CS 305 Project One Template

Document Revision History

Version	Date	Author	Comments
1.0	11/12/2025	Prithvi Ghale	

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

Prithvi Ghale

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?

The Artemis Financial needs its RESTful API and web applications to be modern, very secure, and also protected from any outside attacks. This is because the company handles financial plans such as savings, retirement, investment, and even insurance, so this makes the value of secure communication at a very high level. The client data includes all their personal and also their financial details in it, so the encryption and data protection it must prevent any leaks or theft from happening.

The company might serve international clients, so this means its communications system should follow the U.S. and also the global privacy and data transfer laws like GDPR and PCI DSS. If there are any international transactions, those transactions should be encrypted with HTTPS/TLS and also verified digital certificates.

Some of the possible attacks or threats might be phishing, cross-site scripting (XSS), SQL injection, insecure dependencies, and even man-in-the-middle (MITM) attacks. So in the future, evolving threats like API exploitation and zero-day vulnerabilities may also appear.

Having modernization may require replacing weak or old libraries, adding secure open-source tools with version control, and also keeping software up to date. The web app should start following secure-coding practices and include validation, authentication, and even encryption everywhere.

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.

For the area of security, the main areas I think of that apply to Artemis Financial's web app are:

- Communication Security: Here, I think that secure communications is one of the most important priorities to have as customer data travels all over the internet, so using HTTPS and TLS helps make sure that no one else can look at that information or change anything in the information while it's moving.
- Data Validation and Input Handling: This is another important point because any unfiltered input could end up leading to an attack like XSS or SQL injections. So, since the customers write their names, ID numbers, and other information that they give, the system should always strictly check and clean this input. Doing this helps a lot to stop any attacks from happening or stop attackers from injecting harmful code.
- Authentication and Access Control: For me, this point is another important area. This is because only the right users should be able to access the financial data. So, without having strong access control, anyone could easily pretend to be a user or even an admin and view sensitive information. So, I think this protects both the company and also the customers.
- Dependency Management: I feel like outdated or risky libraries are one of the most common ways an attacker gets in. We can see that Artemis Financial uses open source tools, so this is why I feel like keeping dependencies updated is essential.
- Error Handling and Logging: Another point that is important, because I believe that any errors that occur should be all handled very safely. A mistake on the backend should not

show any detailed information to the user, and this is important as any attackers can use that information to figure out how the system really works. So, Logging is still important, but it should be kept private and very secure.

- Configuration Security: I think that configuration settings play a bigger role than most people think. Different things like default accounts, weak passwords, open ports, or unsecured admin settings, all of these can easily be exploited. So, I feel like a proper configuration will make the app harder for Encryption attackers to attack.
- Encryption: Since we know that Artemis Financial stores things like financial plans and personal details, I think encrypting data both in transit and at rest is necessary. Even if someone intercepts or steals the database, encryption will help keep all the information unreadable for the attackers.

So overall, all these areas are very relevant because Artemis Financial's software deals with very sensitive financial information shared over the internet. So, protecting data in every stage, like input, processing, and storage, is essential.

3. Manual Review

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

So, after I was done looking through the Java files and the project structure, I did find some issues that I feel like could create security issues. Some of them are:

- Missing HTTPS enforcement: I did notice that the application did not force users to use HTTPS, and in my opinion, this is very risky to do because people could accidentally use an unsecured HTTP connection, which means that someone could read or change the data being sent.
- Unvalidated User Input in GreetingController: In the GreetingController, @RequestParam takes the user's input and directly turns it into a message. I feel like this is a risky situation, as the app does not check or clean that input. This means that anyone could enter a harmful code, which could lead to an injection attack.

