****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

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

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **October 15, 2022** | **Lucas de los Santos** |  |

## 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

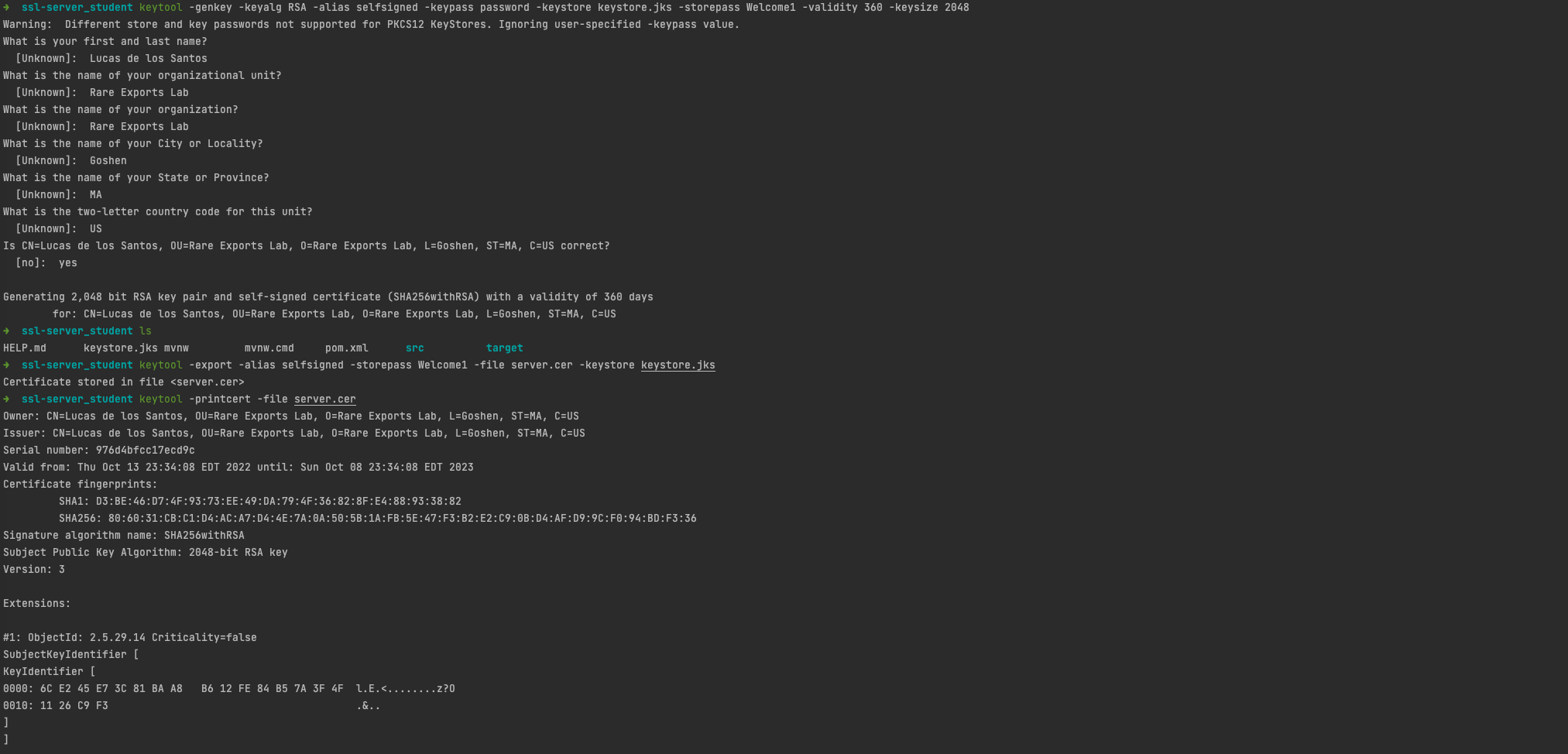
Lucas de los Santos

## Algorithm Cipher

SHA-256 is a creation of the National Security Agency that has withstood collision attacks for some time. SHA-256 is currently being used all over the internet and cryptocurrency. SHA-256 undergoes 64 rounds of data scrambling through a multi-step process. In terms of collision, the total probability of two hashes accidentally colliding is approximately 4.3 \* 10^-60. From that probability, it should tell you that collisions aren’t really considered a factor in the equation.

