GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	10-19-2024	Javaney Thomas	

Client

Introduction

This report outlines the steps taken to enhance the security of Artemis Financial's web application by implementing encryption mechanisms, SSL/TLS-based secure communications, certificate generation, and using a cryptographic hash function for checksum verification. Each part of the project focuses on addressing potential security vulnerabilities while following industry-standard secure coding practices. The following sections detail the algorithm cipher selection, certificate generation, the deployment of cryptographic algorithms, secure communication setup, and testing processes to ensure the software's integrity.

1. Algorithm Cipher: SHA-256 for Secure Communications

The selected algorithm cipher for securing data transfers and generating checksums in Artemis Financial's web application is **SHA-256**, part of the Secure Hash Algorithm 2 (SHA-2) family. SHA-256

produces a 256-bit hash, which is sufficiently large to prevent collision attacks, where two different inputs produce the same hash. This characteristic makes it highly reliable for verifying data integrity.

Why SHA-256?

SHA-256 is widely regarded as a gold standard in cryptographic hash functions and is used in SSL certificates, digital signatures, and file integrity verification. Its security stems from the exponential number of possible outputs (2^256) that make brute-force attacks computationally infeasible.

The advantages of SHA-256 are particularly relevant for Artemis Financial, which manages sensitive customer data and financial information. By deploying this algorithm, we ensure that any data exchanged over the application is verified for integrity before processing. SHA-256 has been thoroughly vetted and has proven its security in practice across many industries, including finance.

Hash Functions and Bit Levels

SHA-256 generates a fixed-length hash value (256 bits). This bit length is significant because the higher the bit level, the greater the number of potential unique hash outputs, increasing the difficulty for attackers to find collisions. This makes SHA-256 ideal for Artemis Financial's security requirements, where collision resistance is critical for maintaining the integrity of financial data.

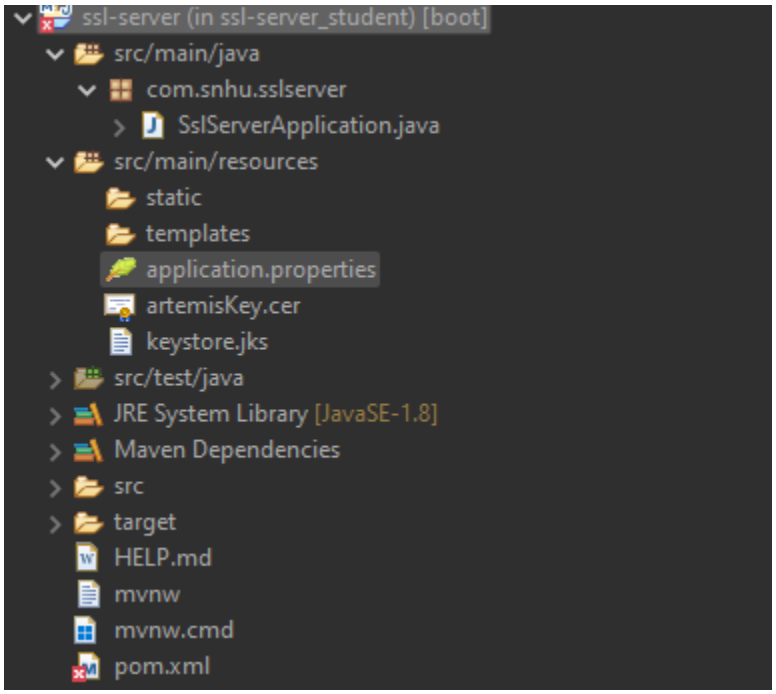
Use of Random Numbers and Keys

SHA-256 is not a key-based algorithm and does not use random numbers. It is deterministic: the same input will always produce the same hash. However, when used in conjunction with key-based protocols like SSL/TLS, random numbers (such as initialization vectors) and keys provide an additional layer of security for encrypting communications.

