CS 305 Project Two
Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	12/11/21	Mark Webster	Updated text sections and added screenshots

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.

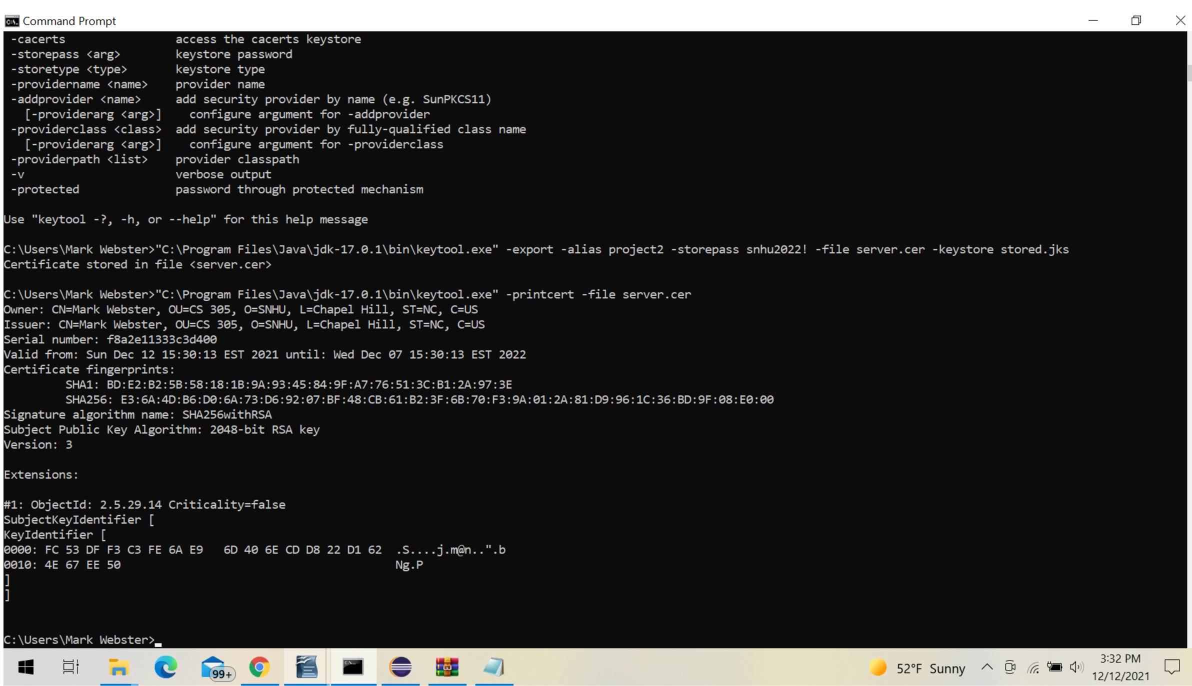
Developer
Mark Webster

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.

I chose SHA-256 because it is currently the most widely used hashing algorithm by both governments and companies. SHA-256 uses a 256-bit key, meaning it's key-space (the number of possible keys that could exist for any one encryption) means that centuries or millenia of computing time are necessary to make any in-roads on cracking it. This also means that the chances of collision (different inputs generating the same hashed outputs) are infinitesimally low. SHA-256 is a symmetric algorithm, in that the same key is used to encrypt and decrypt. Also, using SHA-256, if even one letter is changed in our original information that we want to encrypt, the entire hash is completely different. This is because SHA-256 utilizes compression, padding, and “pseudorandom” random numbers—all of these combine to take large amounts of input, obscure it, and output a succinct hash value. Currently, SHA-256 and SHA-512 belong to the SHA-2 family of hashing algorithms. SHA-1 was no longer recommended by NIST beginning in 2010, and SHA-2 became the standard. SHA-3 is currently in development.

