****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 6](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 7](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 9](#_Toc33111312)

[8. Sources 9](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/12/2021** | **Matthew Oliphant** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Matthew Oliphant

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

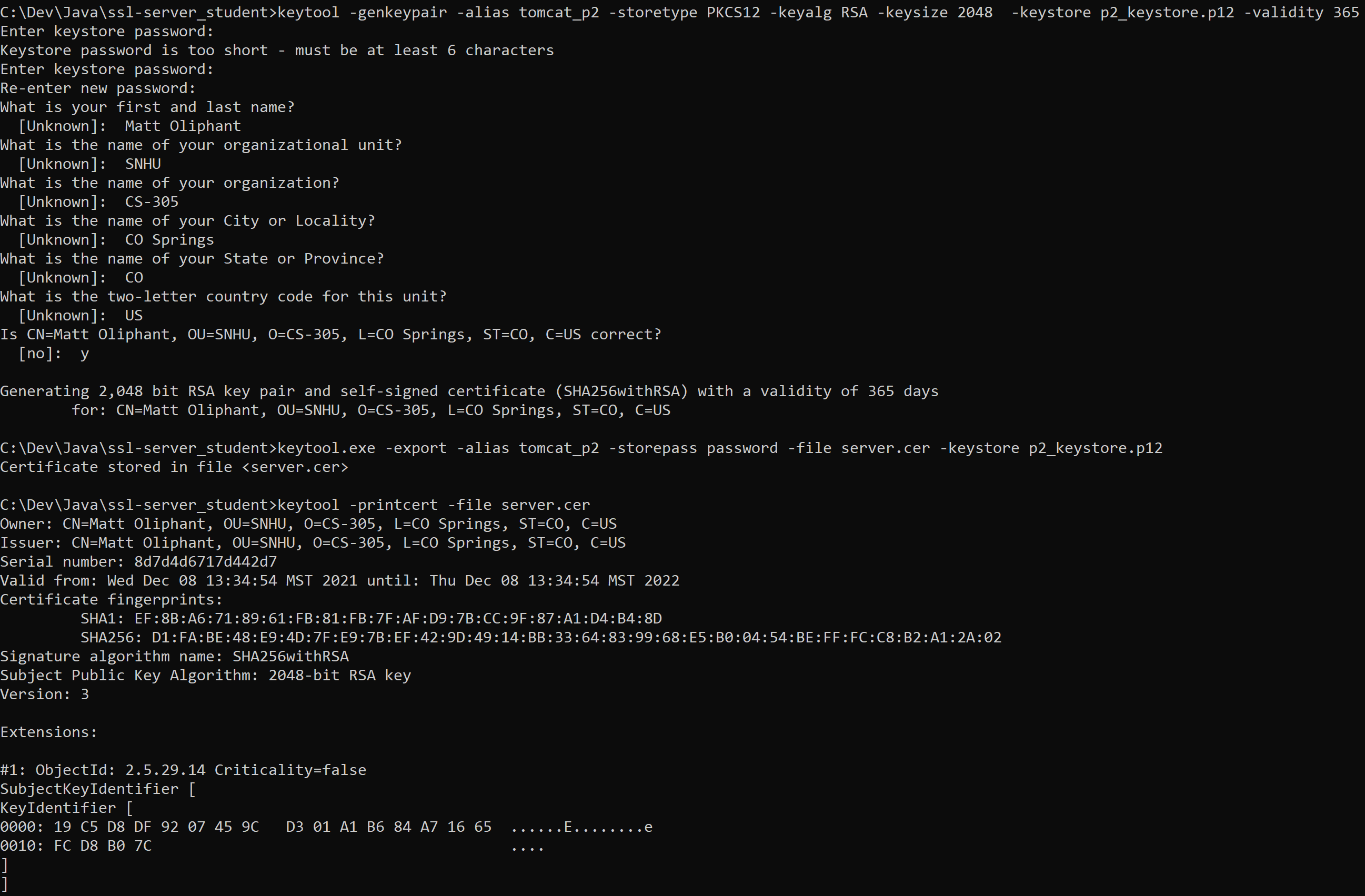
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

