GLOBALRAIN

Practices for Secure Software Report

Kyle Dale

SNHU: CS-305-Software Security

Professor Lyon

February 19, 2023

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	2/16/2023	Kyle Dale	Completed rough draft of title page, Dev, Algorithm Cipher, and Cert Gen, Cipher Deploy.
1.1	2/18/2023	Kyle Dale	Completed rough draft of deploy Secure comms, secondary and functional testing, summary, best practices, references. Tried to fix secure comms.
1.2	2/19/2023	Kyle Dale	Formatted, refined, fixed secure connection.

Client

ARTEMIS FINANCIAL

Developer
Kyle Dale

1. Algorithm Cipher

Artemis Financial is a consulting company that develops individualized financial plans for their customers. This includes savings, retirement, insurance, and investment accounts. Their goal is to modernize their operations, and make the most of current effective software security. When considering security practices and which cipher would best fit Artemis Financial, it is paramount to consider the nature of their business. As Artemis Financial will be dealing with highly sensitive information and data, security will be the primary concern. As such, I would recommend AES-256 when considering encryption for data at rest, and I would recommend SHA-256 when considering the checksum for data in transit.

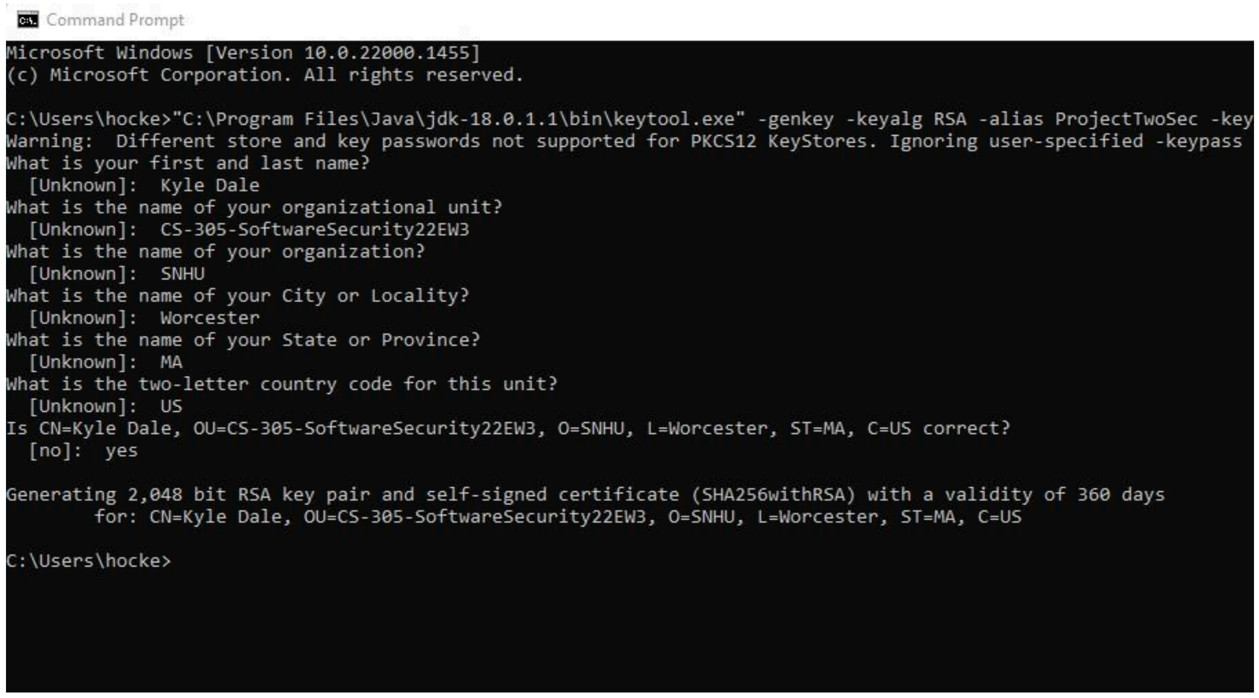
When deciding on the proper cipher for Artemis Financial to employ, it was important to consider a few concepts relating to cryptography. To start, the concept of hashing is important for verifying the integrity of transmitted data. When a message using hashing is sent, the message is imprinted with an original hash digest that takes the form of a unique code (Crane, 2021). This aids in identifying if the message were to be intercepted and the data altered. If this occurs, the hash digest will be altered, notifying the recipient the data has been altered. When referring to bit levels, in essence, the bit level refers to the approximate computations needed to recover the undoctored text. As AES-256 and SHA-256 support 256-bits, this basically means that an adversary would need to perform 2^{256} calculations to recover the plaintext (Staff, 2020). When combined with the additional function of hashing to store passwords as cipher texts, when run through a hashing algorithm, rather than stored just as the plaintext, AES-256 and SHA-256 can provide staunch, layered security for Artemis Financial even when some firewalls are breached (Detlefsen, 2014).

When discussing the use of RNG (random number generation) and symmetric versus non-symmetric keys in cryptography, the topics are heavily interwoven. The main difference between symmetric and asymmetric encryption, is that symmetric encryption utilizes one 'key' to encrypt and decrypt data, whereas asymmetric encryption uses two keys, a public and private key (Arnaud, 2023). The role that RNG takes in this process is to generate the random, unique key. The key is essentially a variable value that is applied to an unencrypted string or block with an algorithm to produce encrypted data (Loshin, 2021). This concept can be applied with encryption to facilitate multiple levels of security around data. The primary danger relating to either symmetric or asymmetric encryption differs slightly. When concerning symmetric encryption, if your key is compromised through mismanagement or breach, all of your encrypted data is at risk of decryption (Arnaud, 2023). Whereas with asymmetric encryption, your private key is never shared, only the public key (Arnaud, 2023). This allows anyone to encrypt a message to you, utilizing your public key, and only you have the ability to decrypt the message using your personal private key. The primary concern with asymmetric encryption is, if you happen to lose your uniquely generated private key, you will be unable to decrypt any of the data encrypted using your public key (Arnaud, 2023). The encrypted data sent to you will then be lost forever.

