****

# 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** | **02/20/21** | **Stephen Blackburn** |  |

## 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

Stephen Blackburn

## 1. Algorithm Cipher

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

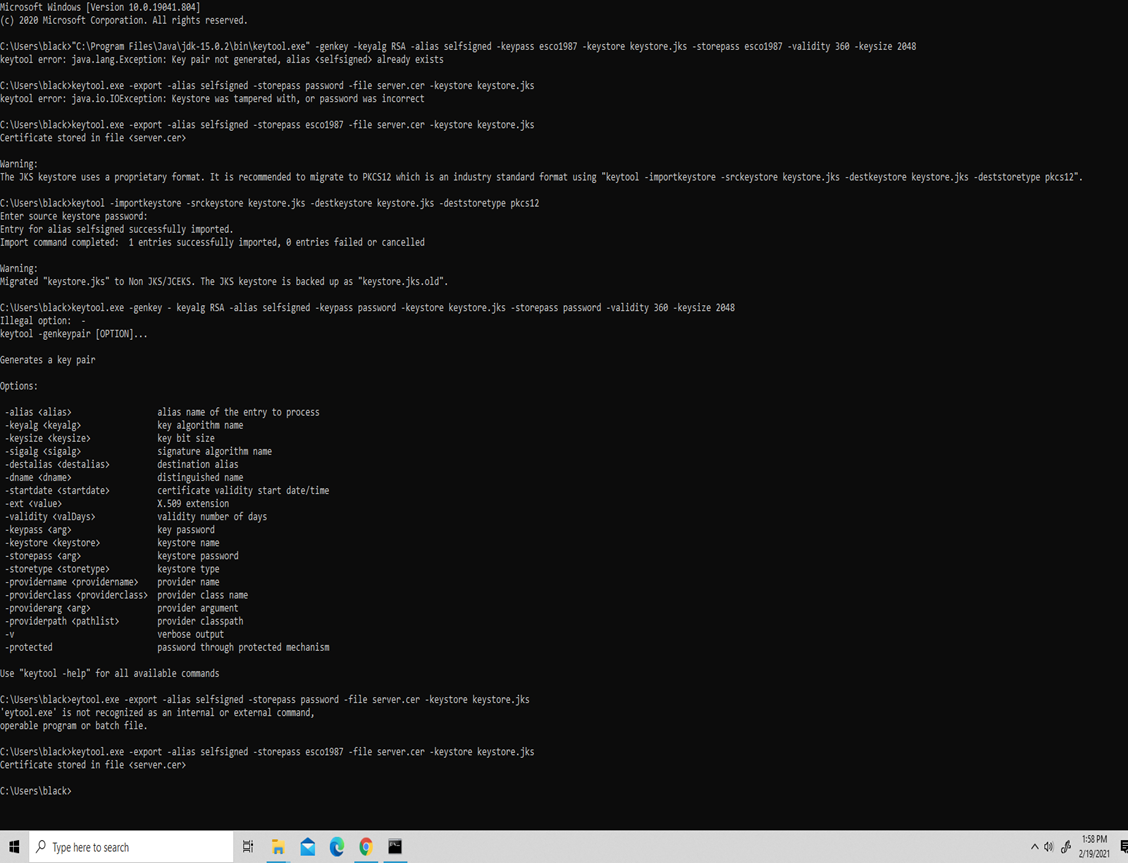
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