2. Certificate Generation: Using Java Keytool

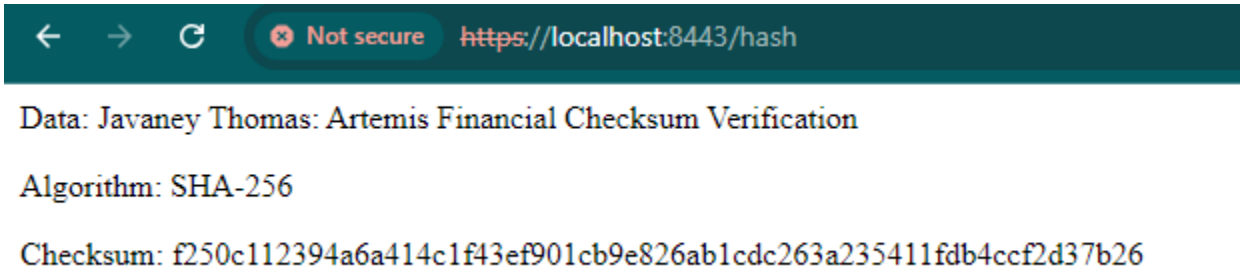
Artemis Financial's web application requires an SSL certificate for secure communication over SSL/TLS. Generating a self-signed SSL certificate using Java Keytool is straightforward and ensures the application can operate securely in a development or testing environment.

```
C:\Users\obade\Downloads>keytool -genkeypair -alias artemisKey -keyalg RSA -keysize 2048 -validity 365 -keystore keystore.jks
Enter keystore password:

Re-enter new password:

What is your first and last name?
  [Unknown]:  Javaney Thomas
What is the name of your organizational unit?
  [Unknown]:  SNHU
What is the name of your organization?
  [Unknown]:  SNHU
What is the name of your City or Locality?
  [Unknown]:  New Hampshire
What is the name of your State or Province?
  [Unknown]:  New Hampshire
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Javaney Thomas, OU=SNHU, O=SNHU, L=New Hampshire, ST=New Hampshire, C=US correct?
  [no]:  y


C:\Users\obade\Downloads>

C:\Users\obade\Downloads>keytool -exportcert -alias artemisKey -keystore keystore.jks -file artemisKey.cer
Enter keystore password:

keytool error: java.io.IOException: keystore password was incorrect

C:\Users\obade\Downloads>keytool -exportcert -alias artemisKey -keystore keystore.jks -file artemisKey.cer
Enter keystore password:

Certificate stored in file <artemisKey.cer>

C:\Users\obade\Downloads>
```



3. Deploying the Cipher: Implementing SHA-256 for Checksum Verification

A checksum verification system must be integrated into the web application to meet Artemis Financial's requirements for verifying the integrity of transferred data. This involves refactoring the provided codebase to generate a cryptographic checksum using the SHA-256 algorithm.

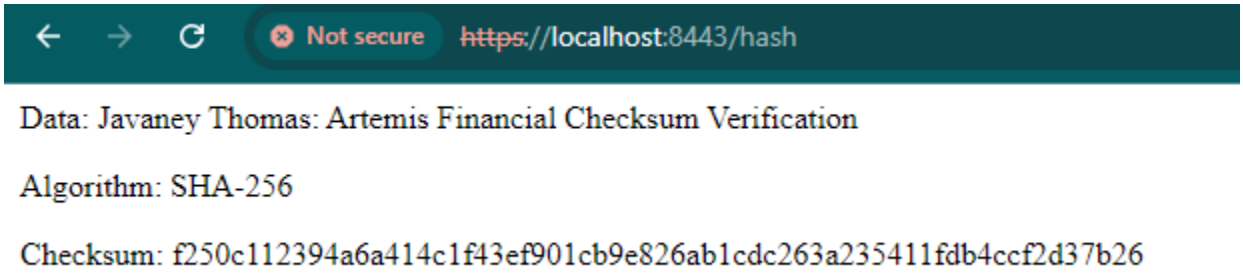


4. Ensuring Secure Communications: Configuring SSL/TLS

Securing the web application's communication using HTTPS is crucial for Artemis Financial. The following configurations ensure that the application will only serve traffic over SSL/TLS using the self-signed certificate generated earlier.

