Secure communications are crucial to Artemis Financial due to the industry they operate in. Clients need to be able to depend on Artemis to maintain secure and encrypted communications to prevent unauthorized access. If Artemis is to delve into international transactions, it must adhere to cybersecurity standards set globally to ensure client data will be protected. Like international laws, governmental restrictions take precedence as well. Adhering to frameworks like HIPAA must be taken into consideration. Keeping systems up to-date and current will be imperative in making sure vulnerabilities are kept at a minimum.

Firstly, data encryption would be priority number one in terms of security. Whether sending or just storing data, it must be always safeguarded. Another must is authentication to said data. Only authorized users should have access to this data in order to keep it safe. Outside of standard safety protocols, its necessary to comply to legal protocols as well.

1. In DocData.java, if SQL queries are made using user-provided data without prepared statements, then this presents the threat of SQL Injections.
2. Also, in DocData.java, the connection details (localhost:3306) are hard coded meaning its possible that this info can be potentially accessed by unauthorized users
3. In CRUDController.java, the CRUD method does not have any input validation posing the threat of injection attacks.
4. In DocData.java, the exception handling in the ‘read\_document’ method is not sufficient, in that it presents the ‘SQLException’ but does not print a proper response.
5. In Customer.java, there are inconsistencies. The ‘account\_balance’ field in the ‘Customer’ class is package-private where ‘account\_number’ is private.
6. Another vulnerability in the CRUD method, includes insecure endpoints. If ‘doc.toString’ contains sensitive information, it could be accessed by unauthorized individuals.
7. While less of a vulnerability and more of a risk of confusion, the “customer.java” class is not properly capitalized.

1. Use prepared statements to prevent SQL injection.

2. Store sensitive information such as database credentials in environment variables or configuration files, not in the source code.

3. Validate and sanitize all user inputs.

4. Implement proper exception handling to ensure errors are logged and handled appropriately.

5. Ensure consistent access control and consider using getter and setter methods for accessing class fields.

6. Carefully review the data returned by endpoints to ensure no sensitive information is exposed.

7. Follow standard naming conventions for better code readability and maintenance.

After running multiple dependency checks, they all came back with no vulnerabilities which cannot be correct. My method of running the check included installing Homebrew due to using a Mac during this project. Next, I installed Maven via Homebrew and proceeded to run the check via command: mvn org.owasp:dependency-check-maven:check and received the check resulting in no vulnerabilities.

For mitigation, based off the manual review, the main focus should be on DocData.java. The threat of SQL injections and the exposed connection details should be remedied for to avoid any exploitations or unauthorized users accessing sensitive information. Speaking more on unauthorized users, the CRUD method’s insecure endpoints should be fixed as well since it is the utmost priority to keep sensitive data secure.