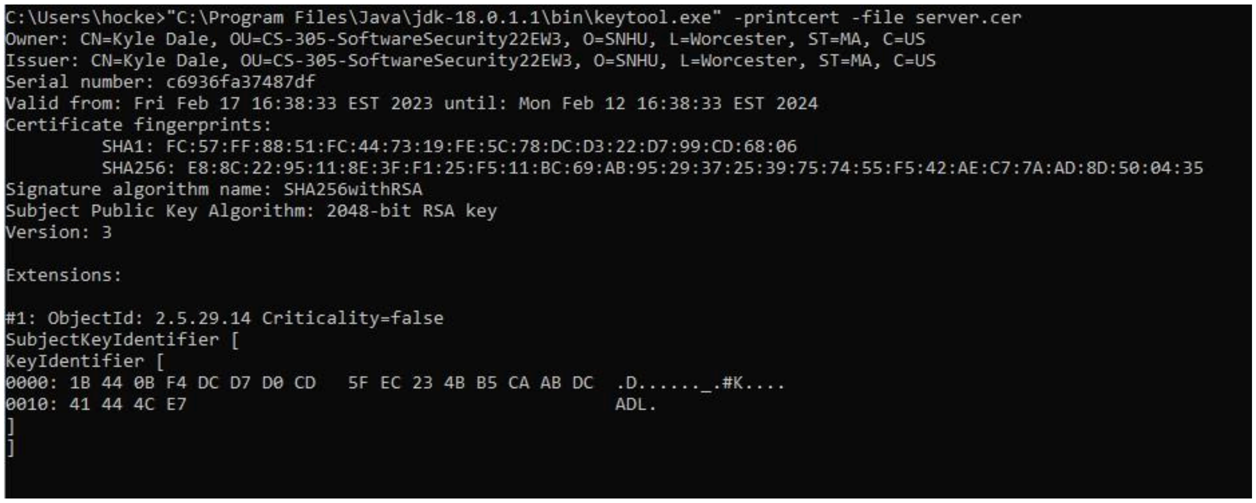
Cryptography, or the art of concealing data, has been a fascination of humankind for nearly 4,000 years, with the earliest recorded evidence dating back to ancient Egypt (Team, 2022). Although this original use of cryptography was to shelter knowledge for the privileged, by 500 BC, Sparta had begun implementing cryptology in military communications (Team, 2022). Although these early methods were technically encryption, they relied primarily on simple unit replacement methods. By World War Two however, with the advent of the German Enigma, encryption had developed in complexity astronomically. By feeding the Enigma a unique key, generated daily, German command could encrypt any message sent via radio or telegram. The nature of the Enigma required attempting a staggering 17,000 different combinations within a 24-hour period to crack the code (Team, 2022). This led to the creation of the Turing Machine during World War Two by Alan Turing and associates, which is largely regarded as one of the foundational models of computability and computer science (De Mol, 2018).

2. Certificate Generation

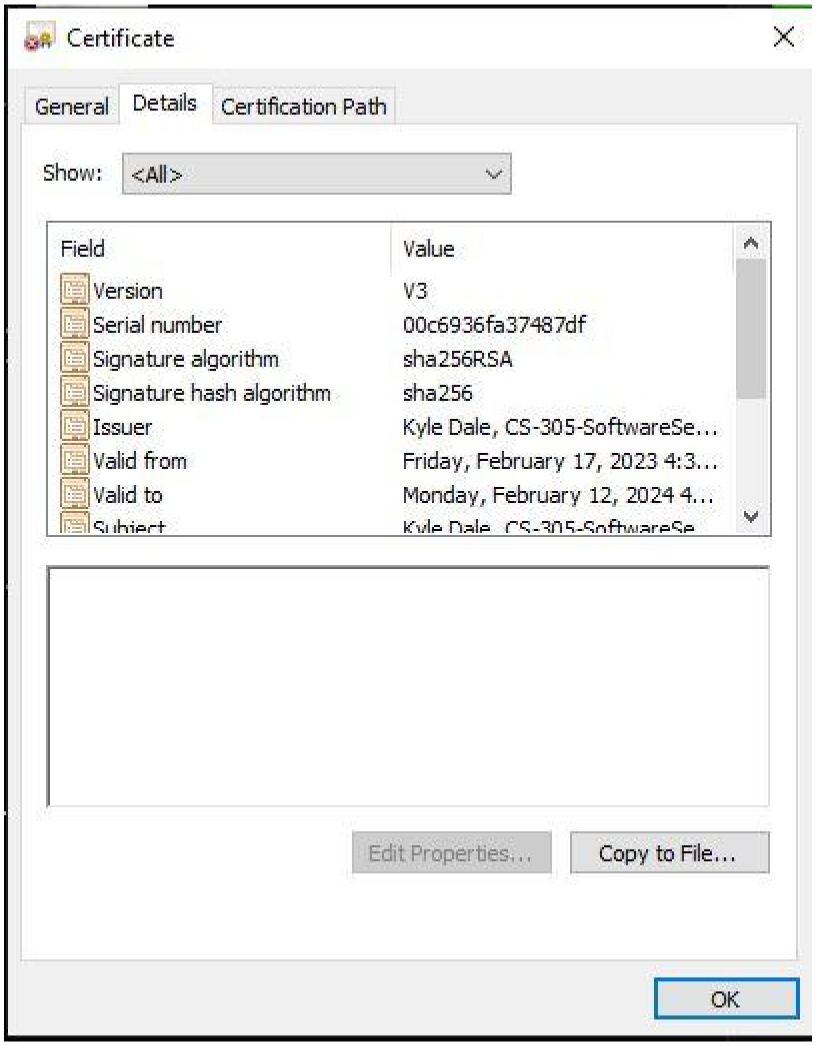
Inserted below are screenshots of the CER file:



```
Microsoft Windows [Version 10.0.22000.1455]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hocke>"C:\Program Files\Java\jdk-18.0.1.1\bin\keytool.exe" -genkey -keyalg RSA -alias ProjectTwoSec -key
Warning:  Different store and key passwords not supported for PKCS12 KeyStores. Ignoring user-specified -keypass
What is your first and last name?
  [Unknown]:  Kyle Dale
What is the name of your organizational unit?
  [Unknown]:  CS-305-SoftwareSecurity22EW3
What is the name of your organization?
  [Unknown]:  SNHU
What is the name of your City or Locality?
  [Unknown]:  Worcester
What is the name of your State or Province?
  [Unknown]:  MA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Kyle Dale, OU=CS-305-SoftwareSecurity22EW3, O=SNHU, L=Worcester, ST=MA, C=US correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 360 days
        for: CN=Kyle Dale, OU=CS-305-SoftwareSecurity22EW3, O=SNHU, L=Worcester, ST=MA, C=US

C:\Users\hocke>
```



```
C:\Users\hocke>"C:\Program Files\Java\jdk-18.0.1.1\bin\keytool.exe" -printcert -file server.cer
Owner: CN=Kyle Dale, OU=CS-305-SoftwareSecurity22EW3, O=SNHU, L=Worcester, ST=MA, C=US
Issuer: CN=Kyle Dale, OU=CS-305-SoftwareSecurity22EW3, O=SNHU, L=Worcester, ST=MA, C=US
Serial number: c6936fa37487df
Valid from: Fri Feb 17 16:38:33 EST 2023 until: Mon Feb 12 16:38:33 EST 2024
Certificate fingerprints:
         SHA1: FC:57:FF:88:51:FC:44:73:19:FE:5C:78:DC:D3:22:D7:99:CD:68:06
         SHA256: E8:8C:22:95:11:8E:3F:F1:25:F5:11:BC:69:AB:95:29:37:25:39:75:74:55:F5:42:AE:C7:7A:AD:8D:50:04:35
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 1B 44 0B F4 DC D7 D0 CD   5F EC 23 4B B5 CA AB DC  .D......_.#K....
0010: 41 44 4C E7                                        ADL.
]
]
```