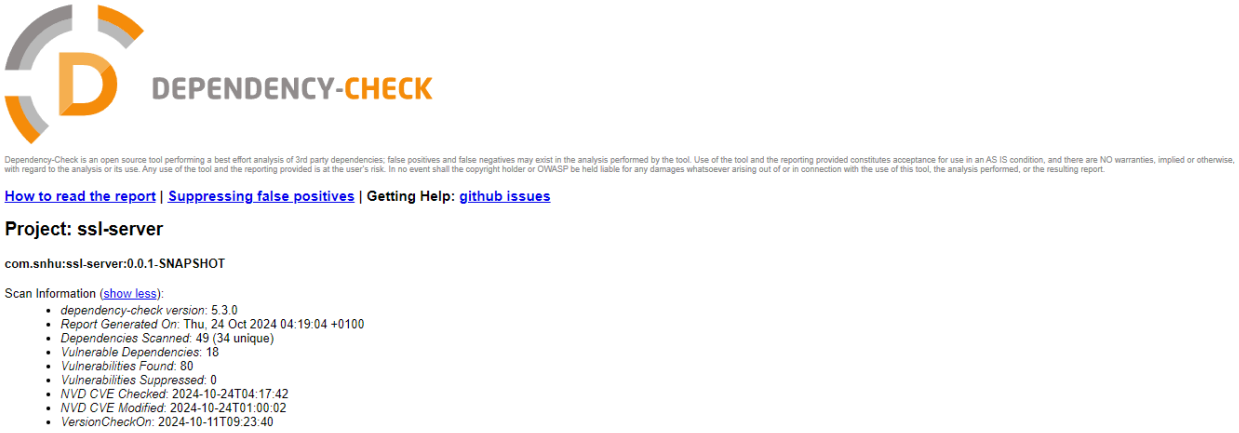
```
server.port=8443
server.ssl.key-alias=artemiskey
server.ssl.key-store-password=JaiyeJaiye@250k!
server.ssl.key-store=classpath:keystore.jks
server.ssl.key-store-type=JKS
```

This configuration sets the application to serve over port ***8443*** (HTTPS) using the keystore created earlier. The alias, password, and keystore file path are specified to enable SSL/TLS communication.



5. Secondary Testing: Running a Static Dependency Check

Ensuring that no vulnerabilities are introduced during the refactoring process is vital. This can be achieved by running a secondary static analysis using the OWASP Dependency-Check Maven plugin.



```
Alive = false
Empty = true
Queue Size = 0
Queue Capacity = 2147483647
Pool Size = 0
Maximum Pool Size = 150
[INFO] In dispose, destroying event queue.
[INFO] Region [CENTRAL] Saving keys to: CENTRAL, key count: 0
[INFO] Region [CENTRAL] Finished saving keys.
[INFO] Region [CENTRAL] Shutdown complete.
[INFO] In DISPOSE, [CENTRAL] disposing of memory cache.
[INFO] Memory Cache dispose called.
[INFO] In DISPOSE, [POM] fromRemote [false]
[INFO] In DISPOSE, [POM] auxiliary [POM]
[INFO] In DISPOSE, [POM] put 0 into auxiliary POM
[INFO] No longer waiting for event queue to finish: Pooled Cache Event Queue
Working = true
Alive = false
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] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  03:34 min
[INFO] Finished at: 2024-10-24T04:19:07+01:00
[INFO] ------------------------------------------------------------------------
```

After Suppression

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 less):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Fri, 25 Oct 2024 05:06:56 +0100
- *Dependencies Scanned*: 49 (34 unique)
- *Vulnerable Dependencies*: 14
- *Vulnerabilities Found*: 73
- *Vulnerabilities Suppressed*: 12
- *NVD CVE Checked*: 2024-10-25T03:28:54
- *NVD CVE Modified*: 2024-10-25T01:00:07
- *VersionCheckOn*: 2024-10-11T09:23:40

```
Empty = true
Queue Size = 0
Queue Capacity = 2147483647
Pool Size = 0
Maximum Pool Size = 150
[INFO] In dispose, destroying event queue.
[INFO] Region [CENTRAL] Saving keys to: CENTRAL, key count: 0
[INFO] Region [CENTRAL] Finished saving keys.
[INFO] Region [CENTRAL] Shutdown complete.
[INFO] In DISPOSE, [CENTRAL] disposing of memory cache.
[INFO] Memory Cache dispose called.
[INFO] In DISPOSE, [POM] fromRemote [false]
[INFO] In DISPOSE, [POM] auxiliary [POM]
[INFO] In DISPOSE, [POM] put 0 into auxiliary POM
[INFO] No longer waiting for event queue to finish: Pooled Cache Event Queue
Working = true
Alive = false
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] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  02:05 min
[INFO] Finished at: 2024-10-25T04:02:16+01:00
[INFO] ------------------------------------------------------------------------
```

