# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **1/25/2025** | **Noelle Bishop** | **Completed sections 1, 2, 3, 4, and 5.** |

## Client



## Developer

Noelle Bishop

**1. Interpreting Client Needs**

Artemis Financial is a consulting company that deals with customers’ personal information to create financial plans for them. Client trust is crucial for the continued success of the company, as customers would likely not stay with or consider Artemis Financial in the event of a data breach. With loss of trust comes loss of profits, so secure software is a must. Considering the global average cost of a data breach is approximately $4.88 million, per IBM’s 2024 report, investing in security ahead of development can save the company money moving forward.

Artemis Financial may have clients globally, meaning they must comply with international standards and laws for data protection. It’s in their best interest to protect their clients’ data so as not to incur a fine or further legal action. For example, violating the General Data Protection Regulation (GDPR), which applies to companies that use data of people in the European Union (EU), results in significant fines in the event of a violating data breach.

Clients’ sensitive financial information is a possible target for attack. Cybercriminals may use this data to commit identity theft or fraud. Artemis Financial may be directly targeted via a ransomware attack in which criminals perform a distributed denial of service (DDoS) attack or implant malware with the intent of receiving a ransom.

Building a RESTful API for a modern web application can be made simpler by using existing frameworks, libraries, and open-source tools, such as the Spring framework for Java. However, these tools must be kept up-to-date and frequently checked for vulnerabilities to remain secure. Frequent static testing is a must, and in-code security considerations (like input validation) must still be implemented to protect against injection attacks, data tampering, and general software misuse.

**2. Areas of Security**

* Input Validation
  + This web application aims to provide an online experience for users of Artemis Financial. Features like creating an account, logging into an account, or submitting data into input fields are likely features given the nature of the services rendered. Any user-inputted data is untrusted data until it is validated and sanitized since it has the potential to perform injection or denial of service (DoS) attacks.
* APIs
  + This is a web application that uses a RESTful API, so ensuring its security is a crucial aspect of development. Straightforward measures can be checked for, like whether the software uses HTTPS rather than HTTP, whether API keys and endpoints are exposed, and whether user input is validated/sanitized before it’s processed.
* Cryptography
  + Given that the web application will be transferring and storing sensitive customer information, strong cryptography is essential. The application must use data encryption, API security, and user authentication controls. A lack of encryption could lead to unauthorized data access or tampering (i.e., a breach) and its costly consequences.
* Client/Server
  + This is a web application that transmits data between users (the client) and Artemis Financial (the server), so it’s a key area to consider for vulnerabilities.
* Code Error
  + It’s important that any errors or exceptions are caught and properly handled by the program to keep the software in control. Unreviewed error messages may give away more information about the innerworkings of the software than intended, possibly providing an “in” for hackers. Proper error/exception handling also provides a smoother user experience by displaying succinct, relevant error messages that guide the user’s actions.
* Code Quality
  + Code can run as intended and be error-free when tested, but that doesn’t mean it follows best practices for security or otherwise. The code must be reviewed to check for secured API features, input validation, and other defenses.

**3. Manual Review**

* APIs & Controllers
  1. Input is collected on line 16 of GreetingController.java, but it is not sanitized before it’s accepted. A lack of input validation equals an insecure endpoint, which can lead to an injection or DoS attack.
  2. The same issue is present on line 13 of CRUDController.java. The controller accepts input but does not validate it.
* Data Access & Models

1. The input accepted at line 13 of CRUDController.java is not validated and, instead, creates a new DocData object with the given input. This puts the database at risk of injection.
2. The database name, user’s username, and user’s password are exposed in the code. This can be seen clearly on line 27 of DocData.java and is direct data exposure that could lead to account theft.
3. The public function showInfo() in customer.java (lines 7-10) returns the customer’s account number when called. Considering this is sensitive information, it should be a private function to help prevent unauthorized access.

* Models

1. There’s a lack of input validation and error handling for the deposit(int a) function of customer.java (lines 12-14). Unintended error messages may reveal too much information, especially regarding a customer’s account\_balance, the variable being altered.

* Plug-Ins

1. The Bouncy Castle dependency is not up to date, resulting in many vulnerabilities discussed in Static Testing (pom.xml, lines 27-31).
2. The Spring Framework is not up to date, resulting in many vulnerabilities discussed in Static Testing (pom.xml, lines 5-10).

