****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.lawew3gwddlc)

[**Client 3**](#_heading=h.eeenau3alk8z)

[**Instructions 3**](#_heading=h.rwoq8esj4p9q)

[**Developer 4**](#_heading=h.foxeww3uykoy)

[**1. Algorithm Cipher 4**](#_heading=h.v6yjgrs4174)

[**2. Certificate Generation 4**](#_heading=h.3o44g6j5z7ka)

[**3. Deploy Cipher 4**](#_heading=h.jihmvckb1trq)

[**4. Secure Communications 4**](#_heading=h.t9n2zisl4dva)

[**5. Secondary Testing 4**](#_heading=h.jlaozkan7dv3)

[**6. Functional Testing 4**](#_heading=h.ilrgomn4q689)

[**7. Summary 4**](#_heading=h.on31ja3gvt9a)

[**8. Industry Standard Best Practices 4**](#_heading=h.2qhra1vkawv4)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **Ryan Grunest** |  |

## 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

Ryan Grunest

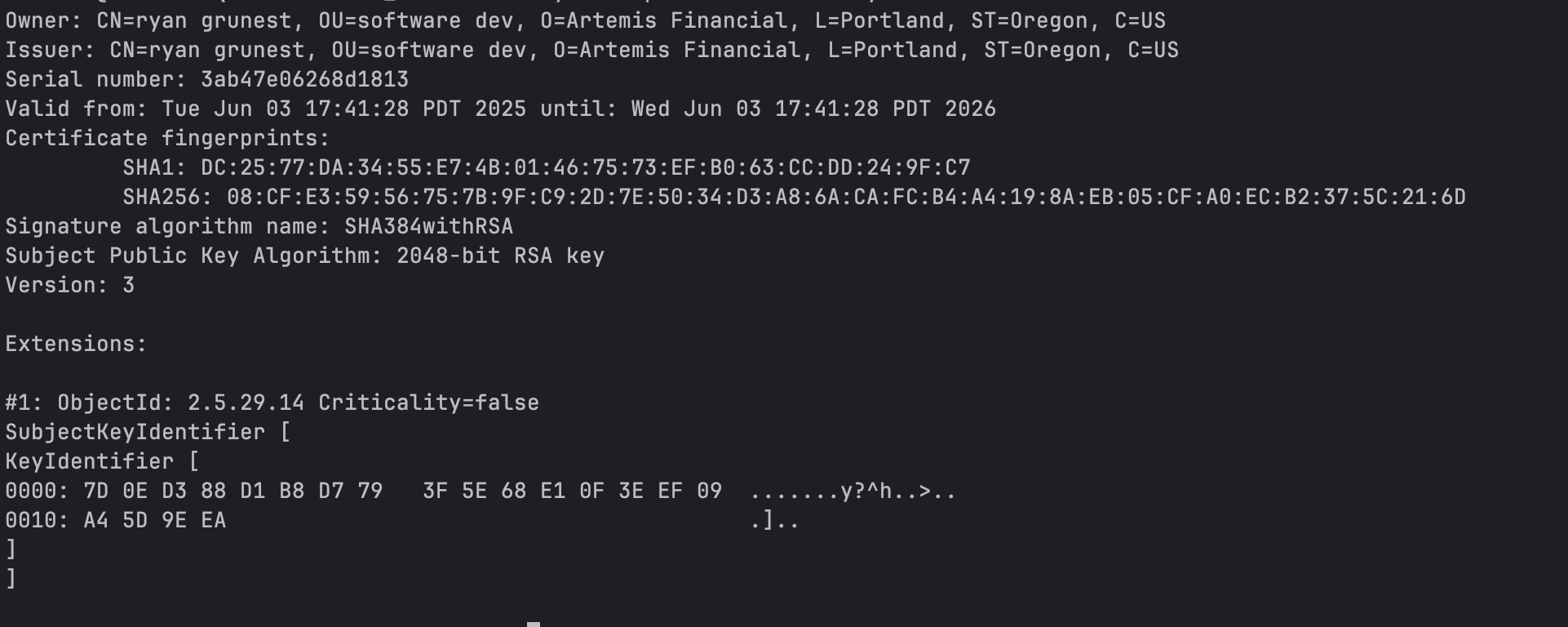
## Algorithm Cipher

Given the security vulnerabilities, I recommend using SHA-256 for checksum validation. This will allow Artemis Financial to verify that data transmitted between the client and server has not been altered. SHA-256 is a hash function that encrypts data using a 256 bit hash value. Because it is a 256 bit hash value, there are minimal collisions, allowing higher security.

