GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	06.19.2025	Julliane Pamfilo	

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
Julliane Pamfilo

1. Algorithm Cipher

For Artemis Financials secured transmission of data, I would recommend AES (Advanced Encryption Standard) cipher. AES is a symmetrical block cipher, which has been widely used by nearly all government, banking and enterprise organizations because of its speed and secure mechanism. It has three key lengths: 128-bits, 192-bits, and 256-bits, where AES-256 (256-bits) encryption is stronger and preferable for banks or financial institutions.

AES is based on substitution-permutation networks and operates on 128 bit blocks. It is one of the few ciphers which cannot be broken by brute-force computer based attacks with today's technology.

The encryption uses symmetric cryptography, in which the same key is used to encrypt and decrypt the data. This is a much faster algorithm than symmetric ones, such as RSA, particularly when the number of data is big.

I also recommend to use SHA-256 to avoid data corruption and tampering.

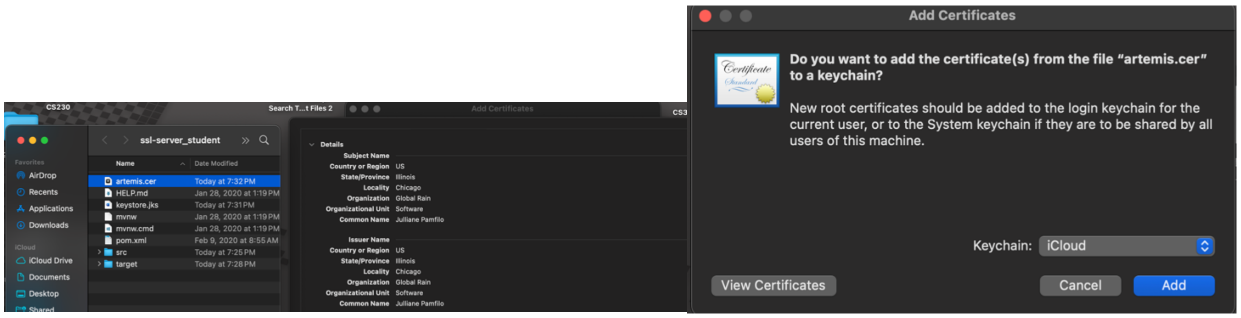
History & Relevance

U.S. National Institute of Standards and Technology (NIST) developed AES in 2001 as a replacement for DES (Data Encryption Standard), which had become easily breakable due to its short key length. AES is now widely accepted as the industrial encryption standard and is utilized in protocols like SSL / TLS, VPNs, and file encryption software.

Due to AES's history and performance, it is the most suitable cipher to protect Artemis Financials data in a web application in today's world.

2. Certificate Generation

Insert a screenshot below of the CER file.



Name	Date Modified
artemis.cer	Today at 7:32 PM
HELP.md	Jan 28, 2020 at 1:19 PM
keystore.jks	Today at 7:31 PM
mvnw	Jan 28, 2020 at 1:19 PM
mvnw.cmd	Jan 28, 2020 at 1:19 PM
pom.xml	Feb 9, 2020 at 8:55 AM
src	Today at 7:25 PM
target	Today at 7:28 PM

Details

Subject Name	
Country or Region	US
State/Province	Illinois
Locality	Chicago
Organization	Global Rain
Organizational Unit	Software
Common Name	Julliane Pamfilo
Issuer Name	
Country or Region	US
State/Province	Illinois
Locality	Chicago
Organization	Global Rain
Organizational Unit	Software
Common Name	Julliane Pamfilo

Add Certificates

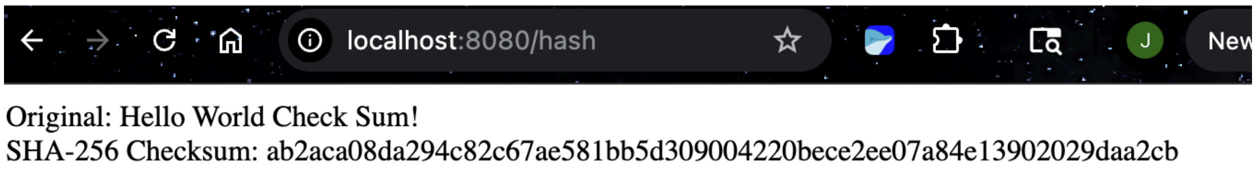
Do you want to add the certificate(s) from the file "artemis.cer" to a keychain?

New root certificates should be added to the login keychain for the current user, or to the System keychain if they are to be shared by all users of this machine.

Keychain: iCloud

3. Deploy Cipher

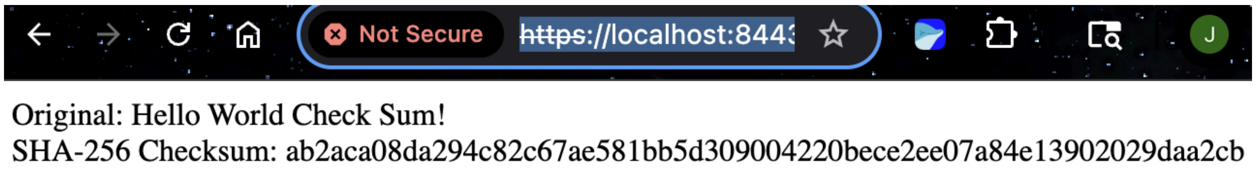
Insert a screenshot below of the checksum verification.