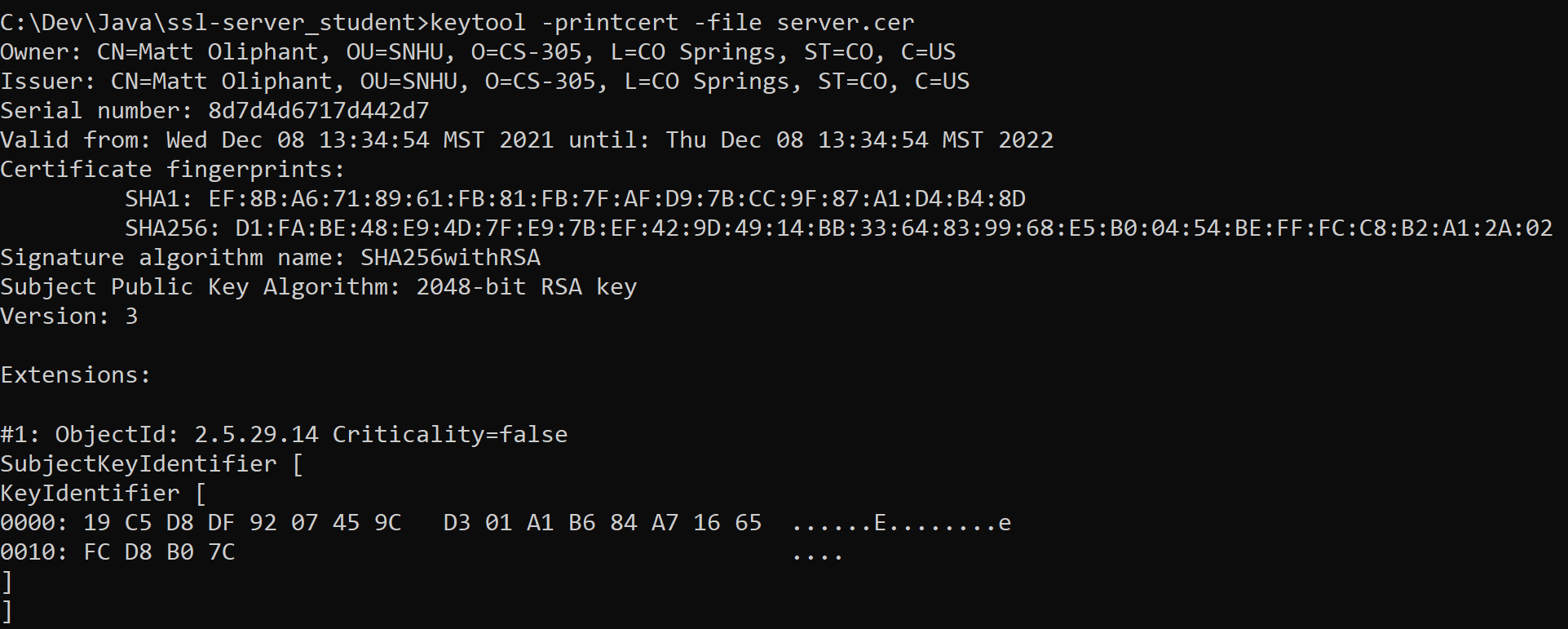
Artemis Financial is seeking to add a file verification step to their web application to ensure secure communications and a data verification step in the form of a checksum when the web application is used to transfer data. Before Artemis Financial releases a file for download, the file needs to be passed through a hash function to create a hash value. The downloaded file can then be passed through the same a hash function, and hashed values can be compared to ensure the integrity of the downloaded file. I would recommend using the SHA-256 encryption algorithm to create the hash values. Each hash value will be 32 bytes (256 bits) in length which is such a large number that current computing cannot even come close to producing a collision. Since the algorithm produces a 256-bit (binary) hash value for each call to it, there are 2256 = 1.16 x 1077 possible permutations for each hashed value, which is so many combinations that even rainbow tables are rendered useless. (Previous hashing algorithms including MD5 and SHA1 have been found to have massive security vulnerabilities and are no longer considered secure.) After input validation, data verification can utilize the same algorithm to ensure that stored data matches the submitted data.

