# CS 305 Project One

## Document Revision History

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 9/22/2024 | Zane Milo Deso | * Interpreted and documented security modernization needs. * Identified and documented relevant areas of security with justification. * Conducted manual code review, Identified and documented vulnerabilities. * Integrated and ran maven dependency check; researched and documented each vulnerability found. * Devised Mitigation and action plan with steps to fix each vulnerability. |

## Client:



## Developer

Zane Milo Deso

1. Interpreting Client Needs

Artemis Financial, a consulting firm specializing in individualized financial plans, requires a robust security upgrade to protect its RESTful web API from external threats. The company's need for security is paramount as it handles sensitive customer information related to savings, retirement, investments, and insurance. Although it’s unclear if Artemis Financial currently operates internationally, Global Rain’s global reach implies that the API must meet international standards to ensure secure communication across borders. This means adopting comprehensive security measures that not only address current vulnerabilities but also anticipate future threats, ensuring the protection of both the company’s and its clients' financial data.

Securing this API is critical to Artemis Financial’s reputability and credibility. Like any financial institution, the trust clients' place in the company is tied directly to its ability to safeguard their personal and financial information. The API must comply with regulatory standards, such as the Code of Federal Regulations in the U.S. (Title 45, Part 170.404), which outlines necessary security and documentation practices. International operations would introduce additional government restrictions on data handling and communication, meaning Artemis Financial must stay vigilant in ensuring that its software complies with these varying regulations.

Given the evolving nature of web applications and the persistence of external threats, Artemis Financial’s modernization plan should include reliance on open-source libraries and software known for their transparency and regular updates. Incorporating trusted open-source solutions helps to keep the API secure, while budgeting for proactive measures like penetration testing ensures that the company remains ahead of emerging vulnerabilities. In this rapidly shifting technological landscape, Artemis Financial must continuously adapt to maintain the security of its systems, safeguarding its reputation and ensuring long-term client trust.

2. Areas of Security

Based on the needs identified in the previous section, several critical security areas must be addressed to ensure the protection of Artemis Financial’s web API. First and foremost, input handling is a significant concern, as improper input validation could lead to injection attacks or denial of service. Input limits and query parameterization are necessary to prevent malicious actors from exploiting vulnerabilities through excessive calls to the server. Ensuring robust input validation at the API level is crucial to safeguarding sensitive financial data and preventing unauthorized access.

The next essential area is API security. Given that the application relies on API communications, proper measures must be in place to control and secure interactions. This includes setting limits on input length and request volume, as well as employing secure communication protocols. Cryptography also plays a vital role, especially in securing data transmission and storage. Encryption methods, such as proper serialization and deserialization techniques, are required to meet global security standards and protect sensitive financial information from external threats.

Finally, client-server communication must be fortified to prevent vulnerabilities between the front and back ends of the web application. Attention must also be given to code quality and error handling; any code errors could lead to unexpected security risks. Following best practices, such as those outlined by OWASP and Oracle, will enhance the maintainability and security of the software. Additionally, encapsulation should be rigorously applied to protect data integrity and prevent unauthorized access to internal components of the application.

