****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_1fob9te)

[Client 3](#_3znysh7)

[Instructions 3](#_2et92p0)

[Developer 4](#_tyjcwt)

[1. Algorithm Cipher 4](#_3dy6vkm)

[2. Certificate Generation 4](#_1t3h5sf)

[3. Deploy Cipher 4](#_4d34og8)

[4. Secure Communications 4](#_2s8eyo1)

[5. Secondary Testing 4](#_17dp8vu)

[6. Functional Testing 5](#_3rdcrjn)

[7. Summary 5](#_26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **06/20/2022** | **Concepción Ardon** | **Integrate AES for appropriate encryption algorithm. Do dependency check and run application for errors. Assess any updates and vulnerabilities in the software.** |

## 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

Concepción Ardon

## 1. Algorithm Cipher

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

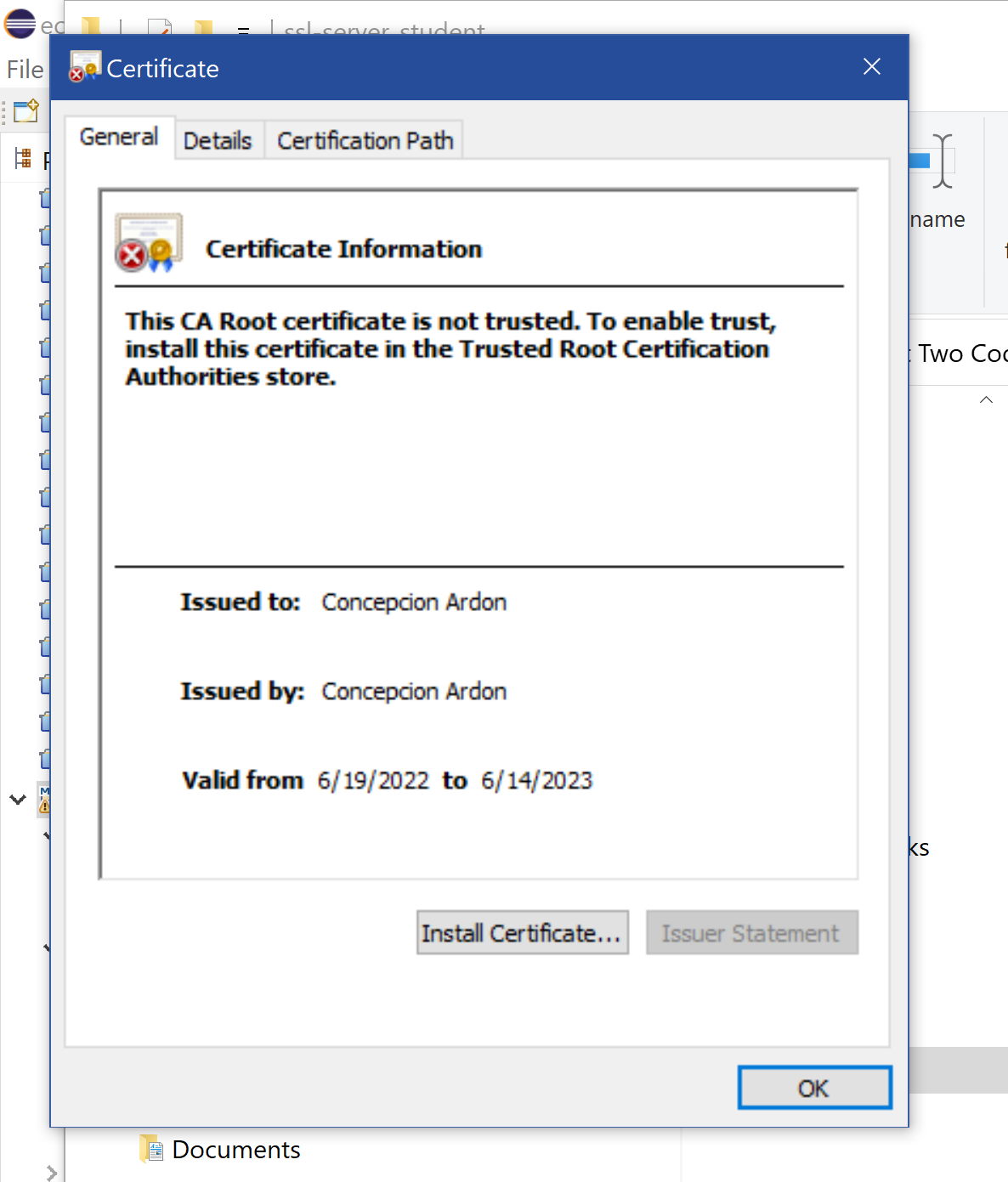
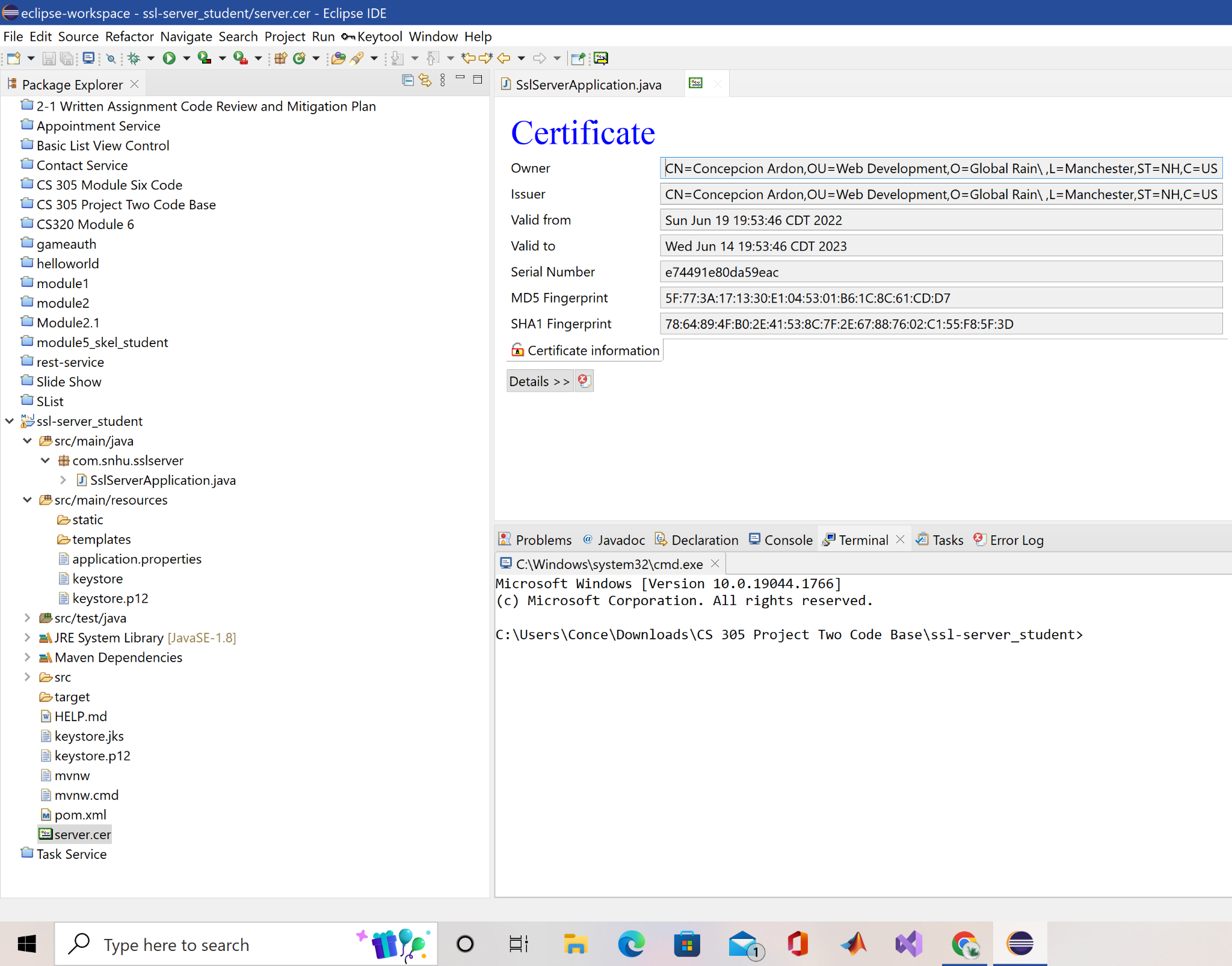
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

