Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	February 20, 2026	Joshua Sevy	

Client

ARTEMIS
FINANCIAL

Developer
Joshua Sevy

1. Algorithm Cipher

For this project, the chosen algorithm was SHA-256. It is a cryptographic checksum function that generates a fixed 256-bit hash value from arbitrary input data. Hash algorithms, unlike encryption algorithms that provide reversible data protection, are intended to verify data validity and produce a unique digest of the source data.

SHA-256 is recommended primarily because of its strong collision resistance and is generally regarded as the industry standard for providing data integrity; the likelihood of collision is extremely low. Though theoretically possible for any hash function to collide, SHA-256 makes colliding with it is virtually impossible to compute in practice. SHA-256 is also the most commonly used hash function for the validation of HTTPS certificates, password hashing, digital signing of documents, etc., which makes it an appropriate, standardized, and practical way to preserve data integrity in the financial system.

Although MD5 and SHA-1 were considered, they were discarded due to published collision vulnerabilities, whereby an attacker could generate two different pieces of data that produce an identical hash value. In contrast, SHA-256 provides a substantially higher level of collision resistance and represents the most current recommendation for ensuring integrity in today's security systems, including TLS certificates, blockchain technology, and secure software distribution.

Hash functions are different from symmetric and asymmetric encryption because they only work in one direction. Symmetric encryption uses the same key for both people, while asymmetric encryption uses a public key and a private key to encrypt and decrypt messages. Because hashing is one-way, it provides integrity verification rather than confidentiality, which is instead provided by encryption mechanisms such as TLS. Secure hashing standards and encryption methods follow NIST cryptographic recommendations (National Institute of Standards and Technology, 2001).

Randomness is important in trusted communication protocols like TLS because it is used during the handshake process. Along with the security from HTTPS, SHA-256 helps keep information private and makes sure it stays accurate at the Application layer. Cryptographic hash functions are important for secure application design and can be used with Java’s Cryptography Architecture framework (Oracle Corporation, 2026).

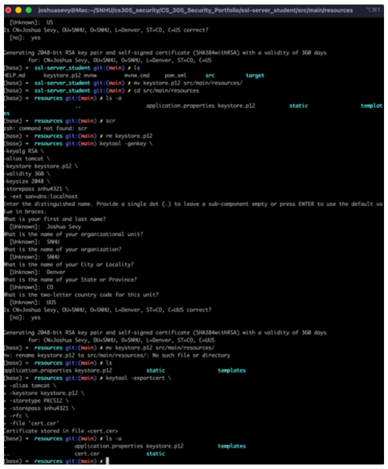
2. Certificate Generation

A self-signed SSL Certificate was created using Java Keytool in the development environment to facilitate secure communication for the Artemis Financial web application (Oracle Corporation, n.d.). Certificates facilitate secure HTTPS communication by providing the server with a cryptographic identity used to authenticate itself to connecting clients.

A public-private key pair was created using the Java Keytool command, which created a JKS (Java Key Store). The private key is safely stored in the application, and the public certificate will be used during the TLS handshake to facilitate secure communication between the client browser and the server.

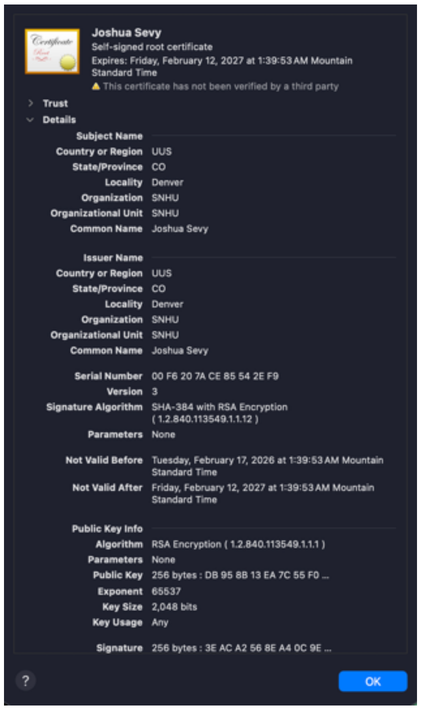
Although self-signed certificates are not recognized by a public Certificate Authority for verification, they are useful for development and testing purposes, as they offer a way for developers to confirm the existence of secure communication functionality without having to register with a public certificate authority. This enables the developer to confirm encryption functionality before moving to a production environment where a trusted Certificate Authority is used.

