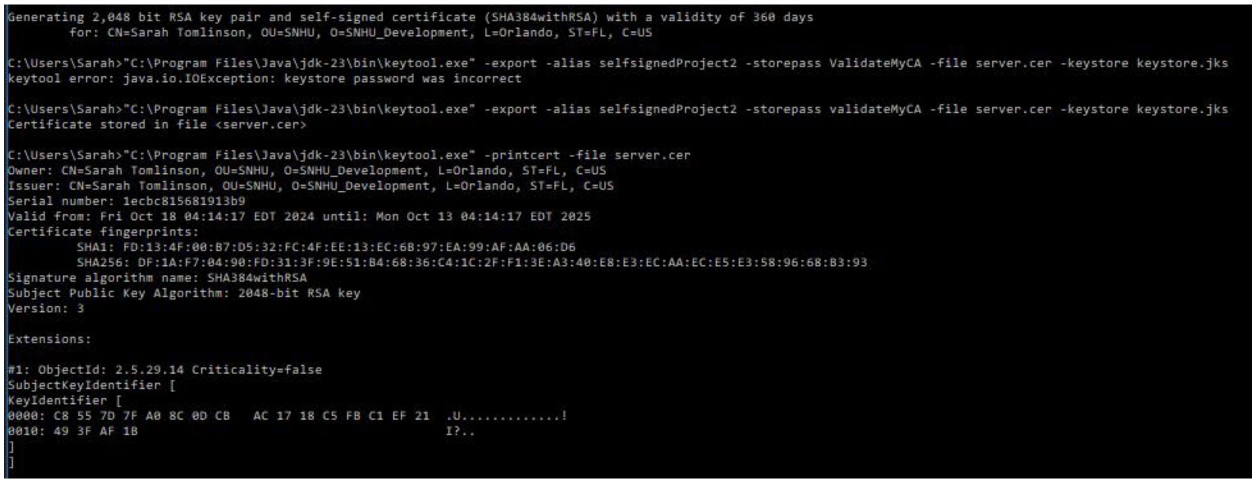


```
Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 360 days
        for: CN=Sarah Tomlinson, OU=SNHU, O=SNHU_Development, L=Orlando, ST=FL, C=US

C:\Users\Sarah>"C:\Program Files\Java\jdk-23\bin\keytool.exe" -export -alias selfsignedProject2 -storepass ValidateMyCA -file server.cer -keystore keystore.jks
keytool error: java.io.IOException: keystore password was incorrect

C:\Users\Sarah>"C:\Program Files\Java\jdk-23\bin\keytool.exe" -export -alias selfsignedProject2 -storepass validateMyCA -file server.cer -keystore keystore.jks
Certificate stored in file <server.cer>

C:\Users\Sarah>"C:\Program Files\Java\jdk-23\bin\keytool.exe" -printcert -file server.cer
Owner: CN=Sarah Tomlinson, OU=SNHU, O=SNHU_Development, L=Orlando, ST=FL, C=US
Issuer: CN=Sarah Tomlinson, OU=SNHU, O=SNHU_Development, L=Orlando, ST=FL, C=US
Serial number: 1ecbc815681913b9
Valid from: Fri Oct 18 04:14:17 EDT 2024 until: Mon Oct 13 04:14:17 EDT 2025
Certificate fingerprints:
         SHA1: FD:13:4F:00:B7:D5:32:FC:4F:EE:13:EC:6B:97:EA:99:AF:AA:06:D6
         SHA256: DF:1A:F7:04:90:FD:31:3F:9E:51:B4:68:36:C4:1C:2F:F1:3E:A3:40:E8:E3:EC:AA:EC:E5:E3:58:96:68:B3:93
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C8 55 7D 7F A0 8C 0D CB   AC 17 18 C5 FB C1 EF 21  .U.............!
0010: 49 3F AF 1B                                        I?..
]
]
```

GLOBAL RAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	10/18/2024	Sarah Tomlinson	Updated Document

Client

ARTEMIS FINANCIAL

Instructions

Submit these completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

- Respond to the steps outlined below and include your findings.
- Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
- Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

