CS 305 Project One

Document Revision History

Version	Date	Author	Comments
1.0	**7/18/2025**	**Tristyn Tate**	

Client

Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

- Respond to the five steps outlined below and include your findings.
- Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
- Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

Developer
Tristyn Tate

1. Interpreting Client Needs
Determine your client's needs and potential threats and attacks associated with the company's application and software security requirements. Consider the following questions regarding how companies protect against external threats based on the scenario information:

- What is the value of secure communications to the company?
- Are there any international transactions that the company produces?
- Are there governmental restrictions on secure communications to consider?
- What external threats might be present now and in the immediate future?
- What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

Artemis Financial needs strong security because they work with private client information like savings, retirement plans, and insurance. Keeping this data safe is important for building trust and following the law. Since the company may work with people in other countries, they have to follow international rules about how data is shared and protected. This means they also need to think about any government rules around using secure tools like encryption. Right now and in the near future, they could face threats like hackers trying to steal data, break into accounts, or mess with the website. As the company updates its system, they'll likely use tools and code made by other developers (called open-source libraries). They'll need to make sure those tools are safe and up to date. They also need to keep up with new technology like online apps and cloud services, which means putting strong protections in place to keep everything secure.

2. Areas of Security
Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial's software application. Justify your reasoning for why each area is relevant to the software application.

1. Input Validation
   This is important because users will be entering personal and financial information into the system. Without input validation, hackers could try to break the system using attacks like SQL injection or cross-site scripting (XSS).
2. Authentication
   Since users log in to access their financial accounts, strong authentication is needed to make sure only the right person can access their data. Weak login systems can lead to account takeovers.
3. Session Management
   Once a user logs in, their session must be securely managed to prevent session hijacking or unauthorized access. Sessions should time out after inactivity and be protected from being reused.
4. Access Control
   Not every user should be able to see or edit everything. Access control makes sure users only see their own financial data and that employees only access the parts of the system they're allowed to.
5. Data Protection
   Artemis handles sensitive data like social security numbers and financial records. This information should be encrypted when stored and when sent over the internet to keep it safe from hackers.

6. Error Handling and Logging
   The application should not show detailed error messages to users, as these can reveal too much about how the system works. Logging should be used to track issues and catch suspicious activity without leaking private data.

3. Manual Review
Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

After manually reviewing Artemis Financial's code base, several security vulnerabilities were identified. First, the LoginController.java file lacks input validation for the username and password fields, leaving the application open to SQL injection and script injection attacks. In UserService.java, passwords are stored using MD5, an outdated and insecure hashing algorithm that should be replaced with bcrypt or Argon2. The DatabaseConfig.java file contains hardcoded database credentials, which is a major security risk and should be replaced with environment variables or a secure vault. In AccountController.java, detailed error messages including stack traces are displayed to users, which can expose internal application logic and should be hidden in production. The TransactionService.java class fails to enforce proper access control, potentially allowing users to view other users' financial data. Additionally, SessionManager.java does not implement session expiration or regeneration, increasing the risk of session hijacking. The UserController.java file has an overly permissive CORS policy that allows all origins, which could allow malicious websites to make unauthorized requests. In AuditLogger.java, sensitive user data is logged in plaintext, which could be exposed if logs are accessed improperly. The FileUploadHandler.java class does not validate uploaded file types, leaving the application open to file-based attacks. Lastly, PaymentService.java lacks rate limiting and CAPTCHA protection, making it vulnerable to brute-force and automated bot attacks. These issues must be addressed to ensure the application meets modern security standards.

4. Static Testing
Run a dependency check on Artemis Financial's software application to identify all security vulnerabilities in the code. Record the output from the dependency-check report. Include the following items:

- The names or vulnerability codes of the known vulnerabilities
- A brief description and recommended solutions provided by the dependency-check report
- Any attribution that documents how this vulnerability has been identified or documented previously

Based on the CS 305 Vulnerability Assessment Process Flow Diagram (Text-Only Version), the security assessment for Artemis Financial's web application should follow a clear and structured approach starting with an Architecture Review. Given that the application is a RESTful API handling sensitive financial data, multiple security areas require thorough assessment.
The flow diagram directs the evaluation through seven key areas:

1. Input Validation
   Users enter personal and financial information, so input validation is essential to prevent attacks such as SQL injection and cross-site scripting (XSS).
2. APIs
   Since Artemis Financial relies heavily on RESTful APIs, securing these endpoints through authentication, authorization, and rate limiting is vital to prevent unauthorized access.

3. Cryptography
   Sensitive data like passwords and personal details must be securely hashed and encrypted. Currently, insecure hashing (MD5) and some unencrypted data storage highlight weaknesses that need to be addressed.
4. Client/Server
   Secure communication channels (e.g., HTTPS) and proper session management prevent interception and hijacking attacks.
5. Code Error
   Detailed error messages are shown to users, exposing system internals. These must be sanitized to avoid revealing potential attack vectors.
6. Code Quality
   Poor coding practices such as hardcoded credentials and insufficient modularity can introduce vulnerabilities and complicate maintenance.
7. Encapsulation
   Sensitive information is sometimes exposed in logs or through interfaces where it should be hidden, increasing the risk of data leaks.

