to 2.5.15 and Maven Dependency Checker version to 9.0.8**

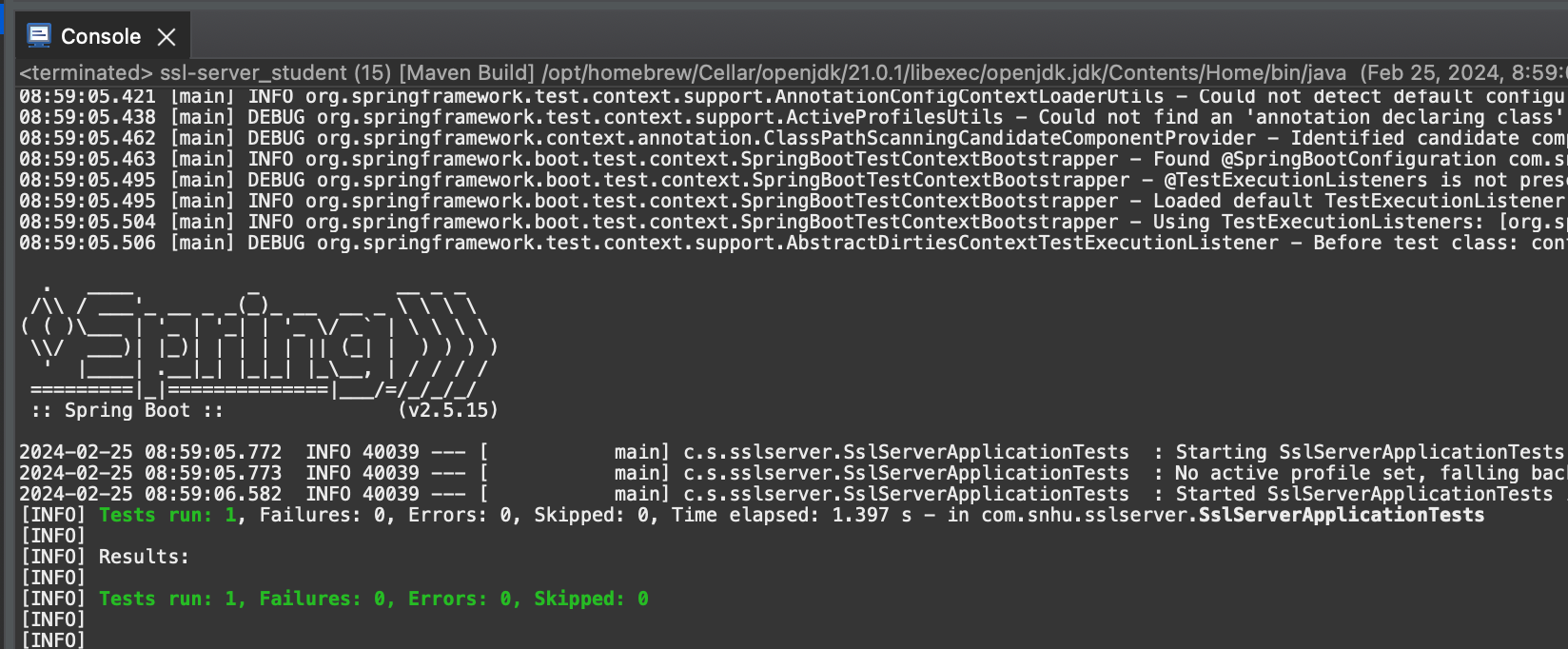
****

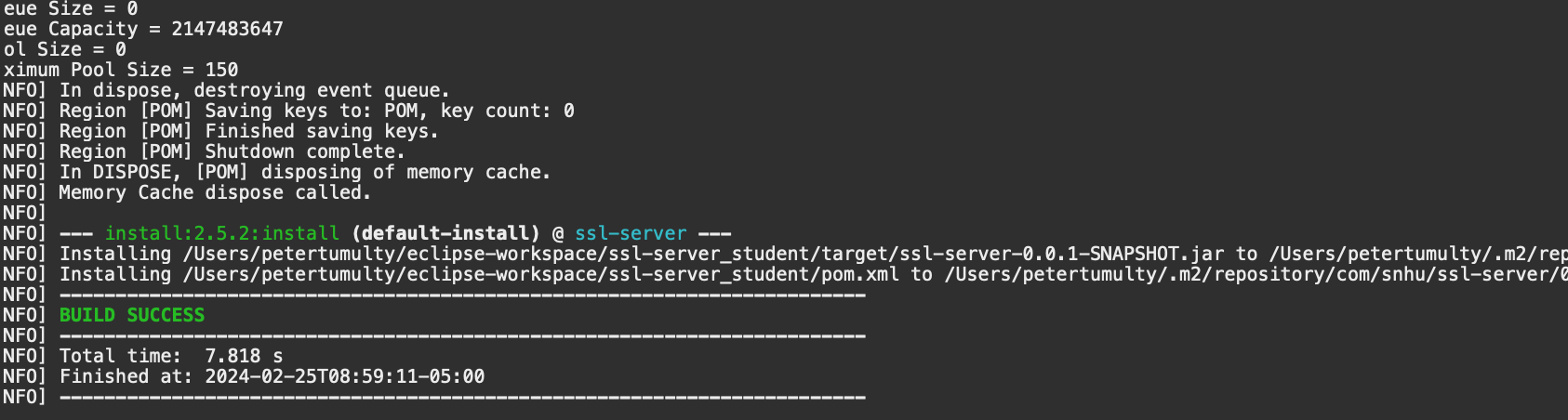


As you can see, after updating the latest stable version of Spring Boot and Maven Dependency Checker that works with Java 1.8 the number of dependencies scanned, vulnerable dependencies, and vulnerabilities was reduced.

## Functional Testing







Overall, the code adheres to secure coding practices from a functional perspective. Based on the screenshots, it can be concluded that the refactored code executes without errors because the test suite runs flawlessly, and the build installation is successful. However, to further enhance the code quality, one should avoid hardcoding the string meant for encryption directly into the Java file. Instead, the message designated for encryption should be imported from a private file. Additionally, although SHA-256 has not yet been compromised, malicious actors could potentially leverage the type of cryptographic hash algorithm as data to infiltrate other aspects of the codebase or other parts of Artemis Financial's system. Therefore, SHA-256 as a string ought to be imported and stored in a private, hidden file, accessible only to the developers.

## Summary

In summary, the refactored code base addresses the following areas of the Vulnerability Assessment Process Flow Diagram: input validation, cryptography, client-server communication, code errors, code quality, and encapsulation. The code base effectively handles input validation because it is written in the Java programming language, a strongly-typed language that ensures the data processed by the API adheres to the data types specified by the developers. Designed to utilize cryptography, the data processed by the API is encrypted using the SHA-256 cryptographic hash function algorithm, while the client interface is secured through a connection with an SSL server certificate. By serving the client interface with an SSL server certificate, the client-server aspects of the Vulnerability Assessment Process Flow Diagram are comprehensively addressed. The aspects of code errors and code quality are covered, as the code employs try/catch blocks to handle unexpected errors, and its writing aligns with the format found in Spring Boot API documentation, ensuring the highest code quality. Finally, the encapsulation aspects, another crucial component of the Vulnerability Assessment Process Flow Diagram, are naturally incorporated due to Java and, by extension, Spring Boot adhering to the Object-Oriented Programming paradigm. Adhering to the pillars of the Vulnerability Assessment Process Flow Diagram ensures the code quality of the application and safeguards the application against vulnerabilities.

