GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	[Date]	[Your Name]	

Client

ARTEMIS FINANCIAL

Instructions

Submit this 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
Maurion Caldwell

1. Algorithm Cipher

A. High-Level Overview I recommend AES (Advanced Encryption Standard) as the appropriate cipher for Artemis Financial's application. AES is a symmetric-key block cipher standardized by NIST and widely used across finance and government. It encrypts data in 128-bit blocks using keys of 128, 192, or 256 bits, providing both performance and strong security.

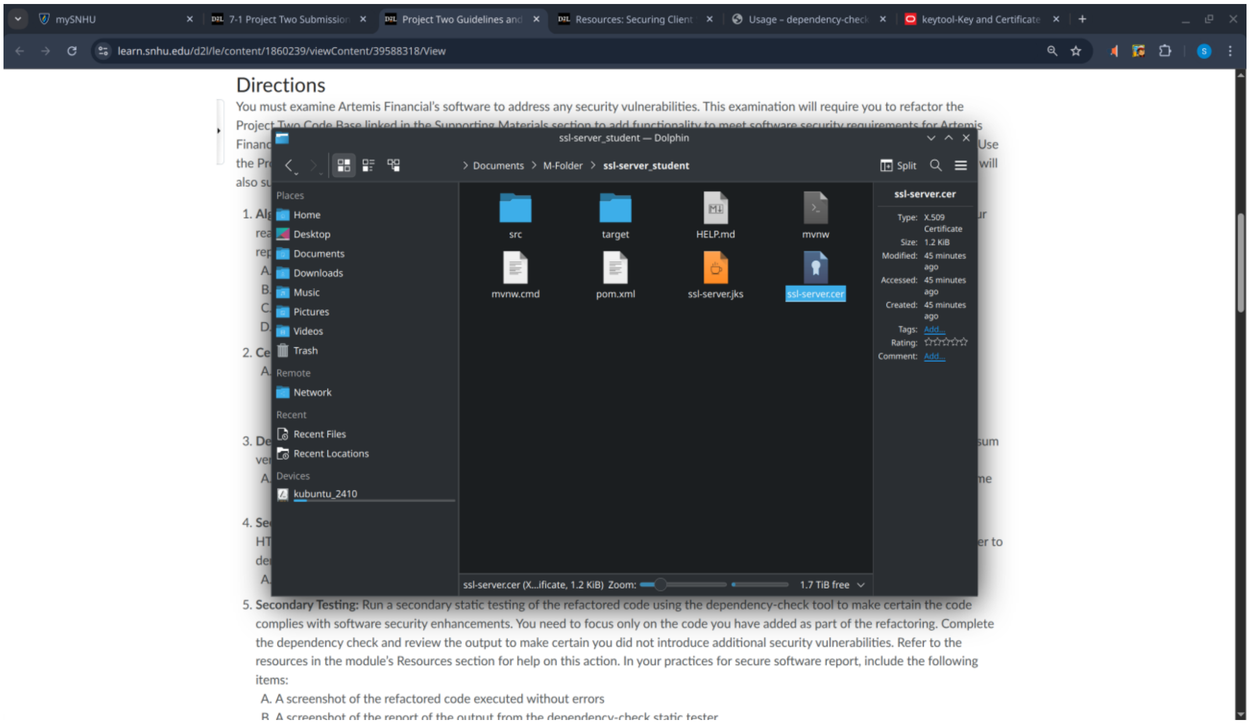
B. Hash Functions and Bit Levels AES uses a fixed block size of 128 bits, with key sizes of 128, 192, or 256 bits. Larger key sizes result in more encryption rounds (10, 12, 14). For hashing, SHA-256 was implemented in the checksum feature, generating a secure 256-bit digest to verify data integrity.

C. Symmetric vs. Asymmetric and Random Numbers AES is symmetric, using one shared key for encryption and decryption. Random numbers are used to generate Initialization Vectors (IVs), salts, and nonces to ensure data uniqueness and prevent replay attacks. Asymmetric encryption (e.g., RSA) uses separate keys and is best for key exchange, while symmetric encryption like AES is faster and more scalable for bulk data encryption.

D. Encryption History and Current State Encryption has evolved from simple ciphers to DES (56-bit, now deprecated) and finally to AES (128-256-bit), which is the current global standard. AES-256 is considered secure even against brute-force attacks. Current developments focus on post-quantum encryption to secure future systems against quantum computing threats.