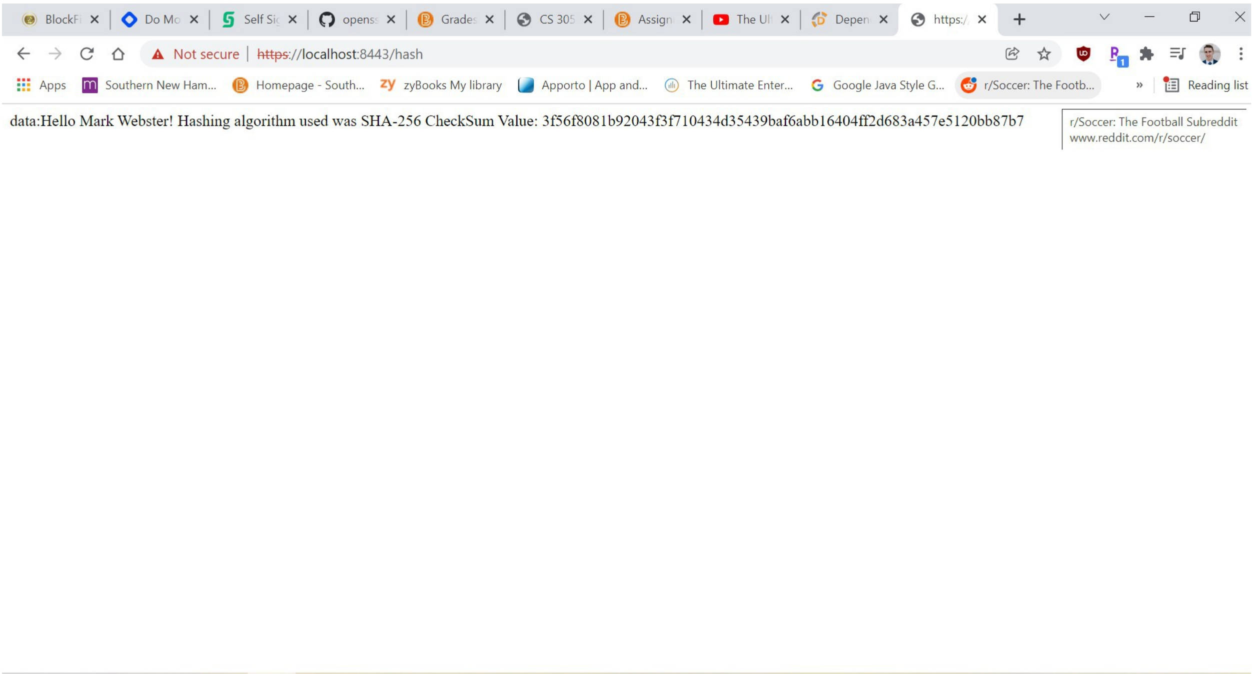
2. Certificate Generation
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.



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.

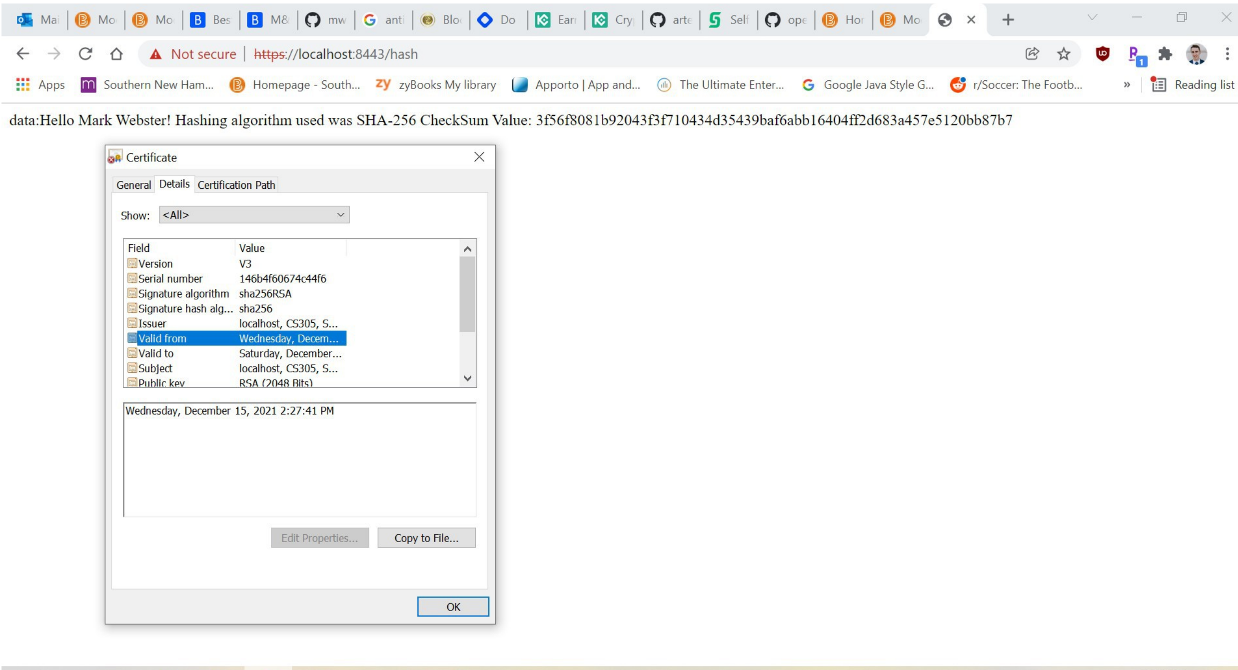
- Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure comecure communication works successfully.

