****

# 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 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **April 15, 2022** | **Christopher Clark** |  |

## 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

Christopher Clark

## 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.*

The recommended cipher for this application is SHA-256. The algorithms are defined in [FIPS PUB 180-4](https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf).

SHA256 is considered an industry standard cryptographic tool. This cipher produces 256-bit keys and is much more secure than other ciphers in use today. With SHA256 there are 2^256 possible output values making it very unlikely for collisions to happen. This also results in the avalanche effect for this cipher. This simply means that if even one bit is changed in the input, then the output is changed significantly. Basically, this cipher has excellent randomization properties. SHA256 has not been cracked, protects against collisions with 256-bit keys, and makes excellent use of the avalanche effect to randomize hash keys. (Wiki, 2021)

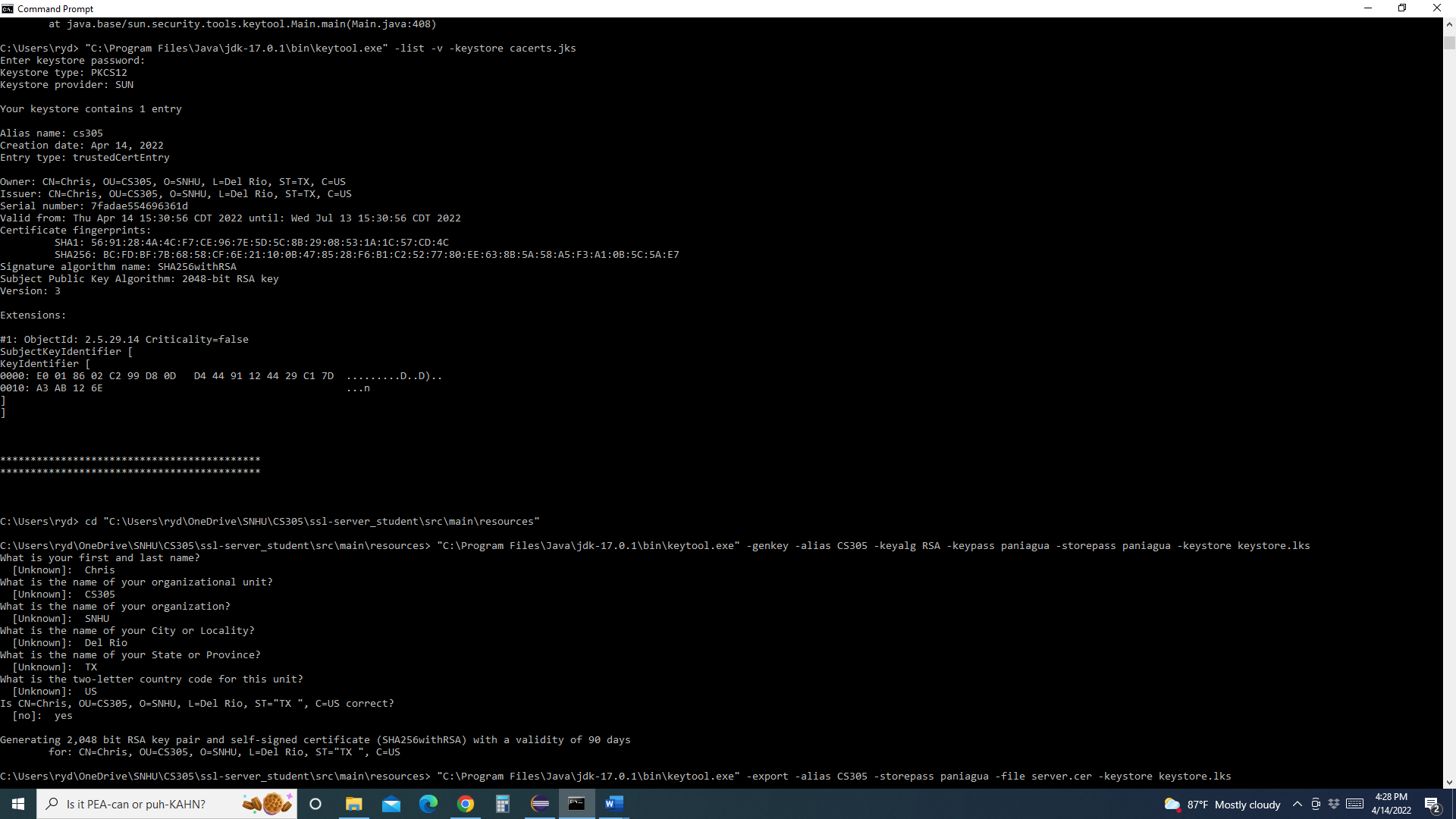
Random number generation is an important part of cryptography. Two types of generators are typically used. This included pseudo-random and true random number generators. Pseudo-random generators use an algorithm to generate seemingly random numbers based on a seed used to initialize the random number. A truly random number generator is a device that uses physical processes like noise to generate random numbers.

