Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	08-17-25	Elizabeth Marticello	Initial Report

Client

Developer
Elizabeth Marticello

1. Algorithm Cipher

Overview

Artemis Financial wants to use the most current and effective software security with its public web interface to protect its client data and financial information. They specifically want to add a file verification step to its web application to ensure secure communication.
I recommend using the cryptographic hash function SHA-256 to ensure data integrity. This cryptographic algorithm can generate a unique fixed-size checksum based on input data. This can be compared before and after transmission or storage to detect errors or tampering. SHA-256 is widely used due to its resistance against collision attacks, where two different inputs produce the same checksum. This algorithm protects user data, like their financial information, against corruption and unauthorized modification (GeeksForGeeks, 2024).

In addition to integrity checks, Artemis Financial also needs to protect the confidentiality of its data. For this I recommend deploying the symmetric encryption algorithm AES-256. AES-256 uses 256-bit keys and processes data in a fixed-size 128-bit blocks, offering a strong balance of performance and security. Unlike hash functions, AES-256 is a cipher that can both encrypt and decrypt data, ensuring only authorized parties can view the information. Its large key size provides resistance against brute force attacks. Together, SHA-256 for verification and AES-256 for encryption create a layered approach to security, addressing both tamper detection and data protection.

Hash Functions and Bit Levels

SHA-256 and AES-256 are both essential to modernizing Artemis Financial's operations but serve very different purposes. SHA-256 is a cryptographic hash function that generates a fixed 256-bit hash output regardless of the input size. It does not use a key, making it a one-way function ideal for data integrity verification (MojoAuth, n.d.-a).

In contrast, AES-256 is a symmetric encryption algorithm that relies on a 256-bit key to encrypt and decrypt data, with its output being ciphertext that matches the size of the original plaintext blocks. While SHA-256 is typically used for checksums, signatures, and ensuring data integrity, AES-256 is designed for confidentiality, securing sensitive data during storage or transmission. Together they provide complementary protection (MojoAuth, n.d.-b).

Random Numbers, Symmetric vs. Non-Symmetric Keys

Random numbers play a critical role in AES encryption by ensuring that generated keys are unpredictable and secure. AES is a symmetric cipher, meaning the same key is used for both encryption and decryption. This approach differs from asymmetric encryption, such as RSA, which uses a public and private key pair. Because symmetric encryption relies on one shared key, strong randomness is essential. If the key is compromised, both the encrypted data and the ability to decrypt it are exposed. By using cryptographically secure random number generators, AES keys can be made unguessable and robust enough to protect Artemis Financial's sensitive client data (Manico & Detlefsen, 2014).

History and Current State of Encryption Algorithms

Some of the earliest encryption methods date back to ancient Sparta, where scrambled letter arrangements protected military communications. During World War I, similar techniques were used to hide military telegram codes. In the 1970s, the Data Encryption Standard (DES) became the first cryptosystem officially used by the U.S. government. However, DES was eventually replaced in 2001 by AES due to its much longer key lengths and stronger resistance to modern hardware attacks (Schneider, n.d.).

Given its proven history, robust key sizes, reliance on strong random numbers, and efficiency in symmetric encryption, AES is the most secure and appropriate cipher for Artemis Financial's long-term file encryption needs.

2. Certificate Generation

Screenshots of generated self-signed certificates using the Java Keytool.

```
Administrator: Command Prompt

Generating 4,096 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 365 days
        for: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -export -alias CS305_Project2 -storepass SNHU_CS305_
7.1 -file server.cer -keystore keystore.jks
Certificate stored in file <server.cer>

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -printcert -file server.cer
Owner: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US
Issuer: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US
Serial number: ae7ebc697603bf0e
Valid from: Fri Aug 22 16:23:06 EDT 2025 until: Sat Aug 22 16:23:06 EDT 2026
Certificate fingerprints:
         SHA1: 7C:69:47:D9:0A:6A:6E:9E:98:29:D0:54:33:F8:D2:ED:6A:AE:D1:E1
         SHA256: 21:9F:5D:B8:F0:2B:63:94:A3:59:84:91:BA:D9:1D:0A:74:91:E5:87:6B:71:A5:59:5C:35:3F:6B:6C:96:31:62
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: DA 92 FA 1F B5 81 06 17   0B 3D 40 CA F5 32 79 B5  .........=@..2y.
0010: 92 34 AF DB                                        .4..
]
]


C:\Windows\system32>
```

```
Administrator: Command Prompt

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -genkeypair -alias CS305_Project2.1 -keyalg RSA -key
size 4096 -sigalg SHA256withRSA -dname "CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, S=NY, C=US" -ext SAN=DNS
:localhost,IP:127.0.0.1 -keystore keystore.jks -storepass SNHU_CS305_7.1 -keypass SNHU_CS305_7.1 -validity 365
Generating 4,096 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 365 days
        for: CN=Elizabeth Marticello, OU=CS305, O=SNHU, L=Syracuse, ST=NY, C=US

C:\Windows\system32>"C:\Program Files\Java\jdk-21\bin\keytool.exe" -export -alias CS305_Project2.1 -file server.cer -ke
ystore keystore.jks -storepass SNHU_CS305_7.1
Certificate stored in file <server.cer>

C:\Windows\system32>
```

3. Deploy Cipher

Screenshot showing the checksum verification.

