

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 5](#_Toc102040758)

[2. Certificate Generation 6](#_Toc102040759)

[3. Deploy Cipher 6](#_Toc102040760)

[4. Secure Communications 7](#_Toc102040761)

[5. Secondary Testing 7](#_Toc102040762)

[6. Functional Testing 9](#_Toc102040763)

[7. Summary 1](#_Toc102040764)0

[8. Industry Standard Best Practices 1](#_Toc102040765)1

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **4/13/2023** | **Charles Breuer** |  |

## 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

Charles Breuer

## Algorithm Cipher

**Algorithm Cipher**

**Choice: SHA3-512**

**Justification**

The type of algorithm needed to implement a checksum data verification step is a hashing function. This type of function produces a unique hash code based on the data’s structure (called the digest). The digest can be used to verify the integrity of data being transferred over insecure networks (Menezes, van Oorschot, & Vanstone, 1996, p. 348). The best recommendation for a hashing function in this scenario is the SHA3-512.

The SHA3-512 hash function is a part of the SHA3 family. This family was created as a response to growing concerns that the SHA2 family may have collision issues in the future (Bernstein & Lange, 2012). While no actionable collision issues have been documented on the SHA2 family, exploits have been found such as the length extension attacks where given H(m), it is easy to find H(m||m’), which makes message authentication not secure. This issue partially motivated the development of HMAC (NIST, 2012). The SHA3 family provides a new internal architecture that reduces the possibility of cryptanalysis attacks and collision attacks.

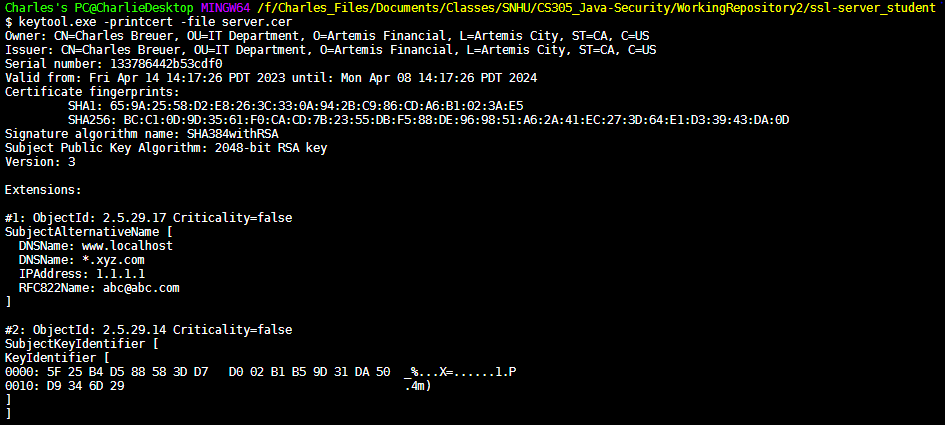
Hashing functions are a family of ciphers that were typically used for indexing and searching for data in early computers. While hash functions are still used in this manner, cryptography utilizes hash functions to create fixed-sized outputs that are difficult to reverse and avoid collisions. Previously popular algorithms included the MD4, MD5, SHA-1, and SHA-2.

The 512 portion of SHA3-512 represents the bit size of the digest and half the block size of data the algorithm processes at one time (NIST, 2015). While SHA3-256 is known for being a faster implementation of the algorithm, the 512 variant provides a higher level of security (Bernstein & Lange, 2012). Since the data being sent includes customer sensitive information, it is important to prioritize security.

Hashing functions do not use random numbers since they take a deterministic approach to create a particular digest for a given set of input data (Menezes et al., 1996). Hashing functions also do not use keys in any capacity since encryption occurs in only one direction.

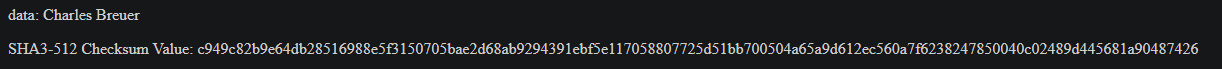
## Certificate Generation

Insert a screenshot below of the CER file.