When a client of Artemis Financial first creates an account, personal data is collected and verified and, in this scenario, the data will be very sensitive. Therefore, it’s not only important to add file and data verification procedures to their web app, but also to ensure that stored data and data in transit is secure as well. Symmetric keys are appropriate for databases since the servers they are stored on only allow admin access. Artemis Financial can use a private symmetric key to keep user data safe while it is in storage. For data in transit, I would recommend using the RSA encryption algorithm; this algorithm uses public and private keys, which are generated based on a defined algorithm that uses large random numbers to prevent duplication. Artemis Financial will generate a public and private key set, submit the private key to a certificate authority, and distribute the public key to their clients. When a client enters the URL to Artemis Financials’ web application in their web browser, a request certificate will be generated and sent to a CA who ensures a site’s validity and makes a connection after comparing the private and public keys.

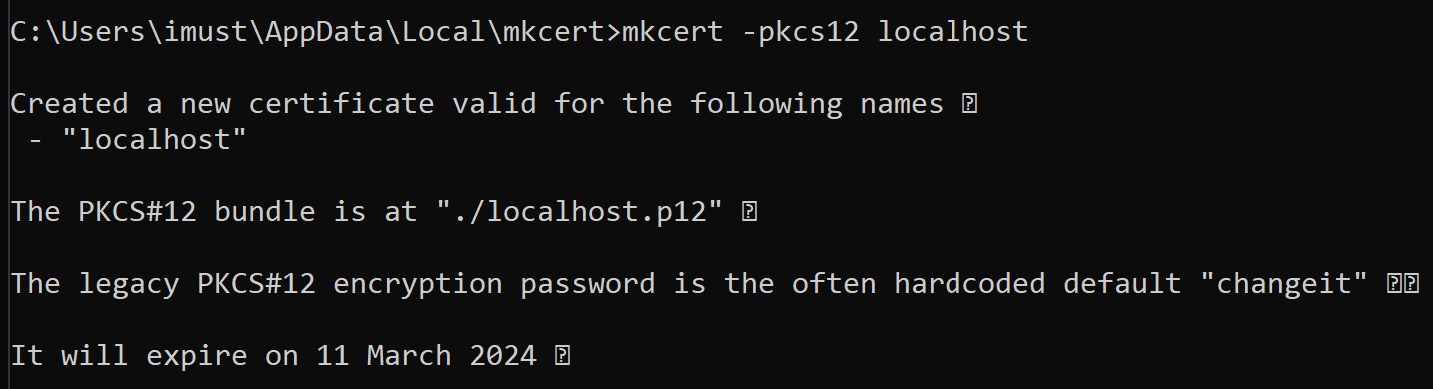
## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.