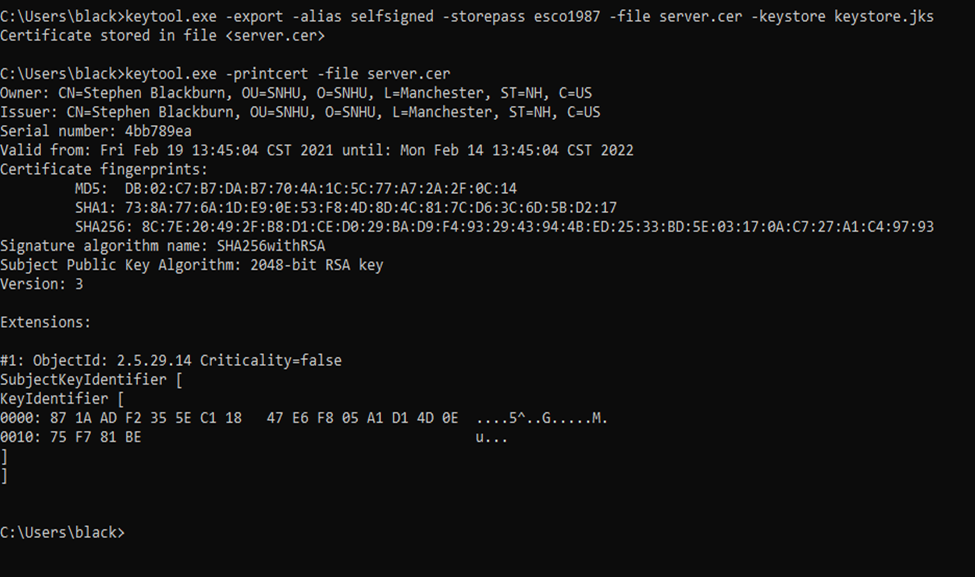
The algorithm cypher used is TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256. TLS 1.2 does not have any known vulnerabilities. EDCHE (Elliptic Curve Diffie-Hellman with Ephemeral keys) is a key exchange that supports forward secrecy. Forward secrecy protects past communications even if a current key has been compromised. The server’s certificate should have an RSA public key and corresponding ECDHE parameters, which provides server authentication. AES\_128 is a symmetric encryption cipher with 128-bit keys. CBC stands for Cipher Block Chaining mode, which is a way of employing a block cipher to encrypt a variable-length piece of data. The Cipher Block Chaining mode has a few vulnerabilities, but still provides reasonable security. SHA256 is a hash function that underlies the Message Authentication code feature. This stops messages from being tampered with along transit. SHA-256 is secure as it is almost impossible to reconstruct initial data. It has a 0.01% chance of collision. An attack would take 2256 attempts to get the initial data and is near impossible for two messages to have the same hash value. This algorithm is still one of the better algorithms to use for security to date.

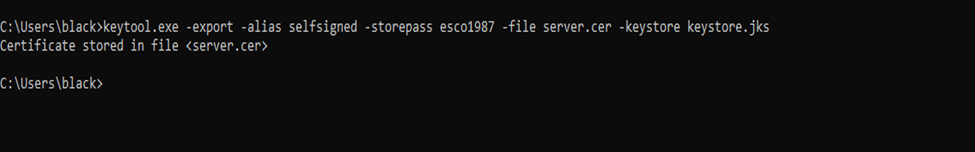
## 2. Certificate Generation

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.