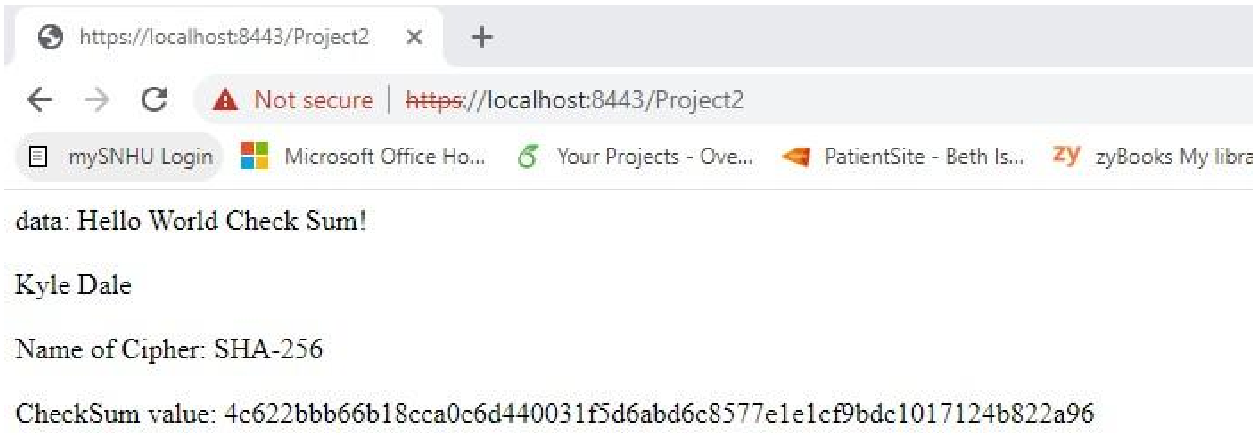


Certificate

```
Field                       Value
Version                     V3
Serial number               00c6936fa37487df
Signature algorithm         sha256RSA
Signature hash algorithm    sha256
Issuer                      Kyle Dale, CS-305-SoftwareSe...
Valid from                  Friday, February 17, 2023 4:3...
Valid to                    Monday, February 12, 2024 4...
Subject                     Kyle Dale, CS-305-SoftwareSe
```

3. Deploy Cipher

Inserted below is a screenshot of the checksum verification:


https://localhost:8443/Project2

Not secure | ~~https~~://localhost:8443/Project2

mySNHU Login | Microsoft Office Ho... | Your Projects - Ove... | PatientSite - Beth Is... | zyBooks My libra

data: Hello World Check Sum!

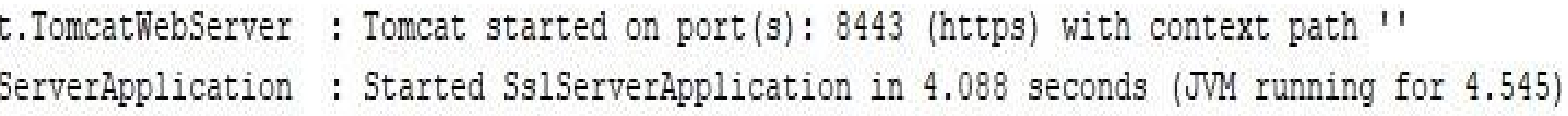
Kyle Dale

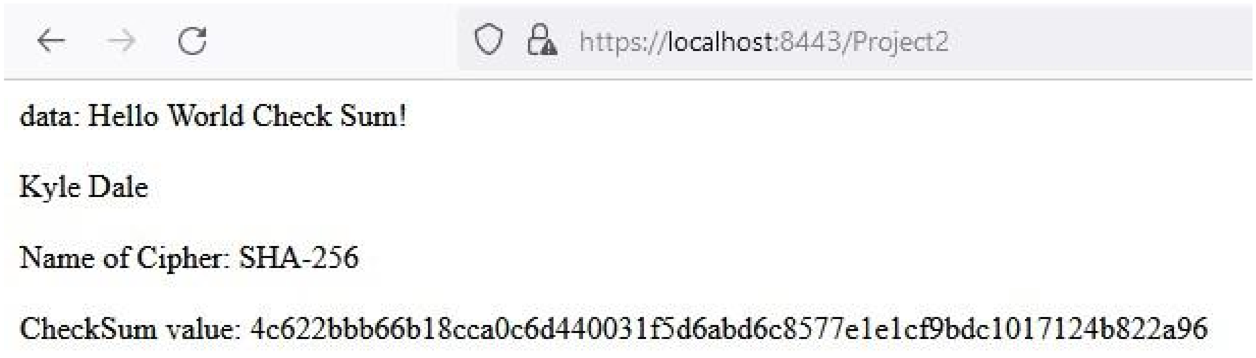
Name of Cipher: SHA-256

CheckSum value: 4c622bbb66b18cca0c6d440031f5d6abd6c8577e1e1cf9bdc1017124b822a96

4. Secure Communications

Inserted below are screenshots of the web browser that shows a secure webpage:


```
t.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
ServerApplication  : Started SslServerApplication in 4.088 seconds (JVM running for 4.545)
```


https://localhost:8443/Project2

data: Hello World Check Sum!