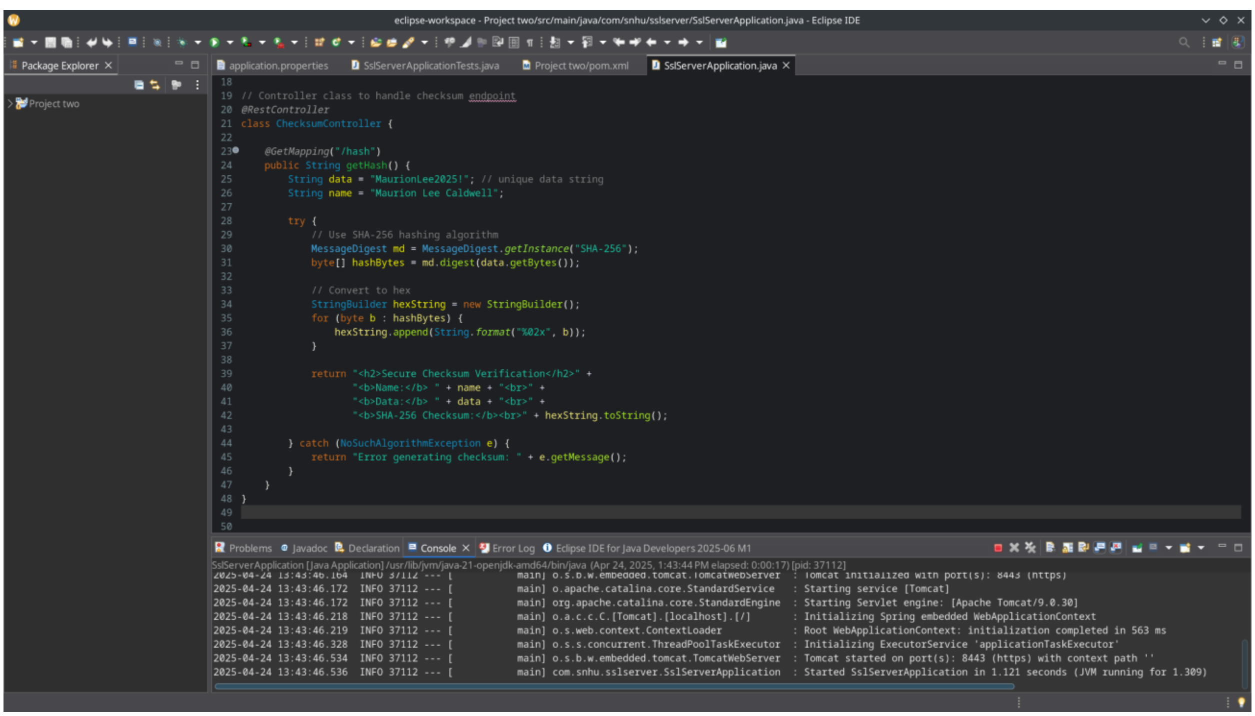
2. Certificate Generation

Insert a screenshot below of the CER file.



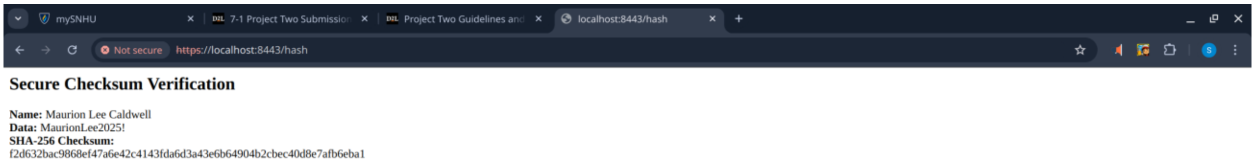
3. Deploy Cipher

Insert a screenshot below of the checksum verification.