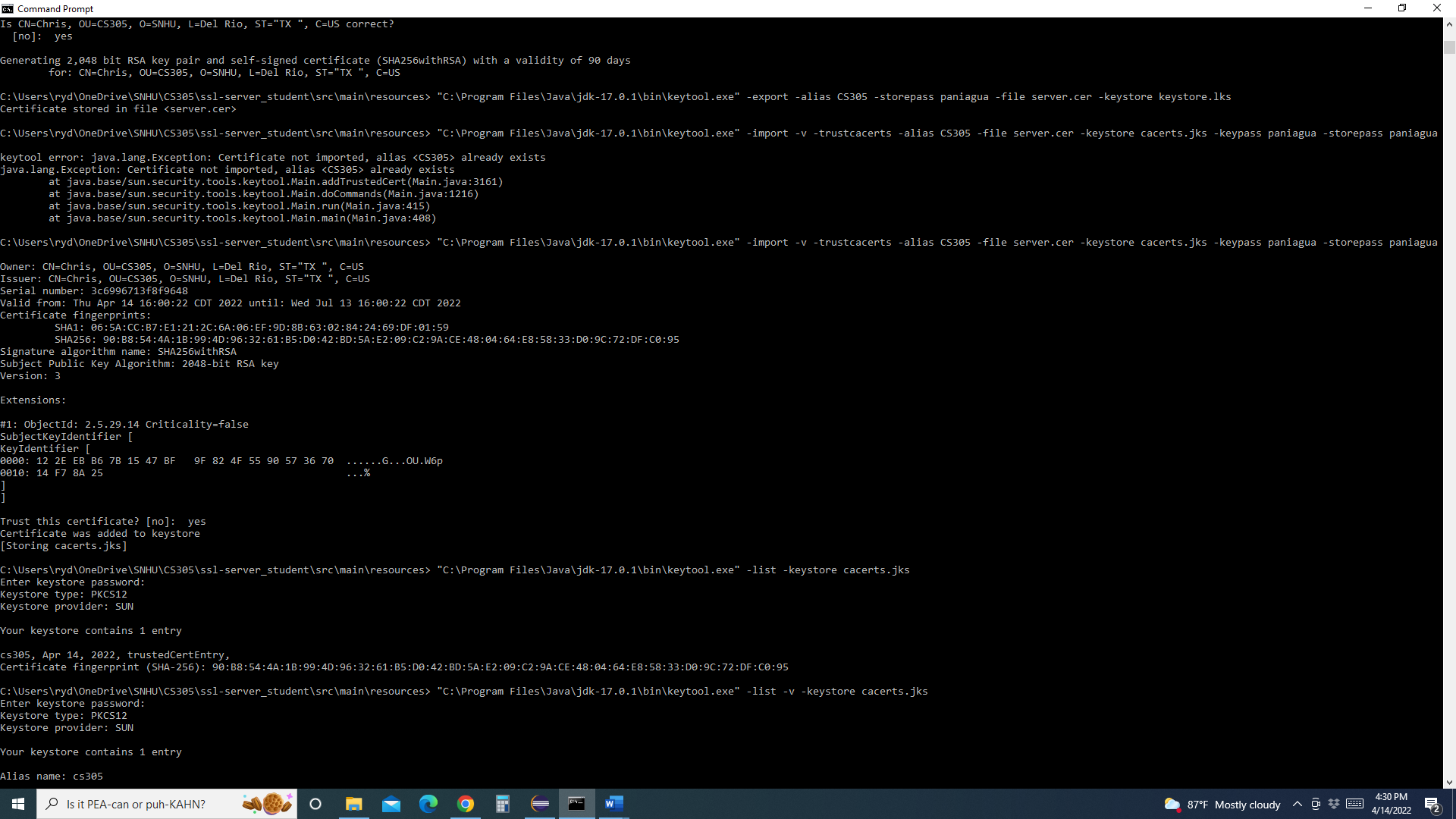
With encryption there is data transfer that requires a key to decrypt data. There are two methods, symmetric and non-symmetric. With symmetric keys, both the sender and the recipient must use the same key to encrypt and then decrypt the data. The key must be shared with all recipients. With non-symmetric keys there are two keys needed to share data. First a public key must be made to allow data encryption. Then a private key must be used at the other end to decrypt the data. Both keys are different, however they are related.