Developer
Sarah Tomlinson

1. Algorithm Cipher

Artemis Financial is a consulting company that helps its customers develop individualized financial plans around savings, retirement, investments, and insurance. They hope to modernize their operations with custom software and use the most current and effective software security available. They have a public web interface and are looking to Global Rain for help protecting client data and financial information. The company wants to add a file verification step to the web application to ensure secure communications. This data verification will need to take place in the form of a checksum, and we will need to choose a suitable encryption algorithm that will protect client data, avoid collisions (the likelihood of creating two different messages such that the starting message and ending message are the same). For this reason, I am recommending the SHA-256 algorithm cipher due to it being a cryptographic hash function with a low chance of collisions.

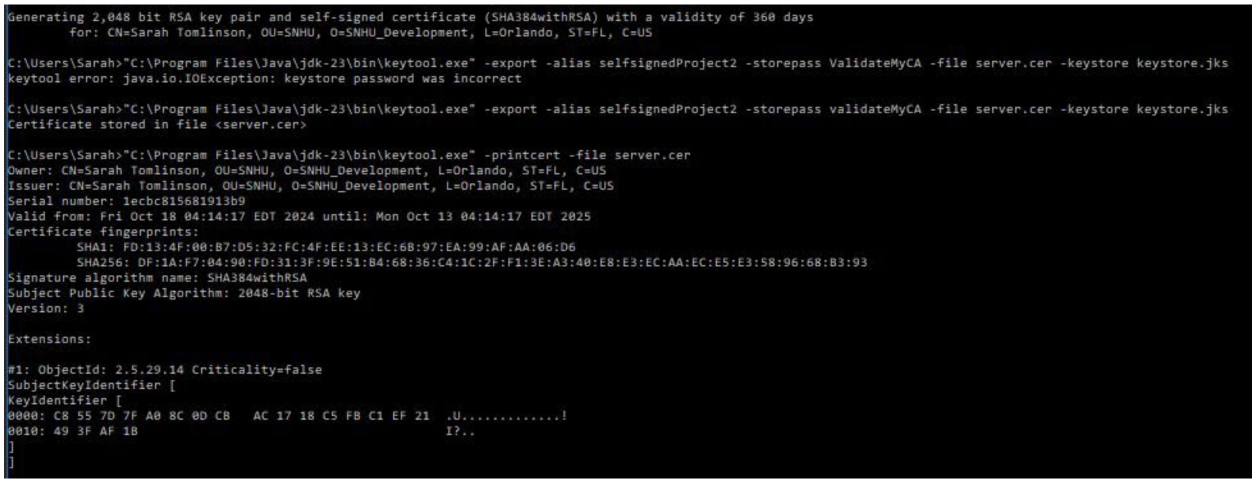
It is important to understand why SHA-256 is the recommended option for Artemis Financial. A low chance of collision is important because the easier it is for an attacker to decrypt transferred data, the more at risk your client's and their financial data is. In general, the longer a hash message is, the harder it is for an attacker to find a collision between the two versions of the message. In addition, part of the security of this encryption algorithm comes from the 256-bit level. The higher the bit level, the longer the hash will be.

SHA-256 was created by the NSA in 2001, and still functions securely. SHA-256 uses a block cipher called SHACAL-2 to encrypt data. It begins by breaking the message down into 0's and 1's, applying compression to the first block to create a chaining variable. This variable is then compressed again, along with the next block. This continues until the entire final message block is reached and appended with a padding block. This final block and the one before it is fed into one more compression function, and the output is the hash is returned. This final hash creates an output that contains lowercase letters or numbers 0 through 9. This means that *each* portion of the ciphertext will have 36 possible characters, and any attacker attempting to brute force the algorithm would have to make 2^{256} attempts to uncover the original message (N-able, 2019). In addition, any minor change to the original data alters the hash value to the extent that it isn't evident the new hash value was derived from similar data, further bolstering its security.

