# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/14/25** | **Michael Ford** | **Vulnerability Report** |

## Client



## Developer

Michael Ford

**1. Interpreting Client Needs**

For this project, I focused on understanding what the client needs in terms of protecting their software and communications. Secure communication is very valuable to any company because it keeps customer data safe and helps build trust. If a company is sending information back and forth between systems or to users, it needs to make sure that data isn’t being seen or changed by someone else. That’s especially true if they deal with financial data or personal info.

If the company does business across different countries, international transactions need to be protected with strong security. There might also be government rules they have to follow depending on where they do business. These rules can affect how encryption is handled or what tools are allowed.

There are also many external threats that companies face now, like hackers trying to get into systems through known software bugs or outdated code. Soon, these threats will probably grow as technology keeps changing.

Modernization is also important. Many applications now use open-source libraries, which are helpful but can introduce risks if they’re not kept up to date. Companies need to pay attention to the tools and frameworks they use and update them often to avoid security problems. Web technologies also evolve quickly, so developers have to stay current to protect their systems properly.

**2. Areas of Security**

When I reviewed Artemis Financials’ web application, I noticed several areas of security that really matter based on what the app is doing.

First, I looked at how the application is set up overall, this is called architecture. It’s important to make sure the design makes sense and doesn’t leave any openings for attackers.

The app also takes input from users, like names or IDs, so input validation is important. If we don’t check that input carefully, someone could enter something harmful that breaks the system or sneaks in.

Since the app uses APIs to share information, those also need to be protected. If anyone can reach those APIs, they might try to steal data or overload the system.

Because the app runs on a server and talks to users through the internet, we also have to think about the connection between the client and server. It should be protected so no one can see or change the data while it’s moving.

I also saw that some parts of the code don’t handle errors the right way. If something goes wrong, the app might crash or show information it shouldn’t.

Some code could be written better, like making sure variables aren’t just public for anyone to change. That’s part of keeping code quality high and protecting how information is stored and used.

I didn’t see any encryption in use yet, but I know that will be important once this app starts handling real customer data.

**3. Manual Review**

I went through the code and found these possible issues:

* In *GreetingController.java*, the user’s input is passed straight into the parser. This could be risky if someone tries to use harmful input.
* In *customer.java*, some fields are public when they should be private. Making them private helps protect the data better.
* In *CRUDController.java*, I didn’t see any checks for what kind of data the user can send in. This could cause problems if someone enters something unexpected.
* There’s no login or security checks in the code, so anyone could use the system without being verified.
* The *application.properties* file allows files up to 200MB. That seems too high and could let someone overload the system.
* The *CRUD.java* file doesn’t handle errors using try/catch blocks, so the program could crash if something goes wrong.
* I didn’t see anything that forces the app to use HTTPS, which could put data at risk during transfer.
* The app doesn’t clean up or check input from the URL, which could be dangerous if used in other parts of the code.
* The file path for uploads is hardcoded, which might not work well on different computers or setups.
* The *myDateTime.java* file has confusing names, which could make it harder to keep the code organized.

**4. Static Testing**

**A screenshot of a computer

AI-generated content may be incorrect.**

After running the dependency check, I was able to see that the application had several known vulnerabilities. The report showed the names and CVE codes of the affected files. Here are some of the key vulnerabilities found:

* **bcprov-jdk15on-1.46.jar** – This had multiple CVEs like CVE-2024-34447 and CVE-2016-1000338. It’s from Bouncy Castle, and it has been flagged for security issues related to cryptography. The recommended fix is to update to a newer version.
* **jackson-databind-2.10.2.jar** – Found issues like CVE-2022-42003 and CVE-2020-25649. These are related to deserialization problems that could allow someone to run code they shouldn’t. Updating to the latest version is suggested.
* **log4j-api-2.12.1.jar** – Had CVE-2020-9488. This vulnerability could let attackers mess with log files. The fix is to update to a safer version.
* **spring-web-5.2.3.RELEASE.jar** – Several CVEs came up here too, including CVE-2022-22965, which is known as Spring4Shell. This can allow remote code execution. The report recommends updating Spring to a newer release.
* **snakeyaml-1.25.jar** – Found CVE-2022-1471 and a few others. These deal with unsafe data loading from YAML files. Updating the library would help fix it.

The dependency-check tool pulled this information from the National Vulnerability Database (NVD), where each of these issues is documented and tracked using a CVE number. That’s how we know these problems are real and have been recognized by others in the cybersecurity community.

**5. Mitigation Plan**

To help fix the vulnerabilities I found through manual review and the dependency-check report, I would first remove or update any outdated libraries. For example, the bcprov-jdk15on dependency is known to have many security issues, so I would replace it with a newer, safer version. I would also avoid using Spring Expression Language directly on user input. Instead, I’d use plain string formatting to safely include user-provided values. Any user input throughout the application should be validated and sanitized to prevent injection attacks. In the future, I would make sure to use strong, updated dependencies and regularly run security scans like the OWASP Dependency-Check. I’d also suggest keeping documentation of any known vulnerabilities that were fixed, so the development team can stay ahead of new threats. By regularly testing the software, writing safer code, and keeping the dependencies up to date, we can make Artemis Financials’ application much more secure.

**References:**

OWASP Foundation. (n.d.). *Dependency-Check*. <https://owasp.org/www-project-dependency-check/>

National Institute of Standards and Technology. (n.d.). *National Vulnerability Database (NVD)*. <https://nvd.nist.gov/>