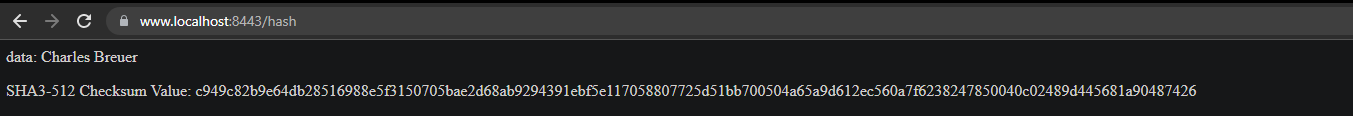
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

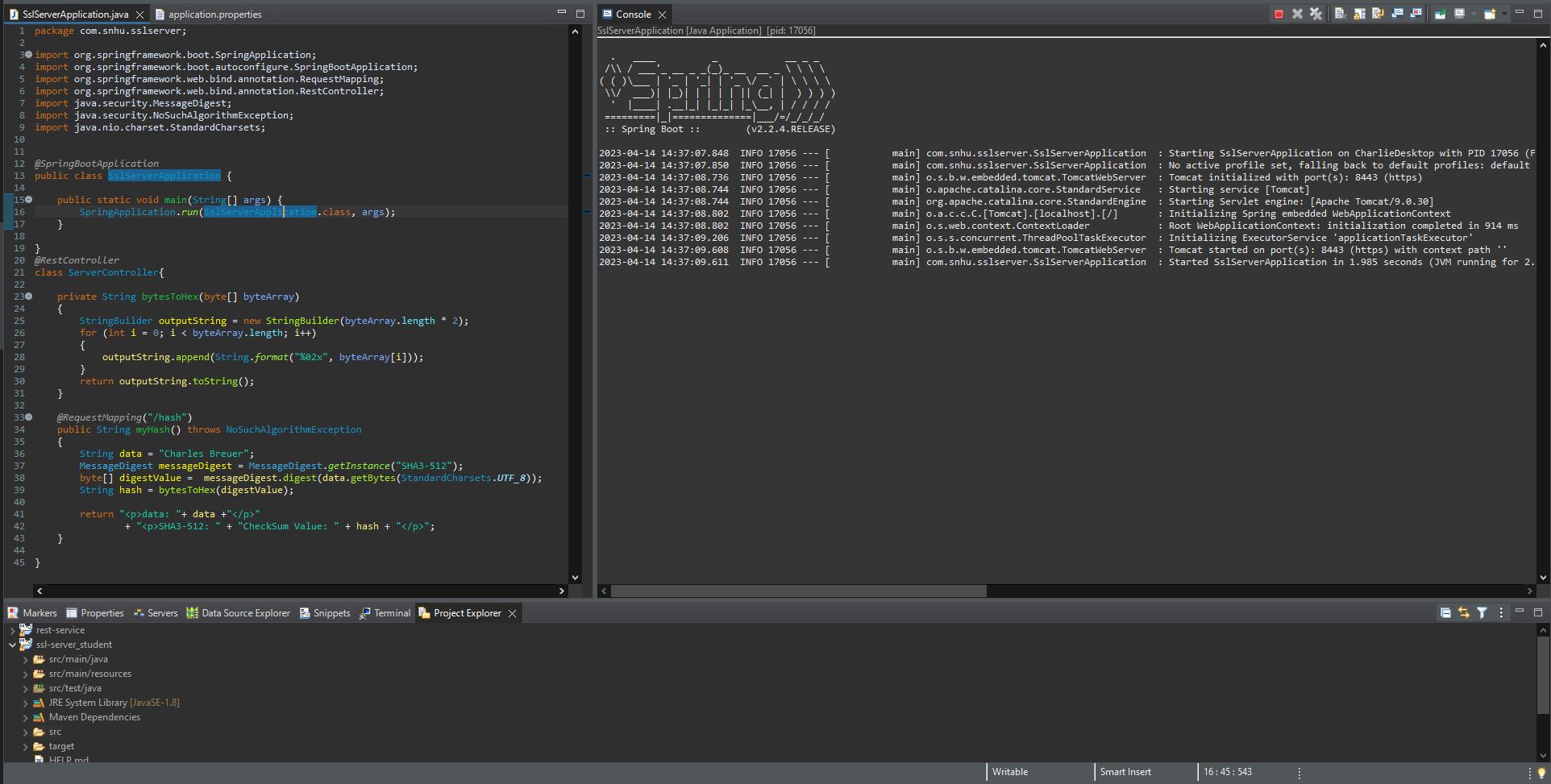
Insert a screenshot below of the web browser that shows a secure webpage.