Historically, cryptographic keys were developed by the Spartans to send secret battle messages using a piece of leather wrapped around a rod. This method was only the beginning as cryptography became more and more advanced through the years. The same principles apply to encrypt and decode messages, even transferring data across the internet. Modern day computer-based encryption started in the 1970s with IBM and the Data Encryption Standard (DES). The United States government used to utilize the DES algorithm with a 56-bit cryptographic key, but now uses AES. As mentioned previously, the time required to crack a bit level is related to the number of bits used, so a key with a higher bit level (such as SHA-256) is more secure. At its core, cryptography still uses the same principles as the Spartans, but it has been improved and adapted across centuries to provide more security in encryption.

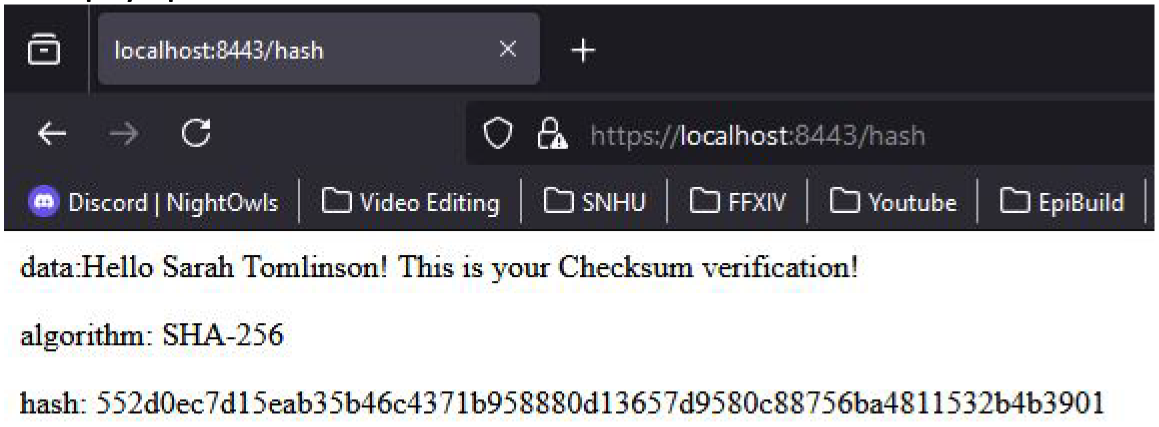
2. Certificate Generation

Screenshot of the CER File.



```
Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 360 days
        for: CN=Sarah Tomlinson, OU=SNHU, O=SNHU_Development, L=Orlando, ST=FL, C=US

C:\Users\Sarah>"C:\Program Files\Java\jdk-23\bin\keytool.exe" -export -alias selfsignedProject2 -storepass ValidateMyCA -file server.cer -keystore keystore.jks
keytool error: java.io.IOException: keystore password was incorrect

C:\Users\Sarah>"C:\Program Files\Java\jdk-23\bin\keytool.exe" -export -alias selfsignedProject2 -storepass validateMyCA -file server.cer -keystore keystore.jks
Certificate stored in file <server.cer>

C:\Users\Sarah>"C:\Program Files\Java\jdk-23\bin\keytool.exe" -printcert -file server.cer
Owner: CN=Sarah Tomlinson, OU=SNHU, O=SNHU_Development, L=Orlando, ST=FL, C=US
Issuer: CN=Sarah Tomlinson, OU=SNHU, O=SNHU_Development, L=Orlando, ST=FL, C=US
Serial number: 1ecbc815681913b9
Valid from: Fri Oct 18 04:14:17 EDT 2024 until: Mon Oct 13 04:14:17 EDT 2025
Certificate fingerprints:
         SHA1: FD:13:4F:00:B7:D5:32:FC:4F:EE:13:EC:6B:97:EA:99:AF:AA:06:D6
         SHA256: DF:1A:F7:04:90:FD:31:3F:9E:51:B4:68:36:C4:1C:2F:F1:3E:A3:40:E8:E3:EC:AA:EC:E5:E3:58:96:68:B3:93
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: C8 55 7D 7F A0 8C 0D CB   AC 17 18 C5 FB C1 EF 21  .U.............!
0010: 49 3F AF 1B                                        I?..
]
]
```

