****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

Table of Contents

[Document Revision History 3](#_Toc32574607)

[Client 3](#_Toc32574608)

[Instructions 3](#_Toc32574609)

[Developer 4](#_Toc32574610)

[1. Interpreting Client Needs 4](#_Toc32574611)

[2. Areas of Security 5](#_Toc32574612)

[3. Manual Review 5](#_Toc32574613)

[4. Static Testing 8](#_Toc32574614)

[5. Mitigation Plan 9](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/17/2022** | **Daniel Giannatsis** |  |

## Client



## Developer

Daniel Giannatsis

## 1. Interpreting Client Needs

* The value of secure communications to the company is high, since they need to protect their customers from the exposure of their data, as data breaches mean customers can have their information or identity stolen and will lead to customer mistrust in the company and affect profits. They also want to protect corporate communications as corporate espionage is a legitimate threat.
* As an international financial consulting firm, Artemis Financial is paid by their patrons for their services, and money is transferred to banks, into investment avenues and to insurance companies through Artemis all over the globe.
* Yes, there are in fact governmental restrictions to consider. Since Artemis Financial deals with consumer financial information, they must comply with the Gramm–Leach–Bliley Act which says that any financial institution must protect customer’s financial data and disclose how that data is shared, to 3rd parties for example. Depending on the customer’s home country, such as European countries, there can also be steep penalties for customer data breaches.
* As a financial company running a web application dealing with many customers’ sensitive financial information, there exists some significant threats. Motivating assets for attackers to pursue are money, credit card numbers, personally identifiable information such as social security numbers, mailing and email addresses and phone numbers, along with trade secrets from the company itself. These assets can be lost through data breaches from a variety of methods. Since web applications are tightly linked to their server-side databases, one of the most dangerous methods is SQL injection that uses carefully crafted messages to change database queries to perform unexpected actions, like returning all information in a table. This kind of attack can lead to not only large data breaches but also an attacker can gain elevated access to the rest of the system!! Aside from databases, data breaches can also occur from the absence of properly encrypted communications. A hacker can easily eavesdrop on corporate communications if the company doesn’t encrypt inbound and outbound data. Data breaches aren’t only bad for the company’s profits, but also customers lose trust, which leads to much more profit loss in the long-term. Hackers could also be motivated to make the company lose money through a denial of service(DOS) attack which shuts down services for an amount of time by overloading the system’s resources.
* The use of open-source tools and databases are critical to evaluating and mitigating threats as we upgrade the companies’ web security. Some of these tools include static code testing tools, such as Maven Dependency Checker or FindBugs. Dynamic “black box” code testing tools are also important as they scan the application for vulnerabilities as it runs. Databases that track known bugs and possible fixes are also extremely useful and open source, such as the National Vulnerability Database. Current web application technologies such as Spring should be considered to streamline the process of securing Artemis’ system. This is because the Spring framework includes many modern and useful web application libraries (many for security purposes) that are easily implemented through Spring.

## 2. Areas of Security

.

* The first area of security to assess is Input Validation: Secure Input and Representations. This area is relevant to Artemis Financials’ web application as input data from customers (or attackers), whether it be login info, text from forms or any other input data needs to be validated that it’s in an acceptable format.
* The second area of security that is necessary to inspect for the company’s application is APIs: Secure APIs. This needs to be inspected because their application uses a RESTful web API.
* The third area to assess is Cryptography: Encryption use and Vulnerabilities. Artemis Financial deals with sensitive customer financial data and should have effective encryption methods to protect their data at rest and in transit.
* The fourth area of security to assess is Code Quality: Secure Coding Practices/Patterns. APIs and Input Validation are code quality areas and it’s important to make sure that code from frameworks like Spring and any patterns have been written correctly.
* The fifth area to assess is Encapsulation: Secure Data Structures since Artemis Financials’ current application uses custom code and it’s important to check that these custom classes are encapsulated correctly.

## 3. Manual Review

**Input Validation**

In this block of code, input is taken in from the greeting webpage’s name parameter without any kind of input validation to validate that the incoming data is safe.

Text

Description automatically generated

Similarly, in this block of code, at the /read endpoint, the name parameter is being used without any kind of input validation. Text

Description automatically generated

**Encapsulation:** the DocData class is correctly encapsulated .

Text

Description automatically generated

The CRUD class’s members are correctly encapsulated, but the getContent() accessor functions should be avoided.