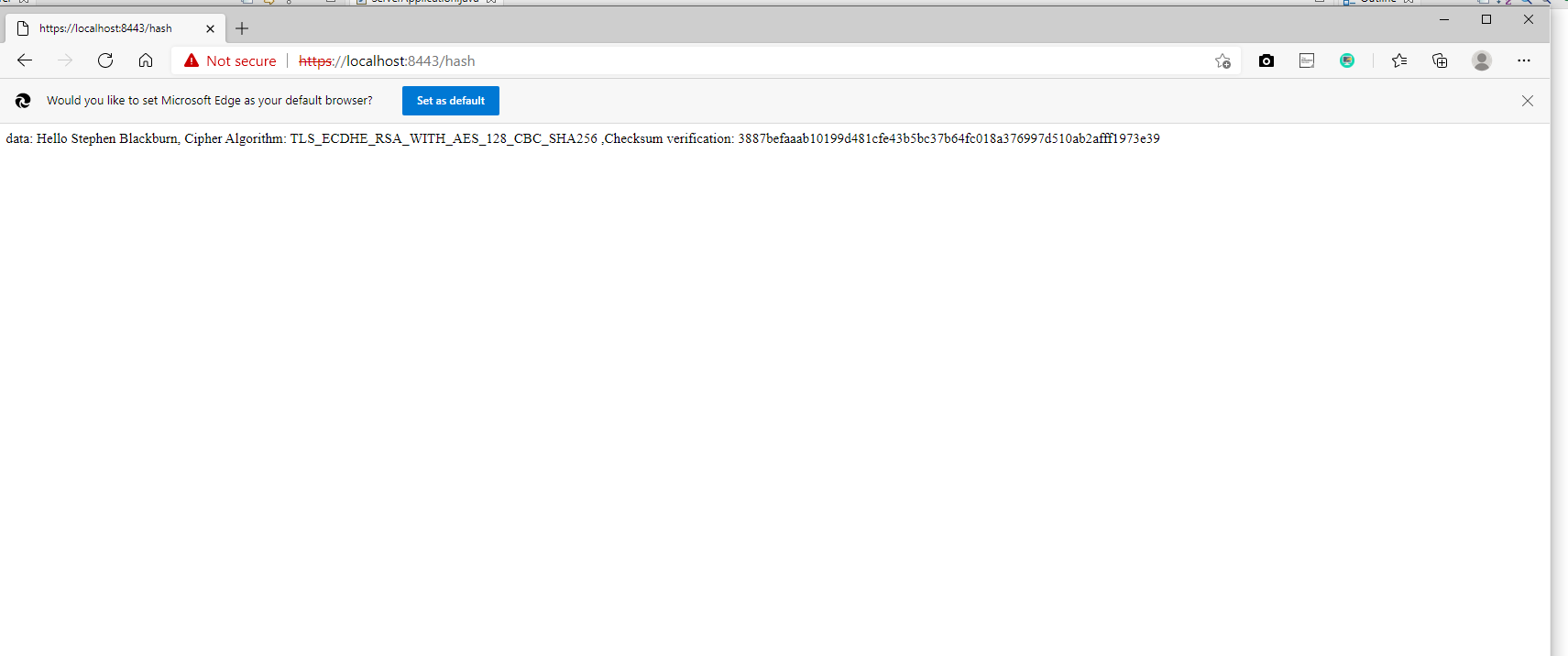




## 3. Deploy Cipher

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.

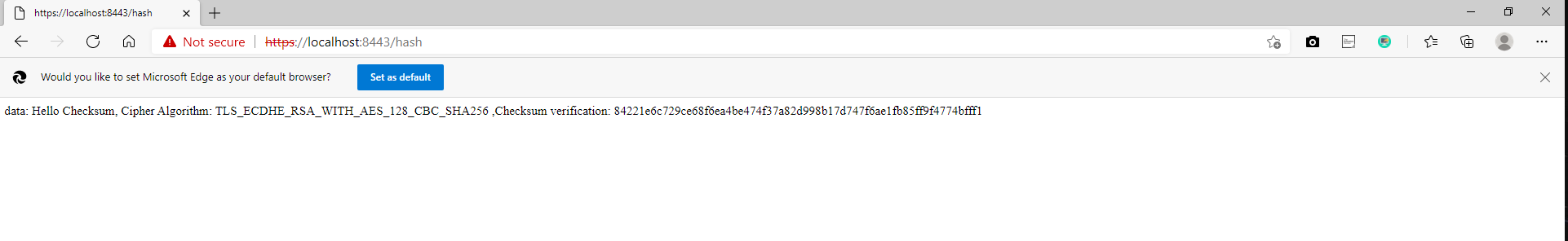
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