Random numbers are used to generate secure keys, initialization vectors, and nonces. This allows values in cryptography to be less predictable and more secure. Symmetric keys use the same key between client and server in order to encrypt and decrypt information. Asymmetric keys use public and private keys in order to encrypt and decrypt information.

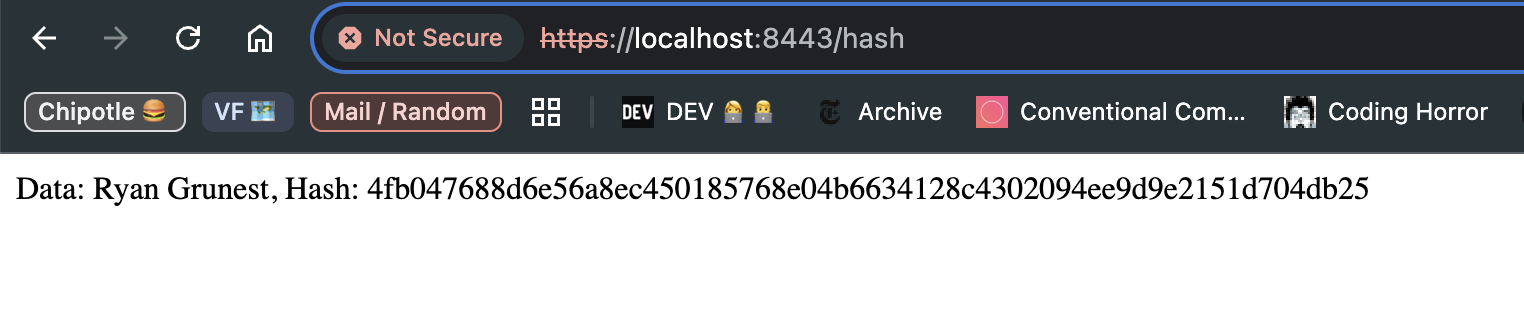
## Certificate Generation

Insert a screenshot below of the CER file.



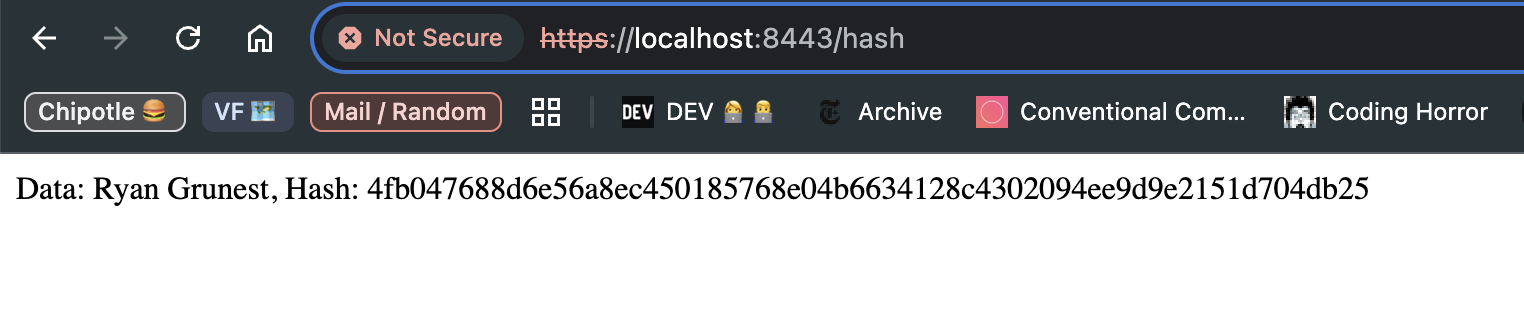
## Deploy Cipher

Insert a screenshot below of the checksum verification.