3. Deploy Cipher

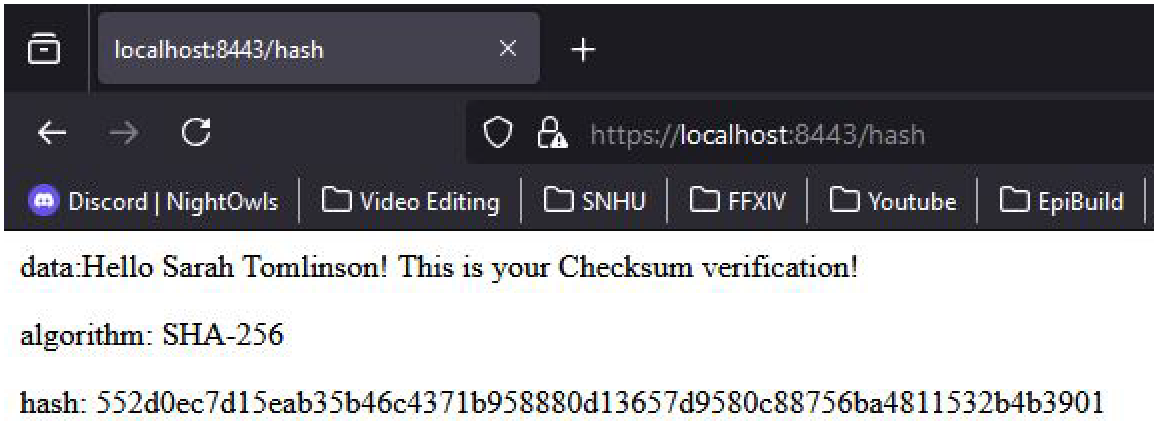


data:Hello Sarah Tomlinson! This is your Checksum verification!

algorithm: SHA-256

hash: 