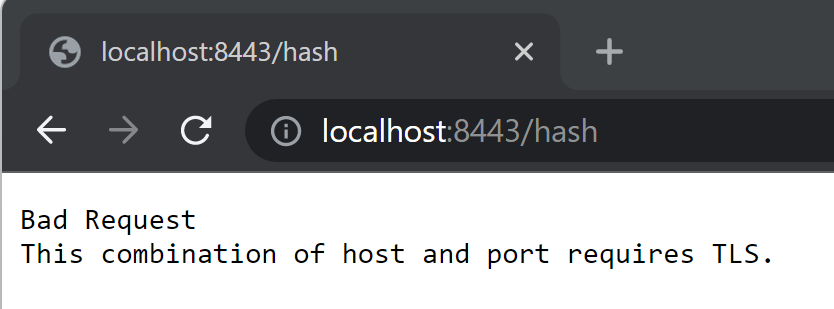


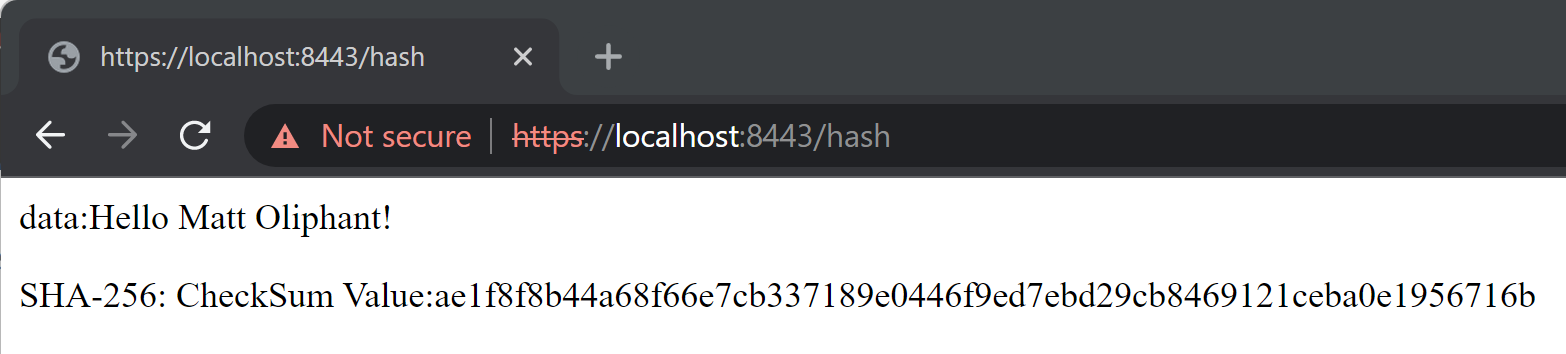
mkcert keystore: (explained below)



## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.





