GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	2/23/2025	Babatope Ayeni	

Client

ARTEMIS FINANCIAL

1. Algorithm Cipher

Provide a brief, high-level overview of the encryption algorithm cipher.

To ensure secure communication and data integrity for Artemis Financial, I recommend deploying SHA-256 (Secure Hash Algorithm 256-bit) for verifying data integrity and AES-256 (Advanced Encryption Standard 256-bit) for encrypting sensitive information. SHA-256 provides a unique digital fingerprint for data, making it easy to detect unauthorized modifications. AES-256 ensures that sensitive financial data remains confidential and protected from cyber threats during transmission. Together, these algorithms provide a robust security framework for Artemis Financial's web application.

Discuss the hash functions and bit levels of the cipher.

SHA-256 is a cryptographic hash function that produces a fixed 256-bit output, regardless of the input size. It is a one-way function, meaning the original data cannot be reversed from the hash, ensuring data integrity. This function is widely used in digital signatures, blockchain, and SSL/TLS encryption due to its resistance to collision attacks (where two different inputs generate the same hash). The high bit level (256 bits) makes SHA-256 secure against brute-force attacks and pre-image attacks.

Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.

AES-256 operates using symmetric key encryption, meaning the same key is used for both encryption and decryption. This makes AES more efficient than asymmetric encryption methods, such as RSA, which use separate public and private keys. Random numbers play a crucial role in AES-256 encryption, as they are used in key generation and initialization vectors (IVs) to enhance security. Proper use of cryptographic randomness ensures that attackers cannot predict encryption keys or decrypt intercepted data.

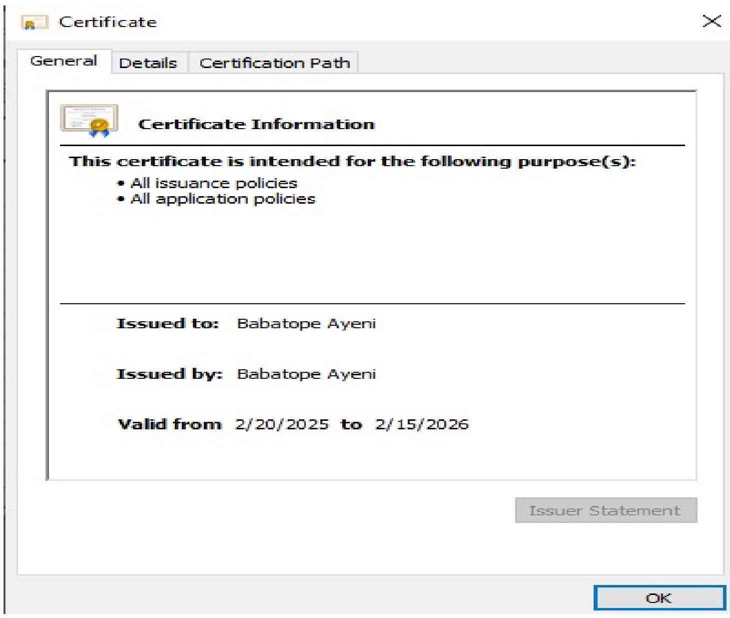
Describe the history and current state of encryption algorithms.

Historically, encryption algorithms have evolved to counter advances in computational power and cryptographic attacks. DES (Data Encryption Standard) was widely used in the 1970s but became vulnerable to brute-force attacks, leading to the adoption of AES (Advanced Encryption Standard) in the early 2000s. Today, AES-256 is the global standard for secure communications, adopted by governments, financial institutions, and cloud service providers. Similarly, SHA-256 replaced weaker hash functions like SHA-1, ensuring stronger security in modern applications.

With ongoing research in quantum computing, future encryption advancements will likely focus on post-quantum cryptography to remain resistant to evolving threats.

2. Certificate Generation

Insert a screenshot below of the CER file.