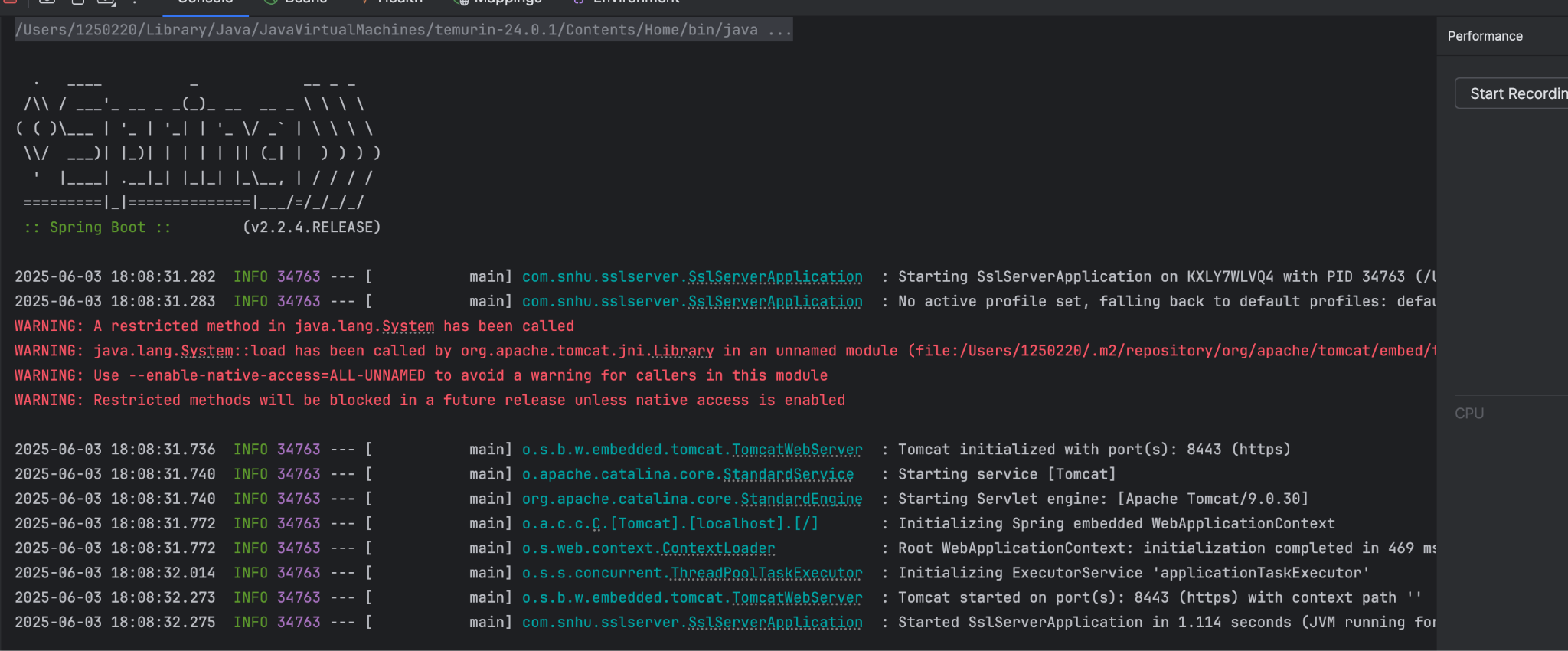
## Secure Communications