3. Manual Review

* Outdated Maven Dependency Check Tool
  + Location: pom.xml
  + Description: The existing Maven Dependency Check tool was outdated by approximately five versions. The outdated tool was removed, and a new, up-to-date plugin was integrated to ensure accurate static testing in the next section.
* Incorrect Package Naming
  + Location: help.md
  + Description: The file contains a note that the original package name, com.twk.rest-service, is invalid and has been replaced by com.twk.rest\_service. This inconsistency requires verification to avoid potential issues in the system, though it is unclear if this constitutes a vulnerability at this stage.
* No Input Validation on API Calls
  + Location: Various API request handlers
  + Description: There is no input validation for API calls. Parameters are not limited, and there are no character standards for the requests, making it difficult to distinguish between legitimate requests and potentially harmful ones. This could lead to injection attacks (such as SQL injections) and denial of service attacks.
* Lack of HSTS and Special Character Handling
  + Location: Various HTTP request handlers
  + Description: The absence of HTTP Strict Transport Security (HSTS) introduces a vulnerability where incorrect formats and special characters in input strings may cause errors during parsing and validation. Improper escaping or the omission of special characters could lead to errors or HTTP request failures. Parsing and canonicalization should be performed before validation to avoid these issues, and invalid data should be rejected at both the header and body levels to prevent processing partial or corrupted requests.
* Use of Unsecured HTTP Requests
  + Location: Various HTTP request handlers
  + Description: The API primarily uses HTTP for data requests instead of HTTPS. This poses a security risk as HTTP is not encrypted, leaving data vulnerable to interception. The application should enforce stricter transport security measures (e.g., HSTS).
* Hard-Coded Database Credentials
  + Location: docdata.java, read\_document() method
  + Description: The method contains hard-coded credentials (localhost/test with username root and password root). This presents a critical security vulnerability as it exposes the database connection to potential attackers.
* Public Access to DocData Class and Methods
  + Location: docdata.java
  + Description: The DocData class, its constructors, and methods are all public, violating the principle of encapsulation. This could lead to unintended access and manipulation of the class from outside, increasing the risk of misuse.
* Improper SQL Handling in DocData
  + Location: docdata.java, read\_document() method
  + Description: The SQL query within the read\_document() method is not using query parameterization, which makes it vulnerable to SQL injection attacks. This vulnerability is compounded by the fact that the method throws stack traces, potentially revealing sensitive information when a connection to the hard-coded localhost fails.
* Absence of Whitelisting and Rate Limiting
  + Location: Input handling across API endpoints
  + Description: The application lacks whitelisting of acceptable data types, rate limits, and length limits for incoming requests. This leaves the system vulnerable to overuse and potential abuse, such as sending excessively large or frequent requests.
* Unused Local Variable
  + Location: docdata.java, read\_document() method
  + Description: A local variable is defined within the try block but is not used. Although not immediately dangerous, leaving unused variables in the code may lead to confusion or unintended behavior in future modifications.
* No Access Control Mechanisms
  + Location: API access control
  + Description: There is no authentication or authorization system in place, and no access control mechanisms such as role-based or attribute-based access controls. The API relies solely on request data to make access decisions, without verifying user identity or permissions.
* Obscurity in Access Control Policy
  + Location: Access control policy
  + Description: The access control policy checks for the presence of localhost in requests, which is hard-coded and insufficient for securing access. This introduces a security vulnerability, as it relies on obscurity rather than proper access control mechanisms.
* No Cryptographic Ciphers
  + Location: Across API handling of sensitive data
  + Description: The application lacks any cryptographic cipher implementation, leaving sensitive data such as client financial information vulnerable to exposure. No serialization or deserialization techniques are implemented to securely handle data.
* No Synchronizer Token Pattern for Transactions
  + Location: API transactions
  + Description: Transactions are not protected by a synchronizer token pattern, which could leave the application vulnerable to cross-site request forgery (CSRF) attacks.
* Lack of Protection Against Cross-Site Scripting and Clickjacking
  + Location: Various HTTP request handlers
  + Description: The application does not implement X-Frame-Options or XSS protection in its HTTP response headers, making it vulnerable to cross-site scripting (XSS) and clickjacking attacks.

