****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_slv0m8twxq05)

[**Client 3**](#_i3y68lkc6u37)

[**Instructions 3**](#_yzafj56fzj3c)

[**Developer 4**](#_1auwesmp8mvf)

[**1. Algorithm Cipher 4**](#_qg91q8z2s0f)

[**2. Certificate Generation 4**](#_wq98yzt59edo)

[**3. Deploy Cipher 4**](#_4saazptb7nhx)

[**4. Secure Communications 4**](#_ni7li8kseym)

[**5. Secondary Testing 4**](#_w1a64jiuoxkk)

[**6. Functional Testing 4**](#_t0924iqqeiyd)

[**7. Summary 4**](#_duabpub40p0c)

[**8. Industry Standard Best Practices 4**](#_yxx7i3onxqw)

## 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

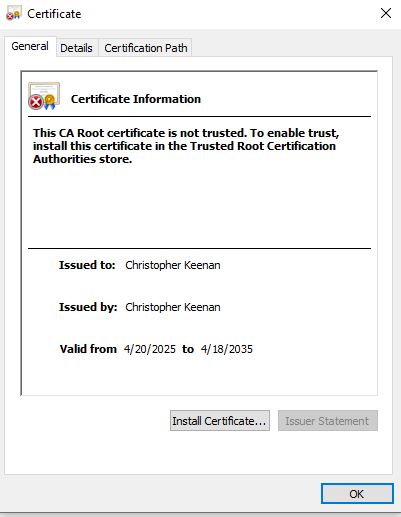
Christopher Keenan

## Algorithm Cipher

For this project, I implemented the SHA-256 hashing algorithm in order to generate a checksum for my chosen string of data. SHA-256 is generally widely trusted for its properties of both balance of security and efficiency. It is currently one of the most common algorithms that is chosen in secure communications. This algorithm generates a 256-bit, or 32 byte hash value, which makes it a strong choice against both collision and brute-force attacks. It does not use encryption keys (unlike AES or RSA) because it's a one-way hash function which is perfect for verifying file integrity. This aligns with Artemis Financial’s goals of ensuring secure data verification.

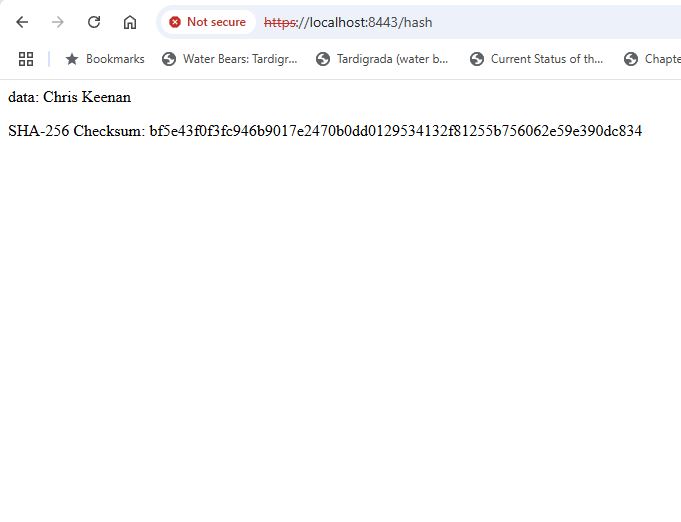
## Certificate Generation

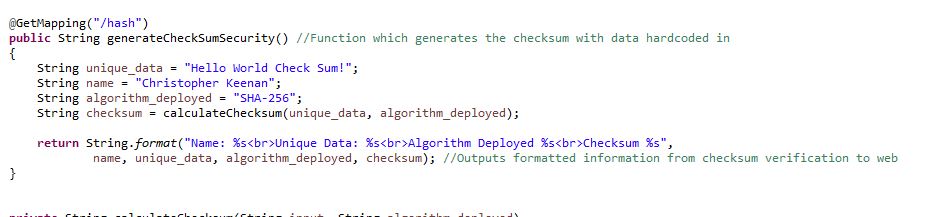
Insert a screenshot below of the CER file.