Kyle Dale

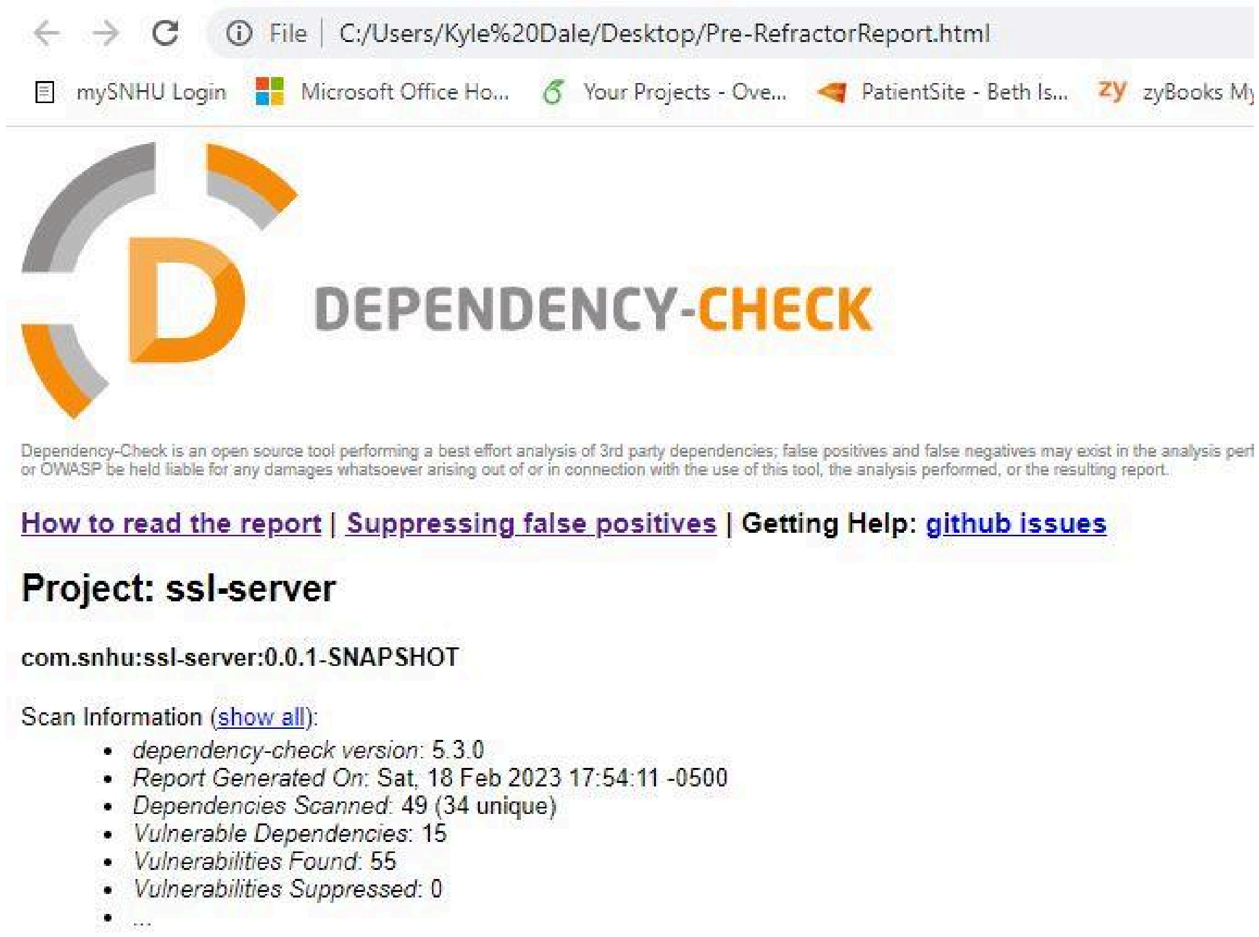
Name of Cipher: SHA-256

CheckSum value: 4c622bbb66b18cca0c6d440031f5d6abd6c8577e1e1cf9bdc1017124b822a96

The images above show I was able to properly have the servlet itself establish connection with port 8443 utilizing a https connection, after much research and error, I was finally able to figure out how to get the browser to validate the self signed certificate. I attempted a multitude of solutions, ranging from importing the server.cer to the trusted root certificates, restarting multiple times to ensure there was no coding error, updating and downloading new Eclipse packages to support development, attempting to upload the certificate to a CA root certificate and run off that, attempting to change the run configuration from ssl server app to tomcat server, attempting to host a separate tomcat server and establish https, configuring chrome and firefox to accept local certificates, and more. Despite this, I was finally able to figure out how to properly implement the CER file locally and have the browser accept the certificate as a valid one. I accomplished this by utilizing ChatGPT to assist me in diagnosing the problem (ChatGPT, 2023).

5. Secondary Testing

Inserted below are screenshots of the refactored code executed without errors and the dependency-check reports:



File | C:/Users/Kyle%20Dale/Desktop/Pre-RefactorReport.html

mySNHU Login | Microsoft Office Ho... | Your Projects - Ove... | PatientSite - Beth Is... | zyBooks My

DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis perf
or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting report.

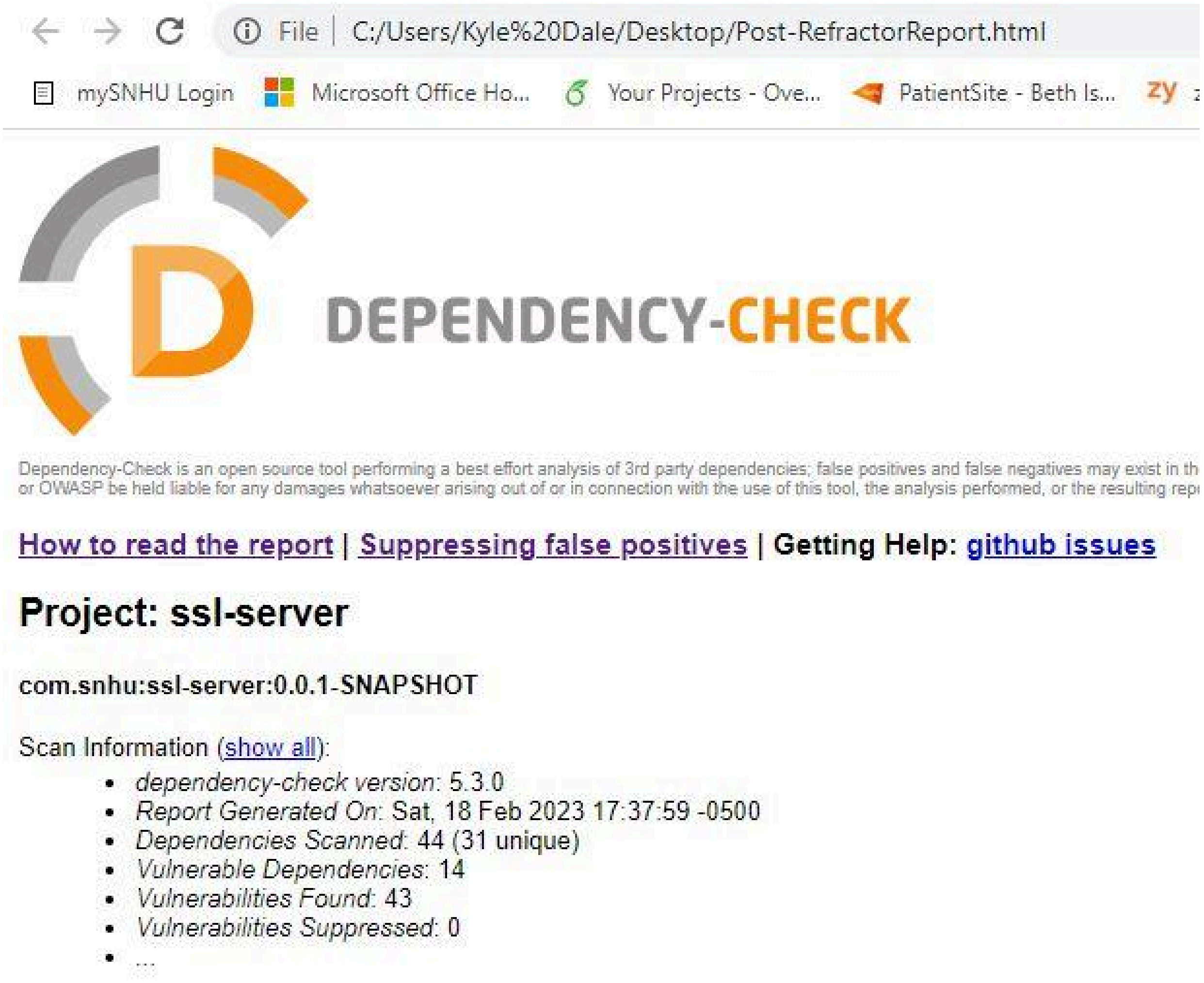
How to read the report | Suppressing false positives | Getting Help: github issues

Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Sat, 18 Feb 2023 17:54:11 -0500
- *Dependencies Scanned*: 49 (34 unique)
- *Vulnerable Dependencies*: 15
- *Vulnerabilities Found*: 55
- *Vulnerabilities Suppressed*: 0
- ...



Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in th
or OWASP be held liable for any damages whatsoever arising out of or in connection with the use of this tool, the analysis performed, or the resulting rep