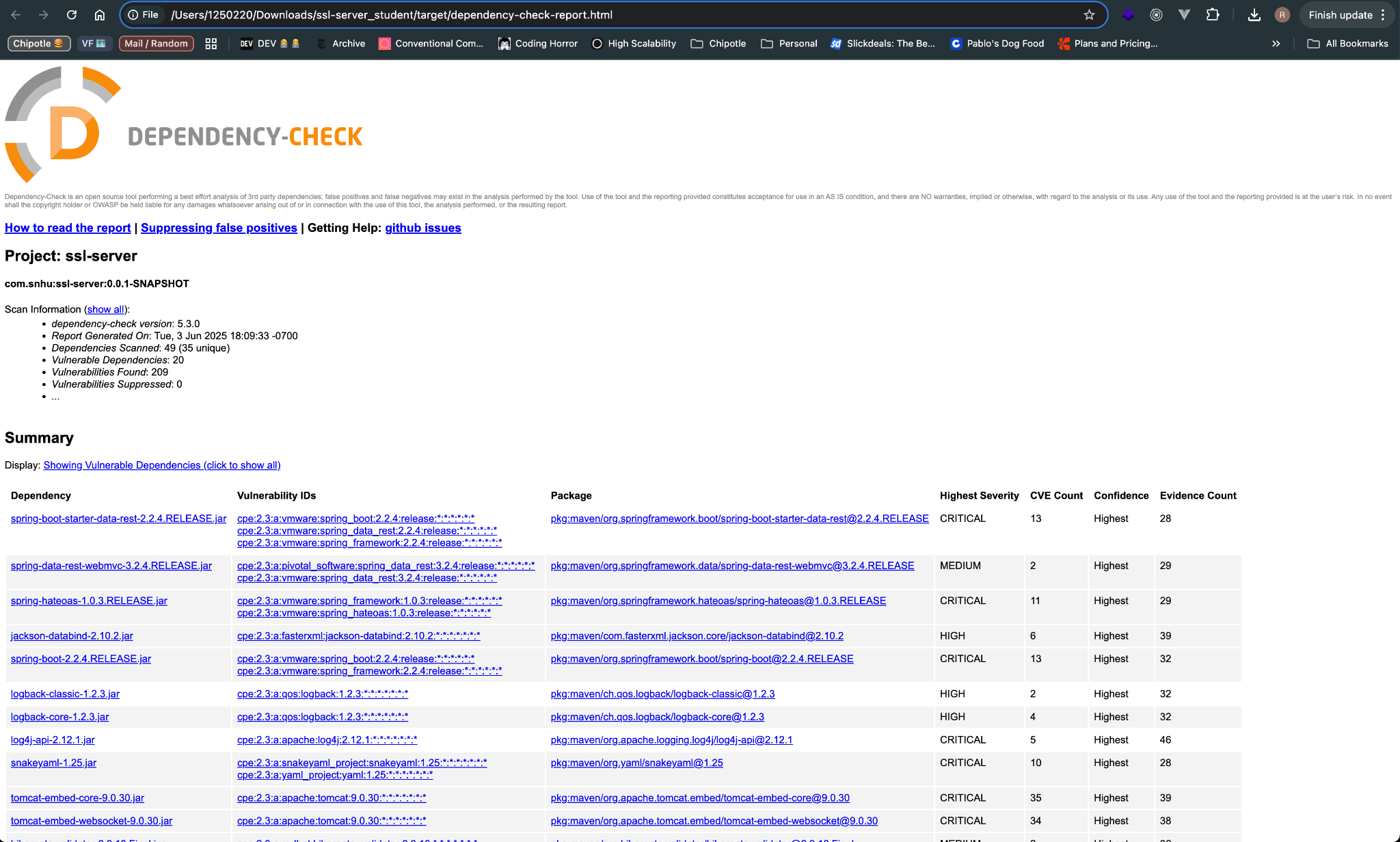
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