Text

Description automatically generated

The Greeting class is correctly encapsulated.

Text

Description automatically generated

The Customer class is not correctly encapsulated, the account balance variable is public.

Text

Description automatically generated

The myDateTime class is not correctly encapsulated, the second, minute and hour variables are public.

Text

Description automatically generated

\*\* No code was found that implements an authentication system to secure the web application.

\*\* No code was found that encrypts data being sent or retrieved from the API.

## 4. Static Testing

Log4j-api-2.12.1

## CVE-2021-44832

* This vulnerability occurs in Log4j-api versions 2.0-beta7 - 2.17.0, where a remote code execution attack can happen if a configuration uses a JDBC Appender with a specific URI if an attacker has control of the target LDAP server. The widely accepted solution is to limit JNDI data source names to java protocol in versions 2.12.4, 2.17.1, , and 2.3.2. This entry was attributed to Apache Software Foundation.

**CVE-2021-45105**

* This vulnerability affects versions 2.0-alpha1 up to 2.16.0 of log4j-api-2.12.1, with the exclusion of 2.12.3 and 2.3.1. The code fails to protect from uncontrolled recursion from self-referential lookups. Attackers that have control over the Thread Context Map data can cause a DOS when fabricated string is interpreted. Fix: update to Log4j 2.17.0, 2.12.3, or 2.3.1. This entry was attributed to the Apache Software Foundation.

## CVE-2021-44228

* This vulnerability is present in the JNDI features which don’t protect against attacker-controlled LDAP and other JNDI related endpoints. If an attacker can control log messages, they can execute code loaded from LDAP servers when message lookup substitution is enabled. There isn’t a widely accepted solution or fixes other than updating to 2.15.0 or later. Attributed to the Apache Software Foundation.

## CVE-2020-9488

* The vulnerability involves improper certificate validation with host mismatch in Log4j’s SMTP appender. This flaw allows the interception of a SMTPS connection that leaks log messages from the appender by way of a man-in-the-middle attack. Solution: Upgrade to 2.13.2 which supports this feature. Previous versions can set the system property mail.smtp.ssl.checkserveridentity to true to globally enable hostname verification for SMTPS connections. Attribution: Apache Software Foundation.

Jackson-Databind-2.10.2

## CVE-2020-25649

This vulnerability was found in FasterXML Jackson Databind, where entity expansion wasn’t properly secured, making it vulnerable to XML external entity attacks. (No widely accepted solution) Attributed to Red Hat, Inc.

Logback-core-1.2.3

## CVE-2021-42550

In logback version 1.2.7 and earlier, an attacker with high enough privileges could craft a malicious configuration allowing code execution from LDAP servers. Solution: Disable write access to logback’s configuration file. Attributed to Switzerland Government Common Vulnerability Program.

Snakeyaml-1.25

## CVE-2017-18640

The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation. Attributed to MITRE. Solution: update to 1.26.

## 5. Mitigation Plan

* To address securing of input for Artemis’ web application, Input validation should be implemented for the input functions above that deal with input from the /greeting and /read endpoints. Code using regular expressions (that aren’t vulnerable to ReDOS) to check against whitelisted patterns, along with other methods such as validating input length should be used to verify acceptable input.
* To add security to the Rest API, a consistent, role-based authentication filter needs to be implemented to force users to log in before performing operations that request data from the server side. Each request would run through a centralized policy enforcement point that would check authorization using a policy decision point. This authentication/authorization scheme can be implemented with Spring.
* To securely encrypt the user’s client-side data being sent through the Restful API, an encryption filter needs to be built so that client’s request is encrypted, decrypted by the filter, passed through the API, then the response is encrypted and passed to the client where it is decrypted.
* To ensure properly encapsulated classes, the account balance member variable in the Customer class should be changed to private. The mySecond, myMinute, and myHour member variables in the myDateTime class should also be changed to private.
* Static report solutions: log4j API: We will use the widely accepted solution to limit JNDI data source names to java protocol. We will also update to version 2.17.0 to avoid the second and third vulnerability. To eliminate the 4th vulnerability, the 2.17.0 update will also be a fix as log4j up to or past 2.13.2 added corrective features. Jackson Databind: there is still not a viable solution to the flaw, so we will keep checking NVD and related sources for any sort of fix in the meantime. LogBack Core: for this vulnerability, we will disable write access to logback’s configuration file as advised. Snakeyaml: we will update to version 1.26 as advised.