Not secure ~~https~~://localhost:8443/hash

data:Hello Liz Marticello!

algorithm: SHA-256

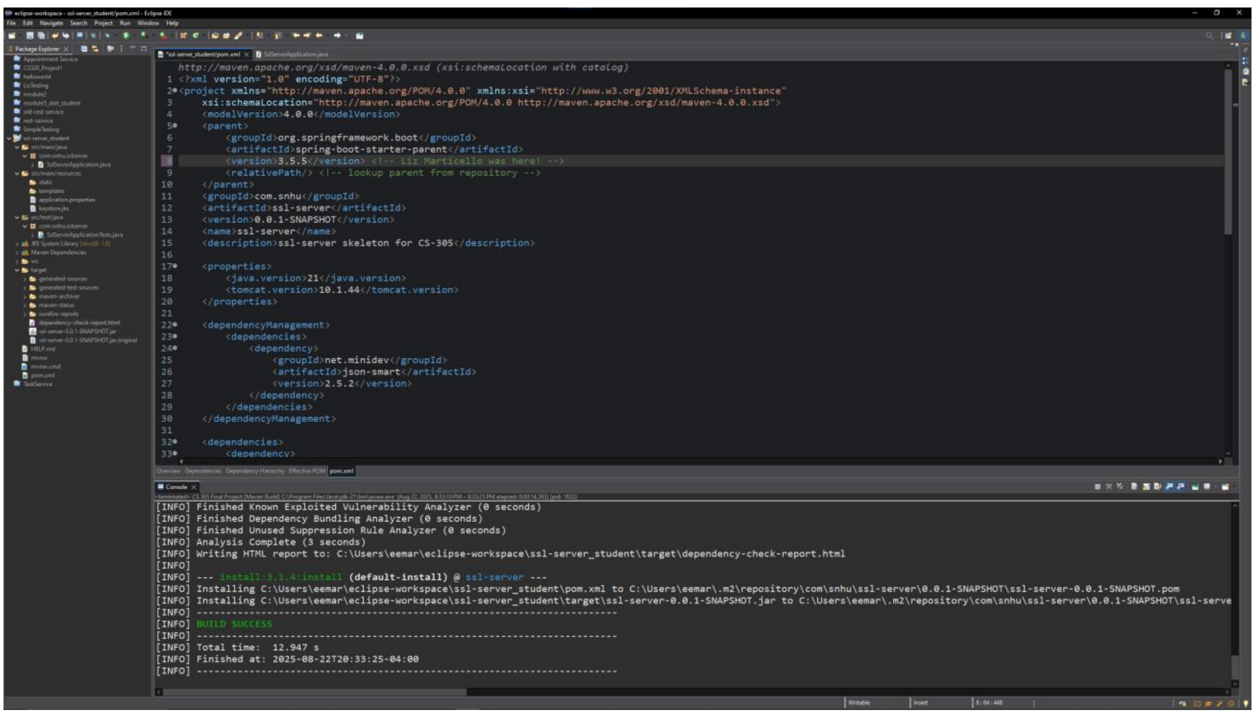
checksum: f9105d3f270abdadf3a5416de7d65fbaac899d0aa92371600ccb5804611289f1

4. Secure Communications

Screenshot of the web browser showing a secure webpage.