**4. Static Testing**

A screenshot of a computer

Description automatically generated

Figure - Dependency Report Overview

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dependency File | CVE(s) | Description/Risks | Solution | Attribution |
| bcprov-jdk15on-1.46.jar | 1. CVE-2024-34447 2. CVE-2016-1000338 3. CVE-2016-1000342 4. CVE-2016-1000343 5. CVE-2024-29857 6. CVE-2016-1000344 7. CVE-2016-1000352 8. CVE-2024-30171 9. CVE-2016-1000341 10. CVE-2016-1000345 11. CVE-2017-13098 12. CVE-2020-15522 13. CVE-2020-0187 14. CVE-2023-33202 15. CVE-2020-26939 16. CVE-2023-33201 17. CVE-2016-1000339 18. CVE-2015-7940 19. CVE-2018-5382 20. CVE-2013-1624 21. CVE-2016-1000346 22. CVE-2015-6644 | 1. Improper certificate validation 2. Possible signature injection 3. Same as #2 4. Weak key generation 5. Excessive CPU consumption 6. Unsupported/unsafe use of ECB mode 7. Same as #6 8. Timing-based leakage 9. Timing attack during signature generation 10. Padding oracle attack 11. “ROBOT”, i.e., exposure of private key 12. Exposure of private key information 13. Local information disclosure without privileges 14. OutOfMemoryError (DoS attack) 15. Sensitive information obtained through exception leak 16. LDAP injection vulnerability 17. Ability to leak information through table access 18. “Invalid curve attack” 19. Compromised BKS keystore 20. Distinguishing and plaintext-recovery attacks 21. Invalid keys reveal private key information 22. Exposure of sensitive information without authorization | Update to the latest version. | National Vulnerability Database (NVD), Open-Source Software Index (OSS Index) |
| hibernate-validator-6.0.18.Final.jar | 1. CVE-2023-1932 2. CVE-2020-1932 | 1. ‘isValid’ method bypass 2. Invalid EL expressions are evaluated as valid (bug) |
| jackson-databind-2.10.2.jar | 1. CVE-2020-25649 2. CVE-2020-36518 3. CVE-2021-46877 4. CVE-2022-42003 5. CVE-2022-32004 | 1. Improperly secured entity expansion 2. Java StackOverflow exception (DoS attack) 3. DoS attack using JsonNode JDK serialization 4. Resource exhaustion, deserialized untrusted data 5. Same as #4 |
| log4j-api-2.12.1.jar | 1. CVE-2020-9488 | 1. Man-in-the-middle attack due to improper certificate validation |
| logback-classic-1.2.3.jar | 1. CVE-2023-6378 2. CVE-2021-42550 | 1. DoS attack using poisoned data 2. Malicious file configuration, LDAP servers’ integrity loss |
| logback-core-1.2.3.jar | 1. Same as logback-classic-1.2.3.jar 2. CVE-2024-12798 3. CVE-2024-12801 | 1. Same as logback-classic-1.2.3.jar 2. Arbitrary code execution possible 3. Forged requests using compromised XML config files |
| snakeyaml-1.25.jar | 1. CVE-2022-1471 2. CVE-2017-18640 3. CVE-2022-25857 4. CVE-2022-38749 5. CVE-2022-38751 6. CVE-2022-38752 7. CVE-2022-41854 8. CVE-2022-38750 | 1. Remote code execution 2. Entity expansion during load operation 3. DoS possible due to lack of nested depth limitation 4. DoS possible via stack overflow 5. Same as #4 6. Same as #4 7. Same as #4 8. Same as #4 |
| spring-boot-2.2.4.RELEASE.jar | 1. CVE-2023-20873 2. CVE-2022-27772 3. CVE-2023-20883 | 1. Application deployed to Cloud Foundry susceptible to security bypass 2. Temporary directory hijacking 3. DoS attack possible using reverse proxy cache |
| spring-boot-starter-web-2.2.4.RELEASE.jar | Same as spring-boot-2.2.4.RELEASE.jar | Same as spring-boot-2.2.4.RELEASE.jar |
| spring-core-5.2.3.RELEASE.jar | 1. CVE-2022-22965 2. CVE-2021-22118 3. CVE-2020-5421 4. CVE-2022-22950 5. CVE-2022-22971 6. CVE-2023-20861 7. CVE-2023-20863 8. CVE-2022-22968 9. CVE-2022-22970 10. CVE-2021-22060 11. CVE-2021-22096 | 1. Remote code execution if using JDK 9+ application 2. WebFlux application vulnerable to privilege escalation 3. Reflected file download defense bypass 4. DoS attack via SpEL expression 5. DoS via authenticated user (if STOMP over WebSocket endpoint) 6. Same as #4 7. Same as #4 8. Case sensitivity mitigates protection 9. DoS possible involving data binding 10. Insertion of data using malicious input 11. Same as #10 |