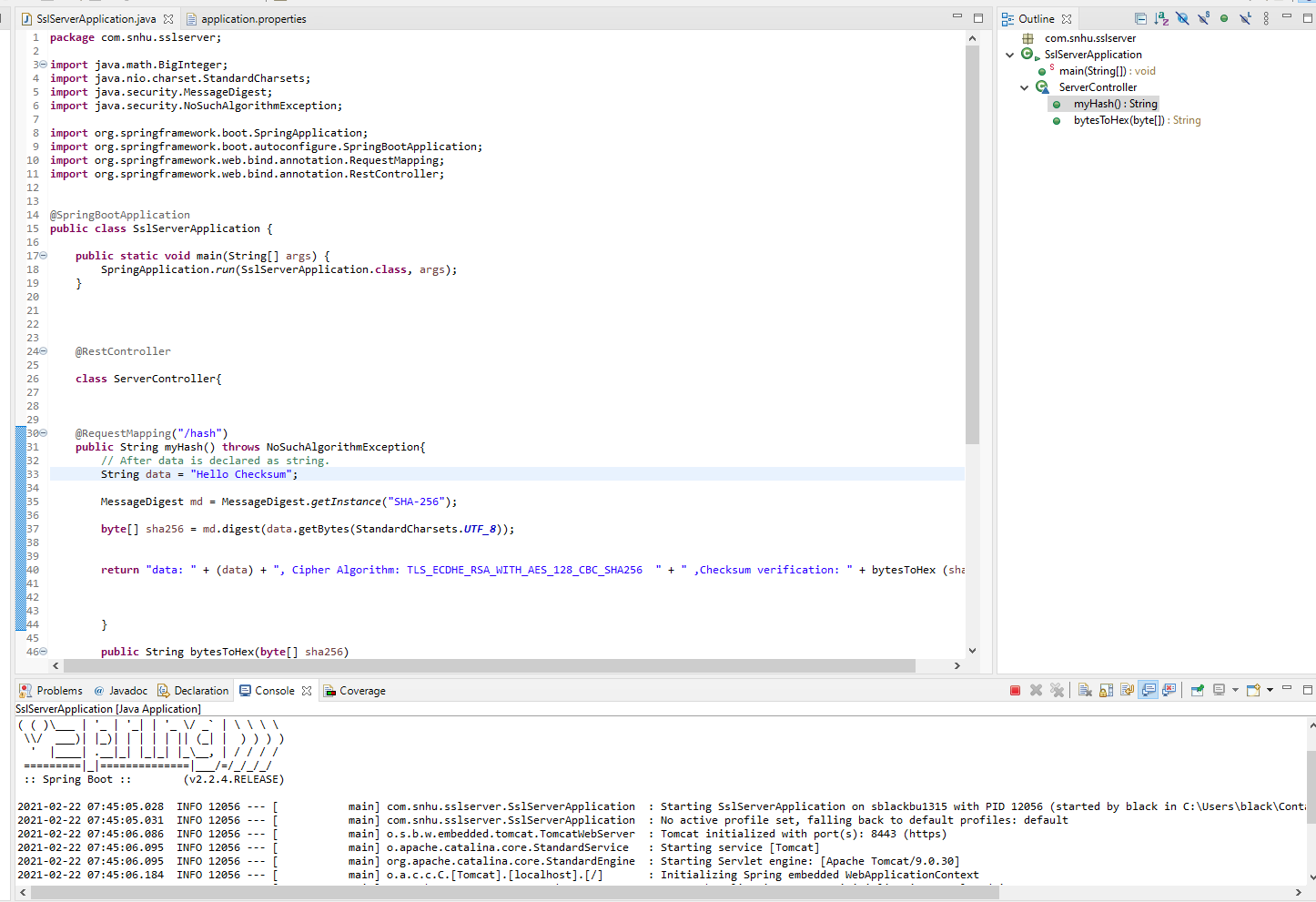


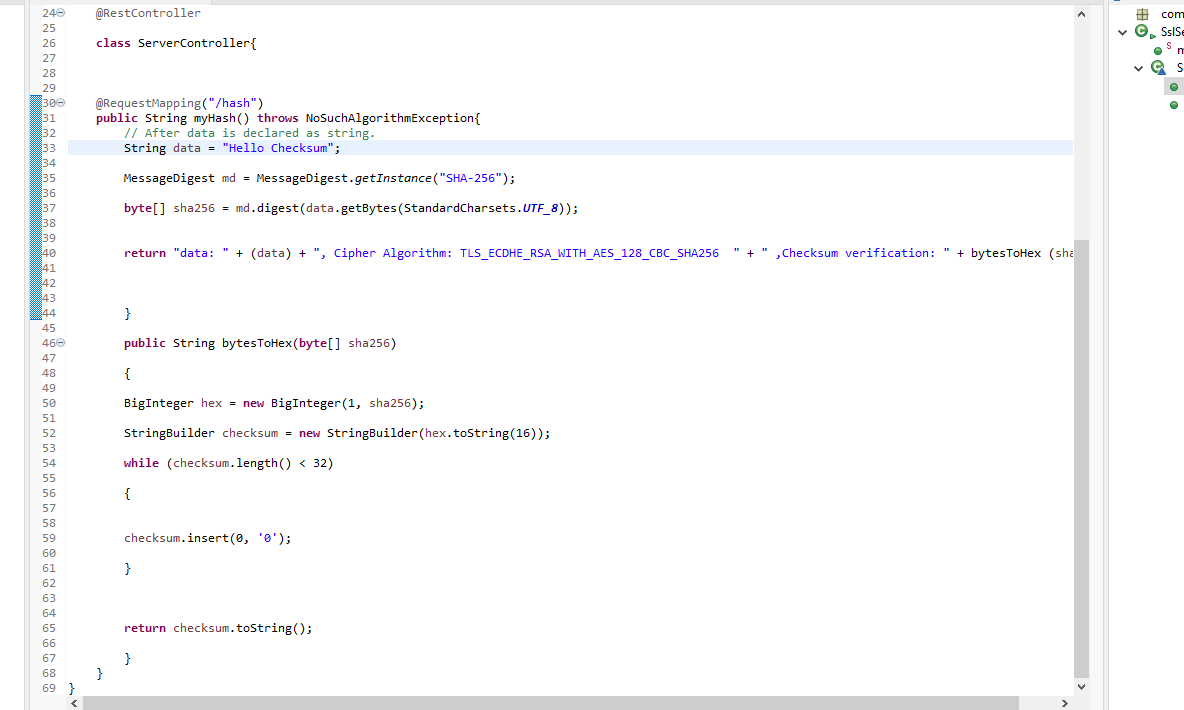
I could not get the communication to become secure. I think that I have too many JDK’s and am probably not working on the right one. However, with time-constraints I could not go and delete and try to fix all of them.