## 

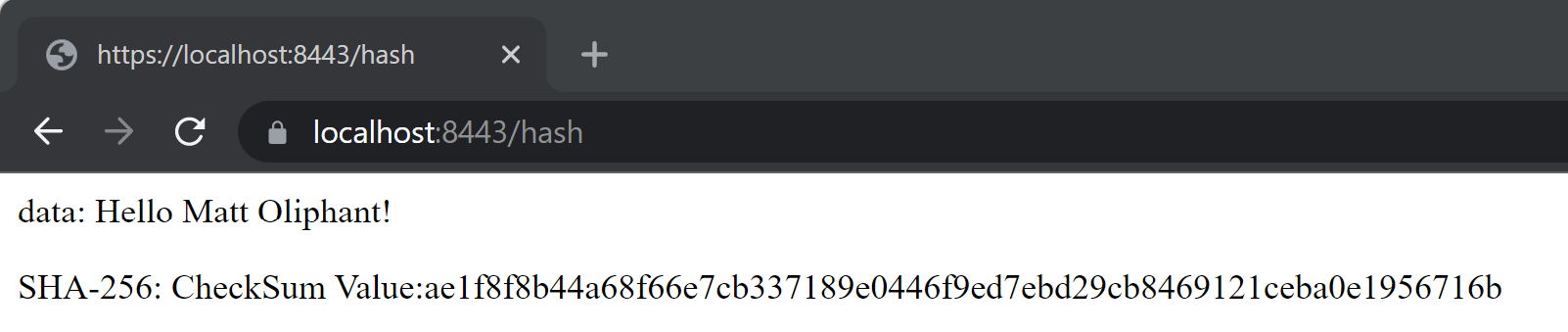
## 4. Secure Communications

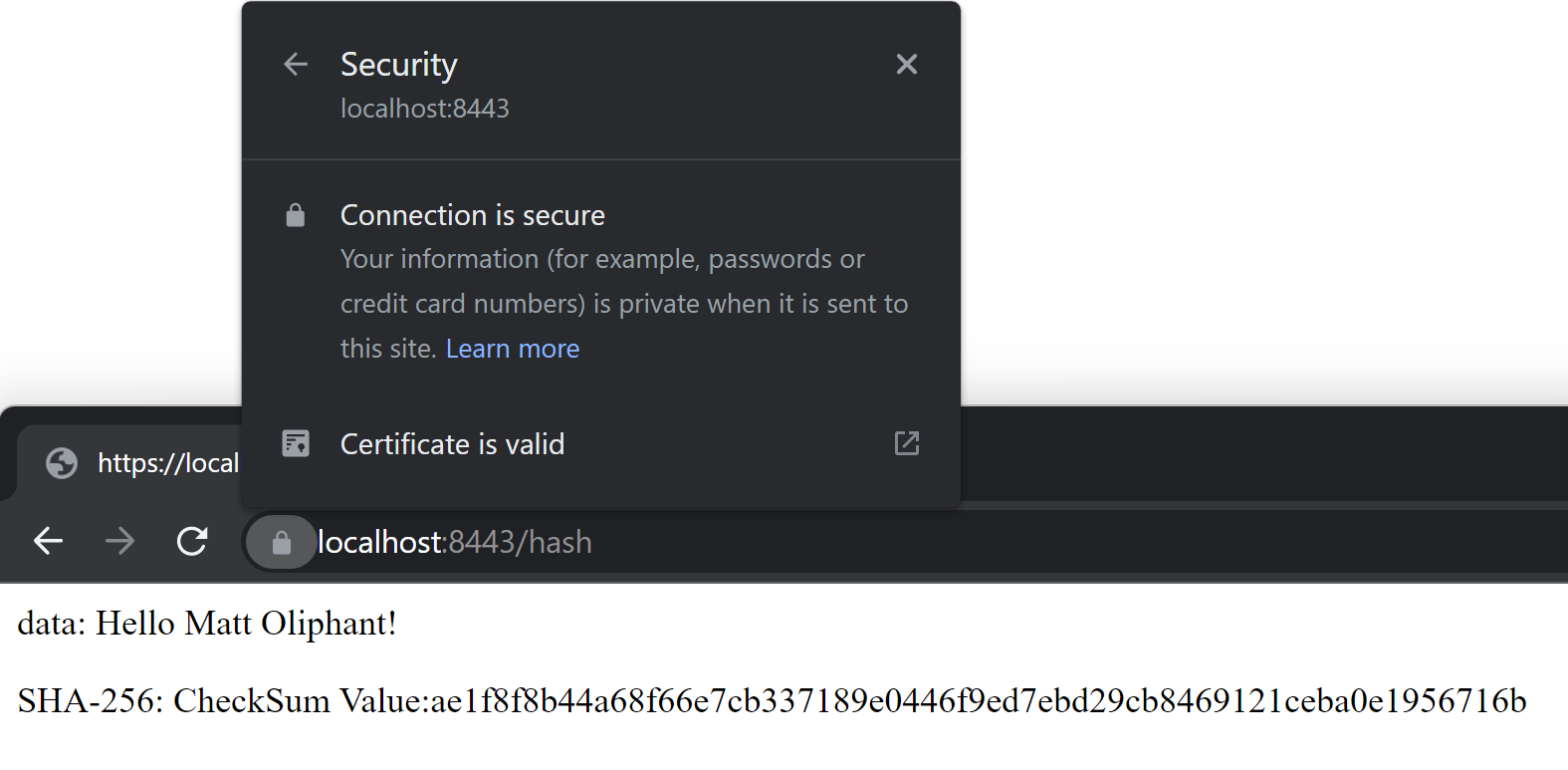
Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

I tried very hard to make a secure connection on Google Chrome using the Java keytool keystore. I generated a keypair and stored it in the keystore.p12 file. I then generated and exported a certificate using that keystore. On google chrome, I set a chrome flag to allow self-signed certificates, and imported my certificate into the trusted root authorities. Numerous sites stated this was supposed to fix the problem, but it did not; I couldn’t get Chrome to trust the certificate and display a secure connection.

To fix this problem, I installed chocolatey (a package manager) which allowed me to install mkcert (Chocolatey, n.d.). “mkcert is a simple tool for making locally-trusted development certificates. It requires no configuration (Valsorda, n.d.).” According to my research, professional and open source CAs do not certify certificates for localhost connections, but mkcert does. Installing mkcert automatically adds the CA to Google Chrome’s trusted root authorities.