AES, or Advanced Encryption Standard, would be the most suitable cipher for Artemis Financials' needs, as well as its overall system. In my research, AES 256 Bit is considered to be the most secure and unbreakable. It uses the Rijndael block cipher with a symmetric-key algorithm. Such an algorithm is useful for protecting consumer and company data. In order to avoid any risks, it would be necessary to ensure that the system does not leak information, as well as make sure the means of gaining access are secure and updated, such as the firewall. Given that security is one of my top priorities, I wouldn't hesitate to select the most secure cipher. Hash functions are frequently used in conjunction with digital signatures to ensure data integrity. The length of the encryption is determined by the bit levels. One example would be 256-bit encryption, which has 256 different possibilities. Random numbers can be used to generate data encryption keys, simulate and describe complicated events, and select random samples from bigger data sets, among other things. In the statistical analysis and probability theory, random numbers are crucial. Symmetric-key encryption encrypts communication using a key and decrypts the message with the same key, making it simple to use but less secure. It also necessitates a secure mechanism for passing the key from one person to the next. Symmetric keys include AES-256 and are used to encrypt plain text with a key. In symmetric key encryption, resource utilization is low as compared to asymmetric key encryption. Asymmetric Key Encryption is a technique that uses both public and private keys to encrypt data. It encrypts and decrypts the message using two separate keys. Although it is safer than symmetric key encryption, it is substantially lower. Encryption algorithms have a long history, dating back to 500 BC (The history of encryption: the roots of modern-day cyber-security. Tresorit Team, 2022). Today, encryption is employed in almost every aspect of modern life, yet most users are completely unaware of it. Encryption is mostly used to conduct transactions through insecure communication channels, such as the internet.