4. Static Testing

Table 1

*Vulnerabilities found during dependency check*

|  |  |  |
| --- | --- | --- |
| Dependency Name | Vulnerability ID(s) | Description/Solution/Attribution |
| bcprov-jdk15on-1.46.jar | [CVE-2023-33202](https://nvd.nist.gov/vuln/detail/CVE-2023-33202)  [CVE-2016-1000352](https://nvd.nist.gov/vuln/detail/CVE-2016-1000352)  [CVE-2016-1000346](https://nvd.nist.gov/vuln/detail/CVE-2016-1000346)  [CVE-2016-1000345](https://nvd.nist.gov/vuln/detail/CVE-2016-1000345)  [CVE-2016-1000344](https://nvd.nist.gov/vuln/detail/CVE-2016-1000344)  [CVE-2016-1000343](https://nvd.nist.gov/vuln/detail/CVE-2016-1000343)  [CVE-2016-1000342](https://nvd.nist.gov/vuln/detail/CVE-2016-1000342)  [CVE-2016-1000341](https://nvd.nist.gov/vuln/detail/CVE-2016-1000341)  [CVE-2016-1000339](https://nvd.nist.gov/vuln/detail/CVE-2016-1000339)  [CVE-2016-1000338](https://nvd.nist.gov/vuln/detail/CVE-2016-1000338)  [CVE-2018-5382](https://nvd.nist.gov/vuln/detail/CVE-2018-5382)  [CVE-2017-13098](https://nvd.nist.gov/vuln/detail/CVE-2017-13098)  [CVE-2013-1624](https://nvd.nist.gov/vuln/detail/CVE-2013-1624) | * The dependency is bcprov-jdk15on-1.46.jar, which is the Bouncy Castle crypto package. It's a Java implementation of cryptographic algorithms. This jar contains JCE provider lightweight API for Bouncy Castle cryptography APIs for JDK 1.5 to 1.7. Bouncy Castle for Java before 1.7 had a potential denial of service vulnerability within the org.bouncycastle.openssl.PEMPParser class. That class basically encodes streams of certificates. Specifically, the class parses OpenSSL PEM encoded streams containing X.509 certificates, PKCS8 encoded keys, and PKCS7 objects. Parsing a file that has ASN.1 data through the PEM parser causes an out-of-memory error, which enables a denial of service attack. The implemented solution here is for users of the FIPS Java API, bc-fja1.0.2.3 and earlier are affected, but everything from bc-fja1.0.2.4 is fixed. The solution here is to update to version 1.0.2.4. * Some of the other vulnerabilities found included JCE provider version 1.5.5 and earlier. The ECI ES implementation allowed the use of ECB mode. That mode is considered unsafe, and there's no service from or support from the provider. Also, the party DH public key is not fully validated, causing issues as invalid keys can be used to reveal details about other parties' private keys. As of release 1.5.6, the parameters are checked on agreement calculation. So updating to 1.5.6 would be the fix here. * In version 1.5.5 and older, where timing is easily observed, it's possible with enough observations to identify when the decryption is failing due to padding. DH IES/EC IES CBC mode is vulnerable to padding attacks, and ECB mode was allowed in DH IES in version 1.5.5 and earlier. Updating to version 1.5.6 where padding and ECB mode vulnerabilities have been mitigated is the appropriate solution. * In version 1.5.5 and earlier, the DSA key pair generator generates a weak private key if used with default values. In earlier releases, this could be addressed by explicitly passing parameters to the key pair generator. This vulnerability has been resolved in later versions where stronger key pair generation methods have been implemented. * Furthermore, in version 1.5.5 and earlier, EC DSA does not fully validate ASN.1 encoding of signatures on verification, leaving it possible to inject extra elements into the sequence making up the signature and still have it validated. This has led to some cases of introducing invisible data into a signed structure. Updating to version 1.5.6 or later is recommended as it addresses this vulnerability with proper validation mechanisms. * In version 1.5.5, DSA signature generation is vulnerable to timing attacks. This vulnerability exists due to the way signature generation leaks timing data, allowing an attacker to observe these signatures over time. The solution is to update to version 1.5.6, where timing attack vulnerabilities have been mitigated. * In earlier versions, specifically 1.5.5 and older, the AES engine used was AESFastEngine, which had a highly table-driven approach. It turns out that the data channel on the CPU can be monitored, and the lookup table accesses provided sufficient information leakage on the AES key being used. This engine has been modified in later versions, and the recommendation is to update to 1.5.6, where these leaks