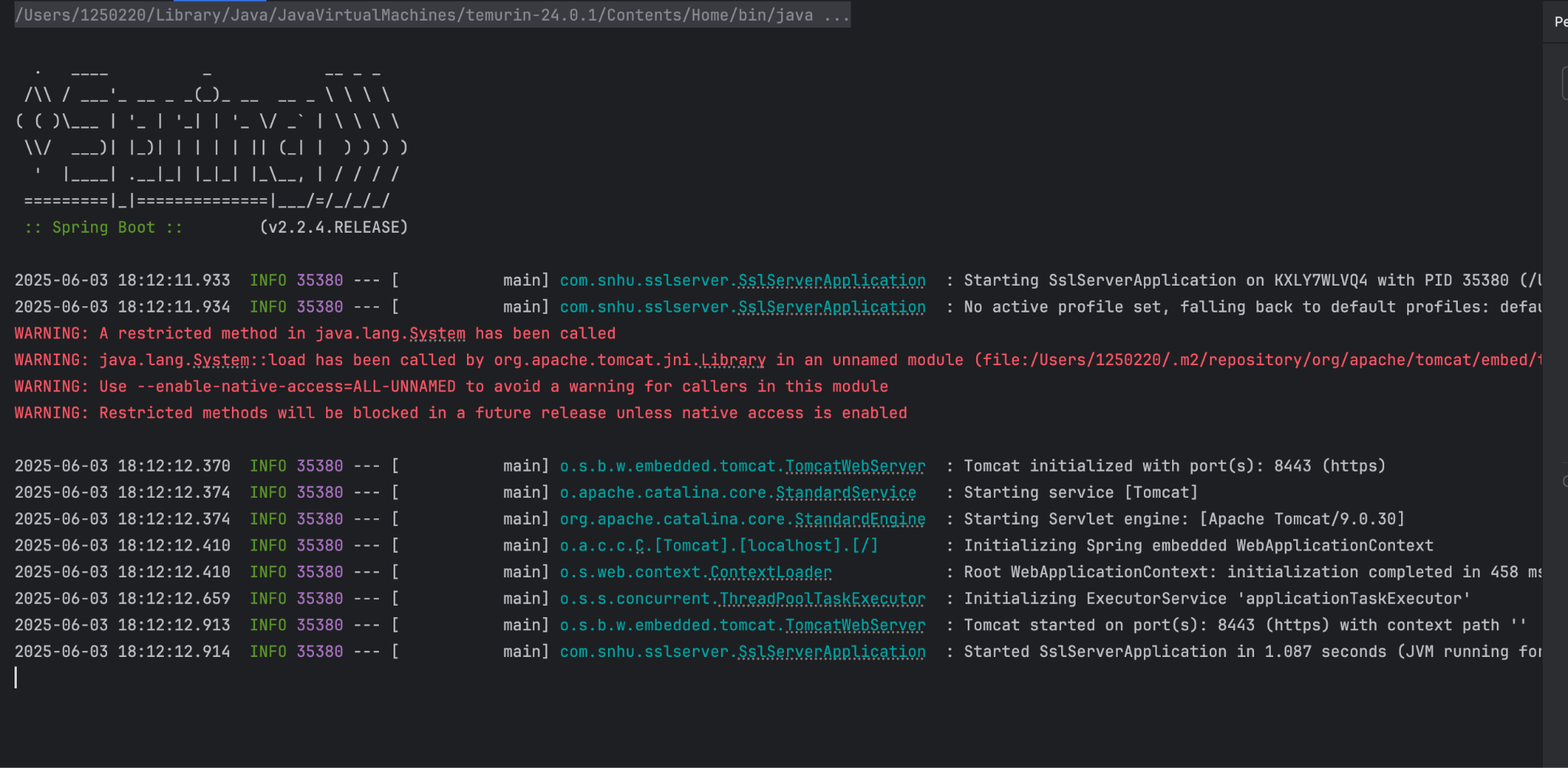
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

I used cryptographic hashing using the SHA-256 hash algorithm to generate a checksum. This demonstrates that the application is capable of securely handling data. The application is also using SSL encryption via a self signed certificate. This allows the client and the server to securely communicate to each other cryptographically.

## Industry Standard Best Practices

This application now uses an industry standard cryptographic hash algorithm, SHA-256, to ensure data integrity. On top of that, the application is now serving data via HTTPS, ensuring that the client and server communicate securely. Using OWASP’s dependency check, I was able to verify that no new vulnerabilities were created when implementing HTTPS and SHA-256. I also implemented a try / catch block in order to log any potential errors if the hash function was not available.