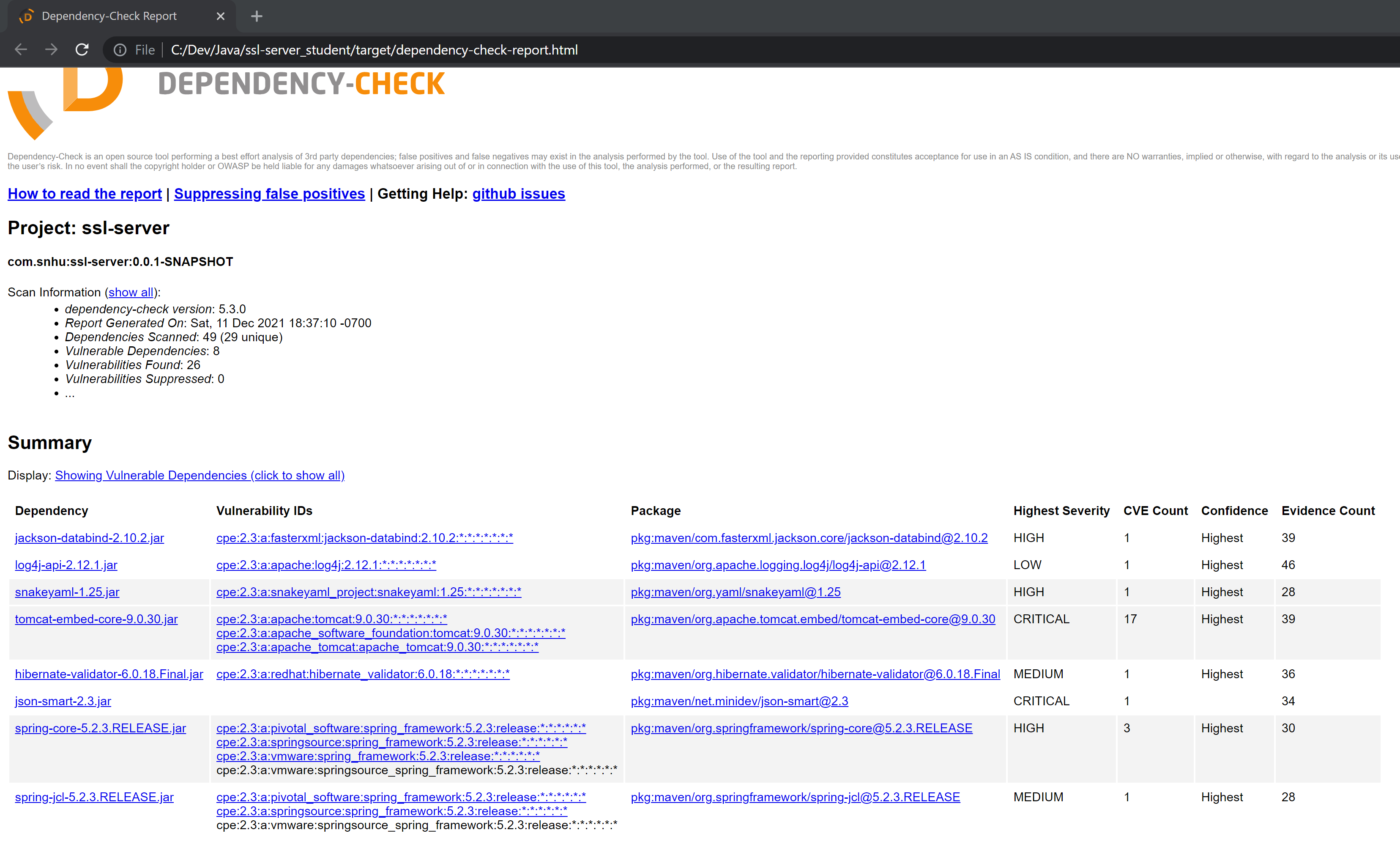
I used mkcert to generate a keystore and then moved that to this application’s resources (localhost.p12). Adding the new keystore details to the application.properties file allows the application to connect securely. This certificate is only valid on machines that have mkcert installed, so will the browser will show unsecure to users who do not have it installed. For this situation, I have commented out the Java keytool keystore in the applications.properties file and left the keystore.p12 file in the ssl-server\_student folder. Removing the comments and commenting out the mkcert keystore will display an unsecure browser, but this certificate does use the Java keytool (as shown in section 3).