After generation, the certificate was exported as a .cer file to verify successful creation and to provide evidence that the application possesses a valid cryptographic identity. Exporting the certificate confirms that the key pair was correctly generated and allows inspection of its metadata, such as the issuer, validity period, and encryption algorithm.



```
joshuasevy@Mac:~/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/src/main/resources
 [Unknown]:  US
Is CN=Joshua Sevy, OU=SNHU, O=SNHU, L=Denver, ST=CO, C=US correct?
 [no]:  yes

Generating 2048-bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 360 days
        for: CN=Joshua Sevy, OU=SNHU, O=SNHU, L=Denver, ST=CO, C=US
(base) ➜  ssl-server_student git:(main) ✗ ls
HELP.md      keystore.p12 mvnw         mvnw.cmd     pom.xml      src          target
(base) ➜  ssl-server_student git:(main) ✗ mv keystore.p12 src/main/resources/
(base) ➜  ssl-server_student git:(main) ✗ cd src/main/resources
(base) ➜  resources git:(main) ✗ ls -a
.                   ..                  application.properties keystore.p12         static               templat
es
(base) ➜  resources git:(main) ✗ scr
zsh: command not found: scr
(base) ➜  resources git:(main) ✗ rm keystore.p12
(base) ➜  resources git:(main) keytool -genkey \
-keyalg RSA \
-alias tomcat \
-keystore keystore.p12 \
-validity 360 \
-keysize 2048 \
-storepass snhu4321 \
> -ext san=dns:localhost
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default va
lue in braces.
What is your first and last name?
 [Unknown]:  Joshua Sevy
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]:  Denver
What is the name of your State or Province?
 [Unknown]:  CO
What is the two-letter country code for this unit?
 [Unknown]:  UUS
Is CN=Joshua Sevy, OU=SNHU, O=SNHU, L=Denver, ST=CO, C=UUS correct?
 [no]:  yes

Generating 2048-bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 360 days
        for: CN=Joshua Sevy, OU=SNHU, O=SNHU, L=Denver, ST=CO, C=UUS
(base) ➜  resources git:(main) ✗ mv keystore.p12 src/main/resources/
mv: rename keystore.p12 to src/main/resources/: No such file or directory
(base) ➜  resources git:(main) ✗ ls
application.properties keystore.p12         static               templates
(base) ➜  resources git:(main) ✗ keytool -exportcert \
> -alias tomcat \
> -keystore keystore.p12 \
> -storetype PKCS12 \
> -storepass snhu4321 \
> -rfc \
> -file 'cert.cer'
Certificate stored in file <cert.cer>
(base) ➜  resources git:(main) ✗ ls -a
.                   application.properties keystore.p12         templates
..                  cert.cer             static
(base) ➜  resources git:(main) ✗
```

Figure 1. Generating the certificate using Java Keytool utility



Joshua Sevy
Self-signed root certificate
Expires: Friday, February 12, 2027 at 1:39:53 AM Mountain Standard Time
⚠ This certificate has not been verified by a third party

› Trust
⌄ Details

Subject Name	
Country or Region	UUS
State/Province	CO
Locality	Denver
Organization	SNHU
Organizational Unit	SNHU
Common Name	Joshua Sevy
Issuer Name	
Country or Region	UUS
State/Province	CO
Locality	Denver
Organization	SNHU
Organizational Unit	SNHU
Common Name	Joshua Sevy
Serial Number	00 F6 20 7A CE 85 54 2E F9
Version	3
Signature Algorithm	SHA-384 with RSA Encryption (1.2.840.113549.1.1.12)
Parameters	None
Not Valid Before	Tuesday, February 17, 2026 at 1:39:53 AM Mountain Standard Time
Not Valid After	Friday, February 12, 2027 at 1:39:53 AM Mountain Standard Time
Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes : DB 95 8B 13 EA 7C 55 F0 ...
Exponent	65537
Key Size	2,048 bits
Key Usage	Any
Signature	256 bytes : 3E AC A2 56 8E A4 0C 9E ...

Figure 2. Certificate within keychain store

The successful generation and export of the certificate enabled the application to support Transport Layer Security (TLS), allowing encrypted HTTPS communication to be configured and verified in later stages of the project.

3. Deploy Cipher

Checksum verification was added to the Artemis Financial app to help protect data and make communications more secure. A checksum lets the system check if data has been changed, damaged, or tampered with while being sent or processed. Instead of encrypting the data, the app creates a fixed-length hash value that represents the original input and can be used to check its integrity.

We chose the SHA-256 hashing algorithm because it is part of the trusted SHA-2 family and is widely used to check data authenticity. SHA-256 creates a 256-bit hash and is built to avoid collisions, so it is extremely unlikely that two different inputs will produce the same hash. This reliability is especially important for financial systems like Artemis Financial, where accuracy and data trust are essential.

SHA-256 offers a good balance between security and speed. Stronger algorithms like SHA-512 create bigger hashes but need more processing power, which is not needed for most web-based checksum checks. On the other hand, older algorithms like MD5 are faster but are no longer secure because of known weaknesses. The table below shows these tradeoffs.

Algorithm	Hash Size	Security Level	Performance	Usage
MD5	128-bit	Weak (collision vulnerabilities)	Very Fast	Deprecated, not recommended
SHA-256	256-bit	Strong collision resistance	Efficient	Industry standard of integrity verification
SHA-512	512-bit	Very Strong	Computationally expensive	Used for high-security or large data systems

SHA-256 provides strong security and is fast enough for real-time web requests, making it a good choice for apps that need to check data often without slowing down. It is used in many real-world systems, such as TLS certificates, digital signatures, software downloads, and blockchain. This keeps Artemis Financial in line with current industry standards.