## 2. Certificate Generation

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

* *To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.*

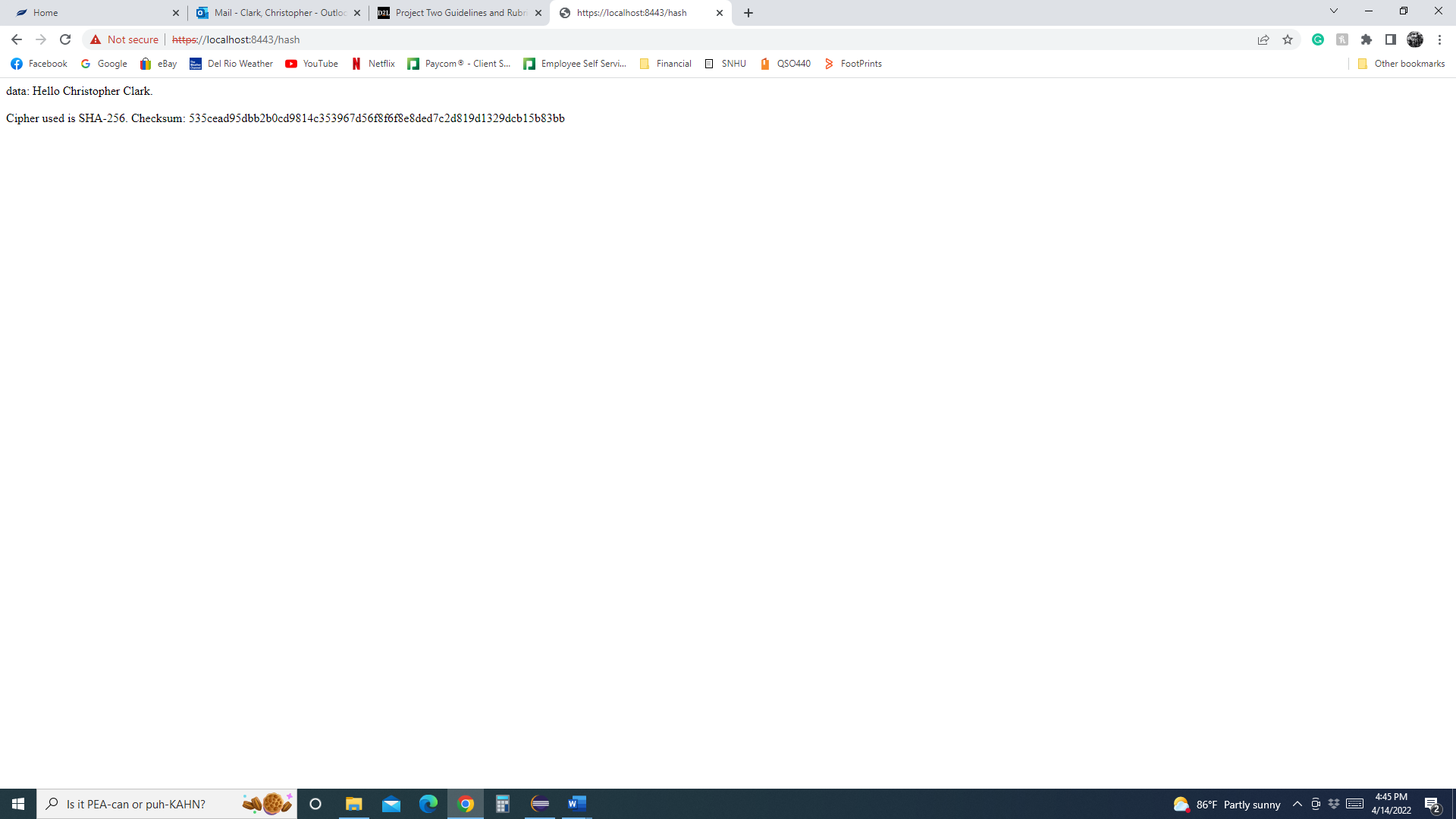




## 3. Deploy Cipher

*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.*

* *Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.*