## 2. Certificate Generation

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

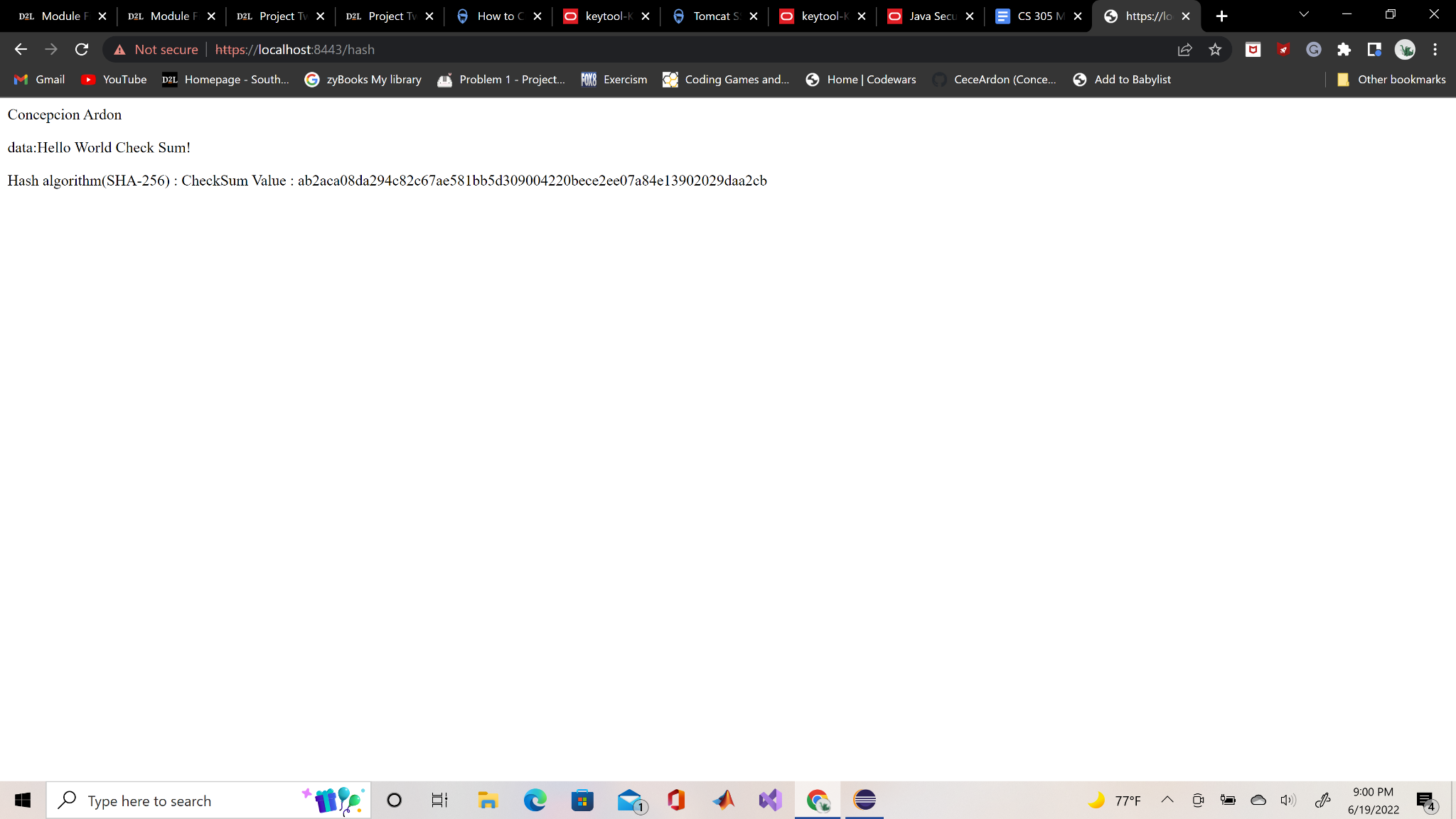
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