To add the checksum feature, we updated the app to use Java's built-in MessageDigest class. When a user sends data to the /hash endpoint, the app creates a SHA-256 hash from the UTF-8 encoded input and turns the result into a hexadecimal string, so it is easy to read and compare (Oracle Corporation, 2026).

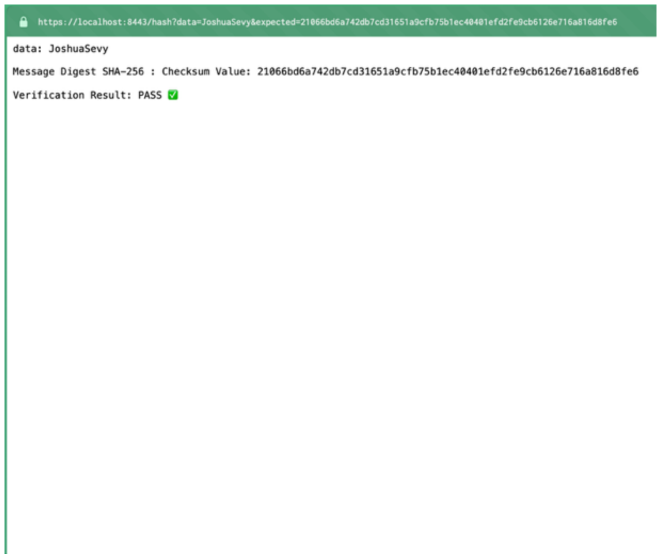
```java
private String createChecksum(String data) {
    try {
       MessageDigest md = MessageDigest.getInstance(HASH_ALGORITHM);

       byte[] hashBytes =
             md.digest(data.getBytes(StandardCharsets.UTF_8));

       return bytesToHex(hashBytes);

    } catch (NoSuchAlgorithmException e) {
       throw new RuntimeException("SHA-256 algorithm not found", e);
    }
}
```

We then added logic to compare the generated checksum with the expected value provided by the user. If the values match, the app confirms the data has not changed. If they do not match, the app reports a verification failure, which could mean the data was changed or tampered with. This upgrade turns the feature from just a hash generator into a tool for checking data integrity.

To make the system even more secure, we added several coding controls:

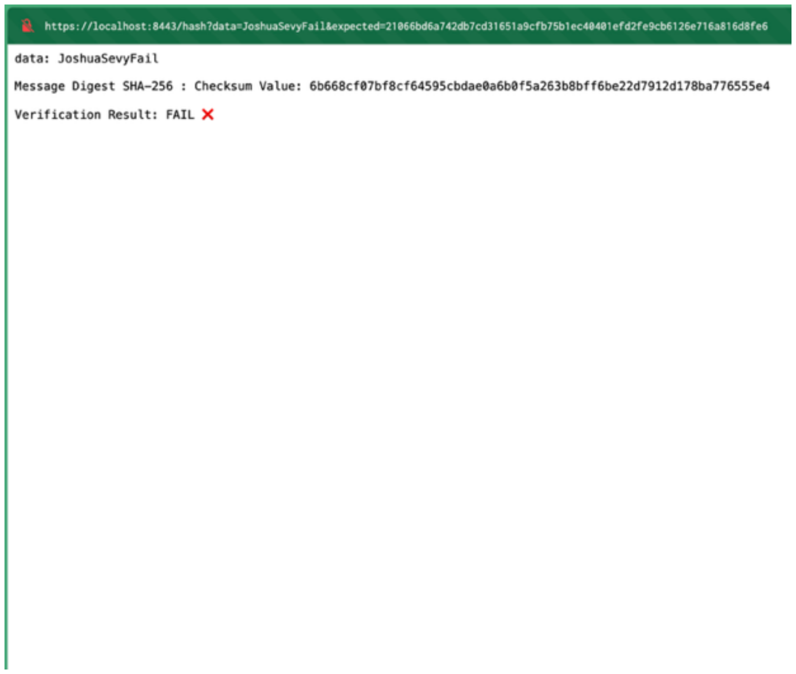
- Input validation prevents null values and excessively large payloads, reducing denial-of-service risks.
- Hash format validation ensures comparison values contain only valid hexadecimal characters and match the expected SHA-256 length.
- HTML encoding is applied before displaying user input to mitigate cross-site scripting (XSS) attacks.
- Controlled output formatting ensures only validated data is rendered in responses.