## Industry Standard Best Practices

To ensure a high-quality and secure codebase, adhering to industry-standard best practices is paramount. Utilizing SHA-256 for encryption is a crucial first step, as it is a well-recognized and robust algorithm that ensures data is hashed in a manner that makes it impossible to reverse-engineer (Jena, 2023). Employing this cryptographic hash function algorithm is pivotal in maintaining data integrity and security, especially in applications where protecting user information is critical. Additionally, keeping Maven project dependencies up to date is considered a best practice in designing secure software (Van Bekkum, 2016). For instance, updating the versions of Spring Boot and Maven Dependency Checker has been shown to reduce the number of vulnerabilities in an application. Implementing a server certificate (SSL) for client-server communication is also regarded as a best practice when delivering a web application to users via the browser. Google released a report stating that they will not promote any web application that does not use SSL (Barret, 2018). Finally, adhering to coding best practices, such as referencing Spring Boot documentation and implementing try/catch blocks in the code, contributes to the overall quality and manageability of the codebase. Employing these strategies and industry best practices significantly reduces security risks from both a user and developer perspective.

**Reference:**  
  
Jena, B. K. (2023, August 29). *What Is SHA-256 Algorithm: How it Works and Applications*. Retrieved from <https://www.simplilearn.com/tutorials/cyber-security-tutorial/sha-256-algorithm>

Van Bekkum, J. (2016, May 20). Keeping dependencies up-to-date in Maven. Xebia. [https://xebia.com/blog/keeping-dependencies-up-to-date-in-maven/](https://xebia.com/blog/keeping-dependencies-up-to-date-in-maven/%5D(https://xebia.com/blog/keeping-dependencies-up-to-date-in-maven/))

Barrett, B. (2018, July 24). Google Chrome Will Start Using a Not Secure Label for All HTTP Sites. *Wired*.<https://www.wired.com/story/google-chrome-https-not-secure-label/>