have been addressed. The AESFastEngine is now only recommended when otherwise deemed appropriate. * The default BKS keystore in versions prior to 1.47 used an HMAC that is only 16 bits long, which could allow an attacker to compromise the integrity of the BKS keystore. With the release of version 1.47, the BKS format was changed to use a 160-bit HMAC instead. This fix applies to any BKS keystores generated before version 1.47. However, for legacy systems that need to create files for backward compatibility, a specific keystore type BKS-V1 was introduced in version 1.49. The use of BKS-V1 is discouraged by the library authors and should only be used when the 16-bit checksum for file integrity checks will not result in a security issue. * Bouncy Castle TLS prior to version 1.0.3, when configured using Java cryptography extension for cryptographic functions, provides a weak Bleichenbacher oracle when any TLS cipher suite using RSA key exchange is negotiated. An attacker can recover the private key from a vulnerable action. This vulnerability is referred to as "Robot." The solution is to update to version 1.0.3 or later to eliminate the vulnerability. * The TLS implementation in the Bouncy Castle Java library before 1.48 and the C# library before 1.8 does not properly consider timing side-channel attacks in non-compliant MAC check operations during the process of malformed CBC padding. This vulnerability allows remote attackers to conduct distinguishing attacks and plaintext recovery attacks via statistical analysis of timing data from crafted packets. Updating to version 1.48 or later is required to mitigate these timing attacks. |
| hibernate-validator-6.0.18.Final.jar | [CVE-2020-10693](https://nvd.nist.gov/vuln/detail/CVE-2020-10693) | * Allows attackers to bypass input sanitation controls by exploiting a flaw in the message interpolation processor. Invalid EL expressions can be evaluated as valid, potentially leading to improper input handling. |
| jackson-databind-2.10.2.jar | [CVE-2023-35116](https://nvd.nist.gov/vuln/detail/CVE-2023-35116)  [CVE-2021-46877](https://nvd.nist.gov/vuln/detail/CVE-2021-46877)  [CVE-2022-42004](https://nvd.nist.gov/vuln/detail/CVE-2022-42004)  [CVE-2022-42003](https://nvd.nist.gov/vuln/detail/CVE-2022-42003)  [CVE-2020-36518](https://nvd.nist.gov/vuln/detail/CVE-2020-36518)  [CVE-2020-25649](https://nvd.nist.gov/vuln/detail/CVE-2020-25649) | * Allows attackers to bypass input sanitation controls by exploiting a flaw in the message interpolation processor. Invalid EL expressions can be evaluated as valid, potentially leading to improper input handling. |
| log4j-api-2.12.1.jar | [CVE-2021-44832](https://nvd.nist.gov/vuln/detail/CVE-2021-44832)  [CVE-2021-45105](https://nvd.nist.gov/vuln/detail/CVE-2021-45105)  [CVE-2021-45046](https://nvd.nist.gov/vuln/detail/CVE-2021-45046)  [CVE-2021-44228](https://nvd.nist.gov/vuln/detail/CVE-2021-44228)  [CVE-2020-9488](https://nvd.nist.gov/vuln/detail/CVE-2020-9488) | * Vulnerability allows for remote code execution (RCE) attacks when the configuration uses the JDBC Appender with a JNDI LDAP data source URI. This issue has been fixed by restricting JNDI data source names to the Java protocol. |
| logback-core-1.2.3.jar | [CVE-2023-6378](https://nvd.nist.gov/vuln/detail/CVE-2023-6378)  [CVE-2021-42550](https://nvd.nist.gov/vuln/detail/CVE-2021-42550) | * Serialization vulnerability exists in the Logback Receiver component, allowing denial of service (DoS) attacks by sending poisoned data. Prior versions allowed attackers with certain privileges to edit configuration files, enabling the execution of arbitrary code loaded from LDAP servers. |
| snakeyaml-1.25.jar | [CVE-2022-1471](https://nvd.nist.gov/vuln/detail/CVE-2022-1471)  [CVE-2022-41854](https://nvd.nist.gov/vuln/detail/CVE-2022-41854)  [CVE-2022-38752](https://nvd.nist.gov/vuln/detail/CVE-2022-38752)  [CVE-2022-38751](https://nvd.nist.gov/vuln/detail/CVE-2022-38751)  [CVE-2022-38750](https://nvd.nist.gov/vuln/detail/CVE-2022-38750)  [CVE-2022-38749](https://nvd.nist.gov/vuln/detail/CVE-2022-38749)  [CVE-2022-25857](https://nvd.nist.gov/vuln/detail/CVE-2022-25857)  [CVE-2017-18640](https://nvd.nist.gov/vuln/detail/CVE-2017-18640) | * A vulnerability exists in the Constructor class, which does not restrict the types that can be instantiated during deserialization. Deserializing YAML content allows attackers to remotely execute code. The recommended fix is to use SnakeYAML's SafeConstructor when parsing untrusted content to restrict deserialization. Upgrading to version 2.0 is advised. * Additionally, parsing untrusted YAML files exposes a denial-of-service (DoS) vulnerability due to missing nested depth limitations for collections. This allows attackers to cause a DoS by providing YAML content with excessive nesting. |