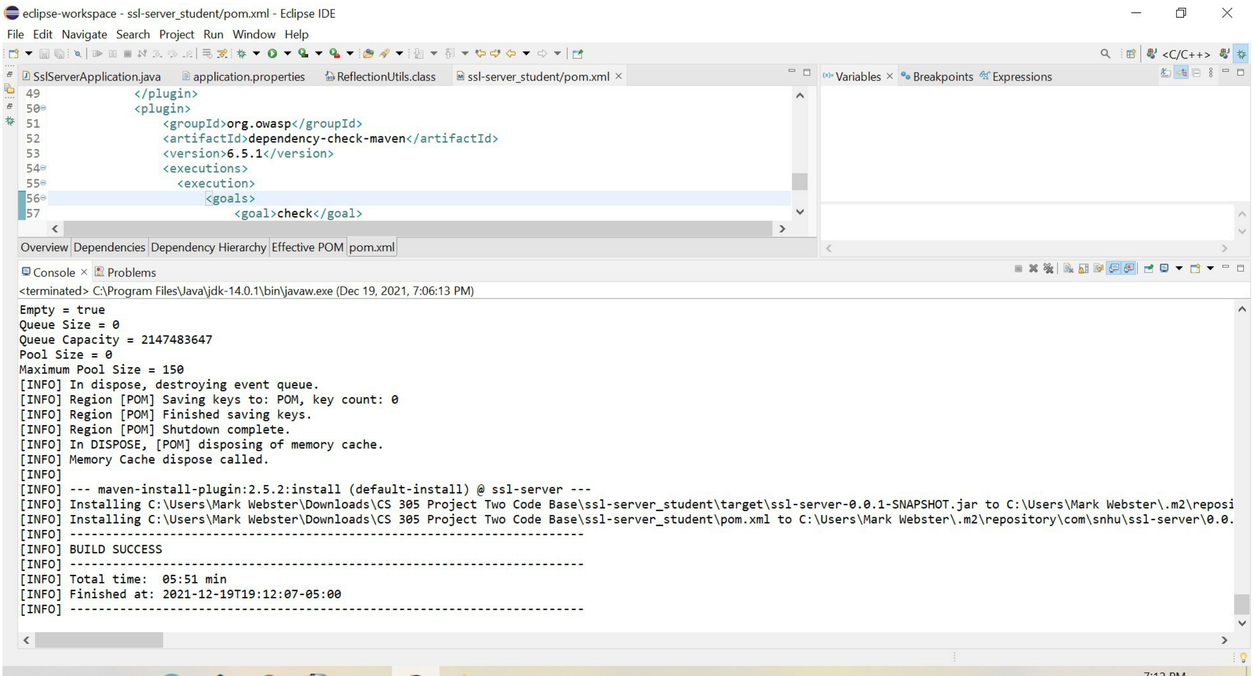
- Insert a screenshot below of the web browser that shows a secure webpage.