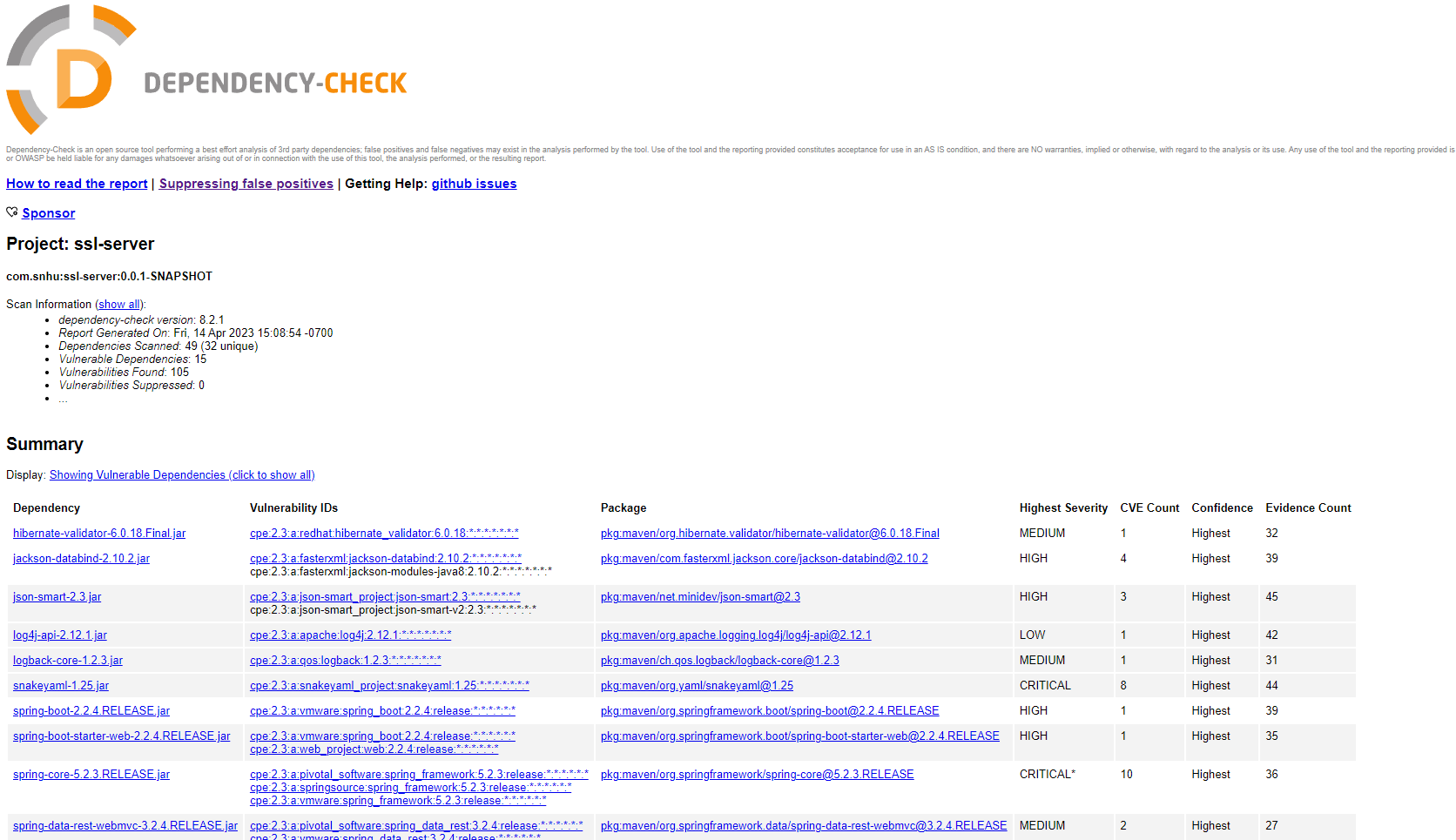
## Secondary Testing

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