## 5. Secondary Testing

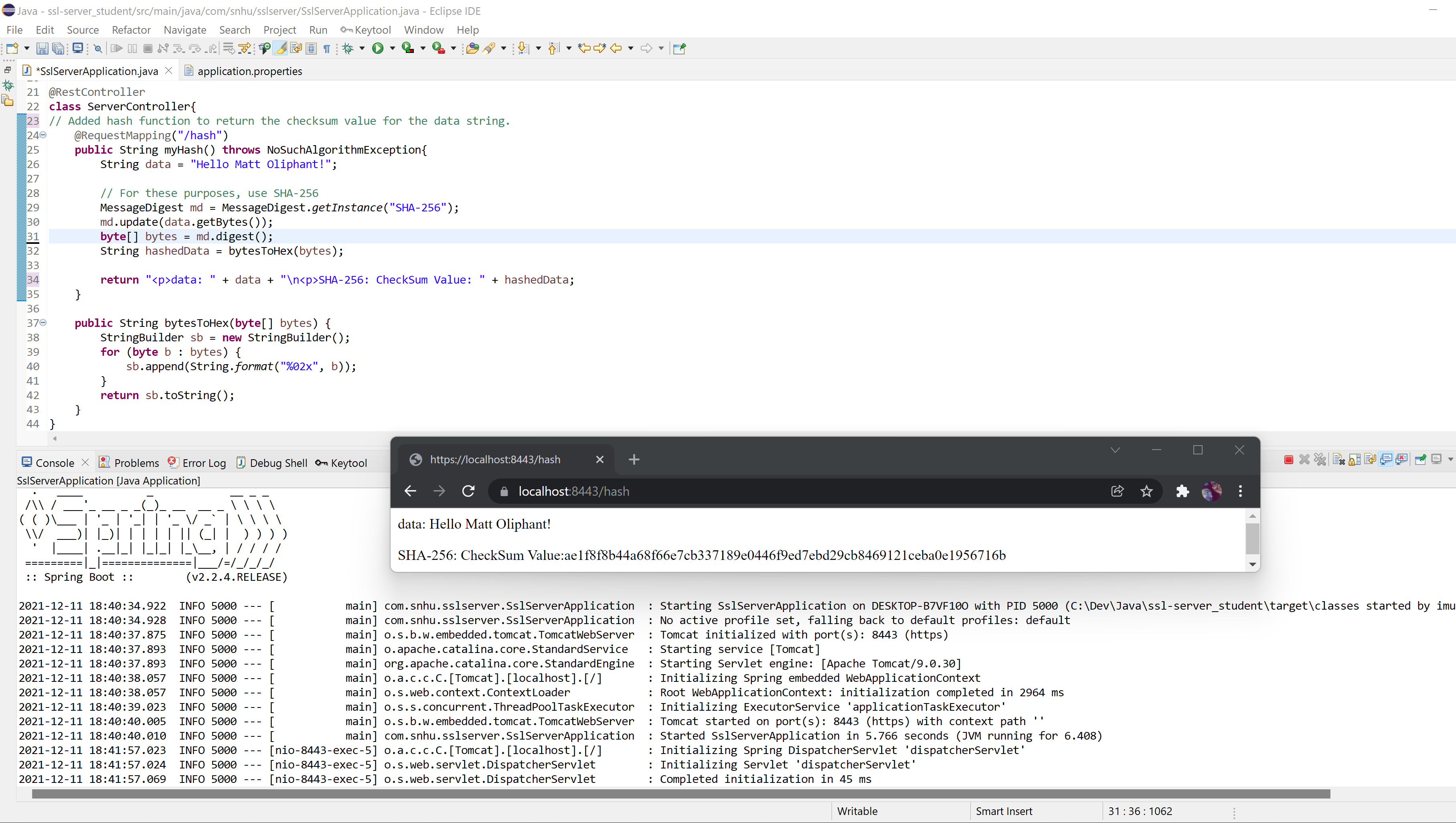
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