552d0ec7d15eab35b46c4371b958880d13657d9580c88756ba4811532b4b3901

4. Secure Communications



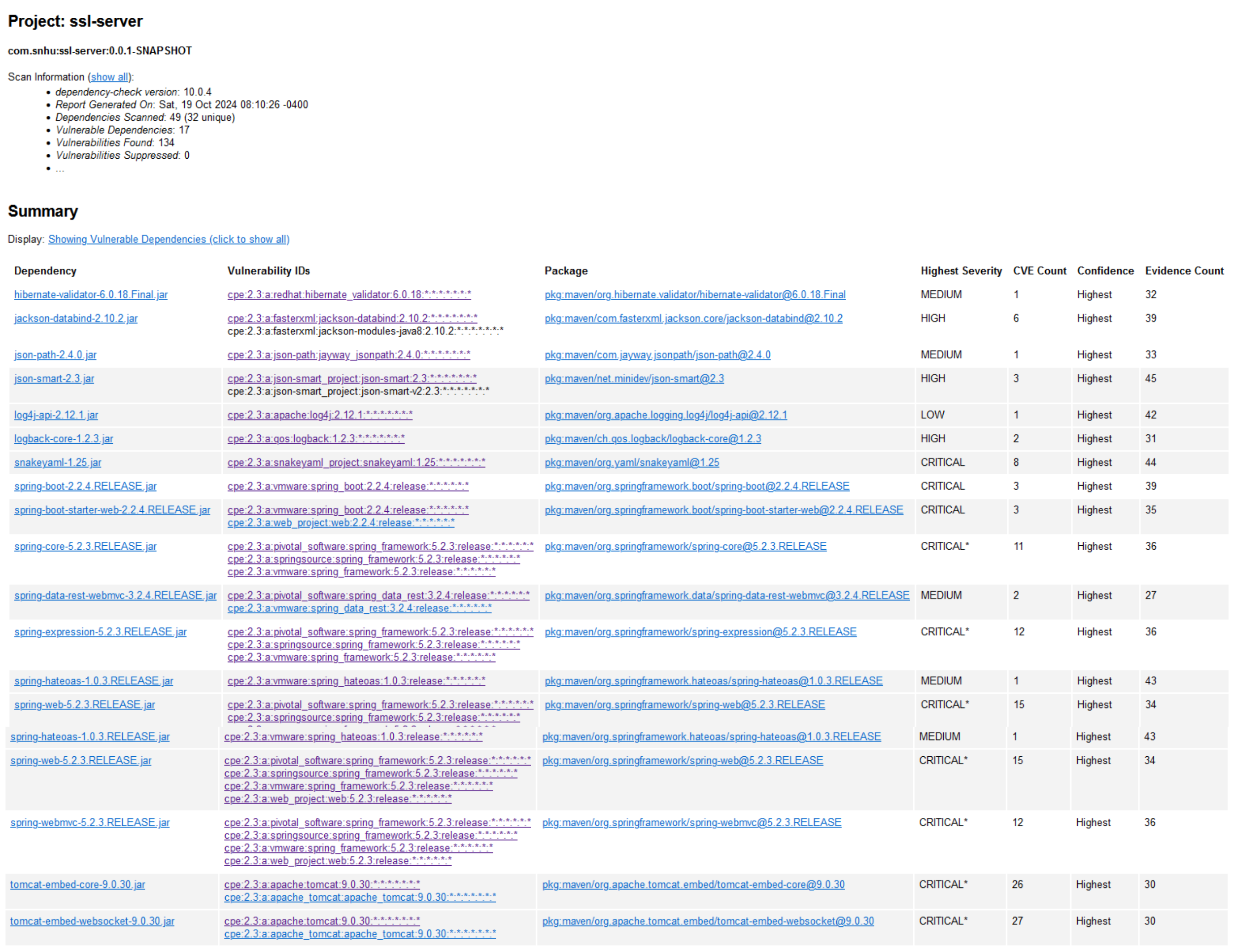
data:Hello Sarah Tomlinson! This is your Checksum verification!