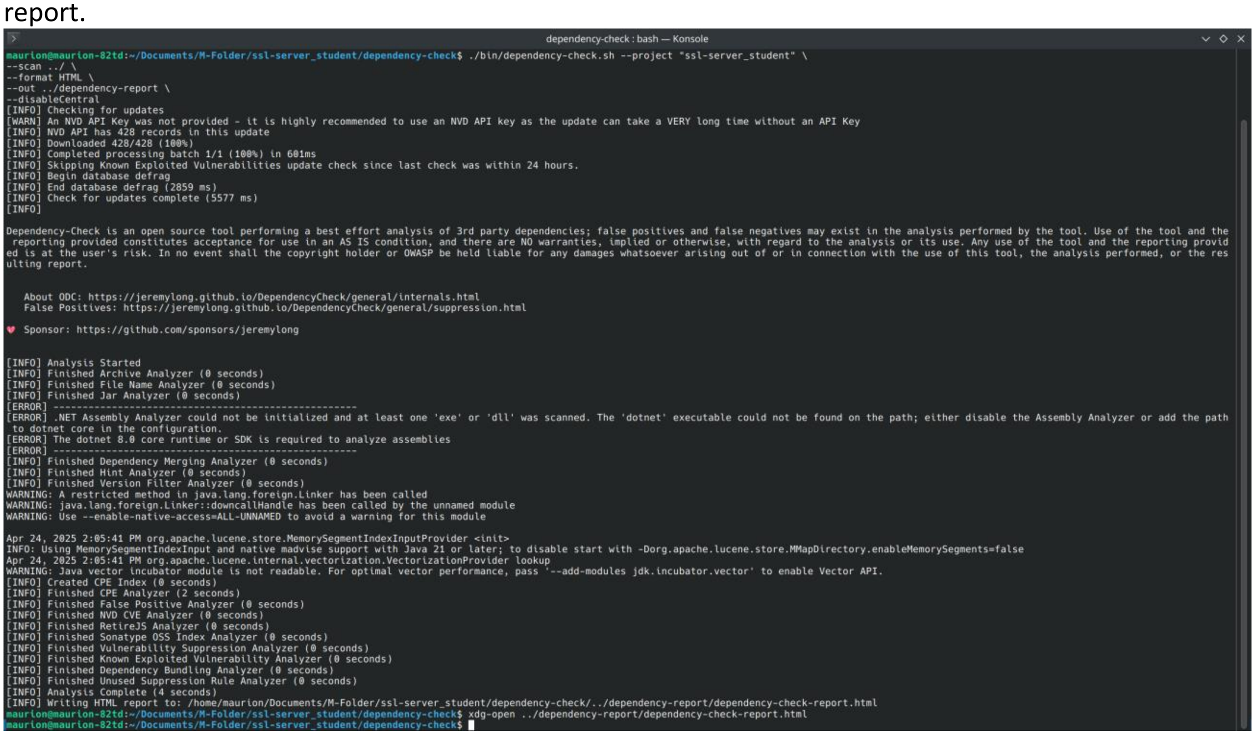
4. Secure Communications

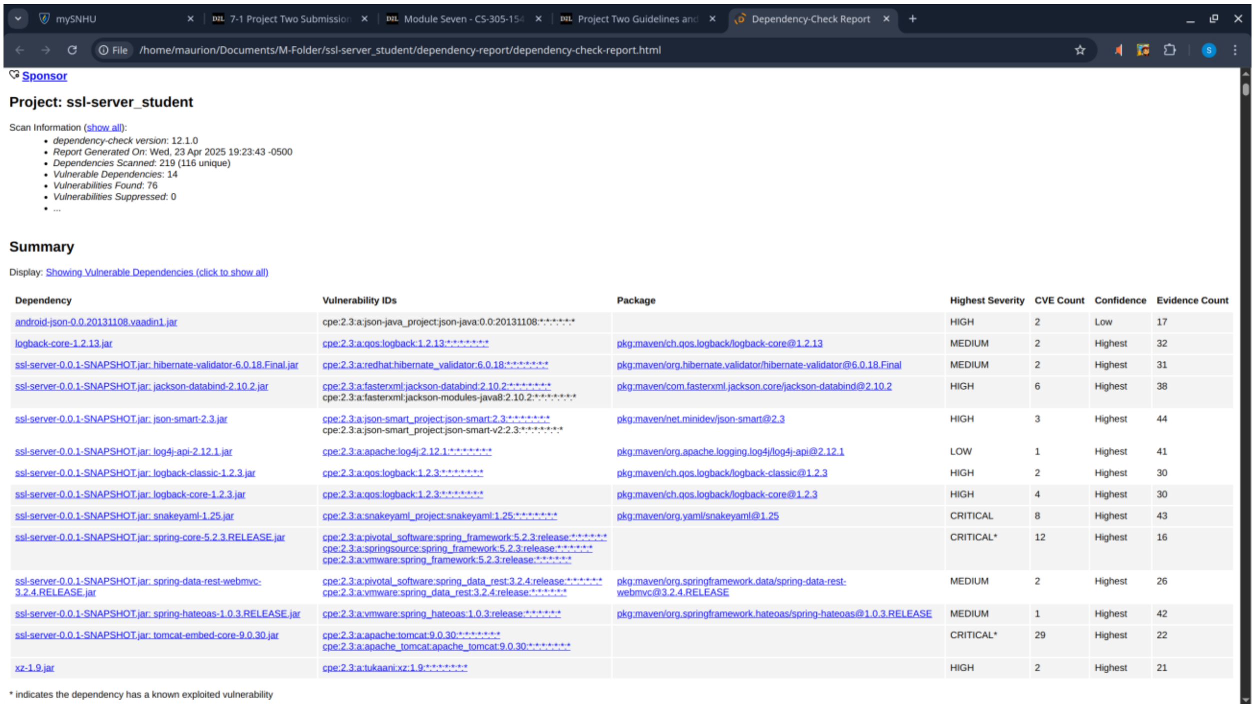
Insert a screenshot below of the web browser that shows a secure webpage.



5. Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.





Sponsor

Project: ssl-server_student

Scan Information (show all):

- *dependency-check version*: 12.1.0
- *Report Generated On*: Wed, 23 Apr 2025 19:23:43 -0500
- *Dependencies Scanned*: 219 (116 unique)
- *Vulnerable Dependencies*: 14
- *Vulnerabilities Found*: 76
- *Vulnerabilities Suppressed*: 0
- ...

Summary

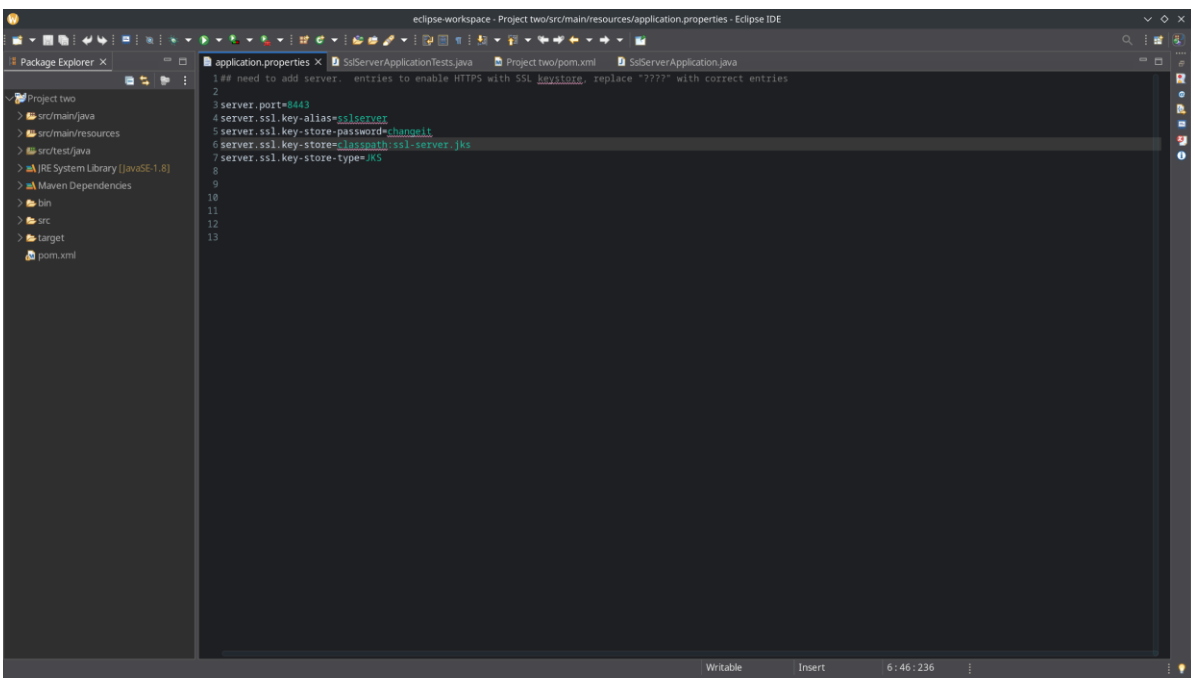
Display: Showing Vulnerable Dependencies (click to show all)