- No Authentication or Authorization: As of right now, I can say that anyone can access the endpoints without logging in. Since this is a financial company, I think that it is a huge problem as sensitive information should only be visible to the users who are allowed to see it and not the public to see.
- Lack of Proper Error Handling: The project does not have any error pages. So, if something were to go wrong, Spring Boot might show the technical information, and I feel like, because of this, attackers could use those details they gained to learn more about the system, and that is very concerning.
- Outdated Dependencies: Before updating the POM, the project was using an older version of Spring Boot and other libraries. I think that having outdated libraries is one of the easiest ways attackers can get in, as they often contain known CVEs.
- Missing Security Headers: Another thing that I noticed is that the app does not set common security headers like Content-Security-Policy, X-Frame-Options, or even CORS restrictions. So, I think that these headers are simple protections that block many basic attacks, and missing them makes the app even more weaker.
- Hardcoded Values in Code: In the Java files, some values, like static strings and also the configuration details, are all written directly inside the Java files, and I feel like this is not very secure to do. For example, let's say someone got access to the code; they would automatically see everything. So, because of this, these should be stored in configuration files instead.

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

After I was done running the Maven Dependency Check plugin, I found known vulnerabilities, which are:

- CVE-2022-22965: This is a very serious remote-code-execution vulnerability in older versions of the Spring Framework. The report flagged this because the project was using a Spring Library, which was outdated. The reason I think this is dangerous is that this flaw lets attackers run their own code on the server, which could completely compromise Artemis Financial's system. For this to be fixed, all we need to do is upgrade to Spring Boot to the latest version available.
- CVE-2021-44228 — Log4j: The dependency check report also pointed out the Log4j issues, which are very common. What this does is that this vulnerability allows attackers to trigger harmful code just by sending a special string into the logging system. This attack can be very dangerous as it has been used worldwide, and it doesn't require authentication. In order to fix this issue, all we have to do is update Log4j to 2.17+ or remove Log4j completely if the project has no use for it.
- CVE-2020-36518 (Jackson-Databind): This is a serialization issue that allows attackers to execute harmful code, and to fix this issue, we have to do is to upgrade to the latest version of Jackson.
- CVE-2022-22950 (Spring Expression Language): So, what this is a vulnerability that allows malicious users to inject harmful expressions if their input is not validated correctly. This is harmful because the project does not validate input in some controllers. I think that this could actually be exploited. To fix this, we have to update Spring Framework to a patched version and also add proper input validation and sanitization.

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.

To fix the vulnerabilities and keep Artemis Financial's system secure, we have to:

- Enforce HTTPS and TLS 1.3: I want to make sure that all the communication is fully secure, since Artemis deals with financial data. So, using HTTPS with the latest TLS version protects users from attackers trying to read or change any of their information. For this to be fixed, we can configure the server to automatically redirect all HTTP requests to HTTPS and use TLS 1.3.

- Validate and Sanitize User Input: As we know that the app takes users' input, like their name parameter in the /greeting endpoint, I think that it is very important to prevent any harmful input, such as scripts or SQL. So, Input validation stops XSS and injection attacks before they even happen. To fix this, we have to use Spring's "@Valid", input filtering, and length/type checks for all the user-provided data.
- Add Authentication and Authorization: I think that right now, anyone can get access to the endpoints, and this is because Artemis stores personal and financial details. So, I think that adding a user login and role-based access is very necessary so that this way only the right person can view or even change any sensitive data. To prevent this, we can enable Spring Security with username/password login and roles.
- Upgrade Dependencies Regularly: I noticed that the dependency check report showed the old Spring Log4j and also Jackson versions with known vulnerabilities. So I feel like this is a big risk, as any attackers or hackers are known to often target outdated libraries. Sto fix this, we just have to update all the dependencies through Maven and set up automatic dependency scans.
- Add Proper Error Handling: In the app, it still shows the White label error page, which can reveal technical details. So, I think that a clean custom error page is much safer because it helps to protect information about the system from anyone. For this, we can create custom error pages and avoid showing any stack traces to the users.
- Enable Security Headers: To protect the frontend from clickjacking, cross-site scripting, or even other web stacks, modern applications need secure headers. So, adding headers like these can help create an extra barrier of protection:

Content-Security-Policy

X-Frame-Options

X-Content-Type-Options

CORS rules

- Remove Hardcoded Values: These hardcoded strings and the values make it so the app is harder to maintain, and it can also expose information if someone looks into the code, which is something we don't want. So I think that sensitive data should always be externalized. For this, we can over secrets in environment variables or application.properties.
- Keep Running Security Scans: I think that scanning the application regularly helps Artemis stay ahead of any attackers, as new vulnerabilities appear all the time.