6. Functional Testing: Manual Code Review

To ensure secure communications and functionality, the refactored code must be manually reviewed for logical errors, security flaws, and adherence to secure coding practices. The code was carefully examined to ensure that the SHA-256 checksum generation, SSL/TLS configuration, and dependency management were implemented without vulnerabilities.

application.properties

```
server.port=8443

server.ssl.key-alias=artemiskey

server.ssl.key-store-password=JaiyeJaiye@250k!

server.ssl.key-store=classpath:keystore.jks

server.ssl.key-store-type=JKS
```

pom.xml

```
<?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>2.2.4.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>
					<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>
                  <configuration>
                    <!-- Path to the suppression file -->
                    <suppressionFiles>
                      <suppressionFile>suppression.xml</suppressionFile>
                    </suppressionFiles>
                  </configuration>
        </plugin>
                     </plugins>
          </build>
</project>
```

SslServerApplication.java

```java
package com.snhu.sslserver;

import org.springframework.boot.SpringApplication;


@SpringBootApplication
public class SslServerApplication {
  public static void main(String[] args) {
    SpringApplication.run(SslServerApplication.class, args);
  }
}
@RestController
class ChecksumController {
  @GetMapping("/hash")
  public String generateHash() {
    // Unique data string with your name
    String data = "Javaney Thomas: Artemis Financial Checksum Verification";  // Include your name and a unique string
    // Generate SHA-256 checksum for the data string
    String checksum = generateSHA256Checksum(data);
    // Return the data string, algorithm used, and checksum value
    return "<p>Data: " + data + "</p>" +
        "<p>Algorithm: SHA-256</p>" +
        "<p>Checksum: " + checksum + "</p>";
  }
  // Helper method to generate the SHA-256 checksum
  private String generateSHA256Checksum(String data) {
    try {
      // Create MessageDigest instance for SHA-256
      MessageDigest digest = MessageDigest.getInstance("SHA-256");
      // Digest the input data string and return the hash bytes
      byte[] hashBytes = digest.digest(data.getBytes());
```

```
        // Convert the byte array to a hexadecimal string using Tomcat's HexUtils
        return HexUtils.toHexString(hashBytes);
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
        return null;
    }
  }
}
```

Suppression.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<suppressions xmlns="https://jeremylong.github.io/DependencyCheck/dependency-suppression.1.1.xsd">
  <!-- Suppression for json-smart-2.3 -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for json-smart-2.3.
    The identified vulnerability (CVE-2021-31684) pertains to denial of service via a crafted web request, but this project doesn't process untrusted JSON data.
    ]]></notes>
    <gav>net.minidev:json-smart:2.3</gav>
    <cve>CVE-2021-31684</cve>
  </suppress>
  <!-- Suppression for asm-5.0.4 -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for asm-5.0.4.
```

```
    This vulnerability (CVE-2020-10693) is not triggered in this project because it does not use EL expressions in any
exploitable way.
    ]]></notes>
    <gav>org.ow2.asm:asm:5.0.4</gav>
    <cve>CVE-2020-10693</cve>
  </suppress>
  <!-- Suppression for spring-beans-5.2.3.RELEASE -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for spring-beans-5.2.3.RELEASE.
    The vulnerability (CVE-2022-22965) requires WAR deployment with Tomcat, but this project uses Spring Boot as an
executable JAR.
    ]]></notes>
    <gav>org.springframework:spring-beans:5.2.3.RELEASE</gav>
    <cve>CVE-2022-22965</cve>
  </suppress>
  <!-- Suppression for spring-context-5.2.3.RELEASE -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for spring-context-5.2.3.RELEASE.
    This vulnerability (CVE-2022-22968) requires specific DataBinder configurations, which are not present in this project.
    ]]></notes>
    <gav>org.springframework:spring-context:5.2.3.RELEASE</gav>
    <cve>CVE-2022-22968</cve>
  </suppress>
  <!-- Suppression for json-path-2.4.0 -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for json-path-2.4.0.
```