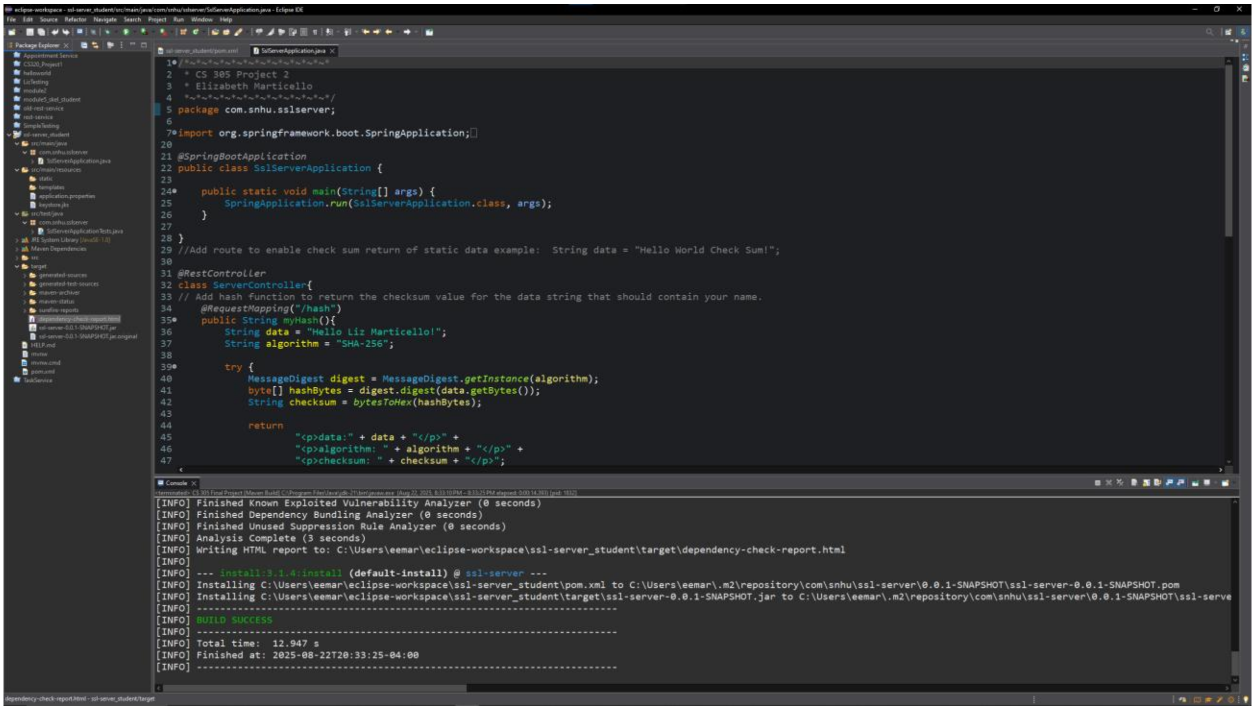
5. Secondary Testing

Screenshots of the refactored code executed without errors and the dependency-check report.



6. **Functional Testing**

Screenshot showing refactored code executed without errors.



7. **Summary**

For Artemis Financial's modernization initiative, the refactored codebase was updated to meet current security requirements by focusing on data confidentiality, integrity, and secure communications. Guided by the vulnerability assessment process flow diagram the application was enhanced with SHA-256 hashing to verify file integrity and AES-256 encryption to protect sensitive client and financial data. Secured communications were ensured by implementing HTTPS by generating and deploying a self-signed certificate. Doing this secures all client-server communication and prevents data interception. Input validation and secure API interactions were reviewed to reduce risks of injection or malformed data and error handling was improved to avoid exposing sensitive information. Static testing with dependency-check tools ensured no new vulnerabilities were introduced during refactoring. Functional testing verified that all features, including checksum verification worked as expected. Using both SHA-256 and AES-256

8. **Industry Standard Best Practices**

Industry Standard best practices, including OWASP guidelines and secure SDLC principles, were carefully applied to preserve the application's existing security while modernizing its overall security. Input validation, parameterized queries, and secure error handling were used to block common attacks like injection. Sensitive data such as encryption keys and credentials were removed from the code and stored securely, and HTTPS was enabled to protect data in transit. Exiting access controls, roles, and sessions were kept in place and tested to make sure no new vulnerabilities were added.

These practices make Artemis Financial stronger by lowering the chance of data breaches and building client trust. They also support compliance with common compliance requirements, help avoid costly remediation, and improve the system's reliability and maintainability. Overall, using secure coding practices keeps the platform safe, protecting both company's reputation and its sensitive financial data.

References

(2024, May 28). *Understanding Checksum Algorithm for Data Integrity*. GeeksForGeeks. https://www.geeksforgeeks.org/system-design/understanding-checksum-algorithm-for-data-integrity/

MojoAuth (n.d.-a). *SHA-256 vs SHA3-256*. MojoAuth. https://mojoauth.com/compare-hashing-algorithms/sha-256-vs-sha3-256/

MojoAuth (n.d.-b). *AES-256 Encryption : Java*. https://mojoauth.com/encryption-decryption/aes-256-encryption--java/

Manico, J., & Detlefsen, A. (2014). *Iron-Clad Jaca: Building Secure Web Applications*. McGraw Hill Computing. https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/?sso_link=yes&sso_link_from=SNHU

Schneider, J. (n.d.). *A brief history of cryptography: Sending secret messages throughout time*. https://www.ibm.com/think/topics/cryptography-history