These safeguards align the checksum implementation with secure coding best practices by preventing malicious input from affecting application behavior while still enabling users to verify data integrity. The application displays both the generated checksum and verification result directly in the browser. Matching values produce a **PASS** result, while mismatches produce a **FAIL** result, clearly demonstrating whether transmitted data remains intact.



```
https://localhost:8443/hash?data=JoshuaSevy&expected=21066bd6a742db7cd31651a9cfb75b1ec40401efd2fe9cb6126e716a816d8fe6
data: JoshuaSevy
Message Digest SHA-256 : Checksum Value: 21066bd6a742db7cd31651a9cfb75b1ec40401efd2fe9cb6126e716a816d8fe6
Verification Result: PASS ✅
```

Figure 3. SHA-256 Checksum generation and verification showing successful data integrity validation.



```
https://localhost:8443/hash?data=JoshuaSevyFail&expected=21066bd6a742db7cd31651a9cfb75b1ec40401efd2fe9cb6126e716a816d8fe6
data: JoshuaSevyFail
Message Digest SHA-256 : Checksum Value: 6b668cf07bf8cf64595cbdae0a6b0f5a263b8bff6be22d7912d178ba776555e4
Verification Result: FAIL ❌
```

Figure 4. Showing checksum verification failure

Testing was performed using a unique input string containing the developer's name and custom data. The results confirmed that the refactored application correctly generates SHA-256 hash values and performs checksum validation, providing a reliable mechanism for verifying secure data transmission within the Artemis Financial system.

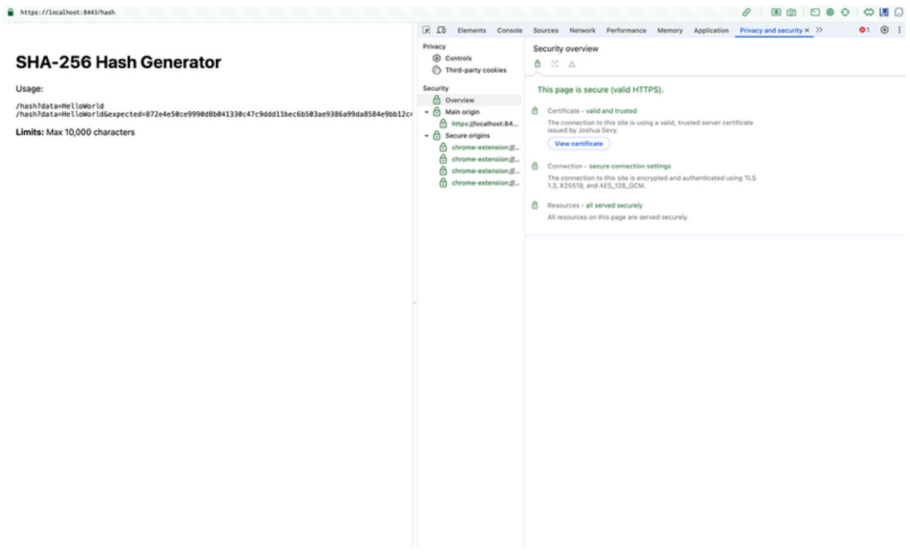
4. Secure Communications

To keep user data safe when sent to the Artemis Financial app, we switched from HTTP to HTTPS with Transport Layer Security (TLS). HTTPS encrypts the data between the browser and server, stopping others from reading or changing it during transfer.

We created a self-signed SSL certificate with the Java Keytool and set it up in the Spring Boot app. Then, we updated the application.properties file to turn on SSL and point to the keystore with the certificate.

```
# use HTTPS with the specified keystore and certificate.
server.port=8443
server.ssl.key-alias=tomcat
server.ssl.key-store-password=snhu4321
server.ssl.key-store=classpath:keystore.p12
server.ssl.key-store-type=PKCS12
```

Once the app was built and started, we checked secure communication by visiting `https://localhost:8443/hash`.


https://localhost:8443/hash

SHA-256 Hash Generator

Usage:

```
/hash?data=HelloWorld
/hash?data=HelloWorld&expected=872e4e50ce9990d8b041330c47c9ddd11bec6b503ae9386a99da8584e9bb12c
```

Limits: Max 10,000 characters

Security overview

This page is secure (valid HTTPS).

Certificate - valid and trusted
The connection to this site is using a valid, trusted server certificate issued by Joshua Sevy.

Connection - secure connection settings
The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.

Resources - all served securely
All resources on this page are served securely.

Figure 5. HTTPS secure communication browser

HTTPS with TLS gives three main types of security:

- Confidentiality: encrypts data so no one else can read it
- Integrity: makes sure data is not changed without being noticed
- Authentication: checks the server's identity using digital certificates

Artemis Financial keeps your financial data safe while it is being sent by using HTTPS. The app also uses SHA-256 checksum validation to secure both your communication and data, following the latest software security standards. These methods follow the secure web application principles described in Iron-Clad Java (Manico & Detlefsen, 2014).