| spring-boot-2.2.4.RELEASE.jar  spring-boot-starter-web-2.2.4.RELEASE.jar  spring-core-5.2.3.RELEASE.jar  spring-expression-5.2.3.RELEASE.jar  spring-web-5.2.3.RELEASE.jar  spring-webmvc-5.2.3.RELEASE.jar | [CVE-2023-20883](https://nvd.nist.gov/vuln/detail/CVE-2023-20883)  [CVE-2023-20873](https://nvd.nist.gov/vuln/detail/CVE-2023-20873)  [CVE-2022-27772](https://nvd.nist.gov/vuln/detail/CVE-2022-27772)  [CVE-2023-20863](https://nvd.nist.gov/vuln/detail/CVE-2023-20863)  [CVE-2023-20861](https://nvd.nist.gov/vuln/detail/CVE-2023-20861)  [CVE-2022-22971](https://nvd.nist.gov/vuln/detail/CVE-2022-22971)  [CVE-2022-22970](https://nvd.nist.gov/vuln/detail/CVE-2022-22970)  [CVE-2022-22968](https://nvd.nist.gov/vuln/detail/CVE-2022-22968)  [CVE-2022-22965](https://nvd.nist.gov/vuln/detail/CVE-2022-22965)  [CVE-2022-22950](https://nvd.nist.gov/vuln/detail/CVE-2022-22950)  [CVE-2021-22060](https://nvd.nist.gov/vuln/detail/CVE-2021-22060)  [CVE-2021-22096](https://nvd.nist.gov/vuln/detail/CVE-2021-22096)  [CVE-2021-22118](https://nvd.nist.gov/vuln/detail/CVE-2021-22118)  [CVE-2020-5421](https://nvd.nist.gov/vuln/detail/CVE-2020-5421)  [CVE-2016-1000027](https://nvd.nist.gov/vuln/detail/CVE-2016-1000027) | * Denial-of-Service via SpEL Expressions: In versions prior to 5.2.24, as well as 6.0.0 through 6.0.6, 5.3.0 through 5.3.25, and 5.2.0.RELEASE through 5.2.22.RELEASE, a user can craft malicious Spring Expression Language (SpEL) expressions that may cause a DoS attack. * DoS Vulnerabilities in STOMP over WebSocket Endpoints: Multiple versions, including those prior to 5.3.20 and 5.2.22, and versions 5.3.0 through 5.3.20, 5.3.0 through 5.3.18, and 5.2.0 through 5.2.22, have vulnerabilities in the STOMP over WebSocket endpoint. An authenticated user could exploit these to cause a DoS attack. * Remote Code Execution via Data Binding: Spring MVC or Spring WebFlux applications running on JDK 9 or later may be vulnerable to remote code execution through unsafe data binding configurations. This exploit requires the application to be deployed on Tomcat as a WAR file. * Privilege Escalation in WebFlux Applications: In versions 5.2.x prior to 5.2.15 and 5.3.x prior to 5.3.7, a vulnerability allows an authenticated attacker to recreate the temporary storage directory. This could enable access to, modification of, and overwriting of files uploaded to the application. * Bypass of RFD Protections and Potential Remote Code Execution: Versions 5.2.0 through 5.2.8, 5.1.0 through 5.1.17, 5.0.0 through 5.0.18, 4.3.0 through 4.3.28, and older unsupported versions may have their protections against Reflected File Download (RFD) attacks bypassed, depending on the browser used. This could lead to remote code execution through Java deserialization of untrusted data. The vulnerability's impact varies based on how the library is implemented and may require authentication. |
| tomcat-embed-core-9.0.30.jar  tomcat-embed-websocket-9.0.30.jar | [CVE-2024-21733](https://nvd.nist.gov/vuln/detail/CVE-2024-21733)  [CVE-2023-45648](https://nvd.nist.gov/vuln/detail/CVE-2023-45648)  [CVE-2023-46589](https://nvd.nist.gov/vuln/detail/CVE-2023-46589)  [CVE-2023-45648](https://nvd.nist.gov/vuln/detail/CVE-2023-45648)  [CVE-2023-44487](https://nvd.nist.gov/vuln/detail/CVE-2023-44487)  [CVE-2023-41080](https://nvd.nist.gov/vuln/detail/CVE-2023-41080)  [CVE-2023-28708](https://nvd.nist.gov/vuln/detail/CVE-2023-28708)  [CVE-2022-42252](https://nvd.nist.gov/vuln/detail/CVE-2022-42252)  [CVE-2021-43980](https://nvd.nist.gov/vuln/detail/CVE-2021-43980)  [CVE-2022-34305](https://nvd.nist.gov/vuln/detail/CVE-2022-34305)  [CVE-2022-29885](https://nvd.nist.gov/vuln/detail/CVE-2022-29885)  [CVE-2021-41079](https://nvd.nist.gov/vuln/detail/CVE-2021-41079)  [CVE-2021-33037](https://nvd.nist.gov/vuln/detail/CVE-2021-33037)  [CVE-2021-30640](https://nvd.nist.gov/vuln/detail/CVE-2021-30640)  [CVE-2021-25329](https://nvd.nist.gov/vuln/detail/CVE-2021-25329)  [CVE-2021-25122](https://nvd.nist.gov/vuln/detail/CVE-2021-25122)  [CVE-2021-24122](https://nvd.nist.gov/vuln/detail/CVE-2021-24122)  [CVE-2020-17527](https://nvd.nist.gov/vuln/detail/CVE-2020-17527)  [CVE-2020-13943](https://nvd.nist.gov/vuln/detail/CVE-2020-13943)  [CVE-2020-13935](https://nvd.nist.gov/vuln/detail/CVE-2020-13935) | * Apache Tomcat has several identified vulnerabilities that could lead to security issues such as sensitive information disclosure, request smuggling, denial of service (DoS), and cross-site scripting (XSS) attacks. It is recommended to upgrade to versions 8.5.64 onwards or 9.0.44 onwards to address these issues. * One vulnerability involves the generation of error messages containing sensitive information. Upgrading to the recommended versions resolves this issue. There is also an improper input validation vulnerability due to incorrectly parsed HTTP trailer headers. Exceeding the header size limit could cause Tomcat to treat a single request as multiple requests, leading to the possibility of request smuggling when behind a reverse proxy. * An incomplete cleanup vulnerability exists when recycling various internal objects in Apache Tomcat. An error could cause Tomcat to skip parts of the recycling process, leading to information leaking from the current request or response into the next one. The HTTP/2 protocol allows for DoS attacks via server resource consumption due to rapid request cancellations and resetting many streams quickly. * A URL redirection to untrusted sites vulnerability is present in the form authentication feature within Apache Tomcat, limited to the root web application. When using the Remote IP Filter with requests received from a reverse proxy via HTTP that include the X-Forwarded-Proto header set to HTTPS, session cookies created by Apache Tomcat did not include the Secure attribute. This omission could result in the session cookie being transmitted over an insecure channel. * Configurations that ignore invalid HTTP headers by setting rejectIllegalHeader to false can make Tomcat susceptible to request smuggling attacks. Tomcat did not reject requests containing an invalid Content-Length header, which is problematic if located behind a reverse proxy that also fails to reject such requests. Additionally, the form authentication example in the example's web application displayed user-provided data without filtering, exposing an XSS vulnerability. * The documentation for EncryptInterceptor incorrectly stated that it enables Tomcat clustering over an untrusted network. While it provides confidentiality and integrity protection, it does not mitigate all risks associated with running over an untrusted network. Tomcat did not properly validate incoming TLS packets when configured to use NIO with OpenSSL or NIO2 with OpenSSL for TLS. Specially crafted packets could trigger an infinite loop, resulting in a DoS condition. * Incorrect parsing of the HTTP Transfer-Encoding request header in some instances led to the possibility of request smuggling when used with a reverse proxy. Tomcat incorrectly ignored the Transfer-Encoding header if the client declared it would only accept an HTTP/1.0 response. A vulnerability in the JNDI Realm allows an attacker to use authentication variations of a validated username to bypass protections enforced by a LockOut Realm. * When responding to new H2C (HTTP/2 over cleartext) connection requests, Apache Tomcat could duplicate headers and portions of the request body from one request to another. This means that User A and User B could both see the results of User A's request, leading to information leakage. Serving resources from a network location using the NTFS file system made certain versions of Apache vulnerable to JSP source code disclosure in specific configurations. The main cause was unexpected behavior of the JRE API File.getCanonicalPath, stemming from inconsistencies in the Windows API FindFirstFileW. * If an HTTP/2 client connecting to Apache Tomcat exceeds the agreed maximum number of concurrent streams for a connection, a subsequent request could contain HTTP headers—including HTTP/2 pseudo headers—from previous requests rather than the intended headers. This could lead to users seeing responses for unexpected resources. Lastly, the payload length in WebSocket frames was not correctly validated in certain versions of Apache Tomcat. Invalid payload lengths could trigger an infinite loop, resulting in a DoS attack. |