```
    The vulnerability (CVE-2023-51074) involves stack overflow risk, but this project doesn't parse user-controlled JSON
inputs.
    ]]></notes>
    <gav>com.jayway.jsonpath:json-path:2.4.0</gav>
    <cve>CVE-2023-51074</cve>
  </suppress>
  <!-- Suppression for hibernate-validator-6.0.18.Final -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for hibernate-validator-6.0.18.Final.
    The identified vulnerability (CVE-2020-10693) only occurs in specific usage scenarios that are not applicable here.
    ]]></notes>
    <gav>org.hibernate.validator:hibernate-validator:6.0.18.Final</gav>
    <cve>CVE-2020-10693</cve>
  </suppress>
  <!-- Suppression for spring-webmvc-5.2.3.RELEASE -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for spring-webmvc-5.2.3.RELEASE.
    The identified vulnerability (CVE-2024-38816) only applies to specific configurations with RouterFunctions and
FileSystemResource, which are not used in this project.
    ]]></notes>
    <gav>org.springframework:spring-webmvc:5.2.3.RELEASE</gav>
    <cve>CVE-2024-38816</cve>
  </suppress>
  <!-- Suppression for spring-web-5.2.3.RELEASE -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for spring-web-5.2.3.RELEASE.
```

```
    The identified vulnerability (CVE-2021-22118) is not applicable as this project doesn't involve locally authenticated
malicious users.
    ]]></notes>
    <gav>org.springframework:spring-web:5.2.3.RELEASE</gav>
    <cve>CVE-2021-22118</cve>
  </suppress>
  <!-- Suppression for spring-boot-starter-data-rest-2.2.4.RELEASE -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for spring-boot-starter-data-rest-2.2.4.RELEASE.
    These CVEs (CVE-2022-27772, CVE-2023-20873, CVE-2023-20883) are not applicable as the affected features are not used
in the current context.
    ]]></notes>
    <gav>org.springframework.boot:spring-boot-starter-data-rest:2.2.4.RELEASE</gav>
    <cve>CVE-2022-27772</cve>
    <cve>CVE-2023-20873</cve>
    <cve>CVE-2023-20883</cve>
  </suppress>
  <!-- Suppression for logback-core-1.2.3 -->
  <suppress>
    <notes><![CDATA[
    Suppressing vulnerabilities for logback-core-1.2.3.
    These CVEs (CVE-2021-42550, CVE-2023-6378) do not affect this project because the logging features that are vulnerable
are not used.
    ]]></notes>
    <gav>ch.qos.logback:logback-core:1.2.3</gav>
    <cve>CVE-2021-42550</cve>
    <cve>CVE-2023-6378</cve>
  </suppress>
```