5. Secondary Testing

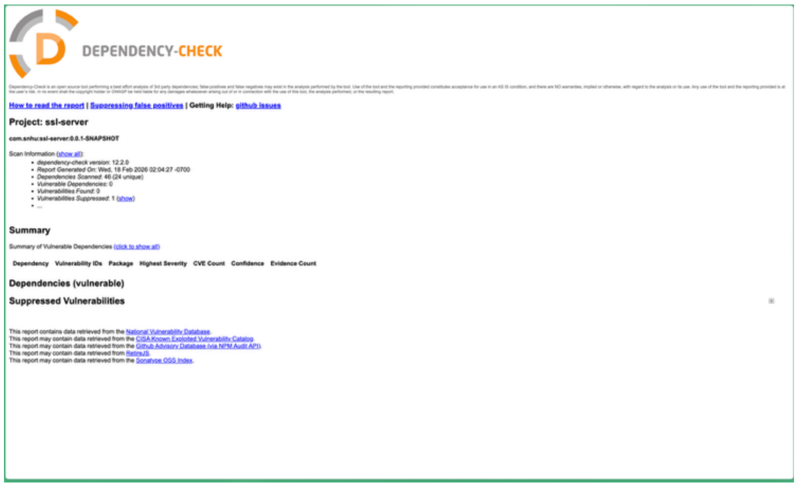
Once the checksum feature was added and HTTPS was enabled, we ran another static security scan with the OWASP Dependency-Check Maven plugin. This helped confirm that the new features and configuration changes did not create new security issues. Scanning for dependency vulnerabilities is a recommended practice, since modern applications often use third-party libraries that can have known security risks (OWASP Foundation, n.d.).

During testing, several vulnerabilities were identified that originated from outdated frameworks and dependencies included in the original project configuration. Rather than ignoring these issues, the OWASP Dependency-Check plugin was updated, the application was migrated to a supported Java version, and the project was upgraded to a newer Spring Boot release. These updates introduced patched dependency versions and resolved multiple known vulnerabilities through standard maintenance and patch management practices aligned with secure software development recommendations (National Institute of Standards and Technology, n.d.).

After making these updates, we ran the dependency check again to see if the changes worked. Most of the reported vulnerabilities were fixed by upgrading dependencies, showing that the refactoring process followed secure maintenance and vulnerability remediation practices.

A manual review of the remaining findings showed that one reported issue was a false positive. The vulnerability was linked to a dependency found through metadata matching, but the affected functionality was not used by the application at runtime. After confirming this, we added a suppression rule using a suppression.xml file created from the dependency-check report. OWASP recommends carefully checking false positives before suppressing them to ensure real vulnerabilities remain visible to developers (OWASP Foundation, n.d.).

The final dependency-check scan confirmed that adding the checksum and enabling HTTPS did not introduce any new publicly known vulnerabilities. This verification step shows that we followed secure development testing protocols and that the updated application has a stronger security posture.


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 performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS IS condition, and there are NO warranties, implied or otherwise, with regard to the analysis or its use. Any use of the tool and the reporting provided is at the user's risk. In no event shall the copyright holder 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*: 12.2.0
- *Report Generated On*: Wed, 18 Feb 2026 02:04:27 -0700
- *Dependencies Scanned*: 46 (24 unique)
- *Vulnerable Dependencies*: 0
- *Vulnerabilities Found*: 0
- *Vulnerabilities Suppressed*: 1 (show)
- ...

Summary

Summary of Vulnerable Dependencies (click to show all)

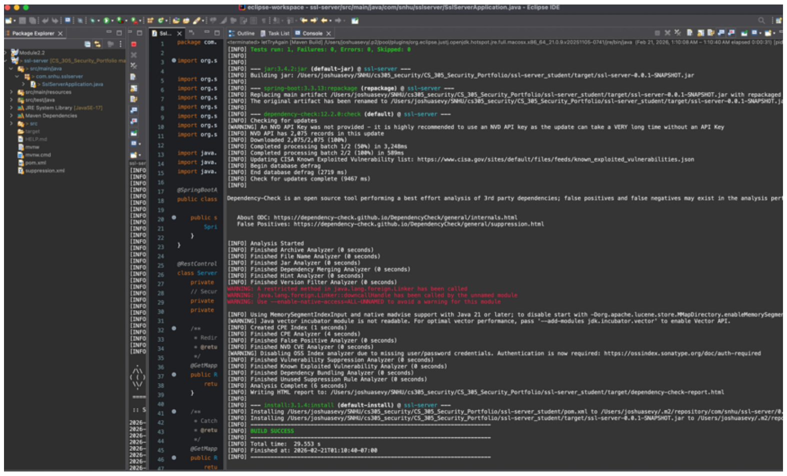
Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count

Dependencies (vulnerable)

Suppressed Vulnerabilities

This report contains data retrieved from the National Vulnerability Database.
This report may contain data retrieved from the CISA Known Exploited Vulnerability Catalog.
This report may contain data retrieved from the Github Advisory Database (via NPM Audit API).
This report may contain data retrieved from RetireJS.
This report may contain data retrieved from the Sonatype OSS Index.