## 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.*

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

Graphical user interface, text, application

Description automatically generated

Certificate generated with Eclipse keytool plugin:

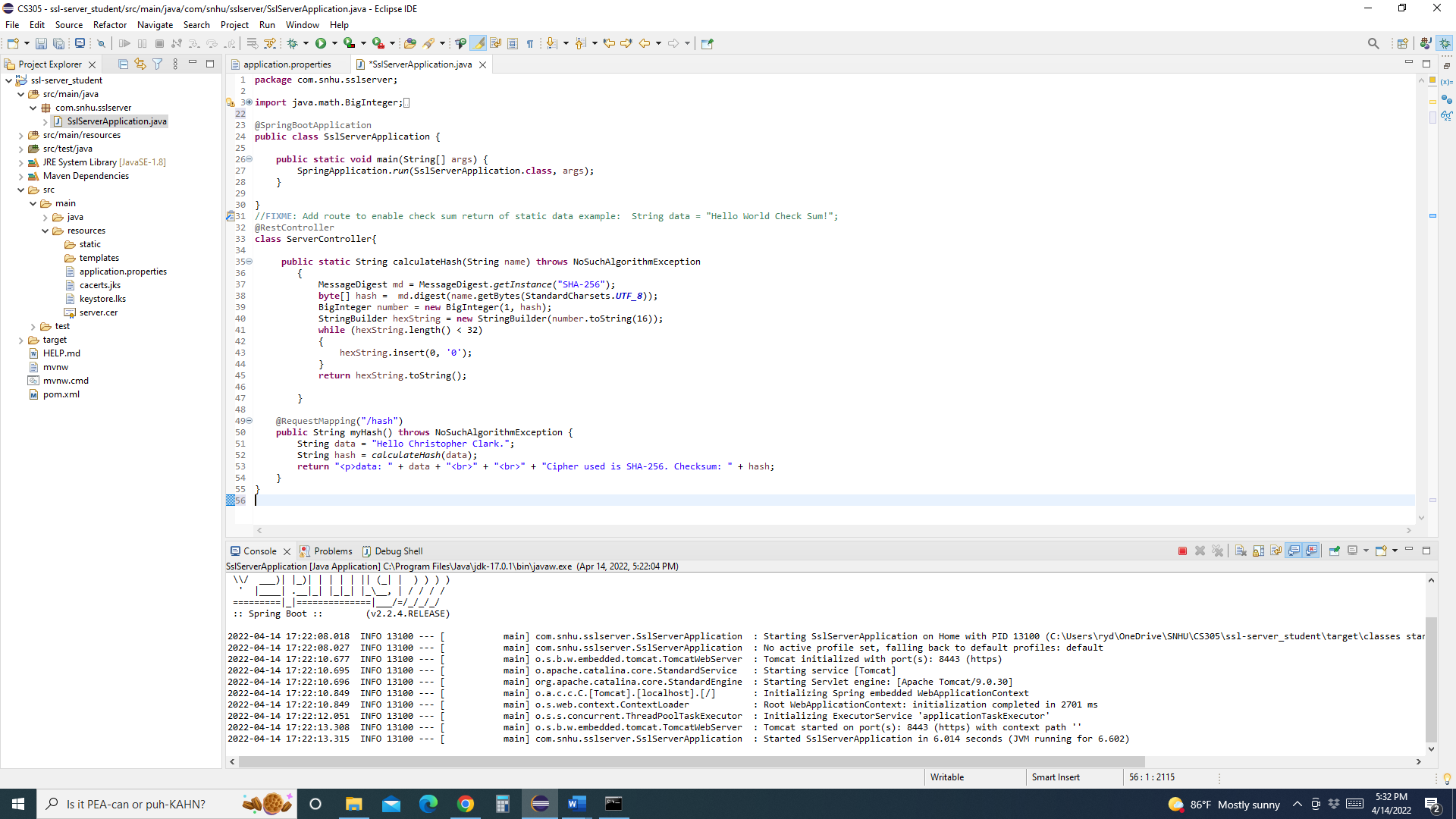
Graphical user interface, text, application, email