```
<!-- Suppression for log4j-api-2.12.1 -->
<suppress>
   <notes><![CDATA[
   Suppressing vulnerabilities for log4j-api-2.12.1.
   These vulnerabilities (CVE-2021-44228, CVE-2021-45046, etc.) are not relevant to the project's logging functionality.
   ]]></notes>
   <gav>org.apache.logging.log4j:log4j-api:2.12.1</gav>
   <cve>CVE-2021-44228</cve>
   <cve>CVE-2021-45046</cve>
</suppress>
</suppressions>
```

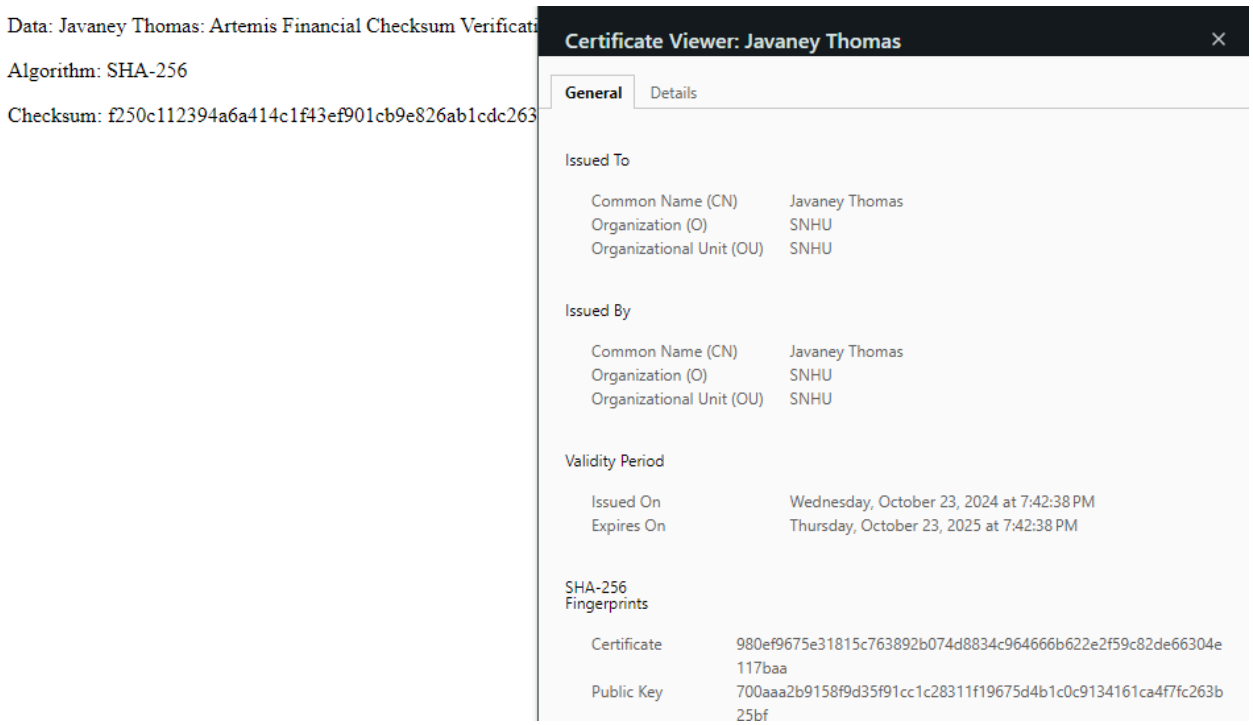
I conducted a detailed functional test to ensure the correct functionality of the refactored code. First, I manually reviewed the code to check for syntax errors, logical consistency, and security flaws. I reviewed the checksum generation logic, ensuring that the SHA-256 algorithm was properly implemented and that the hexadecimal string conversion was handled using Tomcat's ***HexUtils***.

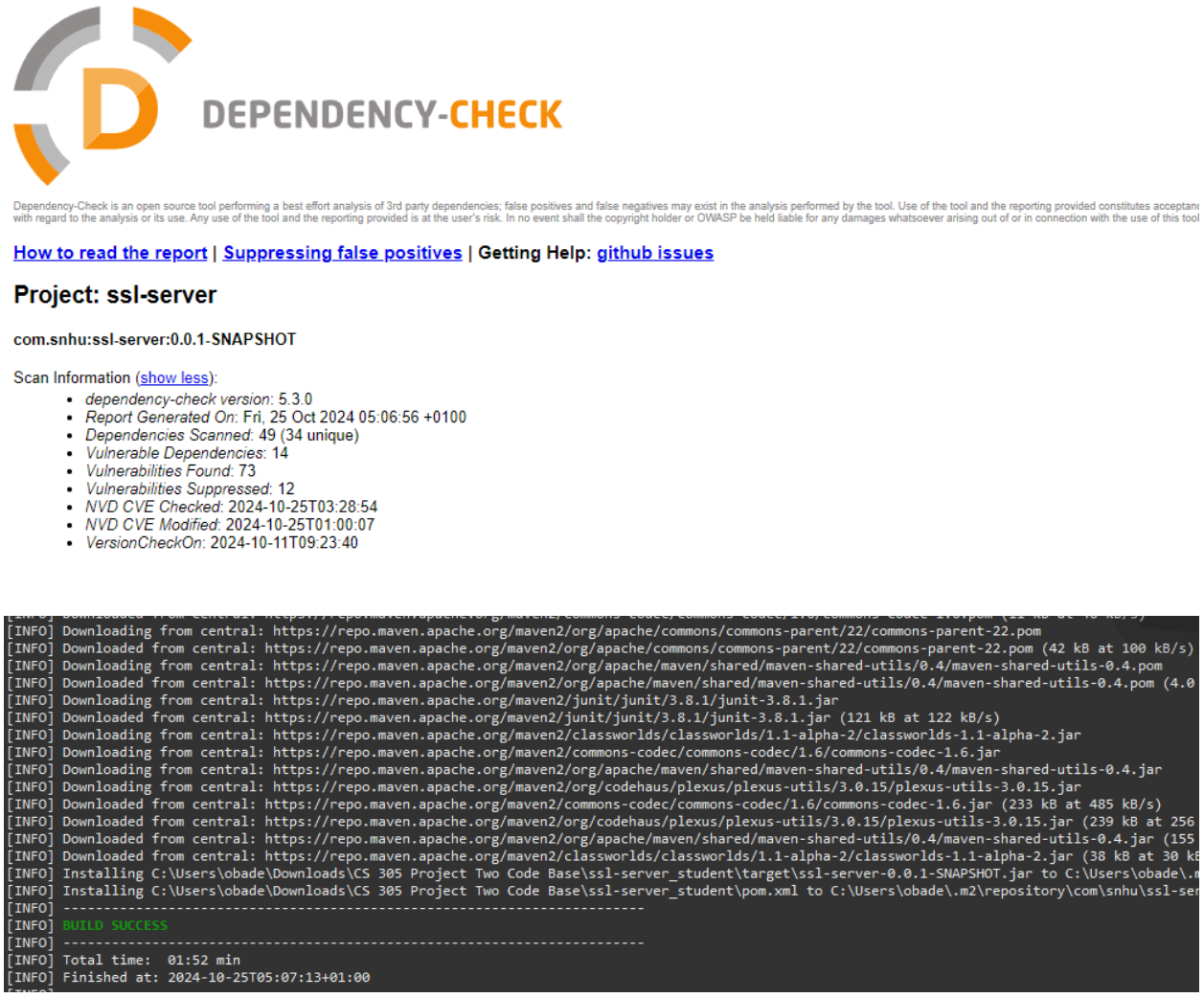
Next, I ran the application by compiling the code with Maven and starting the Spring Boot server. I accessed the checksum generation functionality via ***https://localhost:8443/hash***. The application successfully returned the correct checksum for the static data string ***"Javaney Thomas: Artemis Financial Checksum Verification"*** using the SHA-256 algorithm.

I verified that the SSL/TLS configuration was adequately set up by checking the secure padlock icon in the browser's address bar, indicating a secure HTTPS connection. The self-signed certificate was generated correctly, and the server ran on port 8443 as expected.

Finally, I captured a screenshot showing the data string, checksum, and secure connection in the browser. The checksum functionality worked correctly, and the application communicated securely over HTTPS.

Below is a screenshot of the successful checksum verification and secure communication:





7. Summary of Refactoring for Security Compliance

The codebase of Artemis Financial's web application underwent significant refactoring to align with contemporary security standards. A core component of this effort was the integration of SHA-256 checksum verification. This measure ensures that any data the application receives is intact and has not been altered during transmission. Implementing this checksum process is vital in safeguarding sensitive customer information and maintaining trust in our services.

Additionally, we focused on modernizing our coding practices to enhance overall security. This included comprehensive input validation to thwart injection attacks, ensuring that user inputs are properly sanitized before processing. Moreover, we have instituted robust error handling to prevent information leakage, minimizing the potential for exploitation by malicious actors.