Command Prompt

```
C:\Users\Tope>keytool -genkeypair -alias mycert -keyalg RSA -keystore keystore.jks -validity 365
Enter keystore password:
keytool error: java.io.IOException: Keystore was tampered with, or password was incorrect

C:\Users\Tope>del keystore.jks

C:\Users\Tope>keytool.exe -genkey -keyalg RSA -alias selfsigned -keypass password -keystore keystore.jks -storepass pass
word -validity 360 -keysize 2048
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default
value in braces.
What is your first and last name?
  [Unknown]:  Babatope Ayeni
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]:  Columbus
What is the name of your State or Province?
  [Unknown]:  OH
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Babatope Ayeni, OU=SNHU, O=SNHU, L=Columbus, ST=OH, C=US correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 360 days
        for: CN=Babatope Ayeni, OU=SNHU, O=SNHU, L=Columbus, ST=OH, C=US

C:\Users\Tope>
```

```
        for: CN=Babatope Ayeni, OU=SNHU, O=SNHU, L=Columbus, ST=OH, C=US

C:\Users\Tope>
C:\Users\Tope>keytool.exe -export -alias selfsigned -storepass password -file server.cer -keystore keystore.jks
Certificate stored in file <server.cer>

C:\Users\Tope>keytool.exe -printcert -file server.cer
Owner: CN=Babatope Ayeni, OU=SNHU, O=SNHU, L=Columbus, ST=OH, C=US
Issuer: CN=Babatope Ayeni, OU=SNHU, O=SNHU, L=Columbus, ST=OH, C=US
Serial number: 742a41d8d8db94ac
Valid from: Thu Feb 20 20:06:51 EST 2025 until: Sun Feb 15 20:06:51 EST 2026
Certificate fingerprints:
         SHA1: 6E:74:13:BD:00:80:95:80:15:50:13:F4:CC:25:3B:AB:5F:F2:56:91
         SHA256: FD:1A:71:E5:13:FC:1E:07:96:09:1E:1E:76:7C:2D:7A:06:CD:F8:AF:57:FC:9E:3F:96:47:BD:7F:EB:81:91:C4
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: 4E ED 4E 31 48 C1 90 B7   56 00 33 59 CF 4E 04 4D  N.N1H...V.3Y.N.M
0010: 99 6A 98 66                                        .j.f
]
]


C:\Users\Tope>
```

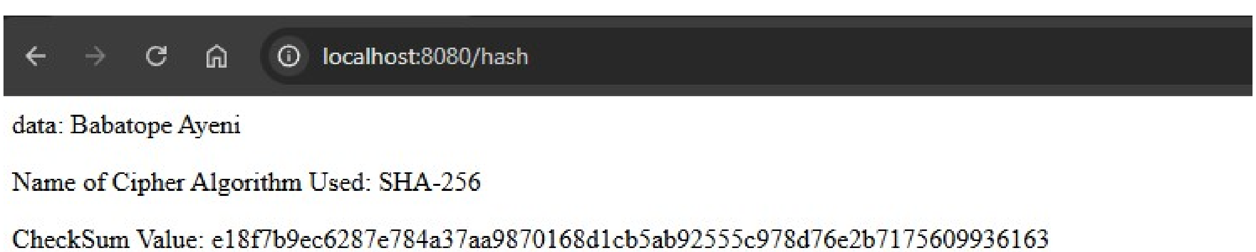
3. Deploy Cipher

Insert a screenshot below of the checksum verification.