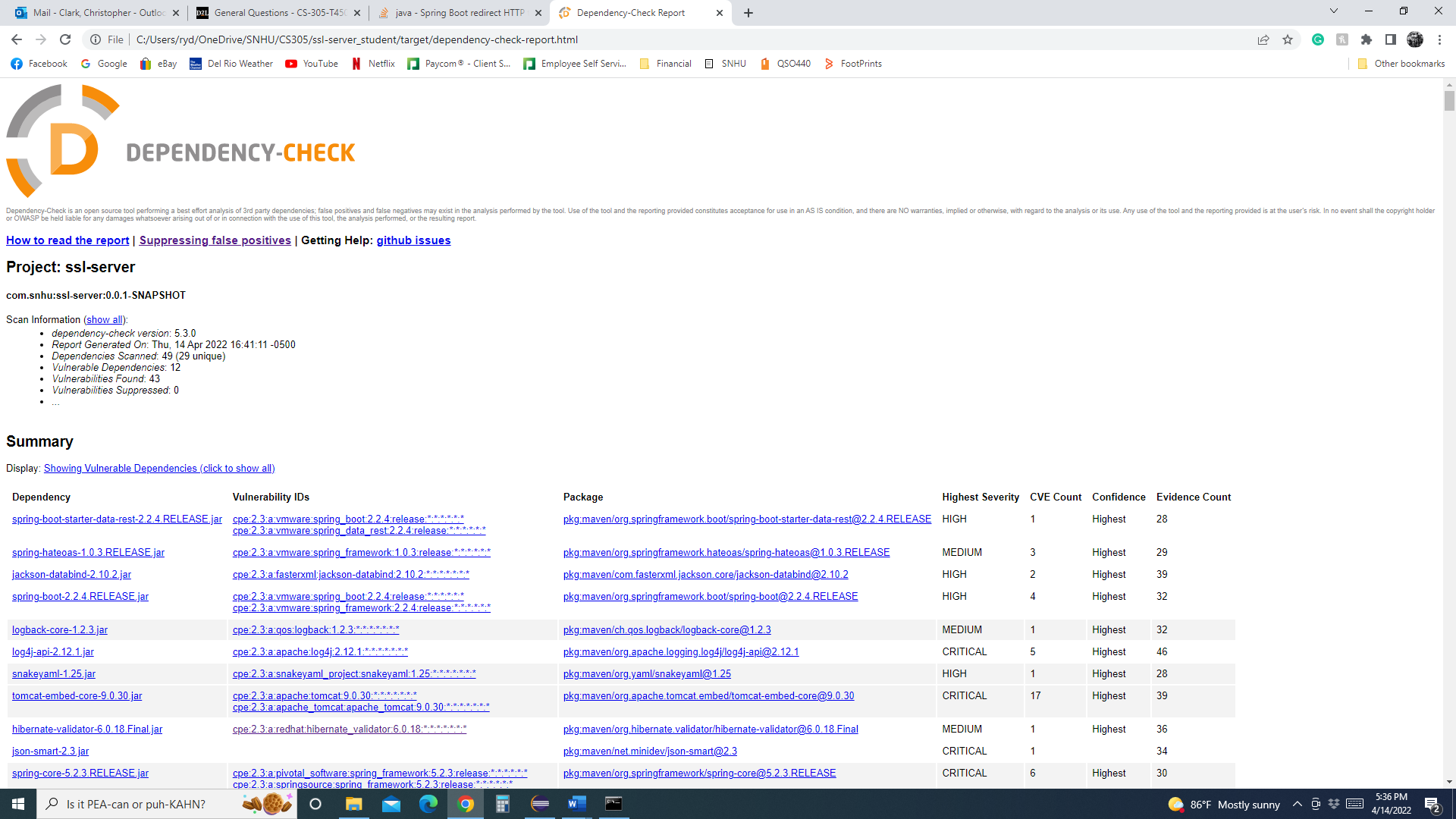
Description automatically generated

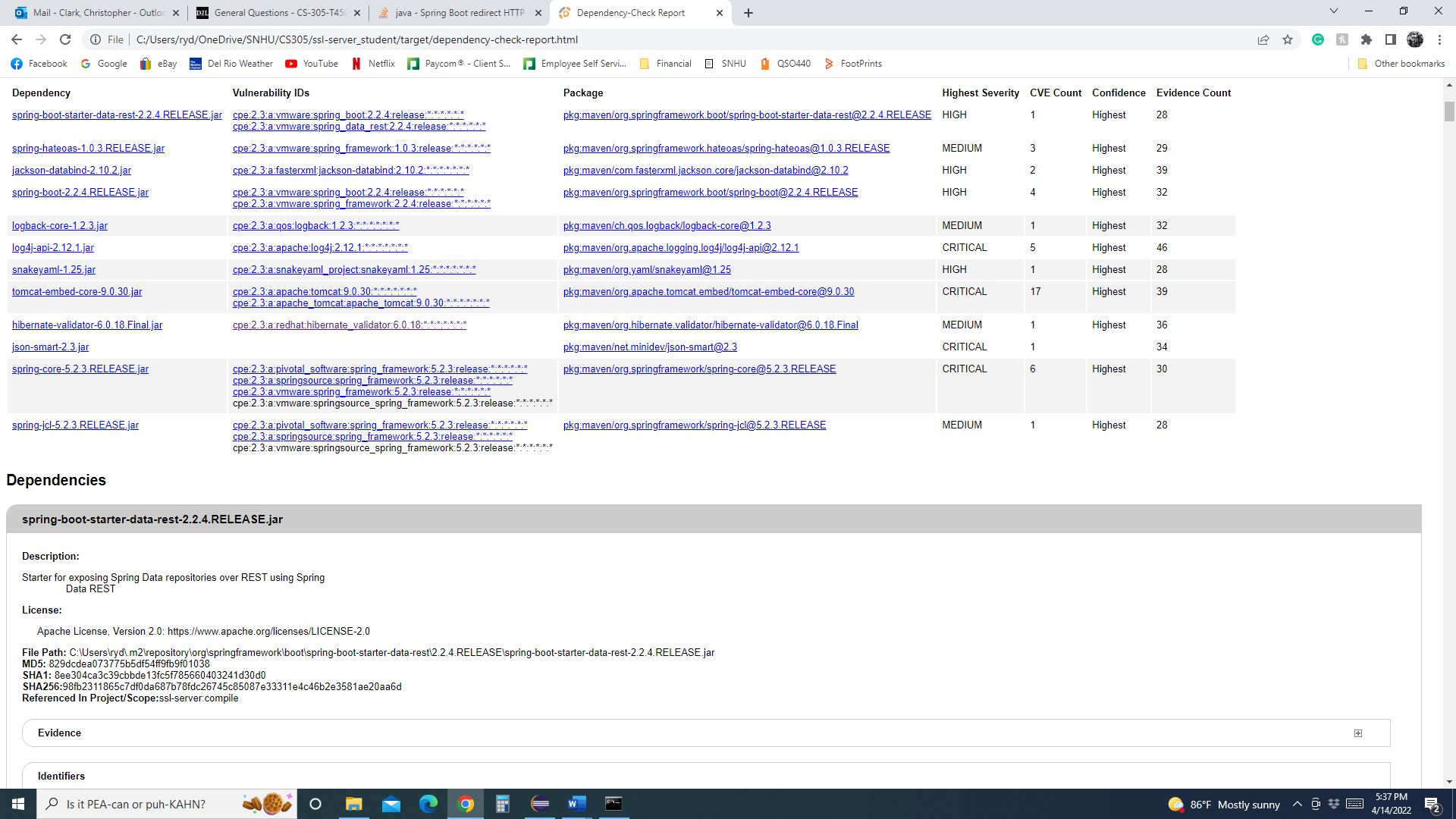
## 5. Secondary Testing

*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.*

* *Include the following below:* 
  + *A screenshot of the refactored code executed without errors*
  + *A screenshot of the dependency check report*







## 6. Functional Testing

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

* *Complete this functional testing and include a screenshot below of the refactored code executed without errors.*

Graphical user interface, application

Description automatically generated

## 7. Summary

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

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

In order to satisfy the needs of the client, an SHA-256 algorithm has been employed. This is evident in the SSLServerAPplication.java file shown above. The code required for implementation of the hash algorithm is in place and operational. A message digest and exception handler were added to output a unique data string and the checksum value issued by the algorithm.