## Deploy Cipher

Insert a screenshot below of the checksum verification.

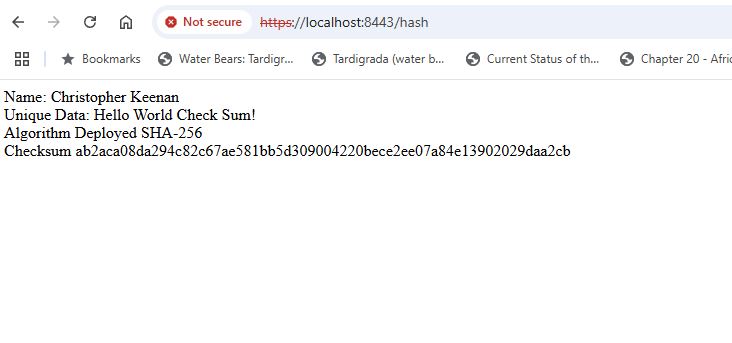




The block I wrote which sets values for the unique data string, my name, the type of algorithm employed, and calls the calculateChecksum function I implemented to hash the data.

## Secure Communications

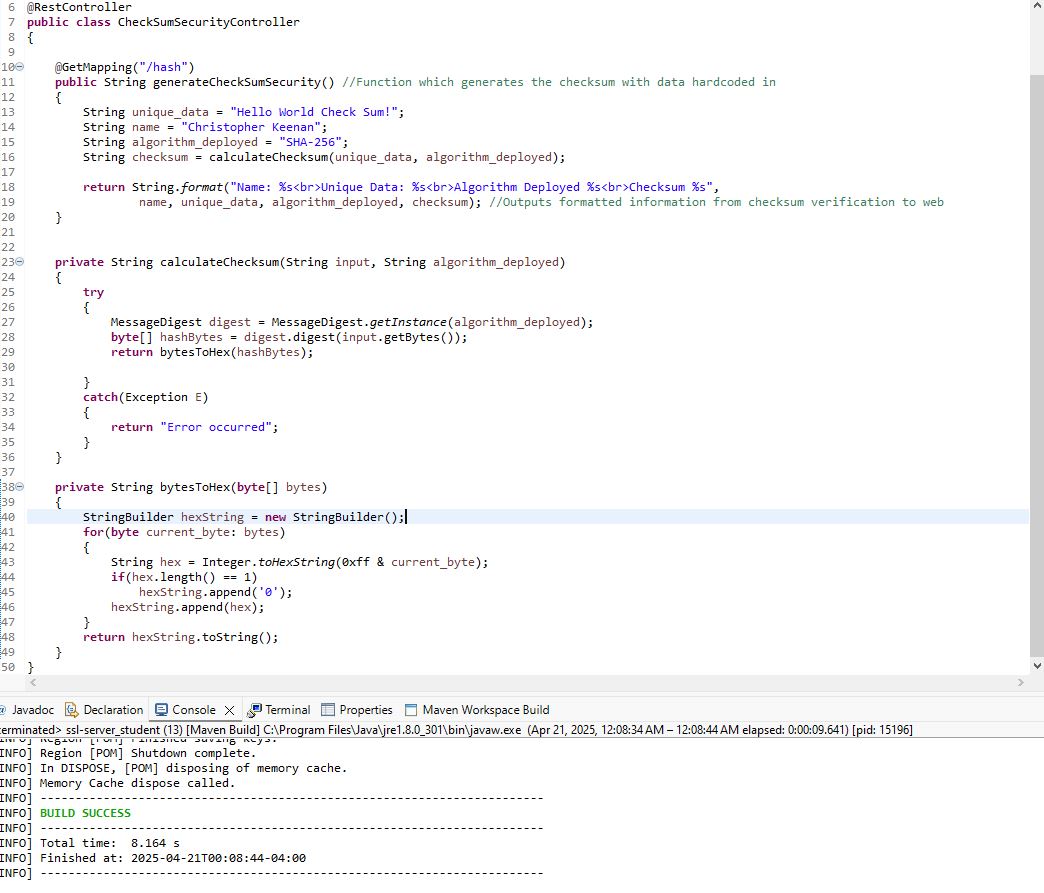
Insert a screenshot below of the web browser that shows a secure webpage.