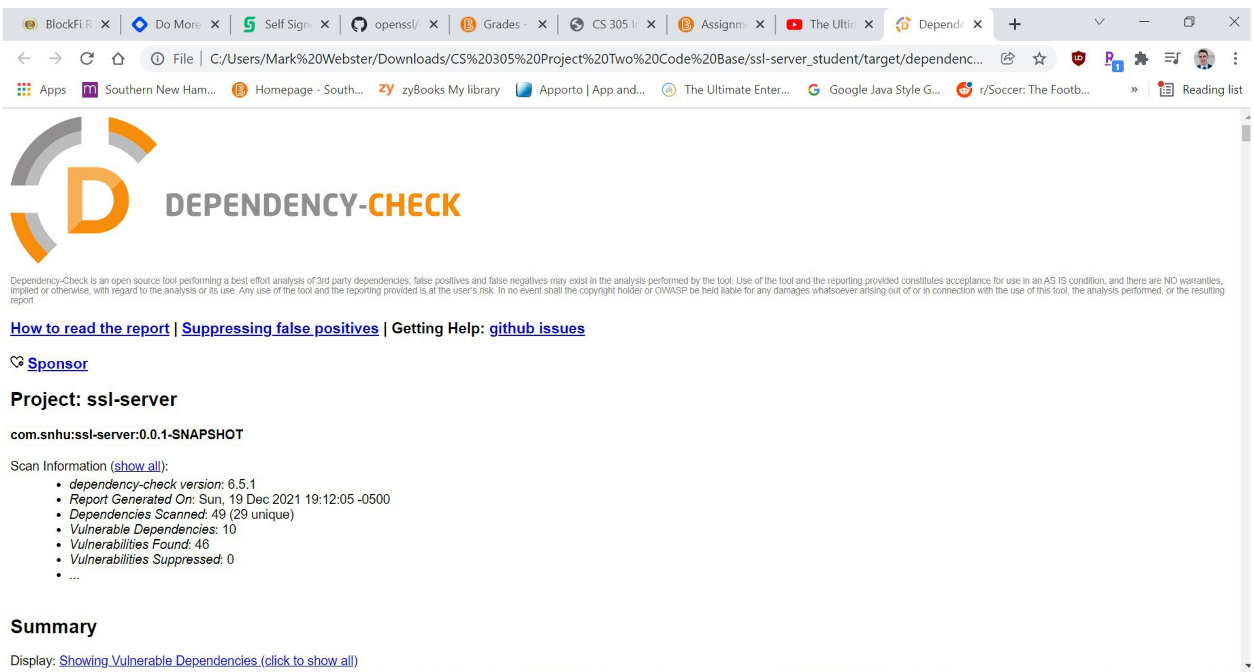


5. Secondary Testing

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.

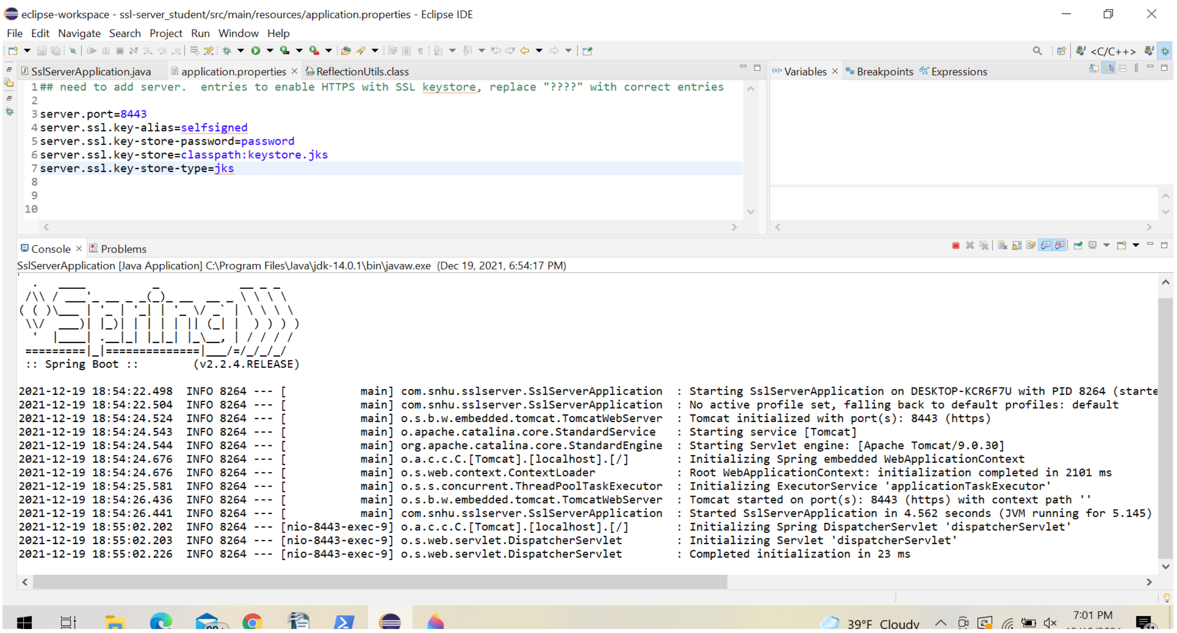
- Include the following below:
  - A screenshot of the refactored code executed without errors
  - A screenshot of the dependency check report


