****

# 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 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/12/2023** | **Joshua Donnelly** |  |

## 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

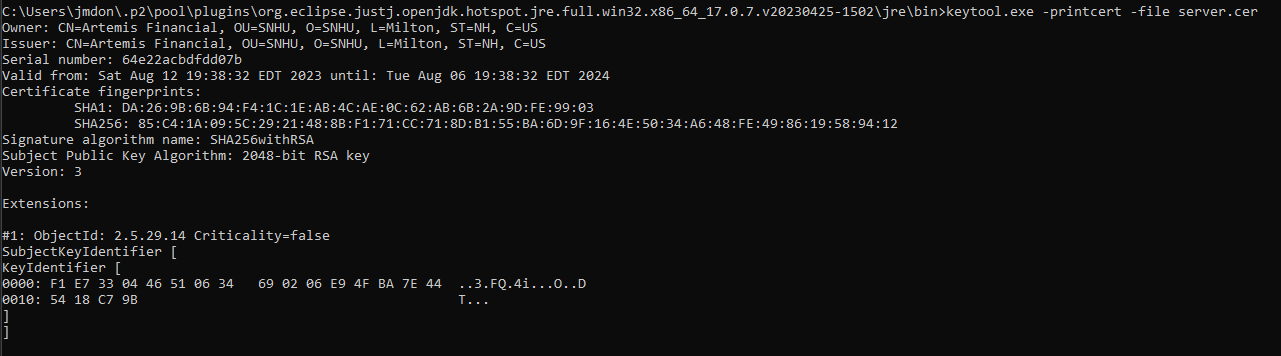
Joshua Donnelly

## Algorithm Cipher

The algorithm cipher I recommend is SSL using a 2048-bit RSA and SHA-256. The 2048-bit RSA covers the requirements to securely transmit and receive data with encryption, ensuring that only the intended recipients can read the data. The SHA-256 uses a hashing algorithm to ensure that none of the data has been altered in any way. This gives a unique checksum for each transaction of data and if the checksum does not match on delivery and receipt the data has been compromised in some way. This combination will allow the use of normal web browsers for users to be able to confidently secure their data transmissions.

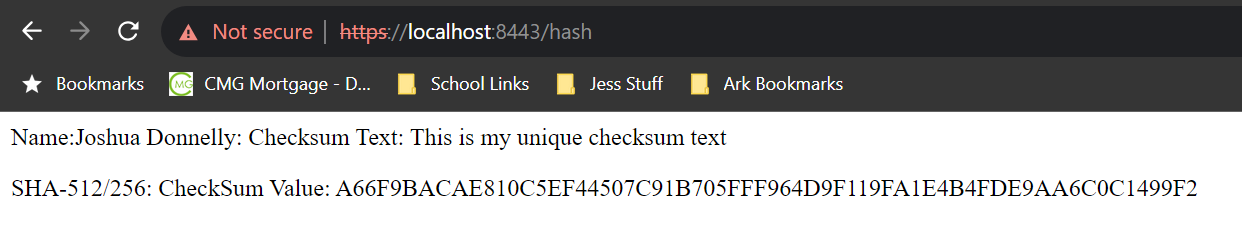
## Certificate Generation

Insert a screenshot below of the CER file.



## Deploy Cipher

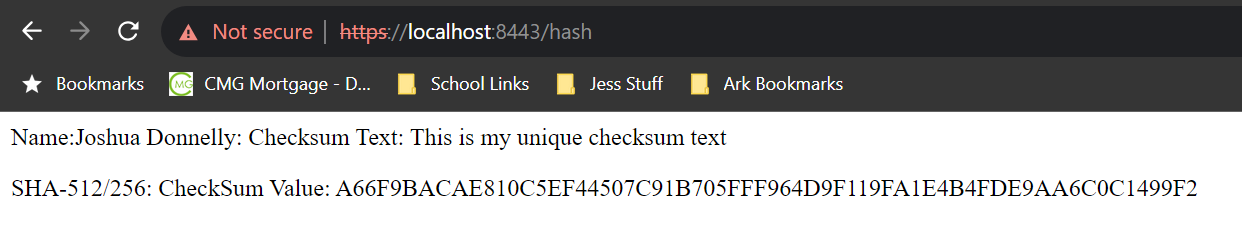
Insert a screenshot below of the checksum verification.

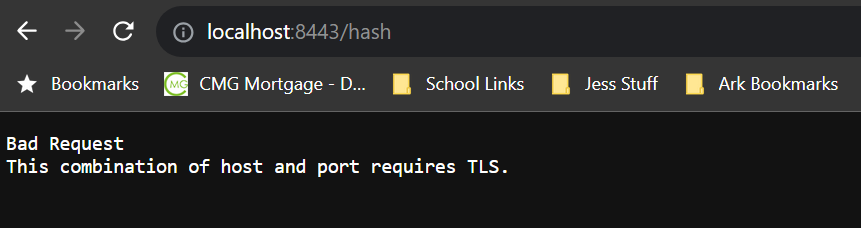


The hash uses the text "This is my unique checksum text" as an example of a checksum that will be received for the text entered every time. If the checksum on delivery and receipt do not equal, they then have been compromised in some way.