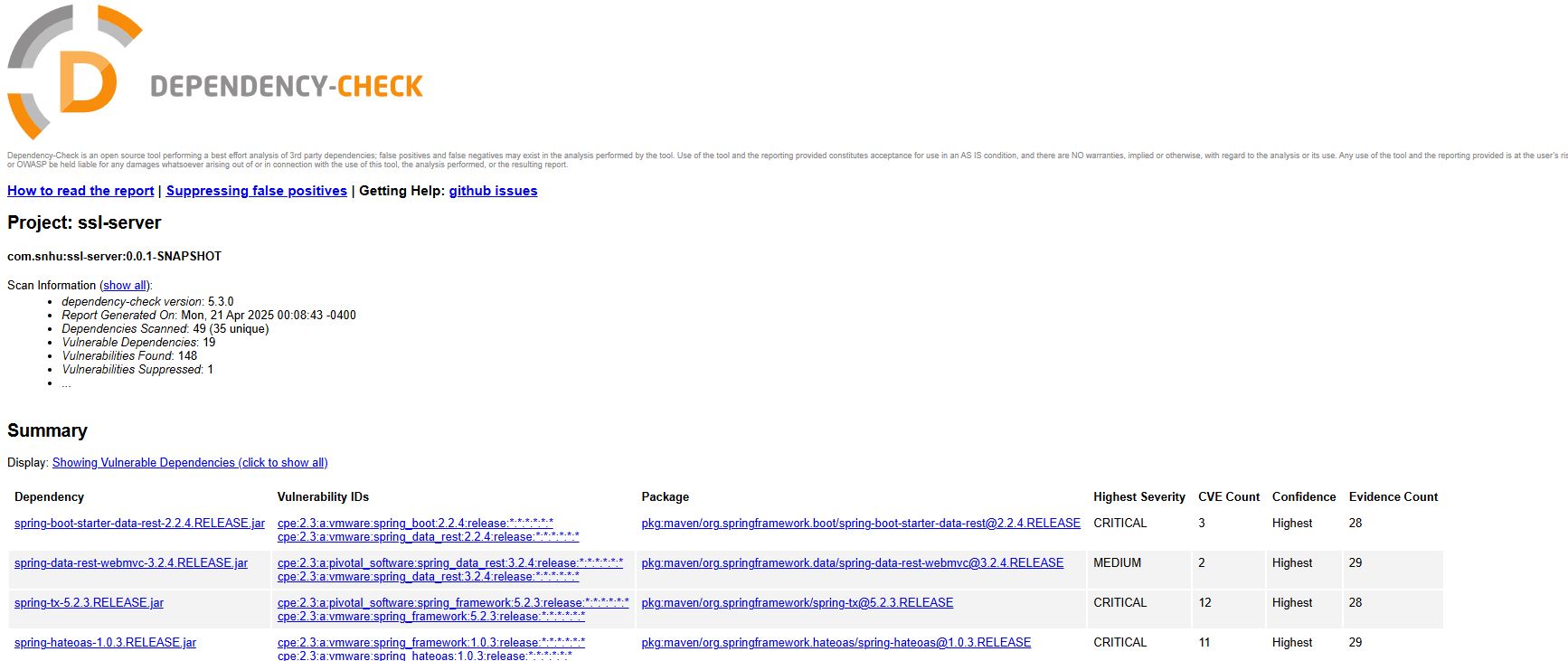
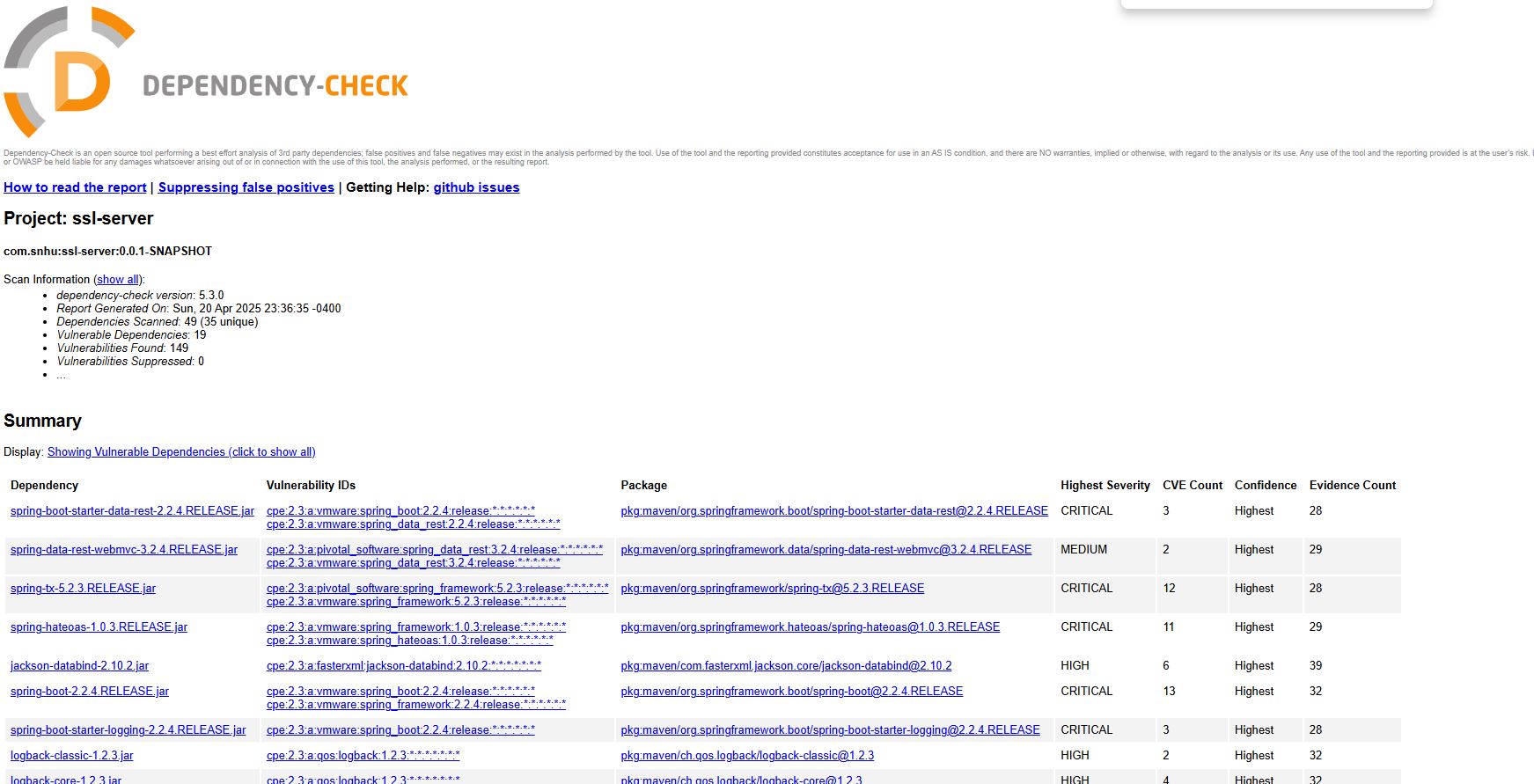
Note - could not get lock icon to display, but it is because I used a self signed certificate to access the local host, and it is still routed through https.