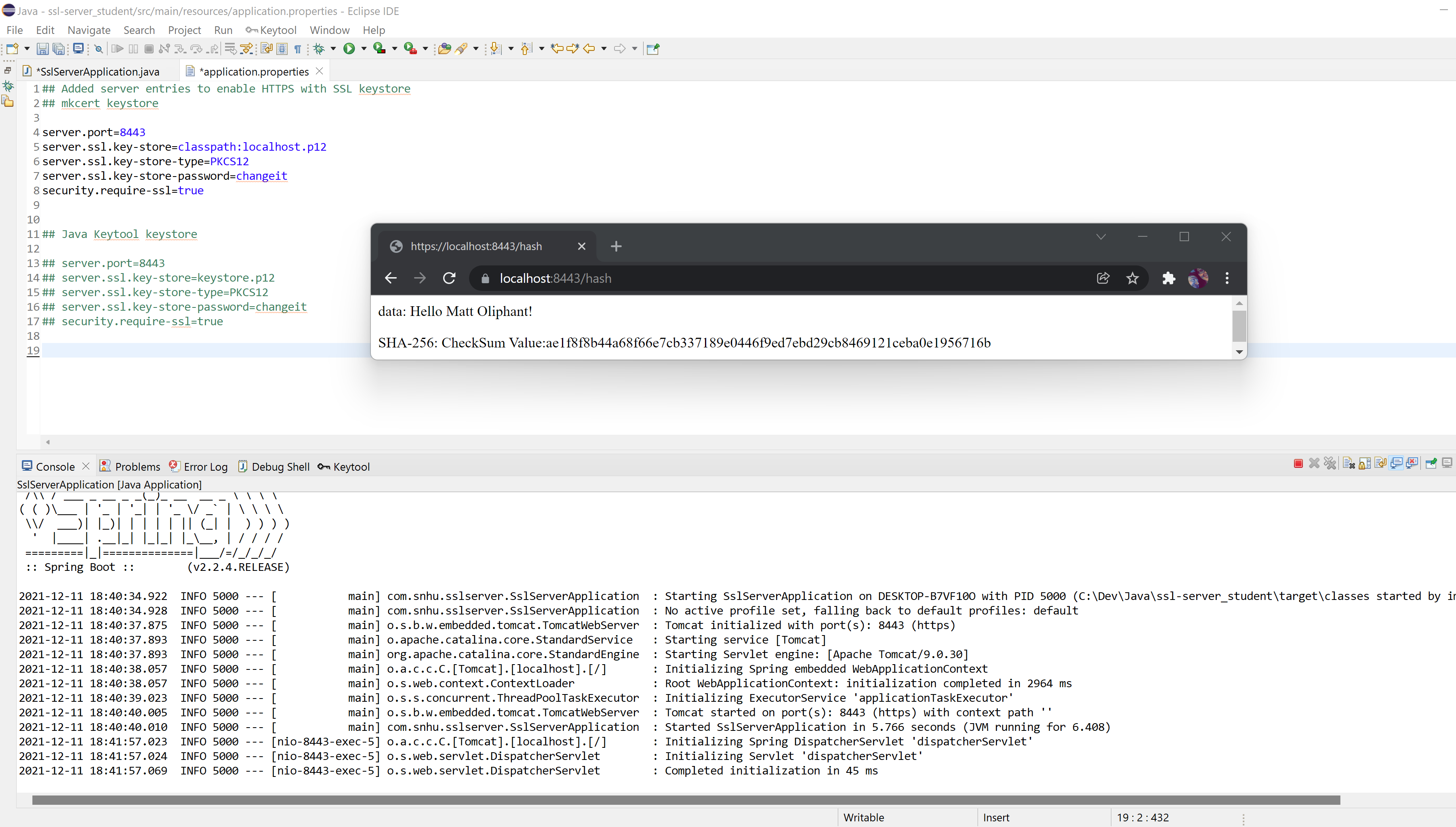


This application’s dependency report shows the exact same vulnerable dependencies as those from module 6, so no extra vulnerabilities were added by adjusting the server data. The log4j-api-2.12.1.jar file can be suppressed since the severity is low.

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.





The code runs without error. The application doesn’t require any input, but does need to securely connect to the spring framework. Cryptography is a vulnerability, although Artemis Financial will need to get a certificate from an authorized CA, rather than the workaround I performed for localhost. Client/ server is probably the largest security risk here, as it’s going to be essential that each client has a secure connection with the server to prevent attacks. Code error, code quality, and encapsulation are each equally important for this application to keep it safe and secure.

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

By refactoring the code, I was able to reduce vulnerabilities related to cryptography, client/ server, code quality, and encapsulation. Forcing a connection to run through https rather than http, provides a layer of security to the software application. Making an app as secure as possible allows users to have peace of mind while using the it and prevents attacks from access to their information. When the application eventually needs input, validation will be required and is a great way to prevent attacks by only allowing regular expressions. Keeping data encrypted while in transit and storage is another requirement. Utilizing hash keys will provide added security by ensuring files for download are the same before and after download.

## 8. Sources

Chocolatey. (n.d.). *Chocolatey - The package manager for Windows*. Chocolatey Software. Retrieved December 11, 2021, from <https://chocolatey.org/>

Valsorda, F. (n.d.). GitHub - FiloSottile/mkcert: A simple zero-config tool to make locally trusted development certificates with any names you’d like. GitHub. Retrieved December 11, 2021, from <https://github.com/FiloSottile/mkcert>