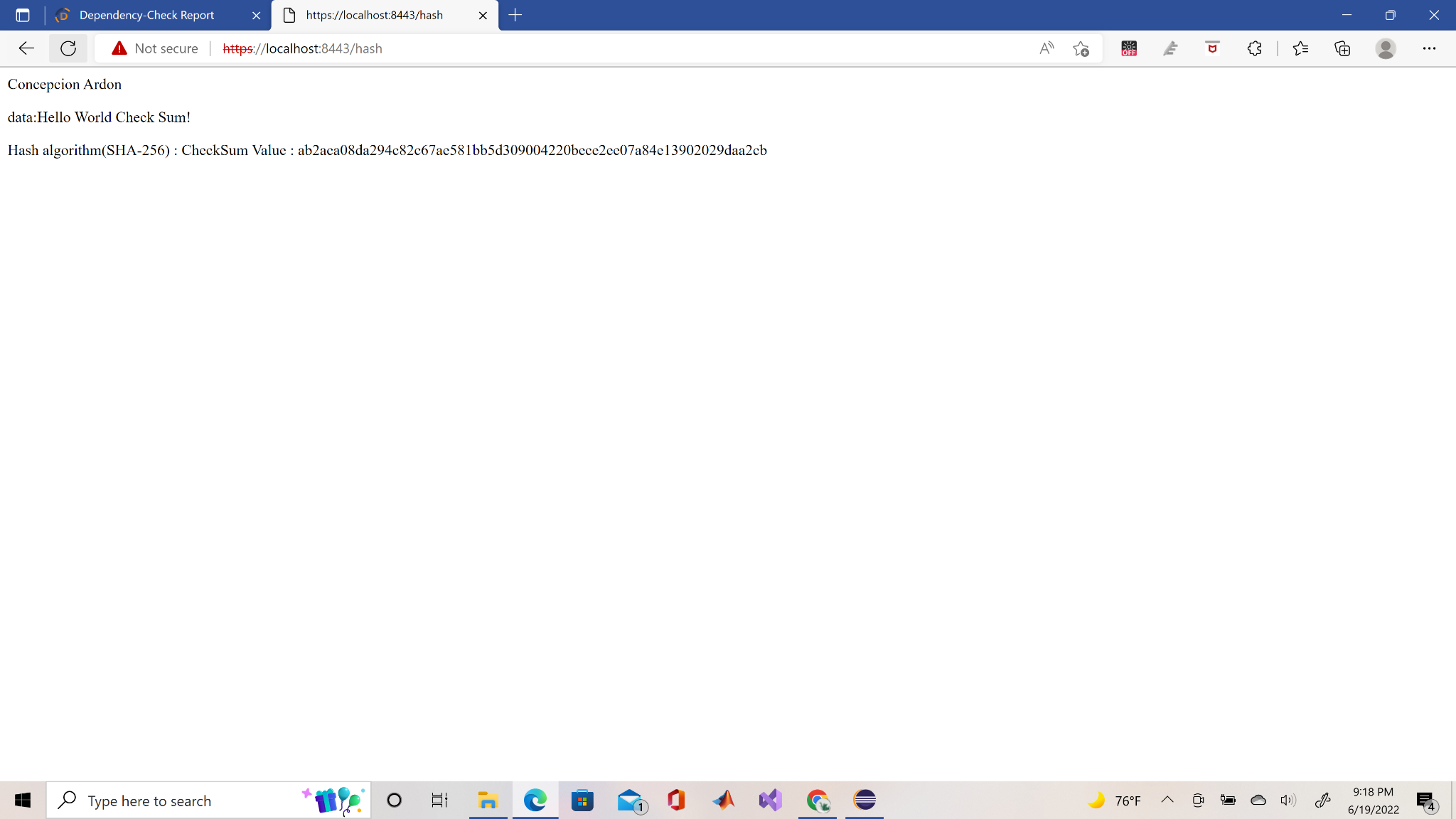
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