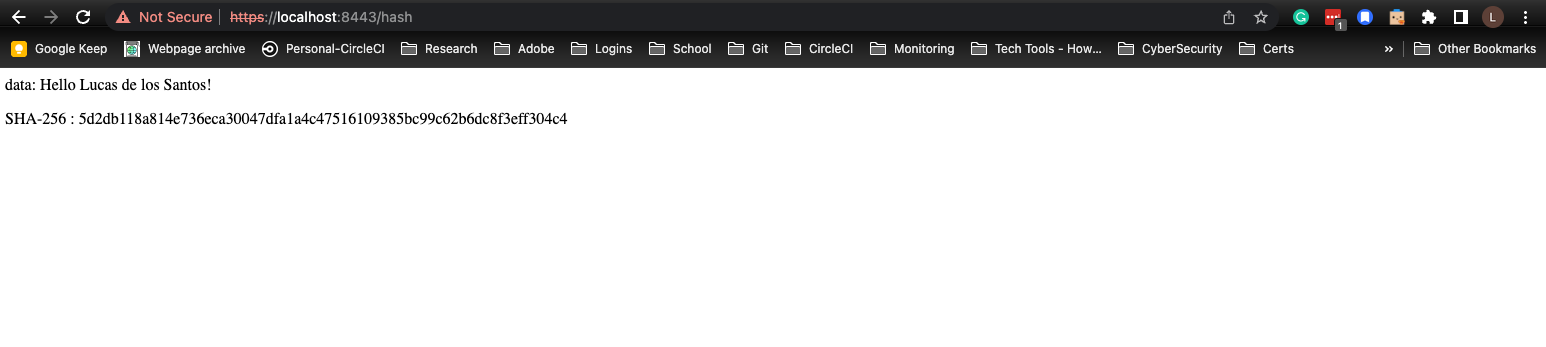
## Certificate Generation

Insert a screenshot below of the CER file.



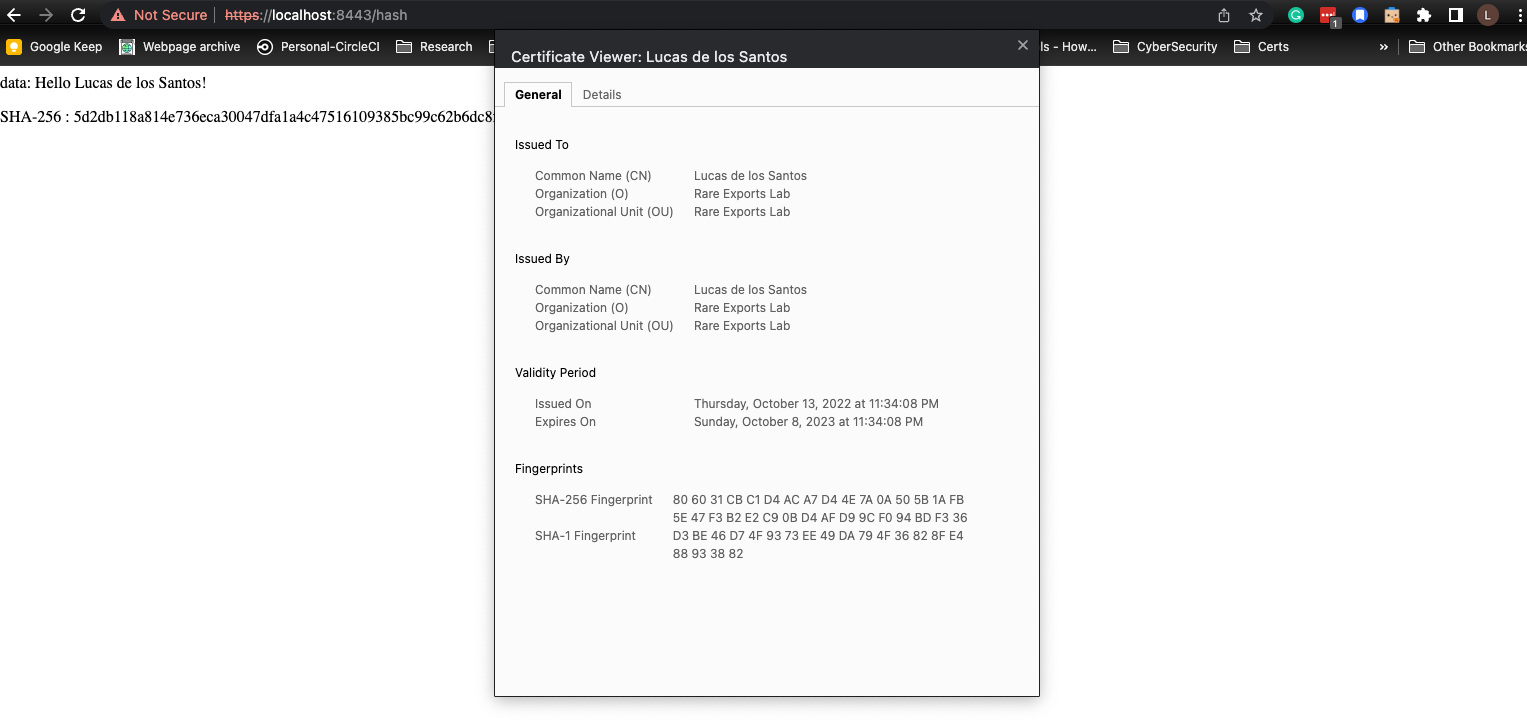
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