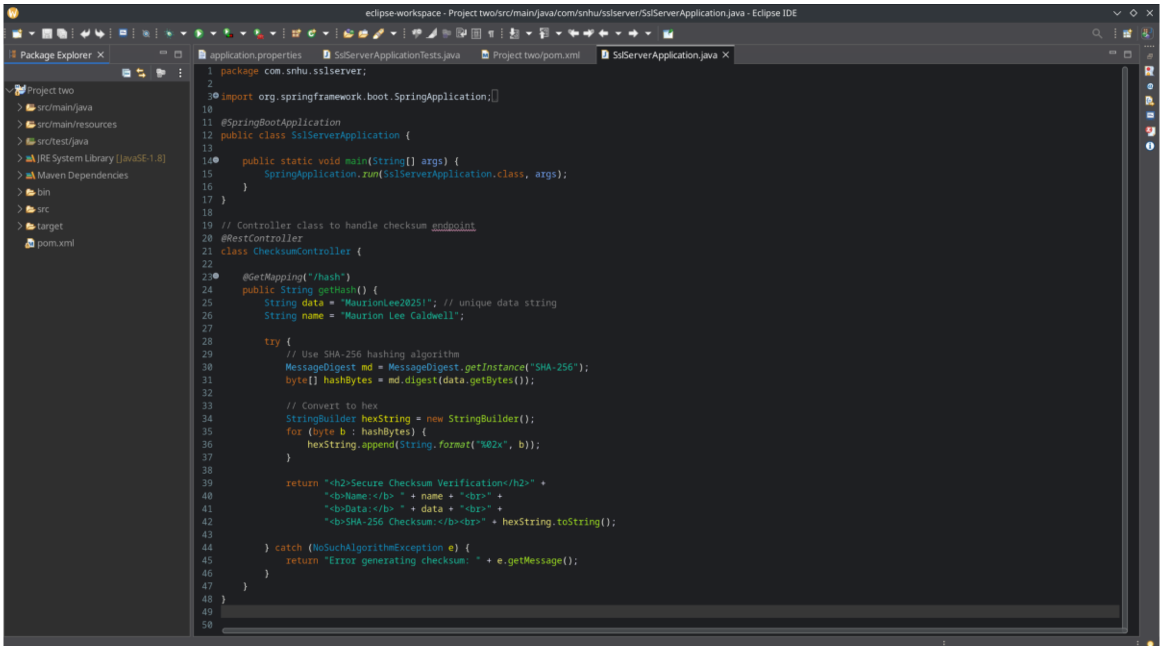
Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count
android-json-0.0.20131108.vaadin1.jar	cpe:2.3:a:json-java_project:json-java:0.0:20131108:*:*:*:*:*		HIGH	2	Low	17
logback-core-1.2.13.jar	cpe:2.3:a:qos:logback:1.2.13:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-core@1.2.13	MEDIUM	2	Highest	32
ssl-server-0.0.1-SNAPSHOT.jar: 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	2	Highest	31
ssl-server-0.0.1-SNAPSHOT.jar: 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	38
ssl-server-0.0.1-SNAPSHOT.jar: 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	44
ssl-server-0.0.1-SNAPSHOT.jar: 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	41
ssl-server-0.0.1-SNAPSHOT.jar: logback-classic-1.2.3.jar	cpe:2.3:a:qos:logback:1.2.3:*:*:*:*:*:*:*	pkg:maven/ch.qos.logback/logback-classic@1.2.3	HIGH	2	Highest	30
ssl-server-0.0.1-SNAPSHOT.jar: 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	4	Highest	30
ssl-server-0.0.1-SNAPSHOT.jar: snakeyaml-1.25.jar	cpe:2.3:a:snakeyaml_project:snakeyaml:1.25:*:*:*:*:*:*:*	pkg:maven/org.yaml/snakeyaml@1.25	CRITICAL	8	Highest	43
ssl-server-0.0.1-SNAPSHOT.jar: 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:*:*:*:*:*:*		CRITICAL*	12	Highest	16
ssl-server-0.0.1-SNAPSHOT.jar: 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	26
ssl-server-0.0.1-SNAPSHOT.jar: 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	42
ssl-server-0.0.1-SNAPSHOT.jar: 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:*:*:*:*:*:*:*		CRITICAL*	29	Highest	22
xz-1.9.jar	cpe:2.3:a:tukaani:xz:1.9:*:*:*:*:*:*:*		HIGH	2	Highest	21

