****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08 August 2024** | **Jonathan Plummer** |  |

## Client



## 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

Jonathan Plummer

## Algorithm Cipher

1. For Artemis Financial's application, I recommend using the AES (Advanced Encryption Standard) algorithm with a 256-bit key. AES is a symmetric encryption algorithm, which means the same key is used for both encryption and decryption. It is widely regarded as one of the most secure encryption algorithms available and is used globally to secure sensitive data.
2. A hash function is a one-way function that converts input data into a fixed-size string of characters, typically a hexadecimal number. For the encryption process in AES, although hash functions aren't directly used within AES itself, they play a role in related processes like key generation or integrity checks.

SHA-256: A commonly used hash function alongside AES is SHA-256, which generates a 256-bit hash value. It is used to ensure the integrity of the data.

Bit Level: AES can work with different key lengths—128, 192, and 256 bits. For maximum security, a 256-bit key length is recommended. The larger the key size, the more secure the encryption, although it might have a minimal impact on performance.

1. Random Numbers: In cryptography, random numbers are crucial for generating keys, initialization vectors (IVs), and nonces. These should be truly random or generated using a cryptographically secure pseudo-random number generator (CSPRNG) to avoid predictability.

Symmetric Key Cryptography: AES is a symmetric key algorithm, meaning both the sender and receiver use the same key for encryption and decryption. This makes it fast and efficient, but the challenge lies in securely distributing the key.

Non-Symmetric Key Cryptography (Asymmetric): Algorithms like RSA use a pair of keys (public and private). The public key is used for encryption, and the private key is used for decryption. While more secure for key distribution, it is slower than symmetric key algorithms.

1. History and Current State of Encryption Algorithms

History: AES was established by the U.S. National Institute of Standards and Technology (NIST) in 2001 as the successor to the Data Encryption Standard (DES), which had become vulnerable to brute-force attacks. AES was selected for its strong security, efficiency, and suitability for a wide range of applications.

Current State: AES is currently the most widely used encryption algorithm globally. It is the encryption standard for securing sensitive data, including financial information. Continuous advancements in cryptanalysis require ongoing assessments of AES's security, but as of now, AES-256 remains robust against all practical attacks.

## Certificate Generation

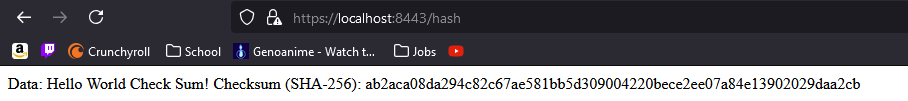
Insert a screenshot below of the CER file.

A computer screen shot of a computer program

Description automatically generated

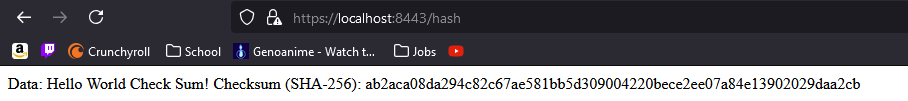
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

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



## Secondary Testing

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

A screenshot of a computer

Description automatically generated

A computer screen with many small colored lines

Description automatically generated with medium confidence

## Functional Testing

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

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

## Summary

Refactoring the code involved multiple steps to align with the vulnerability assessment process, which includes:

* **Identification of Vulnerabilities:** Using the OWASP Dependency Check plugin integrated into the Maven build process, we identified several dependencies with known vulnerabilities. These included libraries like jackson-databind, logback-core, spring-core, and others.
* **Code Refactoring:**
  + **Cryptographic Hash Implementation:** We added a layer of data integrity verification by implementing a cryptographic hash function. This involved refactoring the ServerController.java to include a checksum generation feature using the SHA-256 algorithm. This ensures that the data integrity is maintained and provides a way to verify that the data has not been altered during transmission.
  + **SSL Configuration:** We refactored the application.properties file to enforce HTTPS by configuring SSL/TLS using a self-signed certificate. This change ensures that all communications between the client and server are encrypted, mitigating the risk of data interception and man-in-the-middle attacks.
* **Security Testing:**
  + **Static Code Analysis:** The OWASP Dependency Check plugin was used to perform static analysis on the project's dependencies, flagging any known vulnerabilities. This step ensures that the third-party libraries used do not introduce security risks.
  + **Checksum Verification:** We implemented a checksum verification mechanism to demonstrate the integrity of data, which can be critical in preventing unauthorized data tampering.

