# CS 305 Project One (Kastigar)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/25/2025** | **Alex Kastigar** | **Original Document** |

## 

## Client



## Developer

Alex Kastigar

**1. Interpreting Client Needs**

Artemis Financial handles a lot of personal and financial information for their clients, like insurance and retirement data, so keeping that info safe is really important. If someone were able to access it, that could cause a lot of problems for the business and for customers.

They also might have clients in other countries, so it seems possible that some of their communication or transactions could be international. That might mean they have to follow different security rules depending on where their users live.

Some of the threats they could face are people trying to hack into the system, steal personal data, or even attack weak parts of the software, like input forms or outdated libraries. These things can change over time too, so the threats now might be different in a year.

Since they want to modernize their system, they’ll probably be using more open-source tools and newer tech like REST APIs. That’s great, but it also means they’ll need to stay on top of updates and security patches so old or vulnerable libraries don’t become a problem.

**2. Areas of Security**

Here are the areas I think are the most important for Artemis Financial’s app, based on what it does and what we’ve learned so far:

1. Input Validation: The app takes in a lot of user input, like names and document info. If that input isn’t checked properly, someone could try to mess with the system or sneak in harmful data.
2. Client/Server: Since it’s a REST API, everything is going back and forth between the server and the client. That data should be encrypted and handled safely.
3. Code Quality: There are a few things in the code that could be written better or cleaned up, like unused parameters and weird return values. Poor quality code could make it easier for someone to find holes in the app.
4. Encapsulation: I noticed that sensitive data like SSNs and bank info are stored in the code and not hidden when returned. That kind of info should definitely be protected.

Some of the other areas, like APIs or cryptography, might be more important later, but I didn’t see a lot of that being used in this version of the project.

**3. Manual Review**

While going through the code manually, I found several areas where security could be improved. Most of the issues were related to missing input validation, exposing sensitive data, or returning information that probably shouldn't be shared in an API response. Below are the specific findings, along with the file names and a short explanation of each one:

1. File: CRUDController.java

The method takes in a business\_name from the user but doesn’t validate it or even use it. Leaving unused input like this can be misleading for developers and could become a problem if the variable is used later without proper checks.

1. File: CRUDController.java

The controller returns doc.toString() from a new DocData object. If the toString() method reveals sensitive internal info, this could expose data that shouldn’t be part of the response.

1. File: customer.java

This class includes sensitive fields like ssn, accountNumber, and routingNumber, but doesn’t use any annotations like @JsonIgnore to hide them. If the object is returned in a response, those details could be exposed.

1. File: GreetingController.java

The user-provided name is taken from a query parameter and returned in the greeting without any sanitation. This creates a possible cross-site scripting (XSS) risk if someone inputs a script tag.

1. File: Greeting.java

This class returns both an id and content field as-is. If content includes user-submitted data and there’s no output encoding on the frontend, it could become another XSS risk.

1. File: DocData.java

The class allows setting the documentType with any string and doesn’t validate it. Without limiting the allowed types, this could lead to inconsistent behavior or security issues depending on how that value is used later.

1. File: CRUD.java

The constructor stores whatever is passed from doc.toString() into a variable. If that string includes system-level data or identifiers, it may leak internal structure or debugging info to the client.

1. File: myDateTime.java

This class returns a basic timestamp but isn’t used anywhere in the rest of the codebase. Leaving unused or unclear classes around can lead to confusion or missed security logic over time.

Overall, the code could benefit from better input validation, tighter control over what data is returned in responses, and more consistent handling of sensitive information. These kinds of changes would make the app more secure and also easier to maintain.

**4. Static Testing**

To identify security issues in Artemis Financial’s software application, I ran a static test using the OWASP Dependency-Check Maven plug-in. This tool scanned the codebase and flagged 16 third-party dependencies with known vulnerabilities. The scan completed successfully and generated an HTML report showing the vulnerabilities, their CVE identifiers, and brief descriptions.

Here’s a summary of the vulnerable dependencies detected:

1. bcprov-jdk15on-1.46.jar – An outdated cryptographic library that has vulnerabilities involving key recovery and padding oracle attacks. Vulnerabilities: 18 CVEs (i.e. CVE-2024-34447, CVE-2023-33201)
2. hibernate-validator-6.0.18.Final.jar - Potential issues with unsafe class loading and insecure input validation. Vulnerabilities: 2 CVEs (i.e. CVE-2023-1932)
3. jackson-databind-2.10.2.jar - Known deserialization vulnerabilities that could allow remote code execution. Vulnerabilities: 6 CVEs (i.e. CVE-2020-25649, CVE-2023-35116)
4. log4j-api-2.12.1.jar - Vulnerable to improper deserialization and remote connection issues in certain appenders. Vulnerabilities: 1 CVE (CVE-2020-9488)
5. logback-classic-1.2.3.jar - Logging component with vulnerabilities related to unsafe configurations and access control. Vulnerabilities: 2 CVEs (CVE-2023-6378, CVE-2021-42550)
6. logback-core-1.2.3.jar - A core logging dependency with additional class loading vulnerabilities. Vulnerabilities: 4 CVEs (i.e. CVE-2024-12798, CVE-2024-12801)
7. snakeyaml-1.25.jar - A YAML parser that can be exploited for remote code execution through unsafe deserialization. Vulnerabilities: 8 CVEs (i.e. CVE-2022-1471, CVE-2022-38752)
8. spring-boot-2.2.4.RELEASE.jar - Core Spring Boot library with vulnerabilities allowing attackers to bypass access restrictions. Vulnerabilities: 3 CVEs (i.e. CVE-2023-20873)
9. spring-boot-starter-web-2.2.4.RELEASE.jar - Framework component for web services; inherits multiple Spring-related risks. Vulnerabilities: 3 CVEs (i.e. CVE-2022-27772, CVE-2023-20883)
10. spring-context-5.2.3.RELEASE.jar - Vulnerable to various expression language and configuration injection attacks. Vulnerabilities: 12 CVEs (i.e. CVE-2022-22965, CVE-2023-20861)
11. spring-core-5.2.3.RELEASE.jar - Underlying framework code with several reflection and deserialization flaws. Vulnerabilities: 11 CVEs (i.e. CVE-2022-22968, CVE-2021-22096)
12. spring-expression-5.2.3.RELEASE.jar - Processes dynamic expressions; vulnerable to injection if not properly sanitized. Vulnerabilities: 12 CVEs (i.e. CVE-2024-38808)
13. spring-web-5.2.3.RELEASE.jar - Main Spring web module, vulnerable to multiple CVEs related to XSS, injection, and bypasses. Vulnerabilities: 16 CVEs (i.e. CVE-2024-38828, CVE-2021-22118)
14. spring-webmvc-5.2.3.RELEASE.jar - Spring’s model-view-controller framework; vulnerable to routing and injection attacks. Vulnerabilities: 12 CVEs (i.e. CVE-2024-38816, CVE-2023-20863)
15. tomcat-embed-core-9.0.30.jar – Core Tomcat functionality for embedded applications. Vulnerable to a wide range of issues including directory traversal, denial of service, and remote code execution. Vulnerabilities: 33 CVEs (i.e. CVE-2020-1938, CVE-2024-52316)
16. tomcat-embed-websocket-9.0.30.jar – WebSocket extension for embedded Tomcat. Shares many of the same vulnerabilities as the core module, including serious security risks tied to protocol handling and session hijacking. Vulnerabilities: 32 CVEs (i.e. CVE-2020-1938, CVE-2025-31651)

These results show that although the business logic in the code may seem straightforward, the underlying software stack carries a lot of risk. Many of these CVEs could lead to code execution, denial of service, or data leaks.

**Note on False Positives:**  
It’s also worth mentioning that not every vulnerability listed in the report is guaranteed to be exploitable in this exact project. Some may require certain configurations or input patterns that aren't present here. But relying on assumptions is risky, which is why updating and patching is almost always the safer route.

**5. Mitigation Plan**

Based on the manual review and static testing report, Artemis Financial’s web application is currently using multiple outdated libraries with known security vulnerabilities. While the application itself appears simple, the third-party dependencies introduce serious risk if left unaddressed.

Here are the recommended next steps for mitigation:

1. **Upgrade bcprov-jdk15on**. This cryptography library has over 15 known CVEs, including critical ones related to key exposure and insecure encryption. It should be updated from version 1.46 to the latest stable release (1.77 or later).
2. **Update all Spring-related libraries**. Most of the Spring packages (spring-web, spring-core, spring-webmvc, etc.) are flagged with high or critical vulnerabilities like CVE-2022-22965 (Spring4Shell). These should be upgraded together to a compatible newer version, such as Spring Framework 5.3.30 or Spring Boot 2.7.18+.
3. **Replace outdated logging libraries**. Versions of logback-classic and log4j-api in use have remote code execution and information disclosure CVEs. Upgrade to at least logback 1.4.x and verify logging configuration is hardened.
4. **Replace or patch jackson-databind**. The current version has multiple high-risk CVEs involving unsafe deserialization. Updating to jackson-databind 2.15.3 or newer is strongly advised.
5. **Remove or upgrade snakeyaml**. With multiple CVEs including denial-of-service and unsafe parsing, snakeyaml 1.25 should be replaced with version 2.0 or above. If it’s not actively used in the app, consider removing it entirely.
6. **Review and update embedded Tomcat**. The embedded version (9.0.30) has over 30 known vulnerabilities. Upgrading to tomcat-embed-core 9.0.85+ is critical to ensure secure request handling and avoid path traversal or injection issues.