eclipse-workspace - ssl-server_student/pom.xml - Eclipse IDE
File Edit Navigate Search Project Run Window Help
SslServerApplication.java
application.properties
ReflectionUtils.class
ssl-server_student/pom.xml
</plugin>
<plugin>
<groupId>org.owasp</groupId>
<artifactId>dependency-check-maven</artifactId>
<version>6.5.1</version>
<executions>
<execution>
<goals>
<goal>check</goal>
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Variables Breakpoints Expressions
Console Problems
<terminated> C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe (Dec 19, 2021, 7:06:13 PM)
Empty = true
Queue Size = 0
Queue Capacity = 2147483647
Pool Size = 0
Maximum Pool Size = 150
[INFO] In dispose, destroying event queue.
[INFO] Region [POM] Saving keys to: POM, key count: 0
[INFO] Region [POM] Finished saving keys.
[INFO] Region [POM] Shutdown complete.
[INFO] In DISPOSE, [POM] disposing of memory cache.
[INFO] Memory Cache dispose called.
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ ssl-server ---
[INFO] Installing C:\Users\Mark Webster\Downloads\CS 305 Project Two Code Base\ssl-server_student\target\ssl-server-0.0.1-SNAPSHOT.jar to C:\Users\Mark Webster\.m2\reposi
[INFO] Installing C:\Users\Mark Webster\Downloads\CS 305 Project Two Code Base\ssl-server_student\pom.xml to C:\Users\Mark Webster\.m2\repository\com\snhu\ssl-server\0.0.
[INFO] --
[INFO] BUILD SUCCESS
[INFO] --
[INFO] Total time: 05:51 min
[INFO] Finished at: 2021-12-19T19:12:07-05:00
[INFO] --


C:/Users/Mark%20Webster/Downloads/CS%20305%20Project%20Two%20Code%20Base/ssl-server_student/target/dependenc...
DEPENDENCY-CHECK
How to read the report | Suppressing false positives | Getting Help: github issues
Sponsor
Project: ssl-server
com.snhu:ssl-server:0.0.1-SNAPSHOT
Scan Information (show all):
dependency-check version: 6.5.1
Report Generated On: Sun, 19 Dec 2021 19:12:05 -0500
Dependencies Scanned: 49 (29 unique)
Vulnerable Dependencies: 10
Vulnerabilities Found: 46
Vulnerabilities Suppressed: 0
Summary
Display: Showing Vulnerable Dependencies (click to show all)

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.



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 increase security in the SSL Server Application, we had to secure our API communications and implement the algorithm cipher suite. To the first end, I created a self-signed certificate and imported the information into the application properties. This allowed Spring to run our web application using HTTPS instead of HTTP. While Google no longer accepts self-signed certificates, the certificate was successfully created and Chrome noted its presence. To the second end, I created a ServerController class and built a hashing function that used Java's MessageDigest feature to process a user string into a hashed 256-bit string. I then used Spring's "RequestMapping" function to call the hash function whenever the user navigated to the “/hash” address. This hashed the example string (my name) into an encrypted string.

Artemis Financial and their customers will have increased peace-of-mind knowing that the web application is running using HTTPS, and that data can be encrypted to obscure its content for malicious actors. The government requires that institutions that handle sensitive data have some form of safeguards to protect that data. In addition, we ran a dependency check to see if any of our dependencies had vulnerabilities that could compromise the application. As shown by the recent revelation about Log4j's vulnerabilities, there are many issues that are yet undiscovered. Making ongoing security maintenance a part of the development cycle is crucial to staying on top of this evolving nature. All input must be validated, each connection secured, and every communication turned upside down and shaken to make sure it can be trusted.