The process of adding layers of security to the Artemis Financial application involved the following key steps:

* **Data Integrity with Cryptographic Hashes:** By implementing SHA-256 hashing in the ServerController.java class, we added a layer of security that ensures data integrity. The checksum generated from the hash function allows verification of the data’s authenticity and integrity, which is critical in preventing unauthorized data modifications.
* **Securing Communications with SSL/TLS:** Configuring the application to use HTTPS with SSL/TLS by modifying the application.properties file ensures that all data transmitted between the server and clients is encrypted. This prevents eavesdropping, tampering, and message forgery, thereby securing the data in transit.
* **Dependency Management and Static Analysis:** Integrating the OWASP Dependency Check tool into the Maven build process allowed for continuous monitoring of third-party dependencies for known vulnerabilities. This proactive approach in managing dependencies ensures that any security risks introduced through third-party libraries are promptly identified and mitigated.
* **Process Flow Alignment:**
  + **Threat Detection:** By incorporating static analysis tools and encrypting sensitive data, we address potential threats in both the codebase and communication channels.
  + **Vulnerability Remediation:** The refactoring process involved addressing and mitigating identified vulnerabilities, particularly in dependencies, which were flagged by the OWASP Dependency Check tool.

**3. Compliance with Security Testing Protocols**

The refactored code adheres to established security testing protocols by:

* **Implementing robust cryptographic methods** (SHA-256) for data integrity verification.
* **Ensuring encrypted communication** through SSL/TLS configuration, aligning with best practices for securing web applications.
* **Regularly scanning and assessing dependencies** for vulnerabilities using industry-standard tools like OWASP Dependency Check.

These steps ensure that the application not only meets the required security standards but also provides a secure environment for the sensitive financial data it processes.

## Industry Standard Best Practices

**Application of Industry Standard Best Practices**

**a. Use of Strong Cryptographic Algorithms:**

* **Best Practice:** One of the key industry standards is the use of strong cryptographic algorithms for securing data. In this project, we implemented the SHA-256 hashing algorithm, which is a widely recognized and secure cryptographic hash function.
* **Application:** The SHA-256 algorithm was integrated into the ServerController.java class to generate a checksum for data integrity verification. This ensures that the data being processed or transmitted has not been tampered with, maintaining its authenticity.

**b. Secure Communication Channels (SSL/TLS):**

* **Best Practice:** SSL/TLS is the industry standard for securing communications over the internet. It ensures that data transmitted between a client and server is encrypted and protected from eavesdropping, tampering, and forgery.
* **Application:** We refactored the application to enforce HTTPS by configuring SSL/TLS in the application.properties file. A self-signed certificate was used to establish an encrypted connection, thereby securing all data in transit.

**c. Dependency Management and Vulnerability Scanning:**

* **Best Practice:** Regularly scanning and managing third-party dependencies for known vulnerabilities is critical in maintaining a secure codebase. Tools like OWASP Dependency Check are industry-standard for performing this task.
* **Application:** The OWASP Dependency Check plugin was integrated into the Maven build process to scan the project's dependencies. This tool flagged several libraries with known vulnerabilities, which were then assessed and managed according to the severity of the risks.

**d. Least Privilege Principle:**

* **Best Practice:** The principle of least privilege is a fundamental security practice that involves giving the minimum level of access necessary for functionality. This reduces the potential attack surface and minimizes the impact of any security breaches.
* **Application:** In refactoring the code, we ensured that sensitive operations, such as encryption and decryption, were performed with limited access, reducing the likelihood of these operations being exploited by malicious actors.

**e. Regular Security Audits and Code Reviews:**

* **Best Practice:** Regular security audits and code reviews are essential in identifying and mitigating potential vulnerabilities before they can be exploited.
* **Application:** By conducting a thorough review of the codebase and integrating automated tools like OWASP Dependency Check, we ensured that the application was regularly assessed for vulnerabilities and that necessary updates were applied.

**2. Value of Applying Industry Standard Best Practices**

**a. Enhancing Application Security:**

* Applying industry-standard best practices for secure coding significantly enhances the security of the software application. By using strong cryptographic algorithms, securing communication channels, and regularly scanning dependencies for vulnerabilities, the application is better protected against common and emerging security threats.

**b. Reducing Risk of Security Breaches:**

* Adhering to these best practices reduces the likelihood of security breaches, which can have devastating consequences for the company, including financial losses, damage to reputation, and legal penalties. By proactively addressing vulnerabilities, the company minimizes the risk of such incidents.

**c. Ensuring Compliance with Regulatory Requirements:**

* Many industries are subject to regulatory requirements that mandate specific security measures. By following industry-standard best practices, the company ensures compliance with these regulations, avoiding potential fines and legal issues.

**d. Building Trust with Clients and Stakeholders:**

* Implementing robust security measures builds trust with clients, stakeholders, and partners. It demonstrates the company’s commitment to protecting sensitive data, which is crucial in maintaining and growing business relationships.

**e. Long-term Cost Savings:**

* Addressing security issues during the development and maintenance phases is more cost-effective than dealing with the aftermath of a security breach. By following best practices, the company can avoid the significant costs associated with data breaches, including remediation, legal fees, and loss of business.

**f. Supporting Business Continuity:**

* A secure software application contributes to the overall resilience of the company. By mitigating security risks, the company ensures that its operations can continue uninterrupted, even in the face of potential cyber threats.