## Secure Communications

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

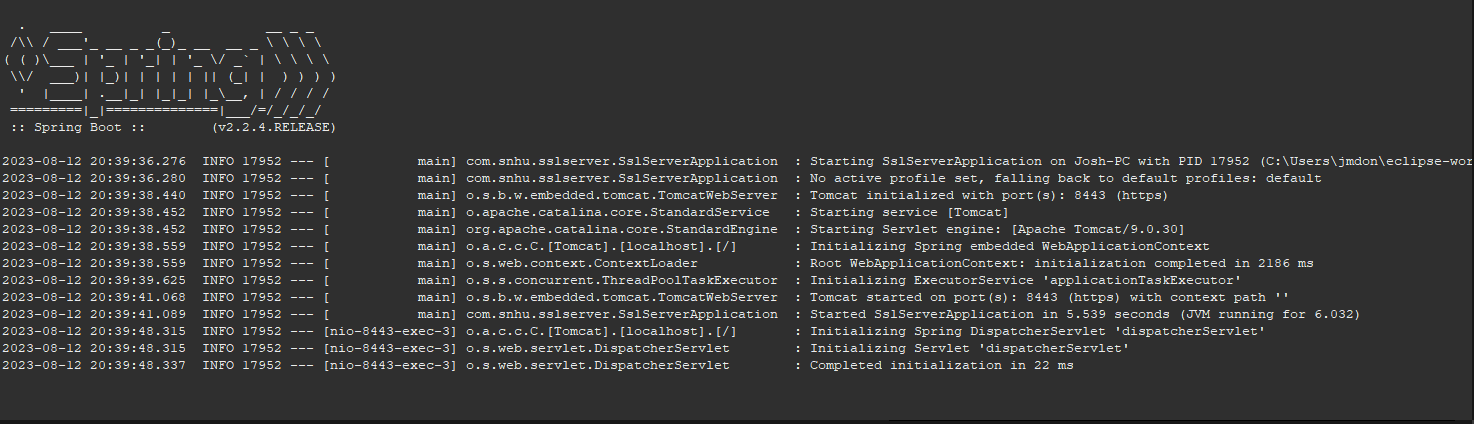


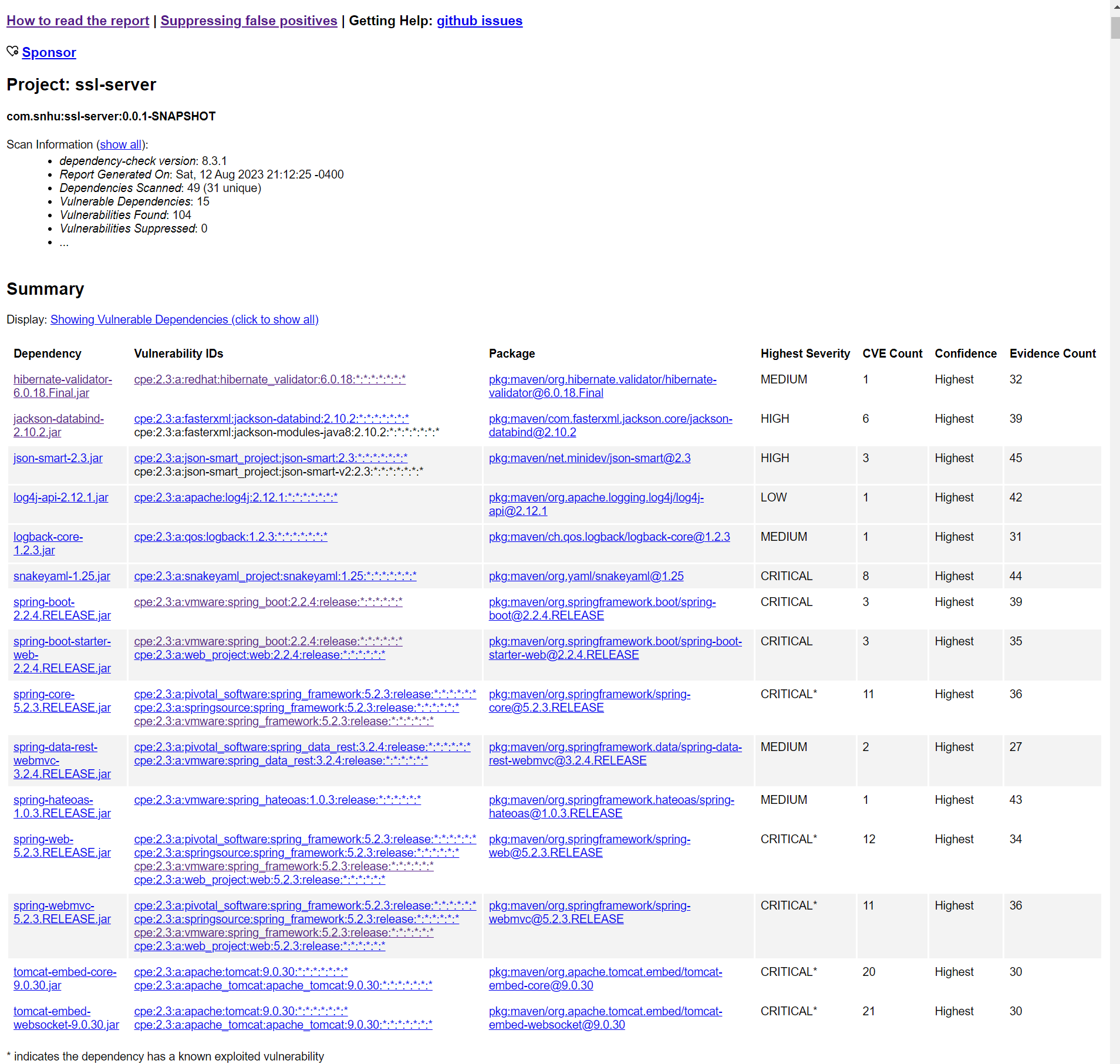


Based on the above screenshots the SSL protocol is in use as shown by using the https protocol and the site prevents the use of the strict http protocol showing an error. The browser will still say that the site is "Not secure" because the certificate has not been verified by a trusted certifying authority and a domain was not included in the certificate even if you do try to trust the certificate on the local machine.

## Secondary Testing

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



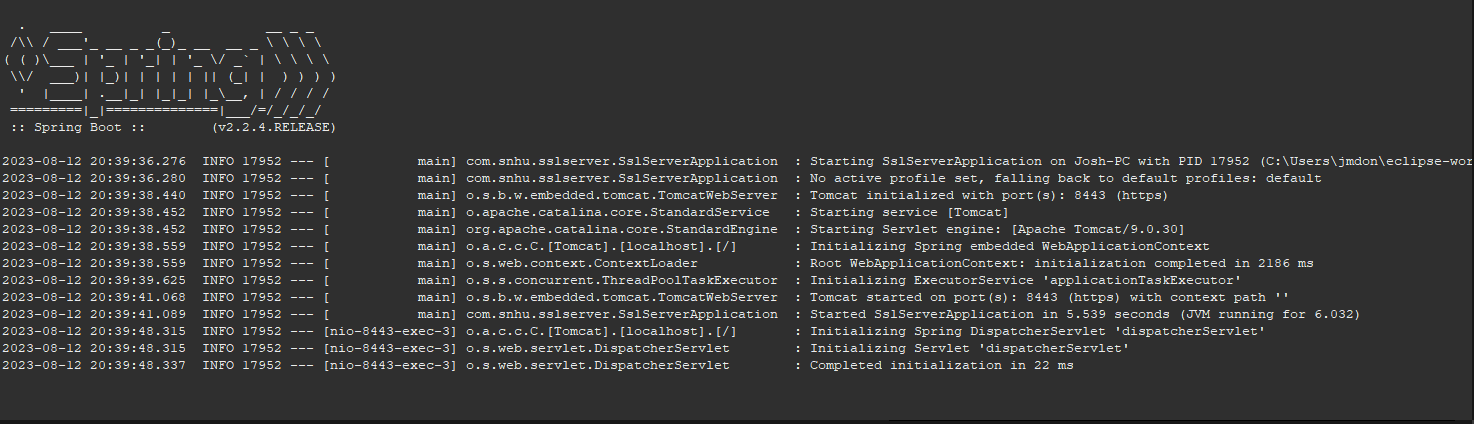




The first dependency report shows the errors prior to any refactoring other than updating the .pom file in order to update the dependency check version. The second shows the same errors and verifies that no new additional security flaws were introduced. The flaws were not corrected as the directions were to only focus on

## Functional Testing

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



The code ran without any errors as requested.

## Summary

The code has been refactored to ensure the cryptography section of the vulnerability assessment flow diagram was addressed by ensuring proper encryption and preventing the addition of new vulnerabilities in the system libraries. The refactored code also addressed the Client/Server section by ensuring that the distributed components are secured in communication between the client and the server. The process to adding layers of security include understanding the architecture of the system to be secured with a security perspective, Lokking at the input points within the system that may be points of vulnerability. Then, API communications need to be secured as any data from outside the system is untrusted. Cryptography is then implemented in order to encrypt and ensure data integrity. The client servers also need to be secured in more detail in order to verify the typically main source of data transmission is secured. After that ensuring that errors within the system are handled appropriately and do not have any unexpected handling situations. Then coding patterns should be analyzed to ensure that code always performs as expected regardless of the situation. Encapsulation also needs to be ensured to make sure that the class and object structures within the code are used appropriately to prevent access to private methods as well. After all of these steps are completed and static testing completed there should be a review to determine where manual reviews should be done in order to prevent vulnerabilities slipping through static testing.

## Industry Standard Best Practices

By using the encryption of the client/server communications and verifying the integrity of said communications, we have followed industry standards and best practices for security. While no system can be considered 100% secure especially when untrusted data is involved, we can ensure that the system is secure enough to protect the majority of users and the system data itself. Securing the website using TLS/SSL prevents bad actors from obtaining data that was not intended for their use and helps to ensure that data integrity is preserved so any communications from the company can be trusted to be from the company.