## *Note.* This table shows the dependencies that were flagged for found Common Vulnerabilities and Exposures from the National Vulnerability Database.

5. Mitigation Plan

A comprehensive mitigation plan is essential to address the security vulnerabilities identified in Artemis Financial’s web application. This plan aims to enhance the overall security posture by implementing robust input validation, securing HTTP requests, strengthening access control mechanisms, preventing SQL injection attacks, improving cryptographic security, managing credentials securely, refining error handling, adopting a secure software development lifecycle, centralizing security management, and establishing a robust incident response plan.

Implementing strict input validation is the first critical step. By enforcing a whitelisting approach, only valid inputs are allowed, while restricting malicious data that could introduce vulnerabilities. Enforcing length restrictions on inputs can prevent attacks like denial of service (DoS) by limiting the amount of data processed. Tools like Apache Stretch can assist in input validation, enhancing the application's defense against malicious input. Additionally, rate limiting should be implemented to control the frequency and size of incoming requests, further protecting against DoS attacks.

Transitioning from HTTP to HTTPS is vital for encrypting data in transit and mitigating the risk of sensitive information being intercepted. Implementing HTTP Strict Transport Security (HSTS) ensures that HTTPS is always enforced, reducing the risk of man-in-the-middle attacks. Switching from GET requests to POST requests for handling sensitive data encapsulates the data within the request body, providing a higher level of security.

Securing credentials is imperative to prevent unauthorized access. Hard-coded database credentials and URLs must be removed from the codebase. Storing sensitive information like usernames and passwords in environment variables or secure vaults, such as HashiCorp Vault or AWS Secrets Manager, reduces the risk of exposure in the event of an attack.