algorithm: SHA-256

hash: 552d0ec7d15eab35b46c4371b958880d13657d9580c88756ba4811532b4b3901

5. Secondary Testing

All vulnerabilities from using old versions of software, none present as vulnerabilities from my refactored code.



Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 10.0.4
- *Report Generated On*: Sat, 19 Oct 2024 08:10:26 -0400
- *Dependencies Scanned*: 49 (32 unique)
- *Vulnerable Dependencies*: 17
- *Vulnerabilities Found*: 134
- *Vulnerabilities Suppressed*: 0
- ...

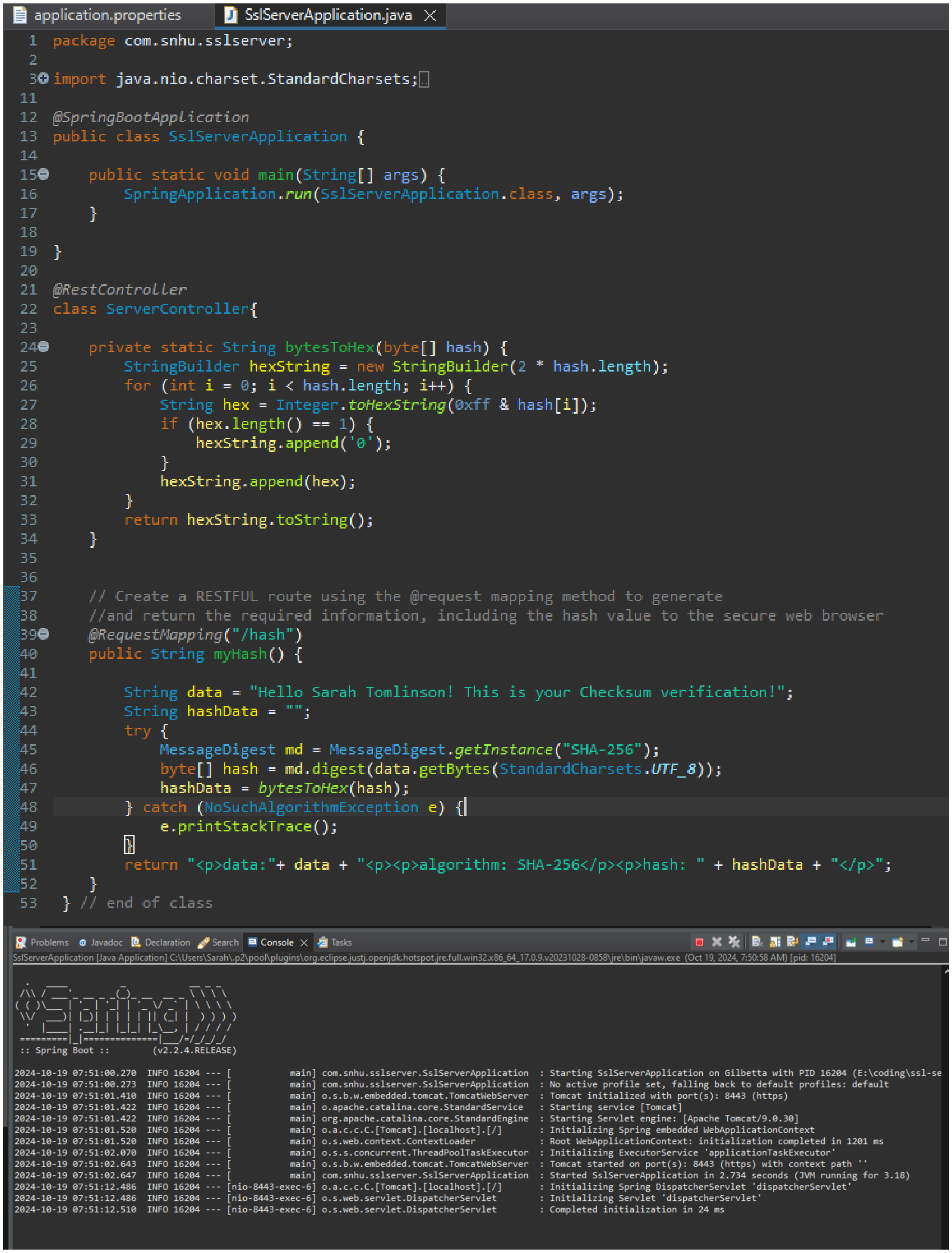
Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
hibernate-validator-6.0.18.Final.jar	cpe:2.3:a:redhat:hibernate_validator:6.0.18:*:*:*:*:*:*:*	pkg:maven/org.hibernate.validator/hibernate-validator@6.0.18.Final	MEDIUM	1	Highest	32
jackson-databind-2.10.2.jar	cpe:2.3:a:fasterxml:jackson-databind:2.10.2:*:*:*:*:*:*:* cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:*:*:*:*:*:*:*	pkg:maven/com.fasterxml.jackson.core/jackson-databind@2.10.2	HIGH	6	Highest	39
json-path-2.4.0.jar	cpe:2.3:a:json-path:jayway_jsonpath:2.4.0:*:*:*:*:*:*:*	pkg:maven/com.jayway.jsonpath/json-path@2.4.0	MEDIUM	1	Highest	33
json-smart-2.3.jar	cpe:2.3:a:json-smart_project:json-smart:2.3:*:*:*:*:*:*:* cpe:2.3:a:json-smart_project:json-smart-v2:2.3:*:*:*:*:*:*:*	pkg:maven/net.minidev/json-smart@2.3	HIGH	3	Highest	45
log4j-api-2.12.1.jar	cpe:2.3:a:apache:log4j:2.12.1:*:*:*:*:*:*:*	pkg:maven/org.apache.logging.log4j/log4j-api@2.12.1	LOW	1	Highest	42
logback-core-1.2.3.jar	cpe:2.3:a:qos:logback:1.2.3:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-core@1.2.3	HIGH	2	Highest	31
snakeyaml-1.25.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.25:*:*:*:*:*:*:*	pkg:maven/org.yaml/snakeyaml@1.25	CRITICAL	8	Highest	44
spring-boot-2.2.4.RELEASE.jar	cpe:2.3:a:vmware:spring_boot:2.2.4:release:*:*:*:*:*:*	pkg:maven/org.springframework.boot/spring-boot@2.2.4.RELEASE	CRITICAL	3	Highest	39
spring-boot-starter-web-2.2.4.RELEASE.jar	cpe:2.3:a:vmware:spring_boot:2.2.4:release:*:*:*:*:*:* cpe:2.3:a:web_project:web:2.2.4:release:*:*:*:*:*:*	pkg:maven/org.springframework.boot/spring-boot-starter-web@2.2.4.RELEASE	CRITICAL	3	Highest	35
