Amanda Purnhagen

13 December 2022

Security Report for Artemis Financial

In this scenario, I work for a software engineering company called Global Rain. We have a client called Artemis Financial that has delegated the task of encrypting their long-term archive files. As their developer and security expert, it is my responsibility to recommend the best encryption practices and which cipher algorithm Artemis Financial should use. Technically, the most secure cipher algorithm is TLS\_ECDHE\_ECDSA\_AES\_256\_GCM\_SHA384. However, using this cipher algorithm would be considered overkill because it is very processing power intensive. Therefore, in this particular scenario, I have come to the conclusion that the best cipher algorithm for Artemis Financial to use for their archive files is TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 because it is a good middle ground between processing power and security, while also providing forward secrecy.

Next, we will break down the components that make up this specific cipher algorithm. According to Rescorla (2008), this cipher algorithm is one of a handful of “elliptic curve cipher suites for Transport Layer Security (TLS).” The “ECDHE” segment of the cipher algorithm title indicates that it is specifically Elliptic Curve Diffie-Hellman Ephemeral. “RSA” stands for the Rivest-Shamir-Adleman cryptosystem. “AES\_128\_GCM” means that it is Advanced Encryption Standard Galois Counter Mode encryption mode with a key size of 128 bits. Finally, “SHA256” is the hashing algorithm used, and it stands for 256-bit Secure Hashing Algorithm.

This cipher algorithm maximizes performance, user experience, and security. It combines the power of multiple encryption methodologies into one without taking up too much time and resources. A 128-bit key size for the encryption is sufficient for this project to avoid excessive latency in the system. For similar reasons, I chose SHA256.

As for the general advice I would give Artemis Financial on best practices regarding software security, I would insist that users never transmit sensitive data using plain text, nor use old and thus less secure algorithms such as RC4, DES, and 3DES, nor use the ANON, NULL, or EXPORT keywords. I do, however, recommend that they employ the use of a cipher algorithm that has a key size of 128 or greater as well as prioritize using ECDHE and DHE as their key agreement method (Manico, 2015).

Of course, there are still risks associated with asymmetric cryptography. However, the risks associated with the cipher itself are minimal. Therefore, they are an industry standard for common users and highly secure applications (e.g., those of government, military, and financial institutions). It would require an unfathomable amount of processing power to “crack” the encryption algorithm (quantum computing excluded). If an attacker was able to take advantage of improper user privileges management, local file inclusion, remote code execution, etc., he could get a hold of the private key (i.e., “certificate” in web development) and he could then decrypt transmitted data. It is very unlikely that the cipher algorithm will be liable for this type of breach. Instead, proper security practices should be employed at the software, network, and system administration level.

In cryptography, asymmetric encryption, as opposed to symmetric encryption, is used to obfuscate the keys even further. Asymmetric encryption involves using a public key and multiple private keys combined with some arithmetic. The weak and outdated symmetric encryption simply uses a single key given to those who need to access the data. Additionally, the use of random numbers is paramount. The cipher algorithm must use random large prime numbers in the generation of public and private keys to nearly eliminate the risk of decryption by an attacker. To get the keys, the attacker must be insurmountably lucky or take an unreasonable amount of time and resources.

The history leading up to the current state of encryption algorithms is fascinating. It tells the tale of a truly inspiring field of mathematics. Humanity is only starting to understand the true complexity of cryptography and the mathematics that accompany it. Briefly put, in 1976, Whitfield Diffie and Martin Hellman published work regarding the Diffie-Hellman key exchange (DH) method which posited the idea of using a cryptographic public key exchange. Just one year later, the Rivest-Shamir-Adleman public key system (RSA) was developed. This method allows two users to send a “mix” of the public key and their individual private key to each other without compromising the security of the final key. These two methods are still highly in use today. The mathematics behind cryptography includes advanced concepts such as elliptical curves, abstract algebra, number theory, statistics, etc.

The SSL server code for Artemis Financial has been modified in a number of ways. First, the pom.xml file has been supplemented to include a higher version of the OWASP dependency checker, version 3.0.0 of the Spring boot starter parent, and a suppression configuration for the OWASP dependency check. Next, the application.properties file now holds the proper configuration for HTTPS communication. The server should now run on port 8443. The SslServerApplication.java file has been updated with a new class that routes /hash. The function for this route returns a formatted string which holds, as a placeholder, a plain text value, “Amanda Purnhagen,” and its checksum, calculated after SHA-256.

Since there is no user input (yet), the relevant fields of the first tier of the Vulnerability Assessment Process Flow Diagram include cryptography, client/server, and code error. The updates included in the refactoring of the code address these fields by adding SSL and exception handling during the hashing. Many potential vulnerabilities were also addressed by updating the versions of the core third party libraries.

My process for adding layers of security to the software application started with manually inspecting the given codebase. I applied common security practices when it comes to web development such as encrypting communications, securing data handling, and static and dynamic analysis via code review and automated scanning.

I applied industry standard best practices for secure coding by updating all third party dependencies to their latest versions to fix known bugs and vulnerabilities. Then I checked if any vulnerabilities remained in the codebase. Additionally, I implemented encryption for the communication and a hashing functionality that could be used later on to secure sensitive data. In case of a breach, we can reassure the users that, even if their data is leaked, it is in a format that requires an unfathomable amount of time and resources to “crack.”

The Artemis Financial company will greatly benefit from consistent practice and application of industry standard best practices for secure coding. It is a financial company, so the data within their servers is especially sensitive and susceptible to attacks. Therefore, it is paramount to incorporate security practices at every layer of the DevOps model, especially when it comes to their web application or any web-facing functionalities. Artemis Financial should prioritize its users’ safety and generally avoid cutting corners. A recommendation would be to have a production version of the application and a development version of the application which is not accessible to anyone but the developers.

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generatedText, letter

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated**References**

Manico, J., Detlefsen, A., Kenan, K., & Smith, M. (2015). Protecting Sensitive Data. In *Iron-clad java: Building secure web applications*. McGraw Hill Education.

Rescorla, E. (2008, August). *TLS elliptic curve cipher suites with SHA-256/384 and AES Galois counter mode (GCM)*. RFC Editor. Retrieved December 12, 2022, from <https://www.rfc-editor.org/rfc/rfc5289>