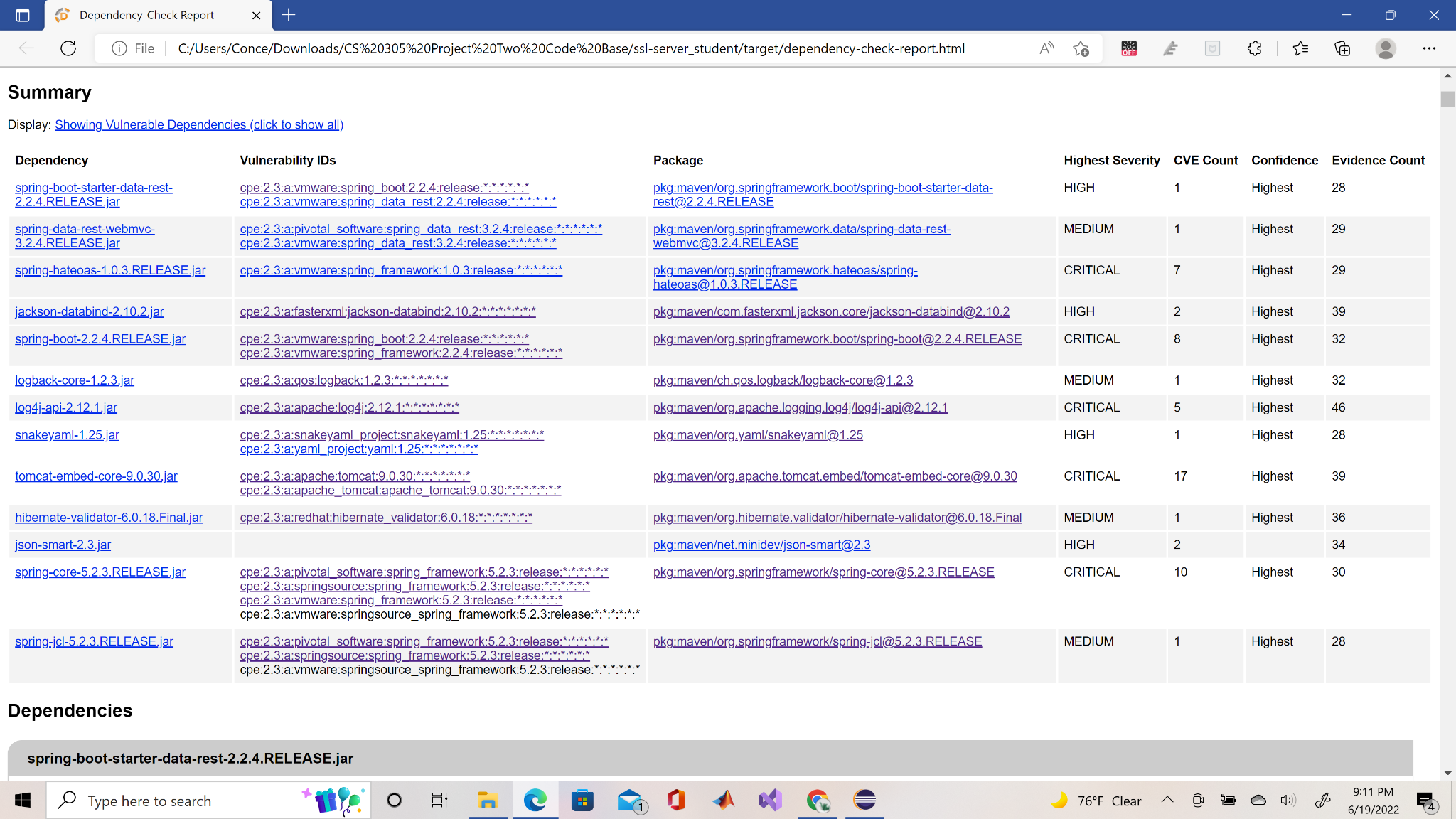
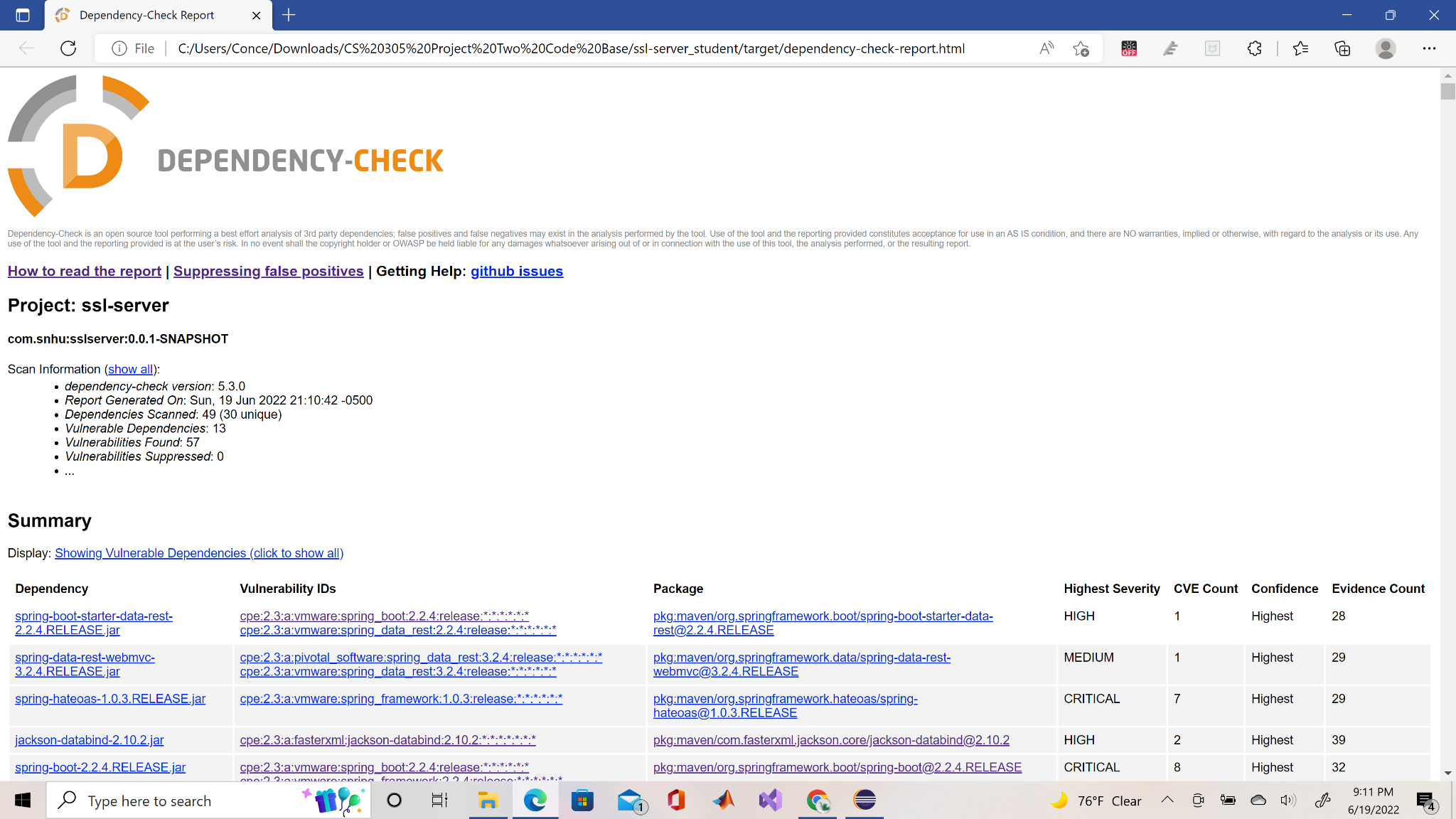