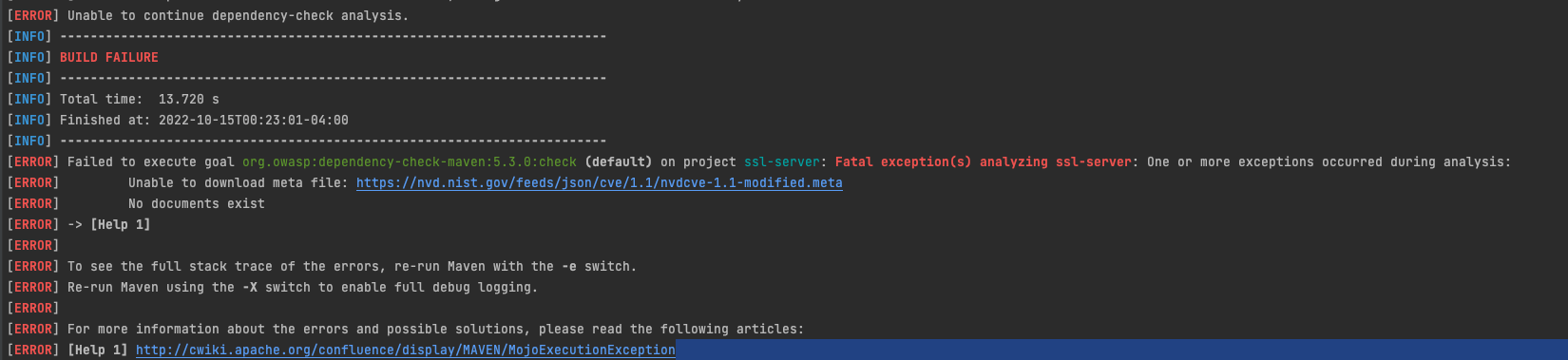
Insert a screenshot below of the web browser that shows a secure webpage.

