****

# 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** | **2/22/2024** | **Jordan Mitchell** | **Final Release** |

## 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

Jordan Mitchell, Global Rain

## Algorithm Cipher

*Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Address the following:*

* *Provide a brief, high-level overview of the encryption algorithm cipher.*
* *Discuss the hash functions and bit levels of the cipher.*
* *Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.*
* *Describe the history and current state of encryption algorithms.*

Artemis Financial is looking to boost protection of its data on their web application via an encryption algorithm. As I’ve previously mentioned, Artemis’ dealings in the financial sector make them an appealing target to bad actors looking for monetary gain. We should look to encrypt the files and make render them useless to outside parties even if they are stolen. I recommend usage of the AES (Advanced Encryption Standard) 128-bit encryption algorithm in CTR (counter) mode. As AES is a symmetrical encryption algorithm, we must secure the keys, and for this I recommend usage of RSA. 128-bit encryption is the length of both the encryption and decryption keys. This is considered incredibly secure as the computational power to crack it required is far beyond reasonable means. It has been said to take thousands or even up to “one billion” years to crack with a supercomputer.

Furthermore, I believe the CTR method of AES128 provides a great advantage in terms of speed and user satisfaction. It allows for parallelization of both decryption and encryption, making the process more efficient. CTR should be used over older methods like 3DES and DES (Triple Data Encryption Standard), which are no longer considered secure by NIST (National Institude of Standards and Technology) as of 2017. The ECB method (Electronic Code Book) should also be avoided in this case, as it creates more obvious repetitive patterns with data larger than 128-bits. My reason for choosing 128-bits over the larger 256 is, although 256 can be considered “more secure”, it had reduced performance. That does not mean 128 bits aren’t secure, as AES128 has never been cracked. If there are large volumes of data, like an international financial corporation like Artemis would have, 128-bits would be the more time-efficient option.

## Certificate Generation

Insert a screenshot below of the CER file.

A screenshot of a computer program

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

## Secure Communications

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

A screenshot of a computer

Description automatically generated

The HTTPS is working, however my certificate is not valid as it is signed by myself.

## Secondary Testing

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

A screenshot of a computer

Description automatically generatedA screen shot of a computer

Description automatically generated

## Functional Testing

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

A screen shot of a computer program

Description automatically generated

## Summary

The code originally lacked usage of the cryptographic hash algorithm, which is essential to ensuring both data security and integrity. I implemented the SHA-256 algorithm and addressed vulnerabilities related to the weak hashing to ensure the data’s integrity. I made sure this aligned with the “implementation of secure algorithms” stage in the Vulnerability Assessment Process Flow Diagram, where algorithms are implemented to mitigate and reduce the risks associated with the tampering or manipulation of data.

I also included HTTPS support. Having secure communication over the network is **essential** to protecting your sensitive data from those who may be spying as well as MITM (man in the middle) attacks. I ensured the application supports HTTPS by configuring the embedded Tomcat server with SSL/TLS, encrypting the information transferred over the network. This stage falls within the “Secure Communication” part of the VAPFD. This step is where secure protocols like HTTPS are used to protect information in transit. This was signed using a personal certificate that was created by me. However, this should eventually be changed to a publicly trusted CA certificate.

The API was created to return the check sum value of a provided string. The check sum can be used to check the “fingerprint” of an object that is digital. This object is run through the hashing algorithm to generate the hash. If the hash is designed well, no collisions happen. This means when there are two differing objects, they do not produce the same the same hash value. So, the hash allows for verification of any file manipulation. The checksum was generated using SHA-256, which has an extremely low probability of colliding (about 2^256). Collisions should be avoided as much as possible, since they allow for another party to spoof a check sum or even reverse engineer the original data (which is especially dangerous if you are hiding sensitive information).

As for the Maven Dependency check, I ensured that I utilized the most up-to-date version. Originally the version was set at 5.3.0, but when I checked it was at the time of writing 9.0.9. This helps make sure the dependency check is as accurate as possible. To maintain the highest levels of security, I suggest routine dependency checks of the application to stay up to date on potential vulnerabilities and can help mitigate and prevent exploitations against the company. It also serves well to keep all plugins found within the pom file updated to their latest versions, to ensure the highest levels of security.

## Industry Standard Best Practices

Industry standard best practices should be followed to ensure that we maximize not only security, but also efficiency of maintenance. For security, there are integrated security testing tools that can be used to conduct regular scans and assessments for vulnerabilities in the code, like OWASP. We proactively reviewed and addressed the vulnerabilities by running a scan to see what vulnerabilities were currently present and have set up for the next step of resolving them. I also adhered to secure coding standards by utilizing the SHA256 cryptographic hash algorithm, which is widely used and accepted. I ensured proper error handling code was used to mitigate the risk of information leaks. I also ensured that the plugins used were updated to their latest iteration. Ensuring the codebase is regularly updated will prevent any known vulnerabilities from being exploited by malicious actors.

Application of industry standard best practices should never be overlooked. Prioritizing security throughout the software development cycle ensures the integrity, availability, and confidentiality of the company’s data and systems. This helps the company foster trust from customers, reduces the risk of regulatory non-compliance, and consolidates brand reputation. By proactively seeking and addressing security vulnerabilities, we can minimize potential financial and operational risks posed by security incidents. Ultimately these practices will contribute to the company’s long-term sustainability and success.

**RESOURCES**

*NIST (2020, June 22). Update to Current Use and Deprecation of TDEA. NIST* [*https://csrc.nist.gov/News/2017/Update-to-Current-Use-and-Deprecation-of-TDEA*](https://csrc.nist.gov/News/2017/Update-to-Current-Use-and-Deprecation-of-TDEA)

*Townsend, P.(2019, Mar. 25). RSA VS AES ENCRYPTION – A PRIMER. Townsend Security*

[*https://info.townsendsecurity.com/rsa-vs-aes-encryption-a-primer*](https://info.townsendsecurity.com/rsa-vs-aes-encryption-a-primer)

*Srivastava, P. (2024, Jan. 8). Rate Limiting a Spring API Using Bucket4j. Baeldung* [*https://www.baeldung.com/spring-bucket4j*](https://www.baeldung.com/spring-bucket4j)

*Baeldung (2024, Jan. 8). Java AES Encryption and Decryption. Baeldung.* [*https://www.baeldung.com/java-aes-encryption-decryption*](https://www.baeldung.com/java-aes-encryption-decryption)

*Griskenas, S. (2023, June 20). What is the SHA-256 algorithm, and how does it work? NordVPN.* [*https://nordvpn.com/blog/sha-256/*](https://nordvpn.com/blog/sha-256/)

*Reddy, P. (2021, May 30). Everything about Cryptographic hash functions. Nerd For Tech* [*https://medium.com/nerd-for-tech/everything-about-cryptographic-hash-functions-e2cd892e2a87*](https://medium.com/nerd-for-tech/everything-about-cryptographic-hash-functions-e2cd892e2a87)