| spring-expression-5.2.3.RELEASE.jar | 1. Same as spring-core-5.2.3.RELEASE.jar 2. CVE-2024-38808 | 1. Same as spring-core-5.2.3.RELEASE.jar 2. Same as Spring Core #4 |
| spring-web-5.2.3.RELEASE.jar | 1. Same as spring-core-5.2.3.RELEASE.jar 2. CVE-2016-1000027 3. CVE-2024-38809 4. CVE-2024-22243 5. CVE-2024-22262 6. CVE-2024-38828 | 1. Same as spring-core-5.2.3.RELEASE.jar 2. Remote code execution using deserialization of untrusted data 3. DoS vulnerability due to lack of size limit on request headers 4. Open redirect or server-side request forgery (SSRF) possible in specific situation 5. Same as #4 6. DoS vulnerability in controller parameter |
| spring-webmvc-5.2.3.RELEASE.jar | 1. Same as spring-core-5.2.3.RELEASE.jar 2. CVE-2024-38816 | 1. Same as spring-core-5.2.3.RELEASE.jar 2. Path traversal attack possible when using WebMvc or WebFlux |
| tomcat-embed-core-9.0.30.jar | 1. CVE-2020-1938 2. CVE-2020-11996 3. CVE-2020-13934 4. CVE-2020-13935 5. CVE-2020-17527 6. CVE-2021-25122 7. CVE-2021-41079 8. CVE-2022-29885 9. CVE-2022-42252 10. CVE-2023-44487 11. CVE-2023-46589 12. CVE-2020-9484 13. CVE-2021-30640 14. CVE-2022-34305 15. CVE-2023-41080 16. CVE-2021-24122 17. CVE-2021-33037 18. CVE-2023-42795 19. CVE-2023-45648 20. CVE-2024-21733 21. CVE-2019-17569 22. CVE-2020-1935 23. CVE-2020-13943 24. CVE-2023-28708 25. CVE-2021-43980 | 1. Remote code execution possible 2. High CPU usage (DoS attack) 3. OutOfMemoryException (DoS attack) 4. Multiple invalid payload lengths cause DoS 5. Possible information leak between requests 6. Same as #5 7. DoS caused by infinite loop 8. DoS risks when running “over any untrusted network” 9. Reverse proxy request smuggling possible 10. DoS caused by rapid stream reset (rapid request cancellation) 11. Same as #9 12. Remote code execution given a specific situation 13. Protection bypass when using the JNDI Realm of Apache Tomcat 14. Cross-site scripting (XSS) attack possible 15. Open redirect possible 16. Source code disclosure 17. Same as #9 18. Incomplete cleanup causes information leakage 19. Incorrect parsing of HTTP trailer headers, possible reverse proxy request smuggling 20. Error message generated containing sensitive information 21. HTTP request smuggling possible 22. Incorrect parsing allowing invalid HTTP headers to be read as valid 23. Violation of HTTP/2 protocol, possible information leak as a result 24. Lack of session cookies’ secure attribute 25. Concurrency bug that could lead to response/client mismatch |
| tomcat-embed-websocket-9.0.30.jar | 1. Same as tomcat-embed-core-9.0.30.jar 2. CVE-2020-8022 | 1. Same as tomcat-embed-core-9.0.30.jar 2. Incorrect default permissions |

**5. Mitigation Plan**

**Manual Review Solutions**

* Implement input validation for GreetingController.java, CRUDController.java, and customer.java at every point in which values are accepted from user input. Some general tips include:
  + Limit the length of all input to help prevent DoS attacks.
  + Disallow certain special characters known to be included in injection attacks when appropriate.
  + Convert all input to lowercase to avoid vulnerabilities regarding case sensitivity.
* Implement error and exception handling via try/catch blocks for every point of user input (same as input validation points mentioned above). This controls the flow of the software, provides an uninterrupted user experience (i.e., the software doesn’t crash), and prevents sensitive information from unintentionally leaking in default error messages.
* Obscure the exposed database name, username, and password data in DocData’s code by storing their values elsewhere, such as in an encrypted file.
* Update the public showInfo() function in customer.java to a private function.

**Static Testing Solutions**

* Update Spring Boot Starter Web to the latest version, 3.4.2, from the Maven Repository. The software is currently operating on version 2.2.4.
* Update Spring Boot Starter Test to the latest version, 3.4.2, from the Maven Repository. The software is currently operating on version 2.2.4.
* Update Bouncy Castle to the latest version, 1.80, from the Maven Repository. The software is currently operating on version 1.46.