spring-core-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.2.3:release:*:*:*:*:*:*	pkg:maven/org.springframework/spring-core@5.2.3.RELEASE	CRITICAL*	11	Highest	36
spring-data-rest-webmvc-3.2.4.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_data_rest:3.2.4:release:*:*:*:*:*:* cpe:2.3:a:vmware:spring_data_rest:3.2.4:release:*:*:*:*:*:*	pkg:maven/org.springframework.data/spring-data-rest-webmvc@3.2.4.RELEASE	MEDIUM	2	Highest	27
spring-expression-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.2.3:release:*:*:*:*:*:*	pkg:maven/org.springframework/spring-expression@5.2.3.RELEASE	CRITICAL*	12	Highest	36
spring-hateoas-1.0.3.RELEASE.jar	cpe:2.3:a:vmware:spring_hateoas:1.0.3:release:*:*:*:*:*:*	pkg:maven/org.springframework.hateoas/spring-hateoas@1.0.3.RELEASE	MEDIUM	1	Highest	43
spring-web-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.3:release:*:*:*:*:*:*	pkg:maven/org.springframework/spring-web@5.2.3.RELEASE	CRITICAL*	15	Highest	34
spring-hateoas-1.0.3.RELEASE.jar	cpe:2.3:a:vmware:spring_hateoas:1.0.3:release:*:*:*:*:*:*	pkg:maven/org.springframework.hateoas/spring-hateoas@1.0.3.RELEASE	MEDIUM	1	Highest	43
spring-web-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:web_project:web:5.2.3:release:*:*:*:*:*:*	pkg:maven/org.springframework/spring-web@5.2.3.RELEASE	CRITICAL*	15	Highest	34
spring-webmvc-5.2.3.RELEASE.jar	cpe:2.3:a:pivotal_software:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:springsource:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:vmware:spring_framework:5.2.3:release:*:*:*:*:*:* cpe:2.3:a:web_project:web:5.2.3:release:*:*:*:*:*:*	pkg:maven/org.springframework/spring-webmvc@5.2.3.RELEASE	CRITICAL*	12	Highest	36
tomcat-embed-core-9.0.30.jar	cpe:2.3:a:apache:tomcat:9.0.30:*:*:*:*:*:*:* cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.30:*:*:*:*:*:*:*	pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@9.0.30	CRITICAL*	26	Highest	30
tomcat-embed-websocket-9.0.30.jar	cpe:2.3:a:apache:tomcat:9.0.30:*:*:*:*:*:*:* cpe:2.3:a:apache_tomcat:apache_tomcat:9.0.30:*:*:*:*:*:*:*	pkg:maven/org.apache.tomcat.embed/tomcat-embed-websocket@9.0.30	CRITICAL*	27	Highest	30