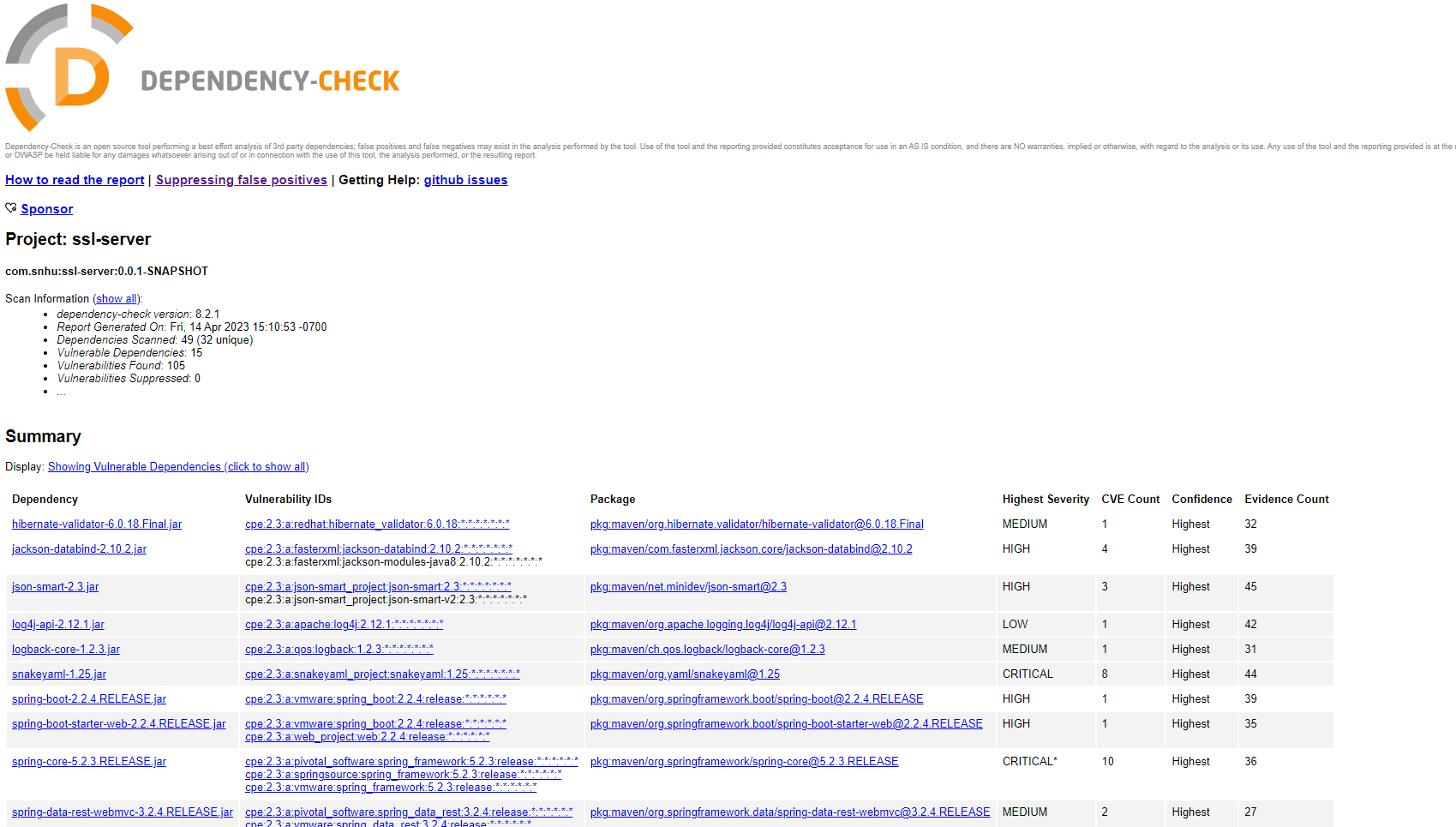
Refactored Code:



Dependency Check Report Before Code Refactoring



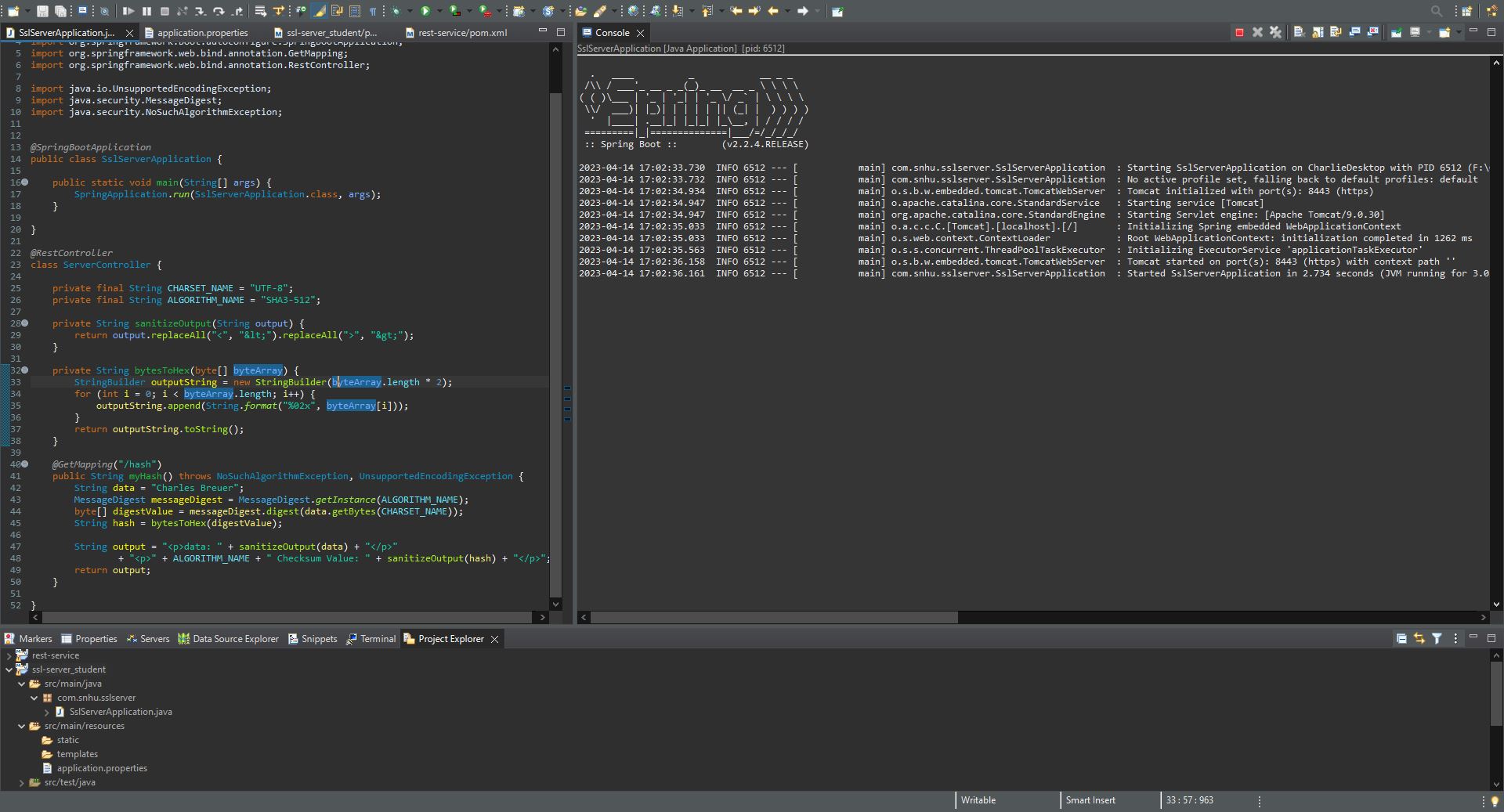
Dependency Check Report After Code Refactoring