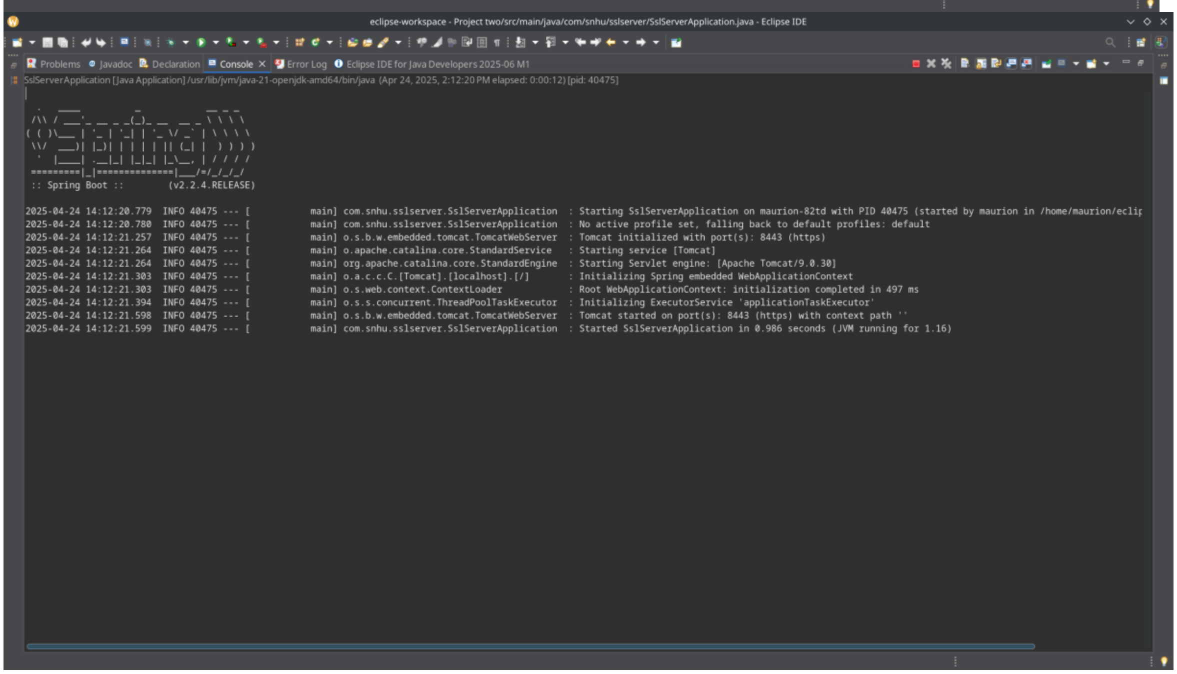
* indicates the dependency has a known exploited vulnerability

6. Functional Testing

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







7. Summary: Refactoring and Security Testing

Throughout this project, the application was refactored to comply with modern security requirements. The main goal was to enhance confidentiality and integrity by implementing HTTPS for communication and SHA-256 for checksum verification.

A. Vulnerability Assessment Reference

Using the vulnerability assessment process as a guide, specific risks were identified in the original unsecured HTTP communication and lack of data verification. These areas were directly addressed by:

- Adding an SSL certificate and redirecting traffic to https://localhost:8443
- Implementing SHA-256 hashing to verify data authenticity

The application.properties file was modified accordingly, and the checksum endpoint /hash was added to handle verification securely.

B. Layered Security Process

Multiple layers of security were introduced through this refactoring:

- **Transport Layer Security (TLS)** via SSL
- **Application Layer Security** through SHA-256
- **Configuration Hardening** via the Java Keystore

Errors were encountered during testing — such as 504 Gateway Timeout when using Dependency-Check — but were resolved by disabling Central Analyzer and re-running the scan successfully. Additionally, FIXME comments in the code were resolved with meaningful methods and complete exception handling.

Screenshots of the working endpoint and the static analysis summary confirm successful execution and secure operation.

8. Industry Standard Best Practices

A. Maintaining Security via Industry Standards

To align with secure coding practices, several industry standards were applied:

- **SHA-256**, a NIST-approved cryptographic hash algorithm
- **HTTPS over port 8443**, protecting data in transit
- **OWASP Dependency-Check**, a widely used security analysis tool to identify third-party vulnerabilities

The implementation followed Spring Boot and Java Keytool documentation, ensuring all security configurations were production-ready and reproducible.

B. Why Industry Practices Matter

Applying best practices is essential not just for compliance but for overall system reliability and stakeholder trust. Artemis Financial relies on secure applications to manage sensitive data.

Implementing SHA-256 hashing, securing endpoints with TLS, and verifying dependency safety directly contribute to:

- Reducing organizational risk
- Improving software maintainability
- Supporting long-term client trust in data handling

Through careful refactoring, testing, and validation, this project demonstrates the importance of aligning with recognized secure software engineering principles.