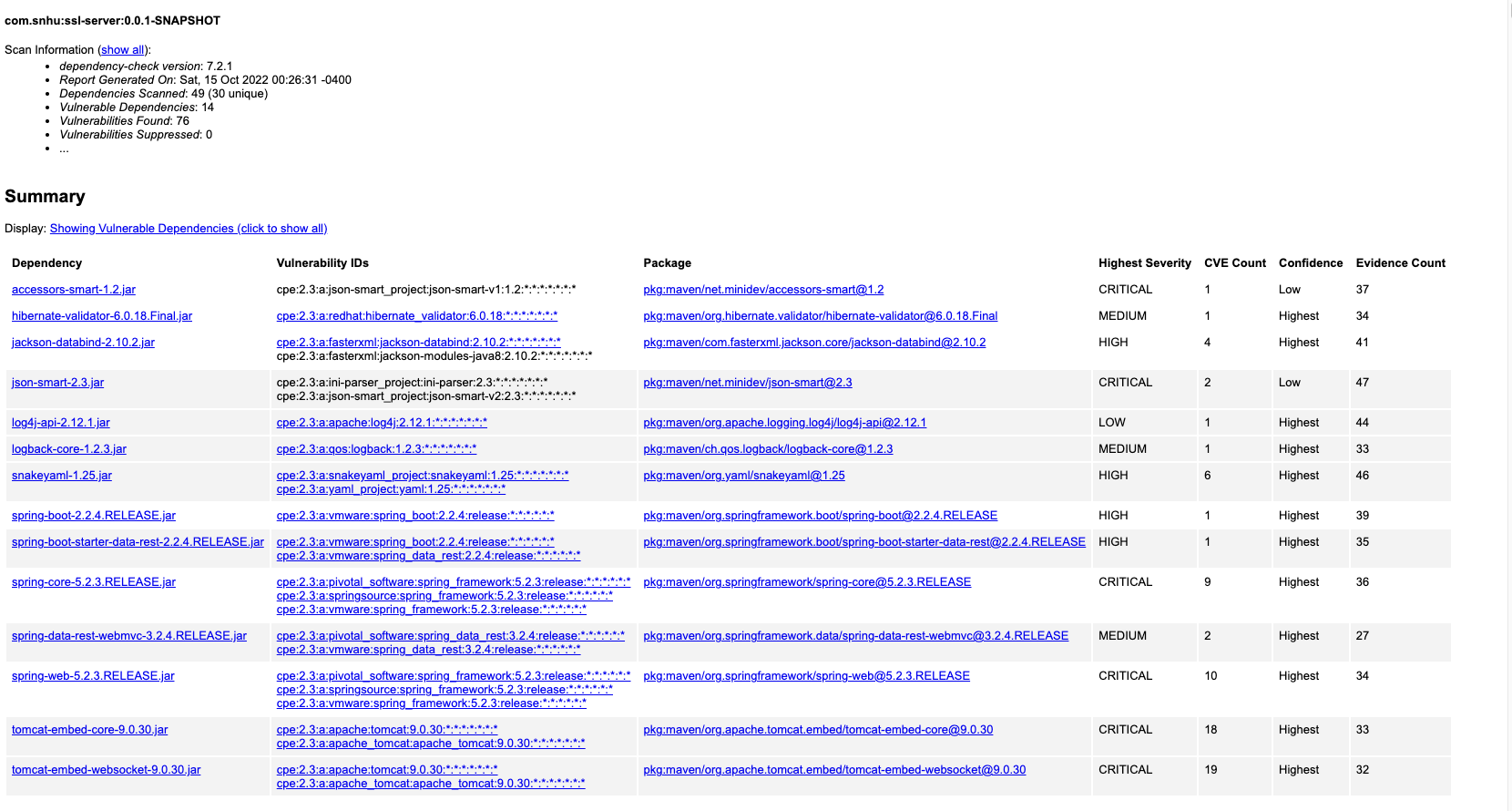


## Secondary Testing

Previous to refactoring (base code):