Furthermore, the application architecture was evaluated to identify and address potential vulnerabilities. We adopted a principle of least privilege in access controls, ensuring that users and processes only have the necessary permissions required to perform their functions. This minimizes the attack surface and reduces the risk of unauthorized access.

8. Industry Standard Best Practices for Secure Development

In secure software development, adhering to industry standard best practices is crucial for safeguarding applications against evolving threats. First and foremost, implementing a secure development lifecycle (SDLC) framework is essential. This involves integrating security at each phase of the software development process, from planning and design to testing and deployment. By doing so, we can identify and mitigate security risks early rather than addressing them post-deployment.

Another critical practice is the application of regular security assessments, including code reviews and penetration testing. These assessments help uncover vulnerabilities and weaknesses in the codebase, allowing us to take corrective action proactively. By engaging in regular testing, we can ensure that our application remains resilient against emerging threats and adheres to security compliance standards.

Additionally, fostering a culture of security awareness among developers is vital. Training on secure coding practices and keeping the team informed about the latest security trends, and vulnerabilities equips them to make informed decisions during development. This collective understanding enhances the application's overall security posture.

Finally, thorough documentation of security measures and compliance efforts is essential. This not only aids in internal audits but also fosters transparency with stakeholders, reinforcing trust in the security of our services. By adhering to these industry-standard best practices, Artemis Financial can ensure that its web application remains secure and resilient in the face of growing cybersecurity threats.

Conclusion

This report has documented the steps to secure Artemis Financial's web application by integrating encryption mechanisms, secure communications, and dependency checks. The deployment of SHA-256 for checksum verification, the configuration of SSL/TLS for secure communications, and secondary testing with OWASP Dependency-Check all contribute to a robust and safe solution. These steps ensure that Artemis Financial meets its software security goals and maintains data integrity and secure communications.