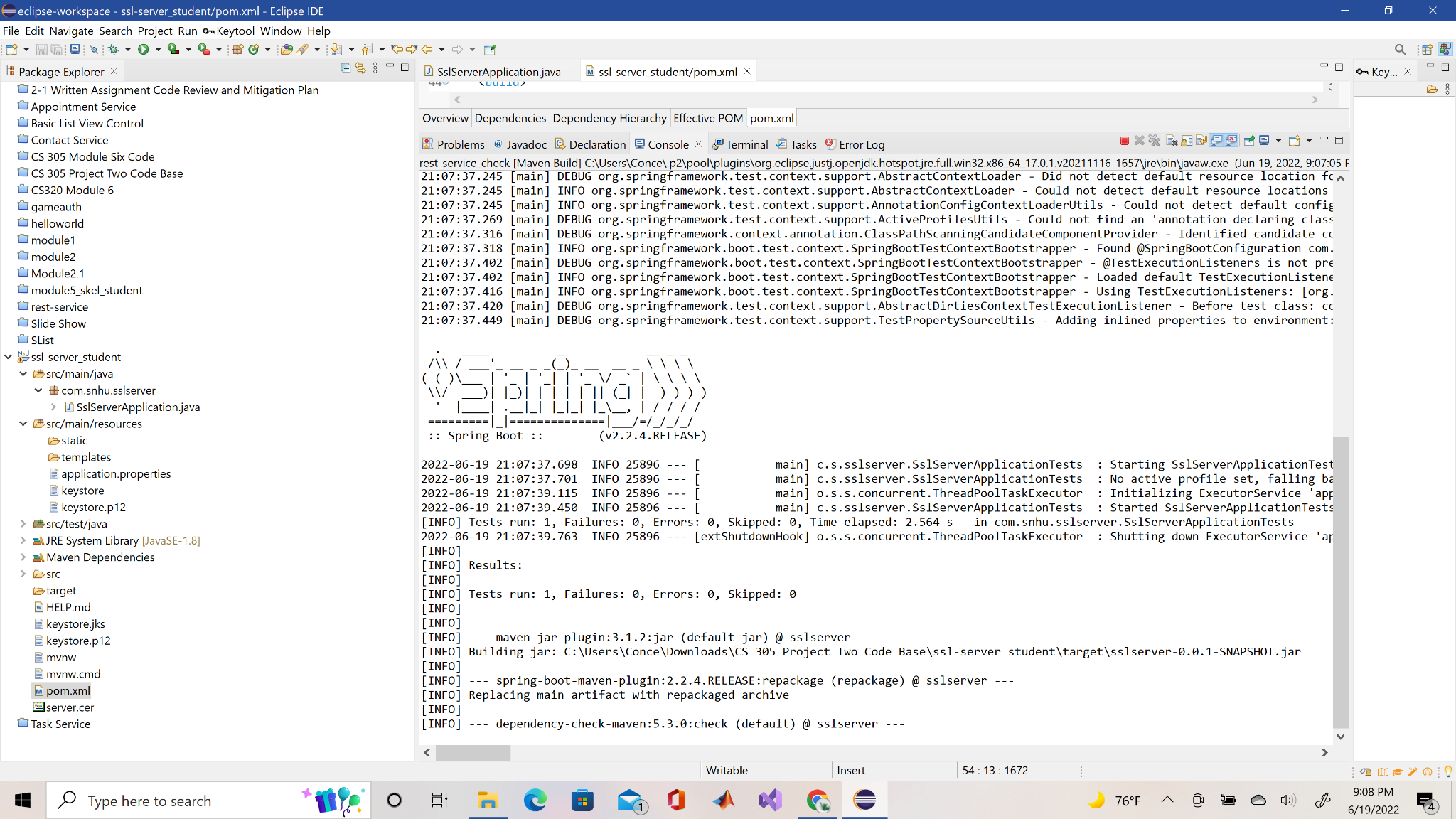
* Insert a screenshot below of the web browser that shows a secure webpage.