Figure 6. Secondary vulnerability report


eclipse-workspace - ssl-server/src/main/java/com/snhu/sslserver/SslServerApplication.java - Eclipse IDE
Package Explorer
Outline
Task List
Console

```
<terminated> letTryAgain [Maven Build] /Users/joshuasevy/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_21.0.9.v20251105-0741/jre/bin/java (Feb 21, 2026, 1:10:08 AM – 1:10:40 AM elapsed 0:00:31) [pid
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jar:3.4.2:jar (default-jar) @ ssl-server ---
[INFO] Building jar: /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/target/ssl-server-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot:3.3.13:repackage (repackage) @ ssl-server ---
[INFO] Replacing main artifact /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/target/ssl-server-0.0.1-SNAPSHOT.jar with repackaged
[INFO] The original artifact has been renamed to /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/target/ssl-server-0.0.1-SNAPSHOT.ja
[INFO]
[INFO] --- dependency-check:12.2.0:check (default) @ ssl-server ---
[INFO] Checking for updates
[WARNING] An NVD API Key was not provided - it is highly recommended to use an NVD API key as the update can take a VERY long time without an API Key
[INFO] NVD API has 2,075 records in this update
[INFO] Downloaded 2,075/2,075 (100%)
[INFO] Completed processing batch 1/2 (50%) in 3,248ms
[INFO] Completed processing batch 2/2 (100%) in 589ms
[INFO] Updating CISA Known Exploited Vulnerability list: https://www.cisa.gov/sites/default/files/feeds/known_exploited_vulnerabilities.json
[INFO] Begin database defrag
[INFO] End database defrag (2719 ms)
[INFO] Check for updates complete (9467 ms)
[INFO]

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 per


   About ODC: https://dependency-check.github.io/DependencyCheck/general/internals.html
   False Positives: https://dependency-check.github.io/DependencyCheck/general/suppression.html


[INFO] Analysis Started
[INFO] Finished Archive Analyzer (0 seconds)
[INFO] Finished File Name Analyzer (0 seconds)
[INFO] Finished Jar Analyzer (0 seconds)
[INFO] Finished Dependency Merging Analyzer (0 seconds)
[INFO] Finished Hint Analyzer (0 seconds)
[INFO] Finished Version Filter Analyzer (0 seconds)
WARNING: A restricted method in java.lang.foreign.Linker has been called
WARNING: java.lang.foreign.Linker::downcallHandle has been called by the unnamed module
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for this module

[INFO] Using MemorySegmentIndexInput and native madvise support with Java 21 or later; to disable start with -Dorg.apache.lucene.store.MMapDirectory.enableMemorySegmen
[WARNING] Java vector incubator module is not readable. For optimal vector performance, pass '--add-modules jdk.incubator.vector' to enable Vector API.
[INFO] Created CPE Index (1 seconds)
[INFO] Finished CPE Analyzer (4 seconds)
[INFO] Finished False Positive Analyzer (0 seconds)
[INFO] Finished NVD CVE Analyzer (0 seconds)
[WARNING] Disabling OSS Index analyzer due to missing user/password credentials. Authentication is now required: https://ossindex.sonatype.org/doc/auth-required
[INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
[INFO] Finished Known Exploited Vulnerability Analyzer (0 seconds)
[INFO] Finished Dependency Bundling Analyzer (0 seconds)
[INFO] Finished Unused Suppression Rule Analyzer (0 seconds)
[INFO] Analysis Complete (6 seconds)
[INFO] Writing HTML report to: /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/target/dependency-check-report.html
[INFO]
[INFO] --- install:3.1.4:install (default-install) @ ssl-server ---
[INFO] Installing /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/pom.xml to /Users/joshuasevy/.m2/repository/com/snhu/ssl-server/0.
[INFO] Installing /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfolio/ssl-server_student/target/ssl-server-0.0.1-SNAPSHOT.jar to /Users/joshuasevy/.m2/repo
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  29.553 s
[INFO] Finished at: 2026-02-21T01:10:40-07:00
[INFO] ------------------------------------------------------------------------
```

Figure 7. Vulnerability report maven build

6. Functional Testing

Along with automated dependency scans, we also conducted manual functional testing to identify potential syntax, logic, or security issues in the updated application. This review made sure the new checksum and secure communication features worked as intended and followed secure coding standards.

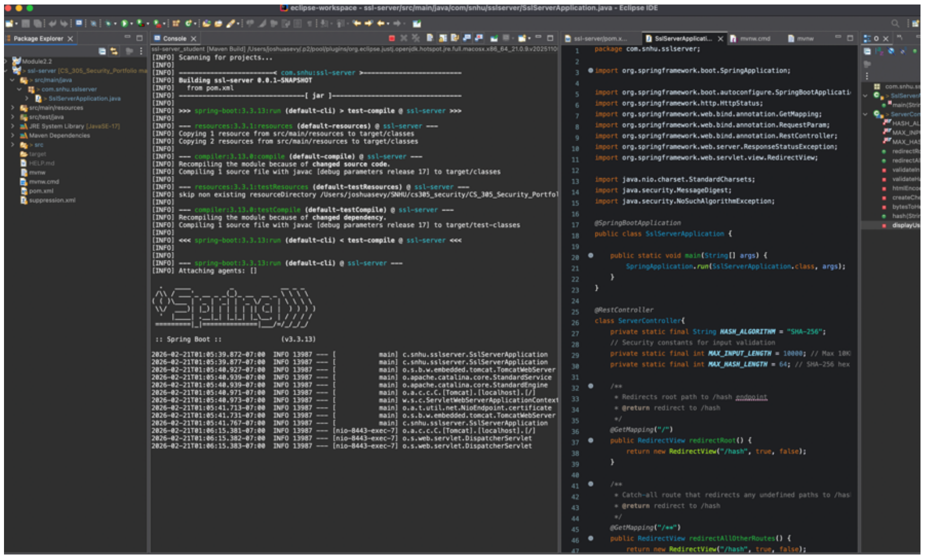
Syntax testing showed that the application compiled and ran successfully after the changes. The Spring Boot server started up without errors, and all endpoints worked as expected during testing.