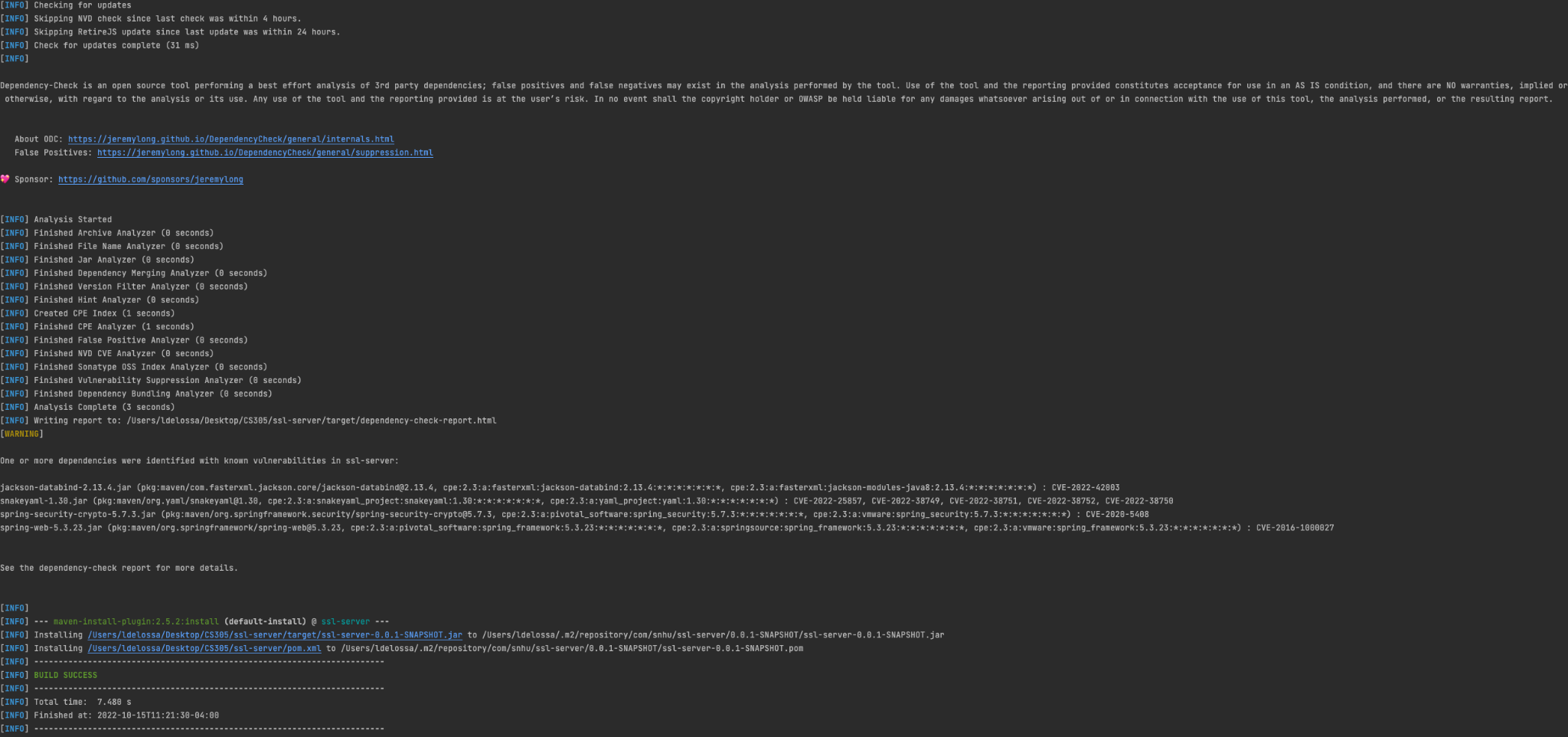
* Code failed to compile more often than not due to v5.3.0 OWASP dependency-check plugin.
* Solution: Upgraded plugin to v7.2.1
* Original code reported 76 vulnerabilities.





Post refactoring:

* Added Spring-boot-starter-security plug v2.7.4
* Updated Spring-boot-starter-parent to v.2.7.4
* Added keystore.p12 to resources
* Updated application.properties for SSL connection
* created SecurityConfig class under config/ that requires a secure channel for all requests.
* Added ServerController class to define SHA-256 Checksum creation of static string.
* Created HelloController class for the main greeting if entering just <https://localhost:8443>