Access control is a critical priority to ensure that only authorized users have access to sensitive functions and data. Implementing role-based access control (RBAC) or attribute-based access control (ABAC) dynamically manages user permissions. Utilizing frameworks like Spring Security allows for real-time role adjustments and permission updates. To resolve issues with obscurity in access control, hard-coded checks for localhost should be replaced with robust authentication and authorization mechanisms that ensure proper verification across environments. Encapsulating unauthorized access and limiting public modifiers in classes like DocData ensures that sensitive code is not exposed.

Addressing SQL injection vulnerabilities is essential for safeguarding the database. Query parameterization must be applied to all database interactions, particularly in classes like DocData that handle database connections. This technique ensures that user inputs are treated as data rather than executable code, preventing attackers from manipulating queries. Using frameworks such as Hibernate or Java Persistence API (JPA) supports secure query handling across the application. We may continue using Hibernate after updates and secure adherence is verified.

Strong encryption must be implemented across all communication channels between the client, server, and database to protect sensitive information during transmission and at rest. Upgrading cryptographic libraries like Bouncy Castle to the latest stable versions addresses known vulnerabilities such as AES timing attacks, weak HMACs in the BKS keystore, and robot vulnerabilities in RSA key exchanges. For BKS keystores created before version 1.47, upgrading to the newer format with a 160-bit HMAC is recommended. Integrating strong encryption libraries resolves the absence of cryptographic ciphers in the current implementation. Regularly updating dependencies ensures that the application benefits from security enhancements and patches provided by library maintainers.

Improving error handling prevents the leakage of sensitive system information. Ensuring that error responses are sanitized and that error details are logged securely without being exposed to the user is essential. Implementing a global exception handler captures and processes all errors in a secure manner, minimizing the risk of information exposure. This approach enhances the application's resilience against attacks that exploit error messages.

Adopting a Secure Software Development Lifecycle (SSDLC) integrates security into every stage of the development process. This includes regular security audits, continuous code reviews, and frequent testing to identify and address vulnerabilities early. Security training for developers and implementing security-focused design principles help ensure that Artemis Financial’s software remains resilient against emerging threats. Updating tools like the Maven Dependency Check plugin within the pom.xml file guarantees proper static testing during development cycles.

Establishing a centralized security management system handles all security operations, including access control decisions, cryptographic operations, and security policies. Centralizing security management ensures consistency across the application and reduces the overall attack surface. Technologies like Spring Security combined with OAuth2 help centralize authentication and authorization securely.

Implementing a robust incident response plan ensures quick and effective action in the event of a security breach. This plan includes monitoring for suspicious activity, alerting the security team, and maintaining logs for analysis. Tools like the ELK Stack or Splunk can be employed for real-time logging and monitoring, ensuring that any anomalies are quickly identified and addressed.

To further enhance security, additional measures should be implemented: Cross-Site Request Forgery (CSRF) protection by implementing synchronizer token patterns for transactional endpoints, requiring a unique token for each request to prevent CSRF attacks; Cross-Site Scripting (XSS) and clickjacking prevention by implementing X-Frame-Options headers, a Content Security Policy (CSP), and enabling XSS protection features within the application to prevent malicious scripts from being injected; codebase clean-up by removing unused local variables and standardizing package naming conventions to reduce potential vulnerabilities and ensure consistency; dependency management by updating the Maven Dependency Check tool to its latest version to analyze dependencies for the most accurate and up-to-date vulnerabilities.

By implementing this comprehensive mitigation plan, Artemis Financial can significantly enhance the security of its web application. Addressing vulnerabilities through input validation, secure communication protocols, robust access control, SQL injection prevention, improved cryptographic practices, secure credential management, refined error handling, updated dependencies, a secure development lifecycle, centralized security management, and a robust incident response plan will fortify the application against current and emerging threats. Consistent application of these measures will help protect sensitive data, maintain user trust, and ensure compliance with security standard