Logic testing checked that the application handled user actions safely and as expected. We tested the `/hash` endpoint with valid, missing, and invalid inputs to ensure the checksum logic returned the correct **PASS** or **FAIL** result. We also checked that users were safely redirected from undefined routes to the correct endpoint, without exposing any unexpected behavior.

We also did a manual security review to find any risks from how user input is handled. During testing, we checked several security controls:

- Input validation prevents null values and excessively large payloads, reducing denial-of-service risks.
- Hash comparison inputs are validated to ensure proper SHA-256 hexadecimal formatting.
- HTML encoding is applied before displaying user-provided input, mitigating cross-site scripting (XSS) attacks.
- Controlled response formatting ensures only validated data is rendered in browser output.

Testing showed that these security measures worked as intended and the application ran smoothly with both valid and invalid inputs. This manual review shows that the updated application works correctly and follows secure coding practices.



```
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------< com.snhu:ssl-server >------------------------
[INFO] Building ssl-server 0.0.1-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot:3.3.13:run (default-cli) > test-compile @ ssl-server >>>
[INFO]
[INFO] --- resources:3.3.1:resources (default-resources) @ ssl-server ---
[INFO] Copying 1 resource from src/main/resources to target/classes
[INFO] Copying 2 resources from src/main/resources to target/classes
[INFO]
[INFO] --- compiler:3.13.0:compile (default-compile) @ ssl-server ---
[INFO] Recompiling the module because of changed source code.
[INFO] Compiling 1 source file with javac [debug parameters release 17] to target/classes
[INFO]
[INFO] --- resources:3.3.1:testResources (default-testResources) @ ssl-server ---
[INFO] skip non existing resourceDirectory /Users/joshuasevy/SNHU/cs305_security/CS_305_Security_Portfol
[INFO]
[INFO] --- compiler:3.13.0:testCompile (default-testCompile) @ ssl-server ---
[INFO] Recompiling the module because of changed dependency.
[INFO] Compiling 1 source file with javac [debug parameters release 17] to target/test-classes
[INFO]
[INFO] <<< spring-boot:3.3.13:run (default-cli) < test-compile @ ssl-server <<<
[INFO]
[INFO]
[INFO] --- spring-boot:3.3.13:run (default-cli) @ ssl-server ---
[INFO] Attaching agents: []

 :: Spring Boot ::               (v3.3.13)

2026-02-21T01:05:39.872-07:00  INFO 13987 --- [           main] c.snhu.sslserver.SslServerApplication
2026-02-21T01:05:39.877-07:00  INFO 13987 --- [           main] c.snhu.sslserver.SslServerApplication
2026-02-21T01:05:40.927-07:00  INFO 13987 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2026-02-21T01:05:40.939-07:00  INFO 13987 --- [           main] o.apache.catalina.core.StandardService
2026-02-21T01:05:40.939-07:00  INFO 13987 --- [           main] o.apache.catalina.core.StandardEngine
2026-02-21T01:05:40.971-07:00  INFO 13987 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
2026-02-21T01:05:40.973-07:00  INFO 13987 --- [           main] w.s.c.ServletWebServerApplicationContext
2026-02-21T01:05:41.713-07:00  INFO 13987 --- [           main] o.a.t.util.net.NioEndpoint.certificate
2026-02-21T01:05:41.731-07:00  INFO 13987 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
2026-02-21T01:05:41.767-07:00  INFO 13987 --- [           main] c.snhu.sslserver.SslServerApplication
2026-02-21T01:06:15.381-07:00  INFO 13987 --- [nio-8443-exec-7] o.a.c.c.C.[Tomcat].[localhost].[/]
2026-02-21T01:06:15.382-07:00  INFO 13987 --- [nio-8443-exec-7] o.s.web.servlet.DispatcherServlet
2026-02-21T01:06:15.383-07:00  INFO 13987 --- [nio-8443-exec-7] o.s.web.servlet.DispatcherServlet
```

```java
package com.snhu.sslserver;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplicati
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.server.ResponseStatusException;
import org.springframework.web.servlet.view.RedirectView;

import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

@SpringBootApplication
public class SslServerApplication {

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

@RestController
class ServerController{
    private static final String HASH_ALGORITHM = "SHA-256";
    // Security constants for input validation
    private static final int MAX_INPUT_LENGTH = 10000; // Max 10K
    private static final int MAX_HASH_LENGTH = 64; // SHA-256 hex

    /**
     * Redirects root path to /hash endpoint
     * @return redirect to /hash
     */
    @GetMapping("/")
    public RedirectView redirectRoot() {
        return new RedirectView("/hash", true, false);
    }

    /**
     * Catch-all route that redirects any undefined paths to /has
     * @return redirect to /hash
     */
    @GetMapping("/**")
    public RedirectView redirectAllOtherRoutes() {
        return new RedirectView("/hash", true, false);
```

Figure 8. Application running after security enhancements

7. Summary

We updated the Artemis Financial app to include more security layers. These updates protect data, keep communications safe, and lower the risk of security issues. During this work, we used secure development methods to identify and mitigate security risks.