The areas of security that need review for this project would include:

*Input validation*. Given this is a web-based application and the goal is to obtain user input, this information needs validation. This can help secure against insertion attacks and provide for a more secure application.

*APIs*. Given the nature of the project, the use of RESTful API may be required and should be considered to secure data transfer in a client/server configuration.

*Cryptography.* For user input and data transfer, proper use of cryptography for data encryption is needed to secure transactions.

*Client/Server*. Should the application require a client/server platform, the use of secure certificates in data transfer is required for secure operation.

*Code error*. Secure operations will require code check and exception handling for data transfer. User input and API access are open to attacks and should be error checked.

*Encapsulation.* Should the user interact with a database or other information internal to the system, encapsulation methods can provide for information hiding and better security.

Having the code in place to add security via cryptography is a best practices solution to the client’s needs to protect data. Utilizing dependency checks and SSL certificates in development helps ensure a secure environment for system operation. The client’s needs will be maintained.

**Citations**

Sönnerup, J. (2022, January 12). *On the difficulty of generating random numbers*. Debricked. Retrieved April 16, 2022, from <https://debricked.com/blog/difficulty-of-generating-random-numbers/#:~:text=Random%20numbers%20are%20used%20in,be%20predicted%20better%20than%20guessing>.

Wikimedia Foundation. (2021, June 9). *Avalanche effect*. Wikipedia. Retrieved March 29, 2022, from https://en.wikipedia.org/wiki/Avalanche\_effect#:~:text=In%20cryptography%2C%20the%20avalanche%20effect,half%20the%20output%20bits%20flip).