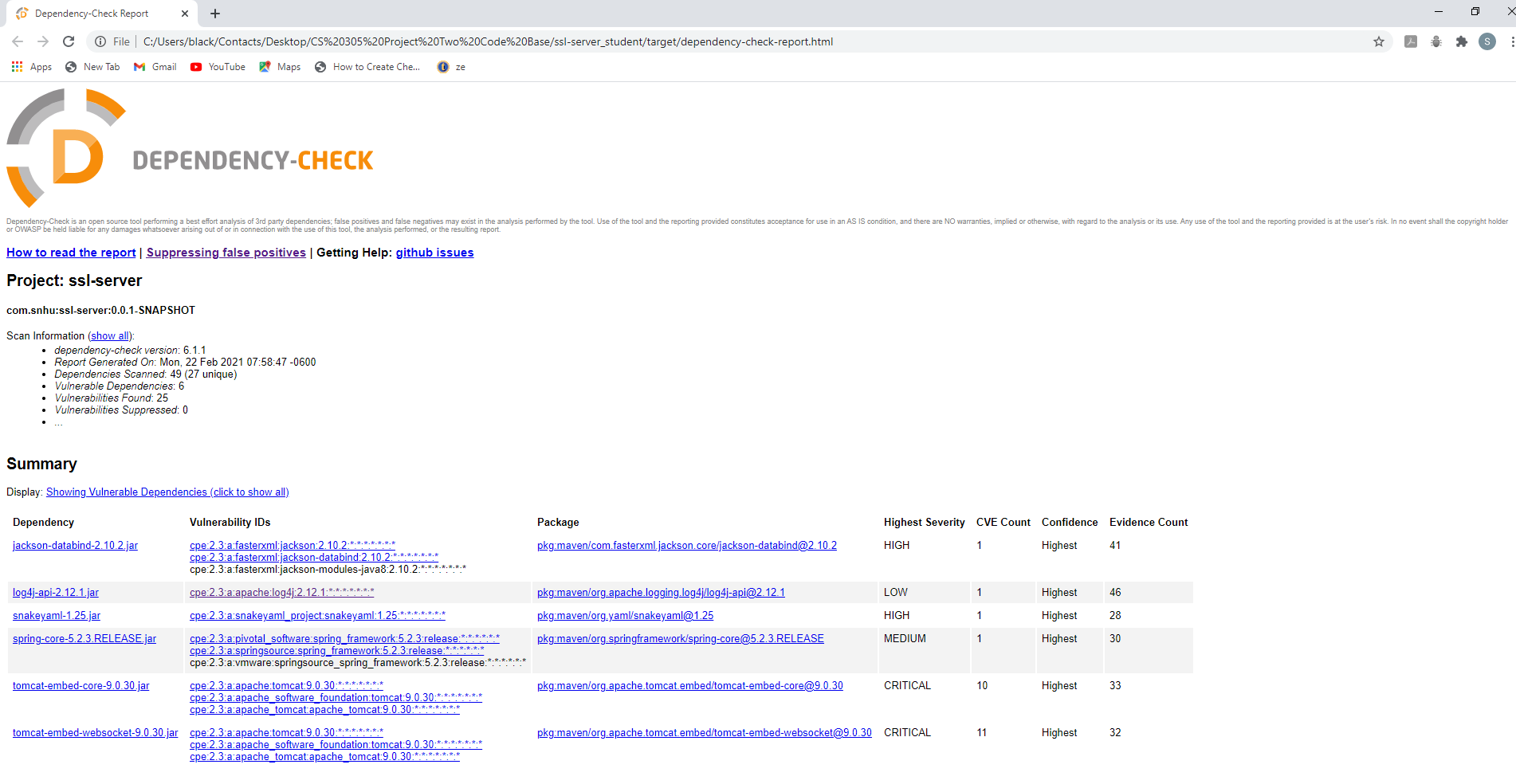
## 5. Secondary Testing

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





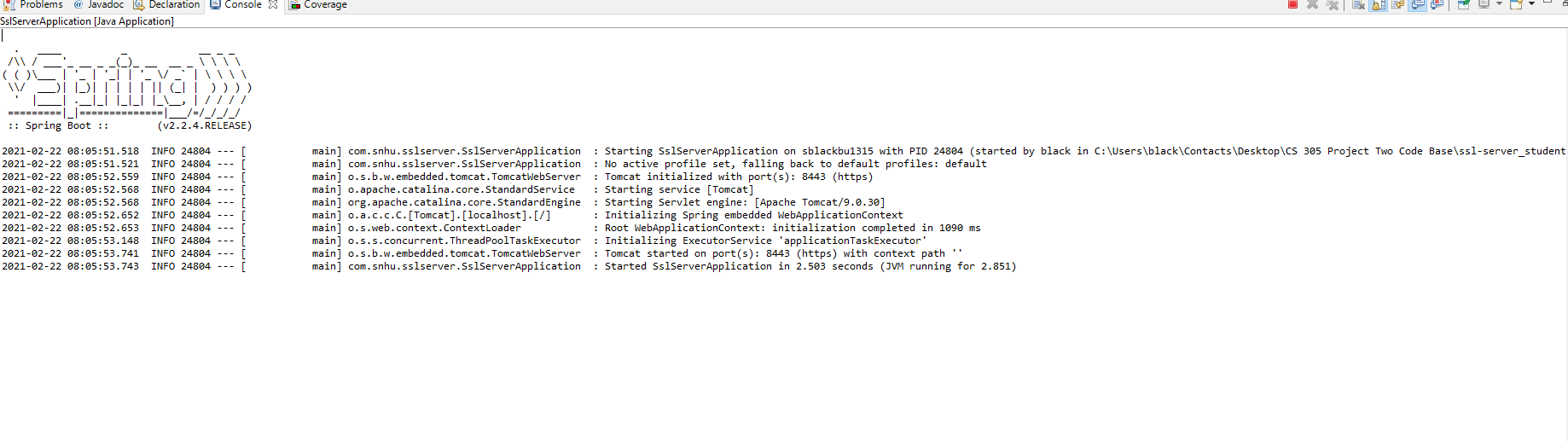


## 6. Functional Testing

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.

There are no errors in the code and the dependency check did not bring up any errors on my part, except that the Java versions are off, which is a problem from switching from old to new technology between classes.



## 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.

When viewing this application from the VAPFD, there are several architectural issues. Most of the API’s are out of date and need to be updated. The server will not allow me to use a trusted certificate, which is probably because my compiler and JDK version are off. I have tried to use an old JDK uninstall tool which did not help. I could have probably written a private data structure for my hashing function as it is not taking in any input, but just hashing the data.

I added layers of security by generating a CA and using a secure cipher algorithm. However, like said before something went wrong with my server certificate and I could not get secure communication. I also made sure that there was no input, which adds an extra layer of security from input attacks. Finally, I researched standard checksums and used what I thought was the best checksum for code.

The biggest problem in the program is all of the old API’s and dependencies being used. If all of these were updated the application would be much more secure. Also, if my certificate would work how it is supposed to there would be a secure connection. I think it is also important to continuously research known vulnerabilities for the cipher algorithm and dependencies being used. Hackers find new vulnerabilities everyday and in cyber-security it is important to stay ahead.

References

SHA-256 algorithm overview. (2019, September 16). Solarwinds MSP. https://www.solarwindsmsp.com/blog/sha-256-encryption