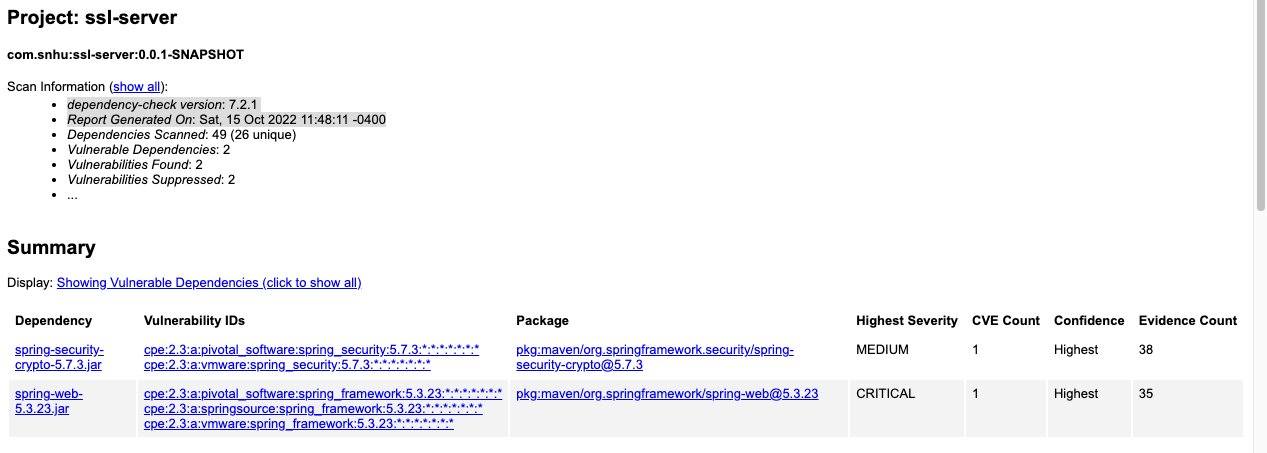


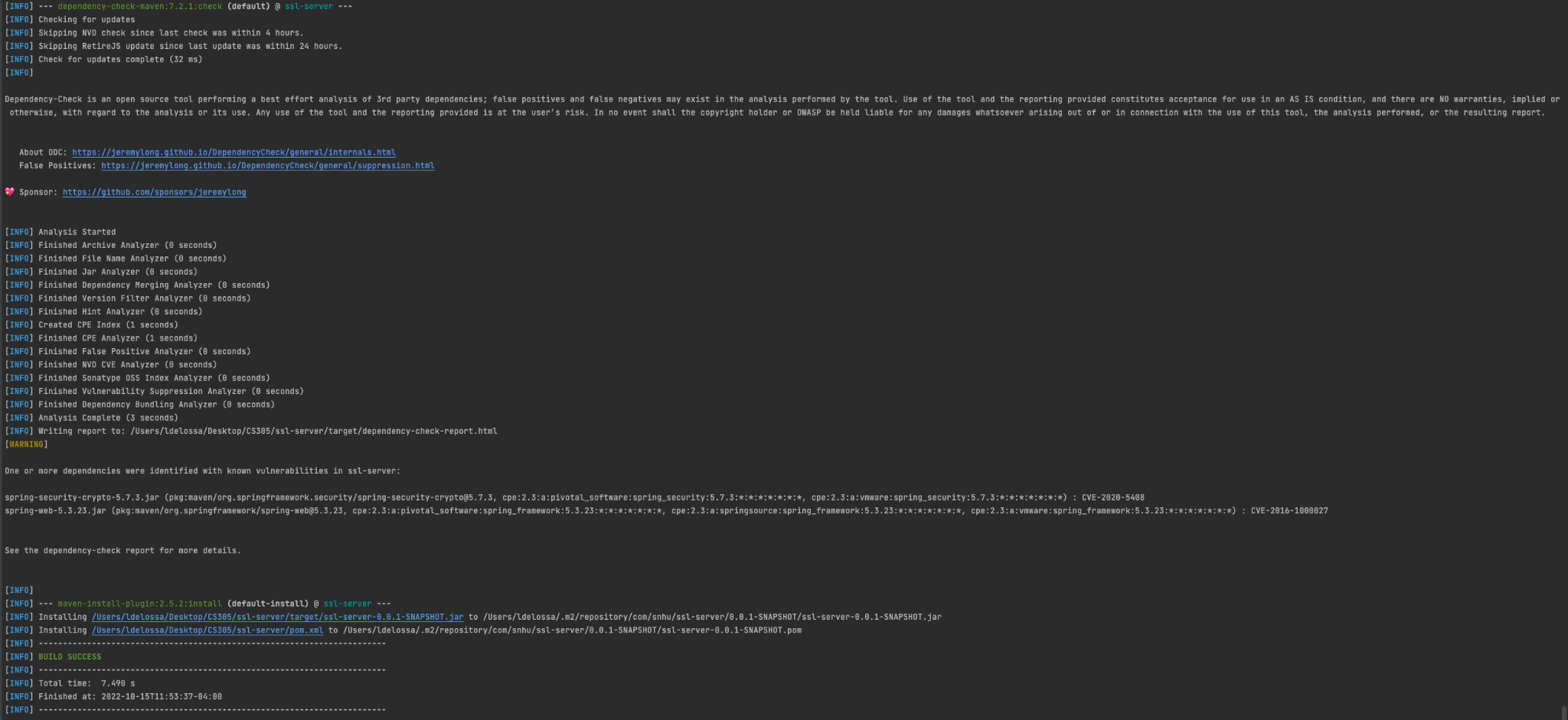


## Functional Testing

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

* False Positive: CVE-2022-42003: jackson-databind-2.13.4.2.jar (Fixes appear to be in the 2.13.4.x versions and backported into 2.12.17.x series)
  + In FasterXML jackson-databind before 2.14.0-rc1, resource exhaustion can occur because of a lack of a check in primitive value deserializers to avoid deep wrapper array nesting, when the UNWRAP\_SINGLE\_VALUE\_ARRAYS feature is enabled. Additional fix version in 2.13.4.1 and 2.12.17.1
* False positive: CVE-2022-38752: snakeyaml-1.3.3.jar
  + Overrided Snakeyaml version to v1.3.3
  + No known vulnerabilities based on <https://nvd.nist.gov/vuln/detail/CVE-2022-38752>
  + OWASP still flags it but the CVE applies to versions up to but excluding 1.3.2, therefore, a false positive so will be suppressed.
* False positive: CVE-2020-5408: spring-security-crypto-5.7.3.jar.
  + Spring Security versions 5.3.x prior to 5.3.2, 5.2.x prior to 5.2.4, 5.1.x prior to 5.1.10, 5.0.x prior to 5.0.16 and 4.2.x prior to 4.2.16 use a fixed null initialization vector with CBC Mode in the implementation of the queryable text encryptor. A malicious user with access to the data that has been encrypted using such an encryptor may be able to derive the unencrypted values using a dictionary attack.
  + To suppress would require excluding multiple identifiers
* Potential False Positive due to lack of HTTP endpoints: CVE-2020-5408: spring-web-5.3.23.jar
  + Based on <https://support.contrastsecurity.com/hc/en-us/articles/4402400830612-Spring-web-Java-Deserialization-CVE-2016-1000027> there has been no fix for the vulnerability when it was originally flagged at version 4.1.4. Only 4.1.4 is listed as vulnerable but it is continuously flagged. As there are no HTTP endpoints that appears to be the remediation.
  + To suppress would require excluding multiple identifiers





## Summary

The base code has been reworked to include Spring Security for handling enforcement of HTTPS-only traffic. A self-signed cert, strictly for development purposes, has been added to the resources that will be used to allow HTTPS traffic. Based on an early scan of the code pre additions it was determined that the use of such older versions of spring-boot-starter-parent flagged 76 vulnerabilities. An update to a more recent version reduced the attack vector to 8 vulnerabilities. At the moment all the input validation is static as the code