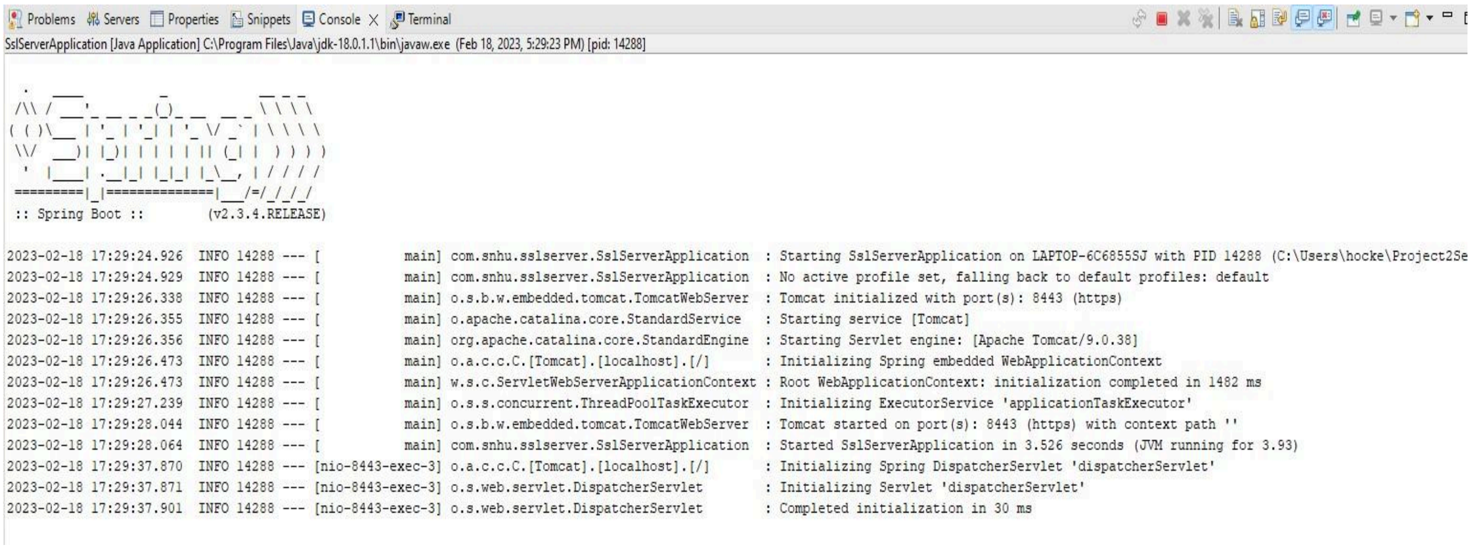
How to read the report | Suppressing false positives | Getting Help: github issues

Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Sat, 18 Feb 2023 17:37:59 -0500
- *Dependencies Scanned*: 44 (31 unique)
- *Vulnerable Dependencies*: 14
- *Vulnerabilities Found*: 43
- *Vulnerabilities Suppressed*: 0
- ...



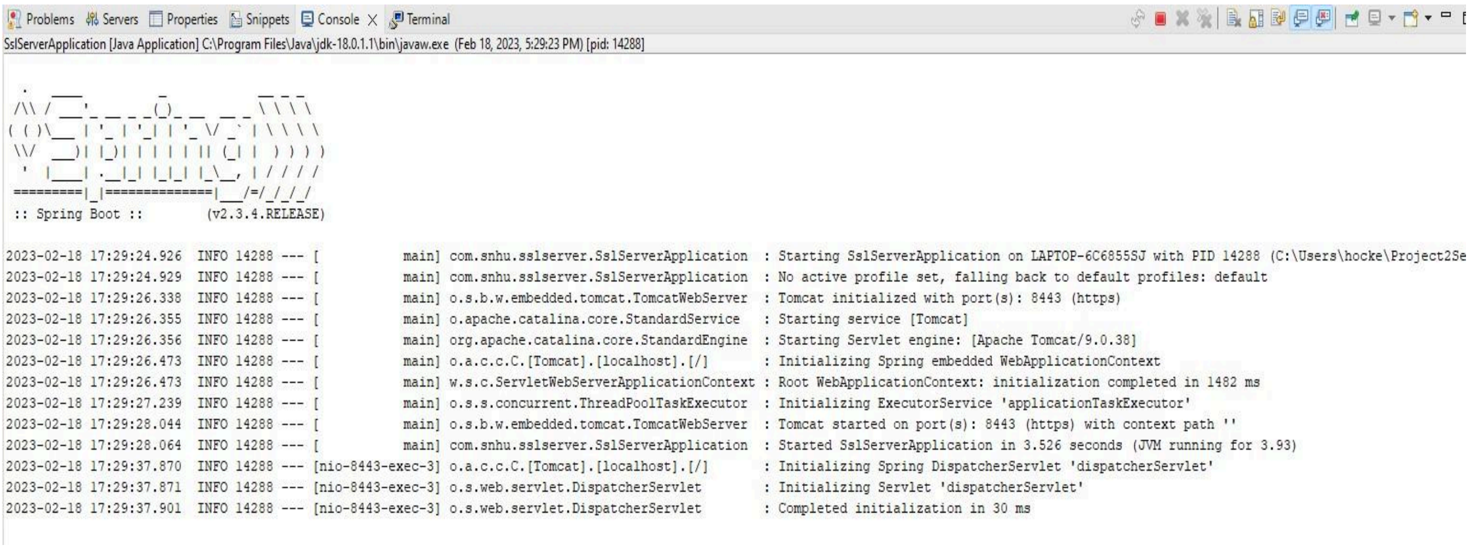
SslServerApplication [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (Feb 18, 2023, 5:29:23 PM) [pid: 14288]

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.4.RELEASE)