After manual code review and static testing, several known vulnerabilities were discovered in third-party libraries. These vulnerabilities have been widely documented and represent significant security risks:

- CVE-2022-22965 – "Spring4Shell" (library: spring-webmvc)
  This critical vulnerability allows remote code execution through data binding in the Spring Framework. It is documented in the National Vulnerability Database (NVD) and requires upgrading Spring Framework to version 5.3.18 or higher.
- CVE-2020-9484 – Apache Tomcat Deserialization RCE (library: tomcat-embed-core)
  An insecure deserialization flaw enables attackers to execute arbitrary code via crafted session data. The recommended fix is upgrading to Tomcat 9.0.37 or later, as noted in NVD and multiple static analysis tools.
- CVE-2021-29425 – Directory Traversal in Apache Commons IO (library: commons-io)
  This vulnerability allows unauthorized file access through crafted paths, especially in file upload features. Upgrading to Apache Commons IO 2.8.0 or later is advised, per NVD documentation.
- CVE-2021-22118 – Spring Core Information Leak (library: spring-core)
  Improper class loading exposes internal application structures, aiding attackers in planning further attacks. Upgrading to Spring Core 5.3.6 or higher addresses this issue, as detailed in Spring Security advisories and NVD.
- CVE-2019-17571 – Log4j SocketServer Deserialization Vulnerability (library: log4j)
  Remote attackers can exploit Log4j's SocketServer via crafted data to execute code. Migrating from Log4j 1.x to Log4j 2.x is essential, according to Apache Log4j advisories.

Following the process flow, the architecture review and static testing outputs guide where to focus manual code reviews, leading to a comprehensive understanding of the application's vulnerabilities. The final step in the process involves compiling a summary of findings and a mitigation plan that addresses each identified issue through updating dependencies, improving coding practices, enforcing security controls, and strengthening communication protocols.
This systematic approach ensures Artemis Financial's software application is evaluated thoroughly and that actionable steps are taken to improve its security posture.

5. Mitigation Plan

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial's software application.

Based on the results of the manual code review and the static testing using the dependency-check plugin, Artemis Financial's software application has several serious security vulnerabilities that need to be addressed to protect sensitive financial data and meet modern cybersecurity standards. The manual review uncovered issues such as lack of input validation, use of weak password hashing (MD5), hardcoded database credentials, overly permissive CORS settings, and missing access controls. The static testing revealed that several third-party libraries in the application—such as Spring, Tomcat, Apache Commons IO, and Log4j contain known vulnerabilities like remote code execution, insecure deserialization, and information leakage.

1. Input Validation
   - Add strong server-side validation using a validation framework (e.g., Hibernate Validator) to sanitize all user input fields.
   - Use allowlists rather than denylists to block unsafe input patterns.
2. Password Security
   - Replace MD5 hashing with strong algorithms such as bcrypt, PBKDF2, or Argon2.
   - Implement password salting and enforce strong password policies.
3. Hardcoded Credentials
   - Remove hardcoded database usernames and passwords from code files.
   - Store credentials securely using environment variables or a secure secrets management tool (e.g., HashiCorp Vault or AWS Secrets Manager).
4. Dependency Updates
   - Upgrade all vulnerable third-party libraries to the latest stable versions:
   - Spring Framework → 5.3.18 or later
   - Tomcat → 9.0.37 or later
   - Apache Commons IO → 2.8.0 or later
   - Replace Log4j 1.x with Log4j 2.x
   - Set up automated tools like Dependabot or OWASP Dependency-Check in the CI/CD pipeline for ongoing vulnerability tracking.
5. Error Handling
   - Replace detailed error messages with generic messages for users.
   - Log full error details securely on the server side only, not exposed to clients.
6. Session Management
   - Set secure timeouts and invalidate sessions after logout or inactivity.
   - Regenerate session IDs upon login to prevent session fixation attacks.
7. Access Control
   - Enforce strict access controls to ensure users only access their own data.
   - Use role-based access control (RBAC) to manage permissions.
8. CORS Policy
   - Limit cross-origin requests by setting specific allowed origins instead of using *.
   - Enable CORS only on endpoints that need it, and use HTTPS.
9. Secure File Uploads
   - Validate file types and scan uploads for malware.
   - Store uploaded files in non-executable directories.
10. Rate Limiting and Bot Protection
   - Add rate limiting on critical endpoints (like login and payment).
   - Use CAPTCHA or reCAPTCHA to reduce automated attack risks.

By following these mitigation steps, Artemis Financial will significantly improve the security of its web application, reduce exposure to known attacks, and better protect its clients' sensitive financial information. Regular security testing and monitoring should also be part of the ongoing software development lifecycle.