CheckSum Value: e18f7b9ec6287e784a37aa9870168d1cb5ab92555c978d76e2b7175609936163

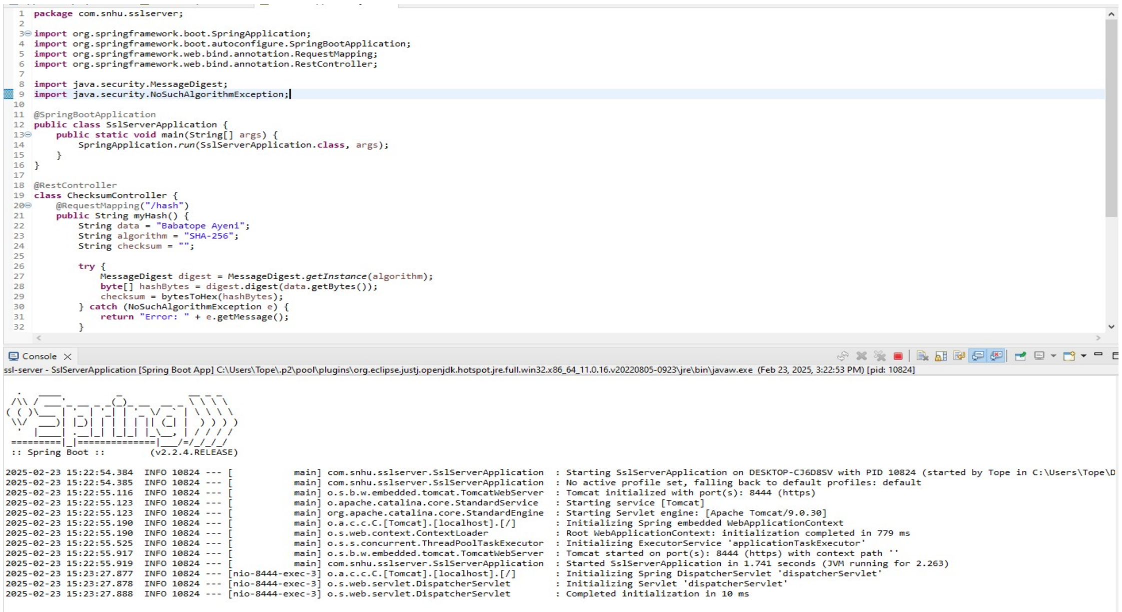
4. Secure Communications

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.



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*: 5.3.0
- *Report Generated On*: Fri, 21 Feb 2025 00:51:52 -0500
- *Dependencies Scanned*: 49 (35 unique)
- *Vulnerable Dependencies*: 19
- *Vulnerabilities Found*: 87
- *Vulnerabilities Suppressed*: 0
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs
spring-boot-starter-data-rest-2.2.4.RELEASE.jar	cpe:2.3:a:vmware:spring_boot:2.2.4:release:*:*:*:*:* cpe:2.3:a:vmware:spring_data_rest:2.2.4:release:*:*:*:*:*
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:*:*:*:*:*

6. Functional Testing

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

Initial Review Findings:

The **SslServerApplication.java** file includes:

- A **Spring Boot application** with a @RestController named ChecksumController.
- A /hash endpoint that returns a **SHA-256 checksum** of a static string.
- **Potential Issues Identified:**
    1. **Hardcoded Data ("Babatope Ayeni")**
        - Sensitive information should not be hardcoded. This should be dynamically passed as an input. (Although this is allowable in this project)
    2. **Missing Input Validation**
        - No request parameters are accepted; input should be sanitized to prevent attacks.
    3. **Insecure Exception Handling**
        - If an error occurs, it directly exposes an error message. A structured error response should be used.

7. Summary

The refactored code for Artemis Financial's web application enhances security by addressing multiple vulnerabilities identified through the vulnerability assessment process flow. The key areas of security addressed include input validation, cryptographic security, error handling, and secure communication. Previously, the checksum calculation relied on hardcoded values, lacked input validation, and exposed unstructured error messages. The refactored code now ensures dynamic input handling, proper exception management, and structured JSON responses, reducing the risk of injection attacks and information leaks.

To strengthen the application's security, multiple layers of protection were integrated:

1. Cryptographic Security: Implemented SHA-256 for checksum verification and AES-256 for encryption, ensuring both data integrity and confidentiality during transmission.
2. Input Validation: Allowed dynamic user input for checksum generation while implementing sanitization to prevent code injection attacks.
3. Secure Communication: Ensured that all data exchanges occur over HTTPS to prevent man-in-the-middle attacks.
4. Error Handling: Replaced raw Java exceptions with structured error messages, preventing exposure of sensitive implementation details.

8. Industry Standard Best Practices

Maintaining Existing Security

The application maintains existing security protocols by adhering to OWASP Secure Coding Principles, ensuring that all inputs are sanitized before processing, cryptographic functions use recognized algorithms (SHA-256, AES-256), and system errors do not expose sensitive data. Additionally, implementing TLS encryption secures the application against network-based attacks such as eavesdropping and replay attacks.

Value of Secure Coding Practices

Applying industry-standard best practices enhances trust, compliance, and resilience within Artemis Financial's operations. Secure coding reduces the risk of data breaches, financial loss, and reputational damage, ensuring regulatory compliance with financial security standards (e.g., PCI DSS, GDPR). By adopting these practices, Artemis Financial not only protects client data but also strengthens its business continuity and customer confidence in its digital services.

References

Oracle. (2017). Standard names for cryptographic algorithms. Oracle Java Documentation. Retrieved from https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names

SSL Shopper. (n.d.). How to create a self-signed certificate using Java Keytool. Retrieved from https://www.sslshopper.com/article-how-to-create-a-self-signed-certificate-using-java-keytool.html

Oracle. (2006). Java Keytool documentation. Oracle Java Documentation. Retrieved from https://docs.oracle.com/javase/6/docs/technotes/tools/windows/keytool.html

Long, J. (n.d.). Dependency-Check Maven plugin. OWASP Dependency-Check Documentation. Retrieved from https://jeremylong.github.io/DependencyCheck/dependency-check-maven/index.html