**There was no difference in the number of vulnerabilities before and after refactoring the code.**

## Functional Testing

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



## Summary

The areas of security that are addressed by the refactoring of the code are input validation, cryptography, client/server, and code quality.

Proper input validation is addressed by sanitizing output of checksum through removing common HTML entities. This prevents HTML injection, which could lead to the website being displayed in an unintended manner. The code utilizes cryptography by selecting a secure hash function that is known to be secure and does not produce collisions. The checksum provides the application a way to verify the integrity of the data being retrieved from a database. The self-signed certificates provide a baseline amount of security that ensures data sent between the client and the server is encrypted properly and uses appropriate ciphers for TLS. The code utilizes best practices for encapsulation of in-memory data and providing trust boundaries between the server’s data structures and output to the client.

The process for adding layers of security is as follows: identify areas of the code that require security (input validation, cryptography, etc…), research best practices for ensuring code is secure, implement code utilizing best practices, test code using static testing (OWASP) and conduct a manual review of the code to identify security vulnerabilities. Additional methods for ensuring security is to implement dynamic testing and adding penetration testing.

## Industry Standard Best Practices

In this application, industry best practices were used to a variety of security concerns. It is known to be best practice to create trust boundaries between the current application and output/other applications. Sanitation was implemented to prevent HTML injection attacks (OWASP Dependency Check, n.d.). The SHA3-256 algorithm is recommended as the current industry standard for hashing functions (Secure Hash Standard, 2015). The SHA3-512 cryptographic hash function was selected as an extra layer of security due to its bit size. Utilizing self-signed certificates provides a baseline amount of security for development code (Self-signed certificates, 2018). Certificate authorities can be added when the software becomes distributed on the internet. The OWASP dependency checker is a well known static testing tool that ensures no vulnerabilities were added during the refactoring of code. Encapsulation and finalization of data in memory is a standard for Java applications that prevents misuse of java classes. Using industry standard best practices provides an easy confirmation that newly written code is not vulnerable to known attack patterns (Secure Hash Standard, 2015; OWASP Dependency Check, n.d.; Self-signed certificates, 2018). Code written without knowledge of industry standards incurs an indeterminate amount of risk for the company.

References:

Bernstein, D. J., & Lange, T. (2012). Hash-based Signatures. Springer Berlin Heidelberg.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC Press.

National Institute of Standards and Technology (NIST). (2015). Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-4. Retrieved from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf

National Institute of Standards and Technology (NIST). (2012). Recommendation for Key Management-Part 1: General (Revised). Special Publication 800-57. Retrieved from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

OWASP Dependency Check. (n.d.). Retrieved from https://owasp.org/www-project-dependency-check

Secure Hash Standard. (2015, August 5). Retrieved from https://csrc.nist.gov/publications/detail/fips/180/4/final

Self-signed certificates: Why are they good for testing and development? (2018, July 23). Retrieved from https://blog.approov.io/self-signed-certificates-why-are-they-good-for-testing-and-development