## 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.

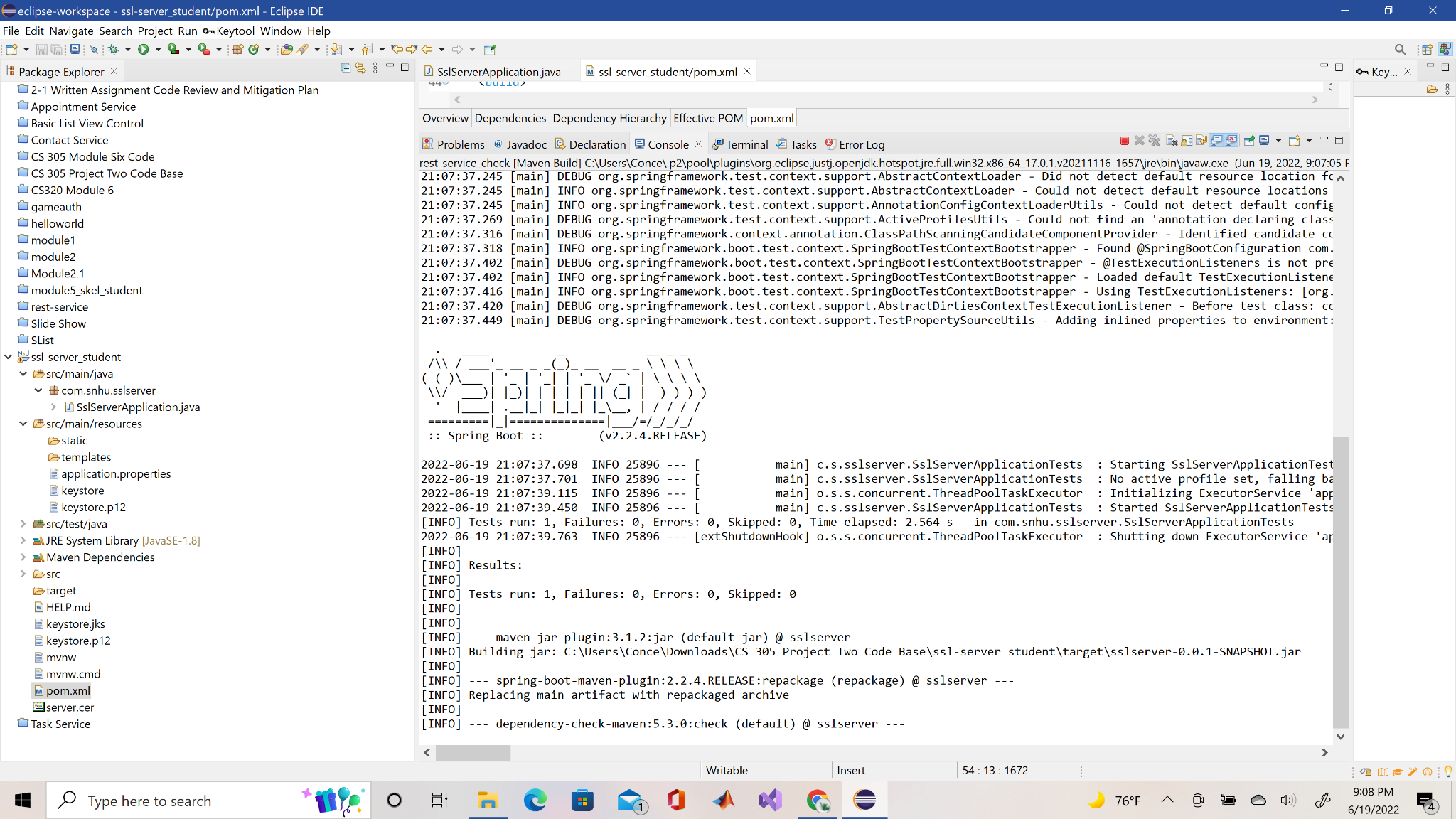
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