2023-02-18 17:29:24.926  INFO 14288 --- [           main] com.snhu.sslserver.SslServerApplication  : Starting SslServerApplication on LAPTOP-6C6855SJ with PID 14288 (C:\Users\hocke\Project2Se
2023-02-18 17:29:24.929  INFO 14288 --- [           main] com.snhu.sslserver.SslServerApplication  : No active profile set, falling back to default profiles: default
2023-02-18 17:29:26.338  INFO 14288 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8443 (https)
2023-02-18 17:29:26.355  INFO 14288 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-02-18 17:29:26.356  INFO 14288 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.38]
2023-02-18 17:29:26.473  INFO 14288 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-02-18 17:29:26.473  INFO 14288 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1482 ms
2023-02-18 17:29:27.239  INFO 14288 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2023-02-18 17:29:28.044  INFO 14288 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
2023-02-18 17:29:28.064  INFO 14288 --- [           main] com.snhu.sslserver.SslServerApplication  : Started SslServerApplication in 3.526 seconds (JVM running for 3.93)
2023-02-18 17:29:37.870  INFO 14288 --- [nio-8443-exec-3] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-02-18 17:29:37.871  INFO 14288 --- [nio-8443-exec-3] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2023-02-18 17:29:37.901  INFO 14288 --- [nio-8443-exec-3] o.s.web.servlet.DispatcherServlet        : Completed initialization in 30 ms
```

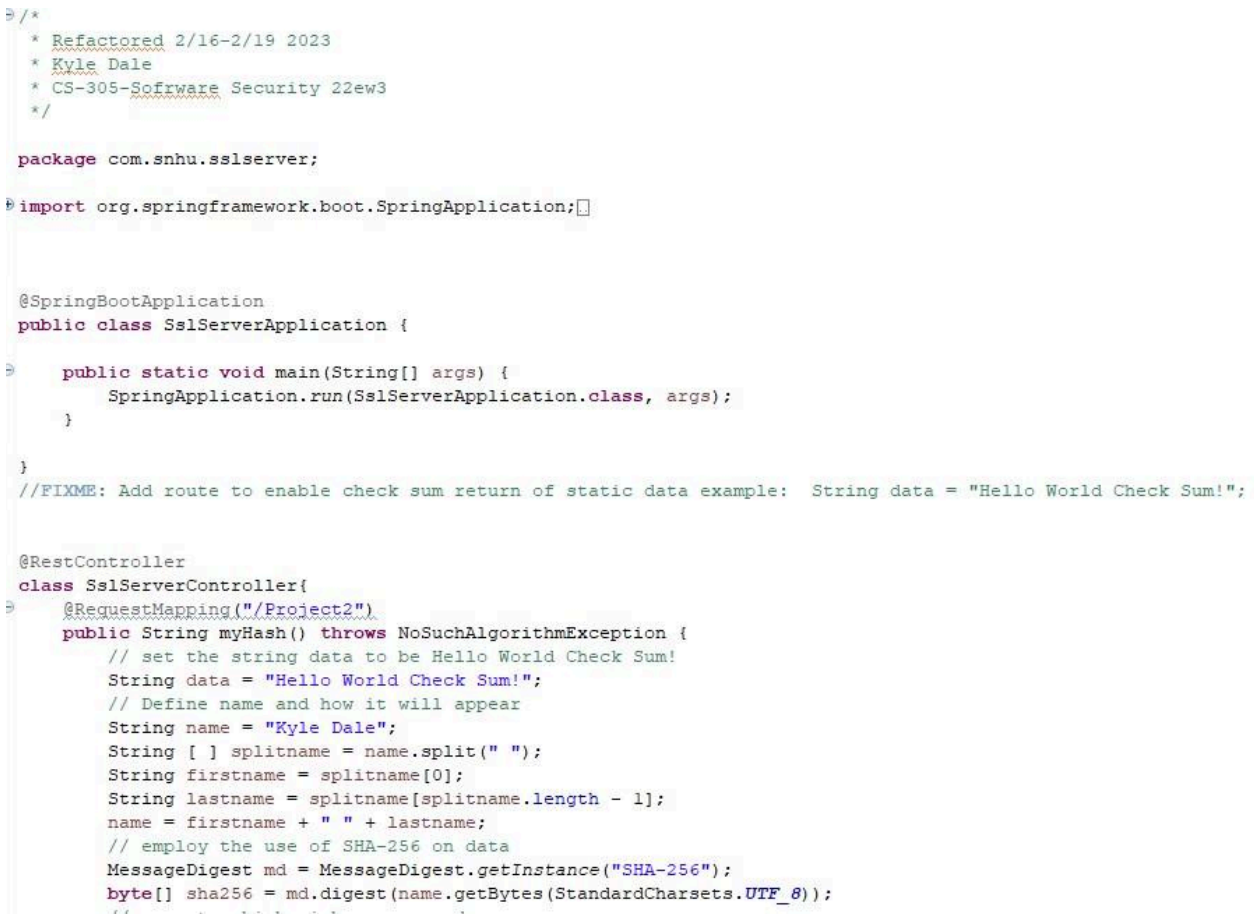
The screenshots included above demonstrate a reduction in the total number of vulnerabilities after my refactoring of the code, as well as the program's execution without error. Please note that the post-refactor report was completed first, as I had forgotten to run one on the untampered code base. I redownloaded a fresh code base for the project and ran a report on it after the fact, and that is why it was created later than the post-refactor report.

6. Functional Testing

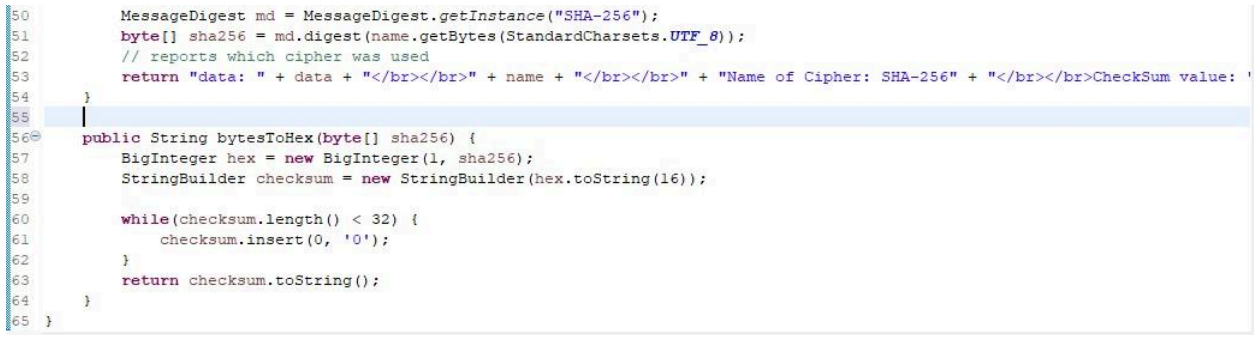
Inserted below is a screenshot of the refactored code executed without errors:



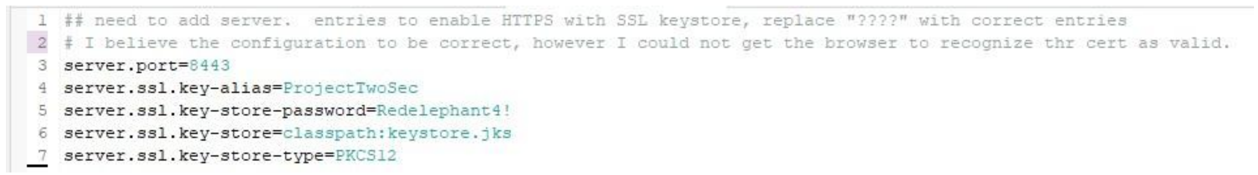
```
SslServerApplication [Java Application] C:\Program Files\Java\jdk-18.0.1.1\bin\javaw.exe (Feb 18, 2023, 5:29:23 PM) [pid: 14288]

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.4.RELEASE)