rendered by Spring Boot is based on static hardcoded values. The APIs across the board all require HTTPS. Error handling in Spring Boot is a feature included so should a user type an incorrect URL it will throw a 404 indicating the page isn’t found. Exceptions are created within the bytesToHex function to throw a secure error should an algorithm unavailable to the environment be requested.

After the coding portion was complete and it was confirmed to run without error both with and without tests enabled it was time to reevaluate the dependency-check report. At this point, 4 dependencies were flagged. After doing research it was determined most if not all were false positives and 2 of 4 have been suppressed. The remaining 2 have multiple identifiers but based on research it seems fairly clear that due to using HTTPS spring-web would not be an issue. As for spring-security-crypto, it appears to be flagging even though it’s at a version higher than CVE includes.

As not part of this assignment but would complete it more would be proper code coverage to better determine code quality. From a review and a SonarLinting of the code, there is nothing being thrown but both unit and integration tests would benefit greatly toward the security goals. From a CI/CD standpoint utilizing 3rd party tools such as Sonarqube(Continous Code Quality tool) and Snyk(Dependency Monitoring) placed within a Jenkins pipeline job to allow for a more extensive scan. Followed by a downstream of rigorous testing such as end-to-end tests, security testing, etc.

## Industry Standard Best Practices

As this is a reasonably small example of code the main implementation was ensuring HTTPS was enforced for all connections. This establishes a guideline of fully encrypted traffic to protect user data. A common theme that comes up amongst multiple secure industry coding practice is utilzing multiple static anaylizers in your code to gain insight and catch potentially insecure coding practices to begin with. In Maven one can use many different options such as Cobertura, Spotbugs, PMD, and Jacoco. In many cases if an issue is found these plugins can be leveraged to fail a build so a developer has visibility and can address the issues. In this way it acts as a gate that discourages poor code from being checked into the main or trunk branch. Putting systems in place and proper tools that give meaningful insight such as code coverage reports, bug finders, syntax error finders, etc makes all the difference in aiding your team to succeed.