## Secondary Testing

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



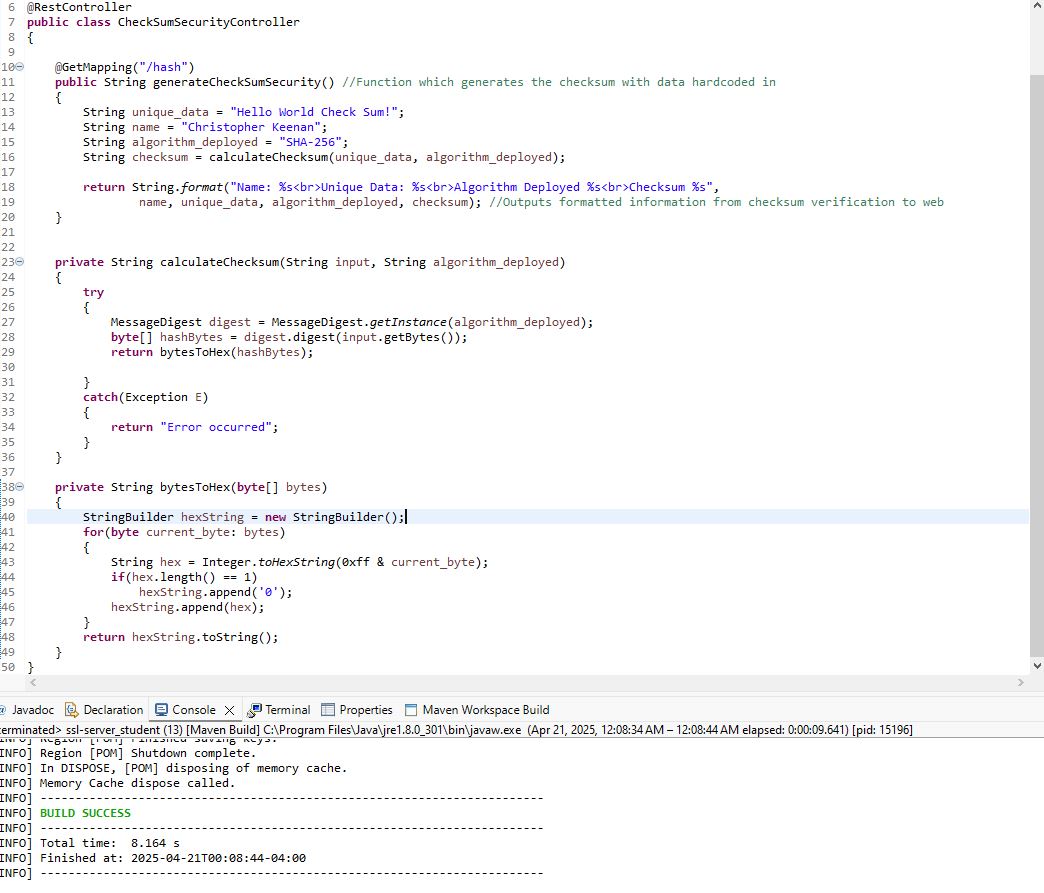
Minus some necessary imports, this is the refactored code which includes all my functions for setting and mutating data. It also includes the output conversion code which translates the hashed values into readable text.



Above are before and after of dependency check reports.

## Functional Testing

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



The dependency check report summaries are included, both before and after suppressing a false positive. Again, this is my code refactored to provide hashing and secure communication.

## Summary

In my effort to meet Artemis Financial’s requirements, I refactored the original codebase by creating a secure checksum verification endpoint which utilizes Spring Boot and Java. To do this, I used SHA-256 to generate the checksum, then exposed the logic via a secure HTTPS endpoint, and ensured the functionality both in the browser and IDE. All of my changes were made following secure coding practices, which ensured proper exception handling, HTTPS configuration, and clean compartmentalized structure. I then conducted a dependency check to confirm that the refactor didn’t introduce any new vulnerabilities.

## Industry Standard Best Practices

During this project, I followed all standard best practices in secure software design learned throughout the course. This includes using a universally reliable hashing algorithm (SHA-256), implementing and deploying HTTPS for a secure connection, and incorporated a static analysis via the OWASP Dependency Check plugin. These steps all served to reduce the attack surface of the application and improve security in the software’s life cycle. Applying these practices doesn’t just make the application resistant from attacks, but also reflects the principles of building for the long run, where security is embedded in the DLC from day one.