One important change was adding SHA-256 checksum checks to ensure data does not change during processing or transfer. This allows us to compare hash values with trusted checksums and catch any tampering or errors. We also set up HTTPS with a self-signed TLS certificate to encrypt data between clients and the server.

We added more defensive controls by using secure coding practices like input validation, limiting payload sizes, checking hash formats, and encoding HTML. These steps help prevent injection and cross-site scripting attacks, reducing risks while keeping the application reliable.
We tested security at every stage of development. The OWASP Dependency-Check tool helped us find and fix issues in our dependencies and made sure new code was safe. We also conducted manual tests to verify code accuracy, logic, and user input handling. Using both automated and manual checks helped us confirm our security upgrades.

By using encryption, secure configuration, dependency analysis, and defensive programming, the updated application now has a layered security model. These improvements meet industry standards by integrating security into every stage of development, thereby increasing the reliability and trust in the Artemis Financial system.

8. Industry Standard Best Practices

During the refactoring process, industry-standard secure coding practices were used to lower security risks and make the Artemis Financial application stronger. Instead of relying on a single security control, we added several layers of defense, following secure software development principles that emphasize defense-in-depth (Manico & Detlefsen, 2014).

To verify that the checksum is correct, we use SHA-256 to ensure it is generated in accordance with current cryptographic standards. SHA-256 is a recommended method for verifying the integrity of data because of its resistance to collision attacks and its acceptance by a large number of secure web systems. SHA-256 is significantly more secure than older hashing methods (e.g., MD5 and SHA1), which have many known vulnerabilities, and it is a fast and secure method for verifying the authenticity of data in production environments (Oracle Corporation, 2026).

This approach ensures that document integrity is always verified against a cryptographically secure hash at the point of ingestion. Using established cryptographic libraries for checksum generation reduces risks of implementation errors and enforces consistency with industry practices.

Next, HTTPS (TLS certificates) for secure communication. Encrypted data while in transit protects sensitive financial data from data interception and/or alteration. This is consistent with the OWASP Top 10, which cites cryptographic failures and insecure data transmission as two significant security risks and therefore calls for strong encryption controls (OWASP Foundation, n.d.). Implementing HTTPS with TLS also maps directly to information system and communications protection controls, such as NIST SP 800-53 control SC-13 (Cryptographic Protection). Referencing this formal control alignment provides additional traceability and reassures compliance reviewers of our audit readiness.

Third, secure input handling in the application code using sanitization. Input validation, limited payload sizes, and applied HTML encoding to lower the risk of common web vulnerabilities like injection attacks, denial-of-service, and cross-site scripting (XSS). Checking and cleaning input at the application boundary follows secure development guidelines and helps make sure untrusted data does not harm the system (Manico & Detlefsen, 2014).

Finally, dependency management best practices were followed through a clear stepwise process

1. We used the OWASP Dependency-Check tool to identify known vulnerabilities in third-party libraries.
2. We updated dependencies and frameworks to support secure versions when possible.
3. False positives were addressed by applying suppression rules only after manual verification.

This process aligns with recommended vulnerability management practices that emphasize ongoing monitoring and the responsible handling of software supply chain risks (OWASP Foundation, n.d.; National Institute of Standards and Technology, n.d.).

All of these steps show that we used recognized industry security standards throughout development. This means security controls were built into the software from the start, not added later.

References:

Dworkin, M. (2007). *Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC* (NIST Special Publication No. 800-38D). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-38D

Manico, J., & Detlefsen, A. (2014). *Iron-clad Java: Building secure web applications*. McGraw-Hill Education.

National Institute of Standards and Technology. (2001). *Announcing the Advanced Encryption Standard (AES)* (FIPS Publication No. 197). https://doi.org/10.6028/NIST.FIPS.197

National Institute of Standards and Technology. (n.d.). *National Vulnerability Database (NVD)*. https://nvd.nist.gov

Oracle Corporation. (2026). *Java Cryptography architecture (JCA) reference Guide*. https://docs.oracle.com/en/java/javase/21/security/java-cryptography-architecture-jca-reference-guide.html

Oracle Corporation. (n.d.). *Keytool: Key and certificate management tool*. https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

OWASP Foundation. (n.d.). *OWASP Dependency-Check*. https://owasp.org/www-project-dependency-check/

OWASP Foundation. (n.d.). *OWASP Top Ten Web Application Security Risks*. https://owasp.org/www-project-top-ten/