## 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.
* spring-boot-starter-data-rest-2.2.4.RELEASE.jar is outdated and if needed update it to the newer version 2.7.0. The dependency report also shows a false positive vulnerability that needs to be suppressed since no vulnerabilities were found in the package according to ossindex.sonatype.org.
* spring-data-rest-webmvc-3.2.4.RELEASE.jar shows some vulnerabilities in the dependency check, but it only shows 1 in the ossindex.sonatype.org. It is also outdated with the newer version being 3.7.0.
* spring-hateoas-1.0.3.RELEASE.jar shows some vulnerabilities in the dependency check, but no found vulnerabilities in the ossindex.sonatype.org. It is also outdated, and the new version is 1.12.11.
* jackson-databind-2.10.2.jar shows 2 vulnerabilities in the dependency check, but in the ossindex.sonatype.org it recognizes 3 vulnerabilities. It is also outdated with the newer version beings 2.13.3.
* spring-boot-2.2.4.RELEASE.jar shows many vulnerabilities in the dependency check, but it only shows 1 in the ossindex.sonatype.org. The dependency report also shows a false positive vulnerability that needs to be suppressed. It is also outdated making the newer version 2.7.0.
* logback-core-1.2.3.jar is outdated and needs to be updated to the new version 1.3.0-alpha16. The dependency check recognizes 2 vulnerabilities, but ossindex.sonatype.org recognizes 3.
* log4j-api-2.12.1.jar has no known vulnerabilities in ossindex.sonatype.org, but in the dependency check it shows 5 which are false-positives vulnerabilities. It is also outdated and the newer version is 2.17.2.
* snakeyaml-1.25.jar has 1 known vulnerability recognized by ossindex.sonatype.org, and 2 in the dependency check. It is also outdated making the newer version 1.30.
* tomcat-embed-core-9.0.30.jar has no known vulnerabilities in ossindex.sonatype.org, but in the dependency check, it shows 17 which are false-positives vulnerabilities. There is a newer version 10.1.0-M16.
* hibernate-validator-6.0.18.Final.jar has 1 known vulnerability in ossindex.sonatype.org, but in the dependency check, it shows 1 vulnerability. There is a newer version 8.0.0.Alpha3 which makes the version used outdated.
* json-smart-2.3.jar has 2 known vulnerabilities in ossindex.sonatype.org, but in the dependency check, it shows 2 vulnerabilities. There is a newer version 2.4.8 which makes the version used outdated.
* spring-core-5.2.3.RELEASE.jar has 0 known vulnerabilities in ossindex.sonatype.org, but in the dependency check, it shows 10 vulnerabilities. There is a newer version 5.3.21 which makes the version used outdated.
* spring-jcl-5.2.3.RELEASE.jar has 0 known vulnerabilities in ossindex.sonatype.org, but in the dependency check, it shows 1 vulnerability. There is a newer version 5.3.21 which makes the version used outdated.



## 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.

**Area of Security**

* Input Validation -I tightly validate all input into the function because of constructing a command input function. Validating input correctly will ensure a safe system and prevent injection attacks.
* APIs -If the command input capability has to be accessible from outside the system, I implemented a RESTful API. To make our systems secure, the API must be built in such a way that it precludes unauthorized access. At the API level, we could also perform some input validation.
* Cryptography - If we are going to use an API, we must use adequate encryption to secure and protect our data and that of our customers.
* Client/Server -If we want to give our function API access, we'll need to make sure we utilize the right certificates to keep data safe throughout HTTPS queries.
* Code Error: All command input functions, as well as any API access layer code, will need to be code reviewed.
* Code Quality - I made sure that secure coding practices and patterns were followed to prevent any failures or errors.
* Encapsulation - If our command input function requires access to data stored in our system, we must guarantee that we are appropriately using it and that our data structures are not harmed.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.

The practice of upgrading dependencies and suppressing any false positives in dependencies aids in the maintenance and security of software by preventing defects and vulnerabilities. The importance of updating and having dependency checks for a corporation is that it helps to increase security by preventing any errors or bugs from compromising clients' private information or accounts. If the company's software was not kept up to date, there would be several security flaws in the system.

* Point out best practices for maintaining the current security of the software application to your customer.

The best practices for maintaining the current security of the software application are to patch the software and systems, educate and train users, automate routine tasks, enforce least privilege, create a robust IR plan, document the security policies, segment the network, Integrate security into our SDLC, monitor user activity, and measure. Many attackers take advantage of known flaws in old or out-of-date software. Ensure that all of our systems are patched to prevent common attacks. One of the most effective software security strategies is patching on a regular basis. Our data and assets will be better protected if the company has a well-organized and well-maintained security training curriculum for its workers. Include security coding training for developers and awareness training for all employees. Automated detection of open ports, security misconfigurations, and other issues is used by attackers. As a result, we won't be able to safeguard the company's systems just through manual means. Rather, automate routine security chores like reviewing firewall modifications and device security configurations. By automating routine operations, our security team may concentrate on more strategic security projects. Ascertain that users and systems have the bare minimum of access rights to accomplish their job duties. By removing unneeded access permissions that might lead to a variety of compromises, enforcing the concept of least privilege greatly minimizes our attack surface. Have a comprehensive incident response (IR) plan in place to detect and mitigate the effects of an attack. Maintain a knowledge base with thoroughly documented software security policies. The employees, such as network administrators and security personnel, will be able to comprehend what operations you're undertaking and why. The notion of least privilege is used to segment our network. Attacker movement is limited by proper network segmentation. Determine where the essential data is housed and implement security rules to limit traffic to and from those network segments. From start to finish, incorporate software security efforts into our organization's software development life cycle (SDLC). Architecture risk analysis, static, dynamic, and interactive application security testing, SCA, and pen testing should all be included in this list. Initially, including security in our SDLC will take time and effort. Trust but double-check. To ensure that users are following software security best practices by monitoring their activity. It also allows you to see suspicious behavior like privilege abuse and impersonation. Define meaningful and relevant critical metrics for the company. Metrics that are well-defined can help us assess our security posture over time.

**Note**

The zipped file attachment is the refactor code for the application. It includes the dependency checks, CER, security measures, and updates applied to the application.