2023-02-18 17:29:24.926  INFO 14288 --- [           main] com.snhu.sslserver.SslServerApplication  : Starting SslServerApplication on LAPTOP-6C6855SJ with PID 14288 (C:\Users\hocke\Project2Se
2023-02-18 17:29:24.929  INFO 14288 --- [           main] com.snhu.sslserver.SslServerApplication  : No active profile set, falling back to default profiles: default
2023-02-18 17:29:26.338  INFO 14288 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8443 (https)
2023-02-18 17:29:26.355  INFO 14288 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2023-02-18 17:29:26.356  INFO 14288 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.38]
2023-02-18 17:29:26.473  INFO 14288 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2023-02-18 17:29:26.473  INFO 14288 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1482 ms
2023-02-18 17:29:27.239  INFO 14288 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2023-02-18 17:29:28.044  INFO 14288 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
2023-02-18 17:29:28.064  INFO 14288 --- [           main] com.snhu.sslserver.SslServerApplication  : Started SslServerApplication in 3.526 seconds (JVM running for 3.93)
2023-02-18 17:29:37.870  INFO 14288 --- [nio-8443-exec-3] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-02-18 17:29:37.871  INFO 14288 --- [nio-8443-exec-3] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2023-02-18 17:29:37.901  INFO 14288 --- [nio-8443-exec-3] o.s.web.servlet.DispatcherServlet        : Completed initialization in 30 ms
```



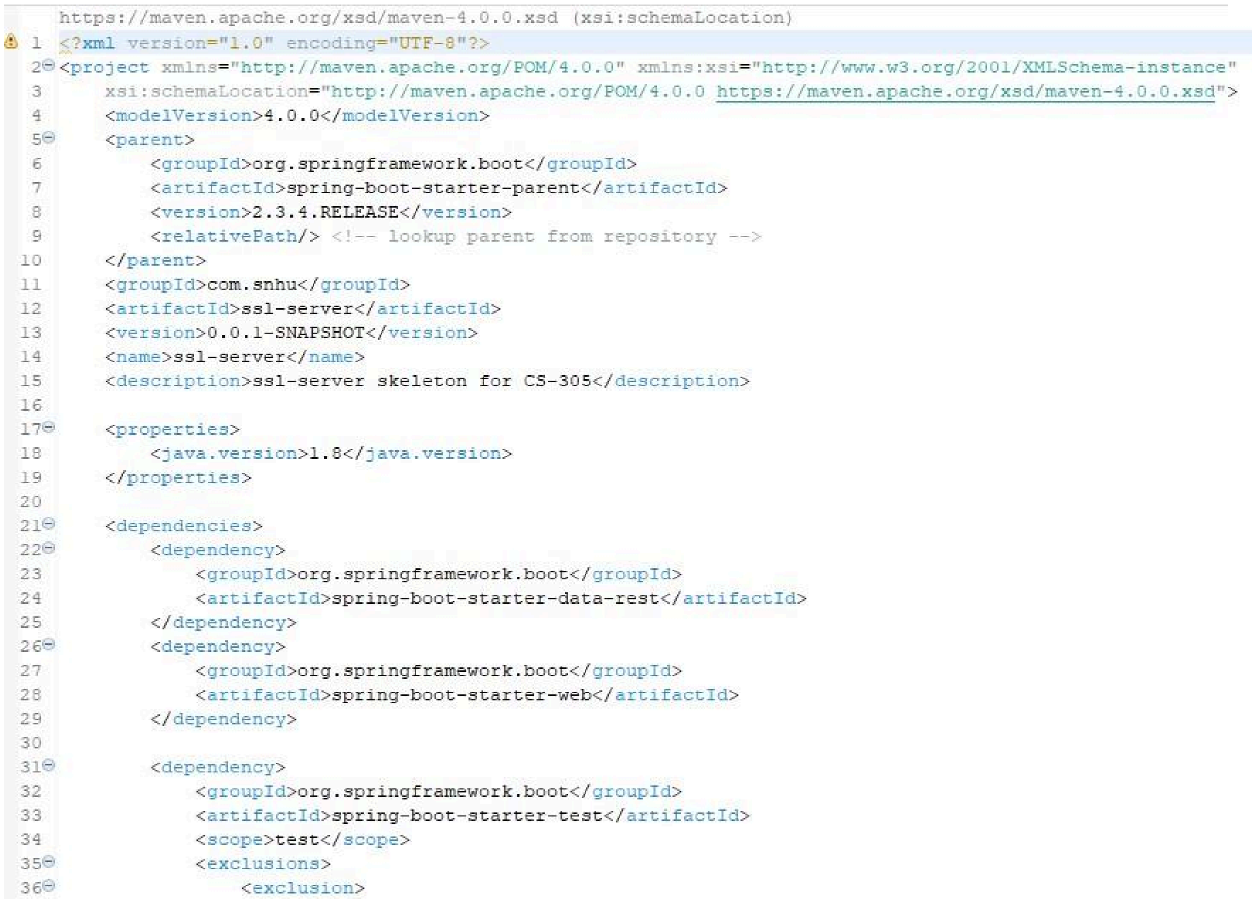
```
/*
 * Refactored 2/16-2/19 2023
 * Kyle Dale
 * CS-305-Sofrware Security 22ew3
 */

package com.snhu.sslserver;

import org.springframework.boot.SpringApplication;


@SpringBootApplication
public class SslServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(SslServerApplication.class, args);
    }

}
//FIXME: Add route to enable check sum return of static data example:  String data = "Hello World Check Sum!";


@RestController
class SslServerController{
    @RequestMapping("/Project2")
    public String myHash() throws NoSuchAlgorithmException {
        // set the string data to be Hello World Check Sum!
        String data = "Hello World Check Sum!";
        // Define name and how it will appear
        String name = "Kyle Dale";
        String [ ] splitname = name.split(" ");
        String firstname = splitname[0];
        String lastname = splitname[splitname.length - 1];
        name = firstname + " " + lastname;
        // employ the use of SHA-256 on data
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] sha256 = md.digest(name.getBytes(StandardCharsets.UTF_8));
```



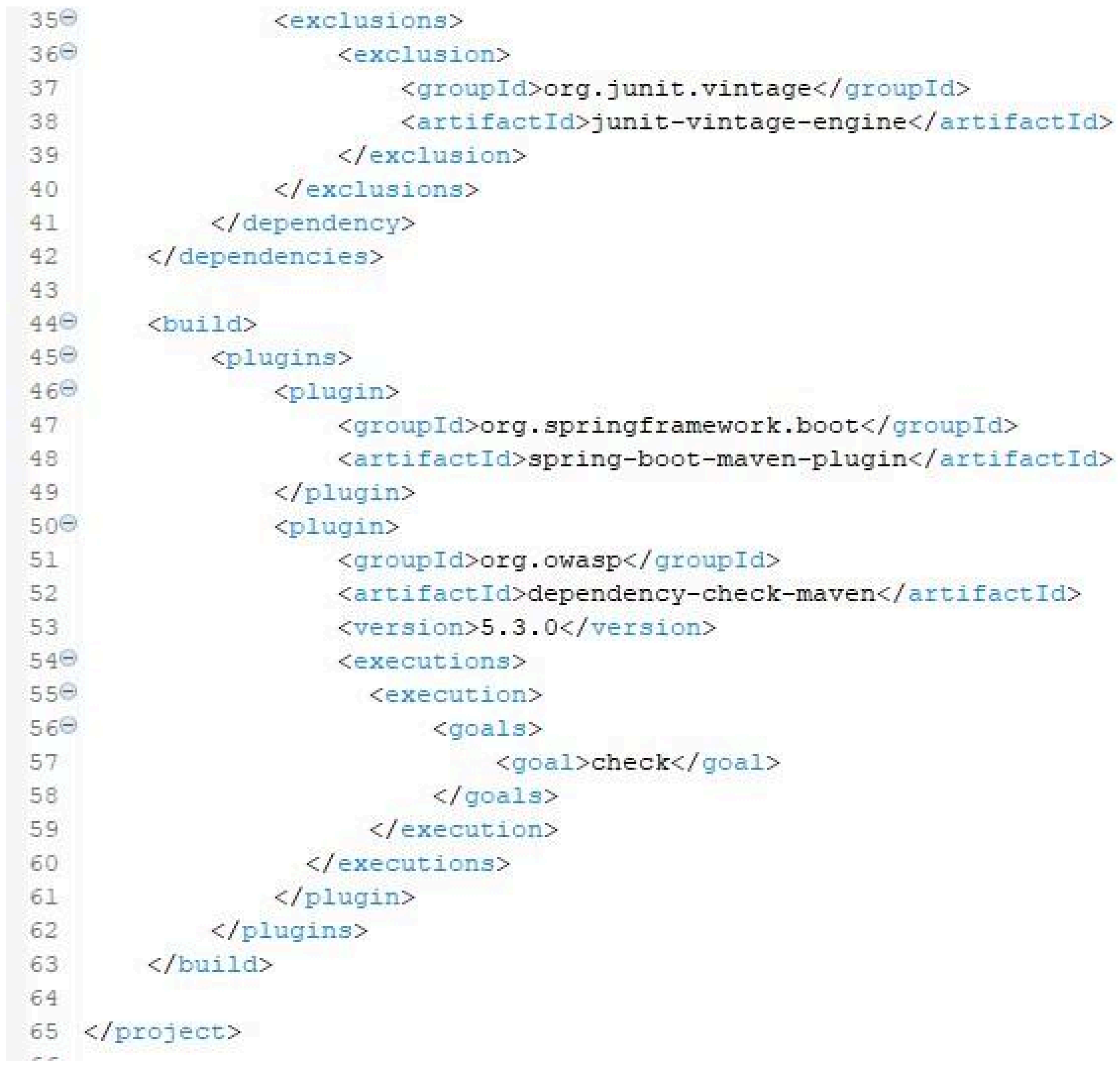
```
        MessageDigest md = MessageDigest.getInstance("SHA-256");
        byte[] sha256 = md.digest(name.getBytes(StandardCharsets.UTF_8));
        // reports which cipher was used
        return "data: " + data + "</br></br>" + name + "</br></br>" + "Name of Cipher: SHA-256" + "</br></br>CheckSum value: '
    }

    public String bytesToHex(byte[] sha256) {
        BigInteger hex = new BigInteger(1, sha256);
        StringBuilder checksum = new StringBuilder(hex.toString(16));

        while(checksum.length() < 32) {
            checksum.insert(0, '0');
        }
        return checksum.toString();
    }
}
```