6. Functional Testing

Insert a screenshot below of the refactored code executed without errors.



application.properties | SslServerApplication.java

```java
package com.snhu.sslserver;

import java.nio.charset.StandardCharsets;

@SpringBootApplication
public class SslServerApplication {

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

}

@RestController
class ServerController{

    private static String bytesToHex(byte[] hash) {
        StringBuilder hexString = new StringBuilder(2 * hash.length);
        for (int i = 0; i < hash.length; i++) {
            String hex = Integer.toHexString(0xff & hash[i]);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }


    // Create a RESTFUL route using the @request mapping method to generate
    //and return the required information, including the hash value to the secure web browser
    @RequestMapping("/hash")
    public String myHash() {

        String data = "Hello Sarah Tomlinson! This is your Checksum verification!";
        String hashData = "";
        try {
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            byte[] hash = md.digest(data.getBytes(StandardCharsets.UTF_8));
            hashData = bytesToHex(hash);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return "<p>data:"+ data + "<p><p>algorithm: SHA-256</p><p>hash: " + hashData + "</p>";
    }
 } // end of class
```

Problems | Javadoc | Declaration | Search | Console | Tasks

SslServerApplication [Java Application] C:\Users\Sarah\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.9.v20231028-0858\jre\bin\javaw.exe (Oct 19, 2024, 7:50:58 AM) [pid: 16204]

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.4.RELEASE)

2024-10-19 07:51:00.270  INFO 16204 --- [           main] com.snhu.sslserver.SslServerApplication  : Starting SslServerApplication on Gilbetta with PID 16204 (E:\coding\ssl-se
2024-10-19 07:51:00.273  INFO 16204 --- [           main] com.snhu.sslserver.SslServerApplication  : No active profile set, falling back to default profiles: default
2024-10-19 07:51:01.410  INFO 16204 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8443 (https)
2024-10-19 07:51:01.422  INFO 16204 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2024-10-19 07:51:01.422  INFO 16204 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.30]
2024-10-19 07:51:01.520  INFO 16204 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2024-10-19 07:51:01.520  INFO 16204 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1201 ms
2024-10-19 07:51:02.070  INFO 16204 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2024-10-19 07:51:02.643  INFO 16204 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
2024-10-19 07:51:02.647  INFO 16204 --- [           main] com.snhu.sslserver.SslServerApplication  : Started SslServerApplication in 2.734 seconds (JVM running for 3.18)
2024-10-19 07:51:12.486  INFO 16204 --- [nio-8443-exec-6] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2024-10-19 07:51:12.486  INFO 16204 --- [nio-8443-exec-6] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2024-10-19 07:51:12.510  INFO 16204 --- [nio-8443-exec-6] o.s.web.servlet.DispatcherServlet        : Completed initialization in 24 ms
```

7. Summary

In refactoring the code for this program, there were multiple steps taken to implement the SHA-256 encryption algorithm. To start, I added a RestController that functions as the secure controller of the program's hash. I also added a RequestMapping method to generate and return the hash value to the secure browser. I also used a self-signed certificate to enable using an HTTPS address, ensure that the website is secure and that users can verify they are connecting to the correct website. Once these parameters were set up, I used Checksum Verification to ensure that the certificates were properly created and gave the proper output (the unique hash created by the program). This unique hash confirmed that the program did scramble the data that we entered as a control test inside the program. For this section, I chose to use the SHA-256 encryption algorithm because it is extremely secure and creates a long hash that has a low chance of collision.

Finally, I tested for vulnerabilities in the program. The vulnerabilities found were all related to the program using older versions of springboot, tomcat, etc. These need to be updated to correct the vulnerabilities found in the program. In addition, to add more layers of security to the program, I would recommend regular dependency checks to ensure that the most up to date versions of software are used and that the program has not been compromised. In the process of refactoring the code, I addressed the cryptography, code error, and code quality areas of security. The cryptography was implemented in the use of SHA-256, the code involves secure error handling and follows secure coding practices and patterns.

8. Industry Standard Best Practices

Industry standard best practices used while developing and refactoring the code for this program involved error handling, strong encryption standards, checksum verification, and adapting the code to utilize HTTPS instead of HTTP by creating a self-signed certificate. I also refactored the code to be reusable and the method bytesToHex only serves function (to generate the hash), meaning it falls in line with the Single Responsibility Principle (SRP). All of these add value to the well-being of Artemis Financial by further protecting client data and financial information. The transfer of costumer data must be taken seriously and protected with secure programming to ensure the trust of past, current, and future clientele. This will add to the company's overall well-being by ensuring that they will maintain the customers they currently have and be able to create new business by ensuring they have a reputation for security within their business practices.