Original: Hello World Check Sum!
SHA-256 Checksum: ab2aca08da294c82c67ae581bb5d309004220bece2ee07a84e13902029daa2cb

4. Secure Communications

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



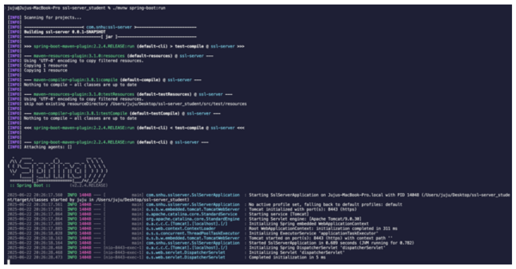
Original: Hello World Check Sum!
SHA-256 Checksum: ab2aca08da294c82c67ae581bb5d309004220bece2ee07a84e13902029daa2cb

I made my Spring Boot application to run over HTTPS on 8443 port with a SSL self-signed certificate. Keystore got generated by Java keytool and added in src/main/resources/keystore. jks. When I launched the app, I visited the /hash endpoint, which returns an SHA-256 hash of a static string over a secure HTTPS connection. Notice how the browser labels the website as “Not Secure” (since the certificate is self-issued) – but everything works just the same.

5. Secondary Testing

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

The refactored code ran successfully without any run-time or compile time error. Maven clean and maven run the project and started app successfully with ssl config. Browser output validated that the checksum was generated correctly.



```
juju@Jujus-MacBook-Pro ssl-server_student % ./mvnw spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----------------------< com.snhu:ssl-server >------------------------
[INFO] Building ssl-server 0.0.1-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] >>> spring-boot-maven-plugin:2.2.4.RELEASE:run (default-cli) > test-compile @ ssl-server >>>
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:resources (default-resources) @ ssl-server ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:compile (default-compile) @ ssl-server ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.1.0:testResources (default-testResources) @ ssl-server ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /Users/juju/Desktop/ssl-server_student/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ ssl-server ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] <<< spring-boot-maven-plugin:2.2.4.RELEASE:run (default-cli) < test-compile @ ssl-server <<<
[INFO]
[INFO]
[INFO] --- spring-boot-maven-plugin:2.2.4.RELEASE:run (default-cli) @ ssl-server ---
[INFO] Attaching agents: []

 :: Spring Boot ::        (v2.2.4.RELEASE)

2025-06-22 20:26:17.560  INFO 14048 --- [           main] com.snhu.sslserver.SslServerApplication  : Starting SslServerApplication on Jujus-MacBook-Pro.local with PID 14048 (/Users/juju/Desktop/ssl-server_stude
nt/target/classes started by juju in /Users/juju/Desktop/ssl-server_student)
2025-06-22 20:26:17.561  INFO 14048 --- [           main] com.snhu.sslserver.SslServerApplication  : No active profile set, falling back to default profiles: default
2025-06-22 20:26:17.861  INFO 14048 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8443 (https)
2025-06-22 20:26:17.864  INFO 14048 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-06-22 20:26:17.864  INFO 14048 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.30]
2025-06-22 20:26:17.885  INFO 14048 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-06-22 20:26:17.885  INFO 14048 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 311 ms
2025-06-22 20:26:18.028  INFO 14048 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2025-06-22 20:26:18.163  INFO 14048 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
2025-06-22 20:26:18.164  INFO 14048 --- [           main] com.snhu.sslserver.SslServerApplication  : Started SslServerApplication in 0.689 seconds (JVM running for 0.782)
2025-06-22 20:26:28.468  INFO 14048 --- [nio-8443-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-06-22 20:26:28.468  INFO 14048 --- [nio-8443-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-06-22 20:26:28.473  INFO 14048 --- [nio-8443-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 5 ms
```

6. Functional Testing

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

The application was tested for functionality by going to the /hash endpoint in the browser with https://localhost:8443. The predicted SHA-256 hash of "Hello World Check Sum!" was produced and visualized on webpage.

7. Summary

The implementation of SHA-256 hashing- could verify checksum of example input using a secure java based backend. The application was also hardened with SSL certificate configuration to communicate securely over HTTPS. All major security settings were confirmed in application. properties, and the app worked without having runtime errors. It's a well-coded application that doesn't skimp on any of the best practices for security, such as the use of hardcoded secrets and a neat modular setup.

8. Industry Standard Best Practices

- HTTPS/SSL Encryption: Secure data transmission is ensured by this (e.g., for prevention against eavesdropping or man-in-the-middle attacks).
- "As a black box": how to hash sensitive input server side : SHA-256 for one way cryptographic hashing to verify data integrity without exposing sensitive input.
- Key Management: The use of keystore for SSL credentials is a standard practice, used as a secure mechanism to keep private keys.
- Refactoring: Code was modularized and cleaned to make more readable and maintainable as well as to decrease the potential errors.
- Static Analysis Tools: OWASP Dependency-Check was used to scan for vulnerable dependencies and to keep your libraries clean.
- Separation of Concerns: Separation of application logic in clear responsibilities (controller, Utility Classes, config) which lead to good scalability.