```
## need to add server.  entries to enable HTTPS with SSL keystore, replace "????" with correct entries
# I believe the configuration to be correct, however I could not get the browser to recognize thr cert as valid.
server.port=8443
server.ssl.key-alias=ProjectTwoSec
server.ssl.key-store-password=Redelephant4!
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-type=PKCS12
```



```
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.4.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.snhu</groupId>
    <artifactId>ssl-server</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>ssl-server</name>
    <description>ssl-server skeleton for CS-305</description>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-rest</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
```



```
                <exclusions>
                    <exclusion>
                        <groupId>org.junit.vintage</groupId>
                        <artifactId>junit-vintage-engine</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>
        </dependencies>

        <build>
            <plugins>
                <plugin>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-plugin</artifactId>
                </plugin>
                <plugin>
                    <groupId>org.owasp</groupId>
                    <artifactId>dependency-check-maven</artifactId>
                    <version>5.3.0</version>
                    <executions>
                      <execution>
                          <goals>
                              <goal>check</goal>
                          </goals>
                      </execution>
                    </executions>
                </plugin>
            </plugins>
        </build>

    </project>
```

The images above demonstrate the refactored code, as well as it running successfully (I had yet to comment on the entire code upon capture of these images, so the submitted code differs slightly from these images, as it contains additional comments). Although the program does function, upon manual review as well as analysis of the dependency reports, there are still a few security vulnerabilities that would need to be addressed prior to launching the program. The first of these is that much of the additional software implemented in the pom.xl file is out of date, and should be updated before implementation, to ensure proper function as well as ensure best security and code practices (Detlefsen, 2014). Secondly, the use of plaintext in the application.properties file, to implement server functionality, could leave Artemis Financial vulnerable. Ideally the keystore password, displayed on line 5 in the application.properties file, would also be run through a cipher algorithm, so if any malicious users got access to the codebase, the data would still possess a layer of security (Detlefsen, 2014). As Artemis Financial is a banking institution, it will be imperative for them to utilize as many layers of security as is possible. As such, encrypting the plain text stored within the code is essential.

7. Summary

The areas of security addressed in the refactoring of this code include:

- APIs:
    - The API section of security was addressed by the implementation of https in server client communication. This will help ensure that the communications between Artemis Financials servers and their clients are secured. Were I actually to be working with a company such as Artemis Financial, I would never recommend they utilize a self-signed certificate as opposed to a CA.
- Input Validation:
    - The input validation section of security was addressed with the implementation of the checksum. This allows users to be confident that the data or site they are accessing was unaltered by a malicious attacker or computational error.
- Code Quality:
    - The code quality section of security was addressed with the manual review of the code, as well as utilizing best practices and clear commentation. This highlighted multiple areas of vulnerability still present in the program that need to be

    addressed prior to implementation, even though the program is currently functional.

- Cryptography:
    - The cryptography section of security was addressed with the https connection utilizing tls to ensure safe transition of data, as well as the implementation of the SHA-256 hash function for creating a checksum, to ensure the data received was not tampered with. Additionally, cryptography is accomplished with the utilization of AES-256 for Artemis Financials data at rest. Both of these methods will serve to ensure Artemis Financial is protected from malicious breach and loss of data.

My process for adding the additional layers of security took a very incremental approach for this project. As the project was rather large in scope, I attempted to lay down the 'bones' of the program before implementing the security measures. This began with ensuring that I could get the servlet to function properly with basic http connection. Once this was accomplished, I included the SHA-256 checksum to ensure validation. Once this portion of the assignment was done, I spent a large amount of time attempting to get my browser to validate the certificate I had generated. This was eventually successful with the assistance of ChatGPT in debugging my work and ensuring the certificate was generated correctly. Finally, I ran the final dependency report and compared the pre and post refactored reports to ensure my modifications had not caused additional error. Once this was done, I completed the manual code review to highlight additional vulnerabilities in the program that, were it to actually be sent to the real company Artemis Financial, I would address forthwith.

8. Industry Standard Best Practices

When developing or refactoring code, it is essential to utilize industry standard best practices at all times. This will cut down on development time, by avoiding the inclusion of additional errors when refactoring code. Additionally, by not introducing new vulnerabilities with refactoring, a developer can steadily reduce the total number of vulnerabilities present in code until none exist. Utilizing best practices also provides a standard form to code, that will allow any developer on a company's team to pick up where another left off, further increasing efficiency and saving time. Another benefit of industry best practices, is through their adherence, one will be assured to abide by industry regulations. A few of these are relevant to Artemis Financial, like the General Data Protection Regulation (GDPR) or the Payment Card Industry Data Security Standard, also known as PCI DSS (ChatGPT, 2023). Best practices are not only beneficial to the program itself, but by building a reputation for developing solid, well secured products, a company can establish a relationship of trust and reliance with their users, boosting the health of the company on many levels. Industry standard best practices exist for a reason, and adhering to them will allow a company to consistently produce efficient and secure programs that their customers can rely on.

References

Arnaud. (2023, January 20). *Symmetric vs asymmetric encryption: What's the difference?*. Mailfence Blog. https://blog.mailfence.com/symmetric-vs-asymmetric-encryption/#:~:text=The%20main%20difference%20is%20that,private%20key%20to%20decrypt%20information.

Crane, C. (2021, January 25). *What is a hash function in cryptography? A beginner's guide*. Hashed Out by The SSL Store. https://www.thesslstore.com/blog/what-is-a-hash-function-in-cryptography-a-beginners-guide/#:~:text=Hash%20functions%20are%20a%20way,a%20means%20of%20identity%20verification.

De Mol, L. (2018, September 24). *Turing machines*. Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/turing-machine/.

Loshin, P. (2021, October 26). *What is an encryption key? - definition from searchsecurity*. Tech Target. https://www.techtarget.com/searchsecurity/definition/key#:~:text=In%20cryptography%2C%20an%20encryption%20key,text%20in%20a%20given%20message.

Manico, J., & Detlefsen, A. (2015). *Iron-clad java: Building secure web applications*. McGraw-Hill Education.

OpenAI. (2023, February 19). Add Exception for Firefox [Online conversation].

https://chat.openai.com/chat/15460a52-548e-4ebf-985f-643e8370f699.

OpenAI. (2023, February 19). Conversation with a User on Security Best Practices https://chat.openai.com/chat

OpenAI. (2023, February 19). File Text Inclusion [Online conversation]. https://chat.openai.com/chat/0e96edd9-5d6e-4745-83dc-bd4907ed1fde

Staff, W. (2020, June 5). *The bit-security of cryptographic primitives*. AWS Wickr. https://wickr.com/the-bit-security-of-cryptographic-primitives-2/.

Team, T. (2022, January 31). *The history of encryption: The roots of modern-day cyber-security*. Tresorit Blog. https://tresorit.com/blog/the-history-of-encryption-the-roots-of-modern-day-cyber-security.