# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/28/2024** | **Rachel Siminski** |  |

## 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

Rachel Siminski

**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 is a company which deals directly with client finances, including bank accounts and insurance information. Due to the highly sensitive nature of the data Artemis Financial handles, secure communication and data handling will be imperative. It is possible that Artemis Financial will handle international transactions, either on behalf of the clientele, or by working with international companies and investment brokers. With regards to governmental restrictions, there is the Gramm-Leach-Bliley Act which requires companies to safeguard sensitive data, the General Data Protection Regulation for the EU which lays out the general regulations for handling, processing, and communicating data, and more (FTC, n.d.)(Intersoft Consulting, n.d.). There are many regulations to consider, some which deal directly with financial data, and some which are general regulations. When handling customer data, especially financial and personal data, there is always the threat of data breaches, via SQL injection, HTML injection, and more. The existence of vulnerabilities in software offers malicious characters to access the internal sections of a program, and potentially to steal sensitive data. Open-source libraries can be incredibly beneficial, but it is important that the ones chosen are secure and up to date. Another potential threat comes from insecure API’s; therefore, it would be important to stay up to date on secure RESTful API guidelines.

**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.

Due to the nature of Artemis Financial’s business, and the data which will be processed and stored by the software, each area of security should be thoroughly considered.

1. Input validation: Artemis Financial will need to store personal and financial data for their clients, meaning there will most likely be a database where this data is held. SQL injection happens when an attacker uses malicious input to essentially ‘trick’ the database into providing data. This can be used to collect information for every client, but validating and sanitizing input before using it within the program can mitigate this threat.
2. APIs: An API is the intermediary between two applications, and it sets the regulations for how these applications communicate. When an API is insecure, whether it be through weak encryption or lack of endpoint protection, it can and will lead to vulnerabilities within the system, allowing malicious characters to access sensitive data. This can be mitigated by ensuring that the API is built securely and by following RESTful API guidelines for a robust API. (Akamai, n.d.)
3. Cryptography: In the event of a data breach, having encrypted data can help mitigate the harm done to clients. A malicious character can exploit a vulnerability and get inside the system, and steal the data, but can’t read or use it. This requires strong encryption to be properly effective, and should be considered a preventative measure, not to be taken lightly. The system should still be made as securely as possible.
4. Client/Server: A system should never assume that another system is as secure and should only share what is vital for a transaction to process correctly, the Principle of Least Privilege. Additionally, “The principle of secure distributed composition states that the composition of distributed components that enforce the same security policy should result in a system that enforces that policy at least as well as the individual components do” (Benzel, et al.).
5. Code Error: As mentioned above, SQL injection occurs when an attacker uses malicious input to cause an error in the system which can result in a database providing sensitive data. What happens here is that the system receives unexpected input and thus behaves in an unexpected way. It’s important to handle errors and unexpected behavior securely, to avoid a system getting ‘tricked’ into providing data to an attacker, or even to a user who has simply made a mistake.
6. Code Quality: The code for the system should be created with security in mind, rather than having security tacked on at the end. Secure coding can help ensure that vulnerabilities or errors are caught early and corrected, before the system is launched and exploited.
7. Encapsulation: This refers to setting up the program to hide data inside a class then using ‘get’ and ‘set’ methods to access and modify the data. This helps to avoid the data being modified unexpectedly, and it allows the program to handle data on its own, rather than allowing users to access internal data. (Busbee & Braunschweig)

**3. Manual Review**

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

1. CRUDController.java:
   1. **public** CRUD CRUD(@RequestParam(value="business\_name") String name)
   2. This line is potentially vulnerable at @RequestParam, if value is not properly validated before use.
2. customer.java:
   1. **public** **int** showInfo()
   2. This method returns the customer account number, which can possibly result in a leak, because there is no access control for who is allowed to ‘showInfo’, and the account number can potentially be logged.
   3. **public** **void** deposit(**int** a)
   4. This method adjusts the account balance in the event of a deposit, but there is no input validation used in this method, meaning that user input is potentially being used here without being secured first.
3. DocData.java:
   1. **public** String getId
   2. This method returns id, which could potentially result in a leak due to a lack of access control and potential logging of the id value.
   3. **public** **void** read\_document(String key, String value)
   4. This method is attempting to read a document from the database using a key and a value. As it is now for testing the key and value are hardcoded as root, which should not be the case once the full system is complete. Additionally, there is no input validation implemented for the key and value. These must be validated before use, otherwise the system will be vulnerable to SQL injection.
4. GreetingController.java:
   1. **public** Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name)
   2. As mentioned above, @RequestParam can be vulnerable to attack if value and defaultValue are not properly validated before use.
5. myDateTime.java:
   1. **public** **class** myDateTime
   2. This method seems straightforward and simple, but it has been discovered that there are ways for attackers to exploit the channels through which atomic time is disbursed. These atomic time transmissions are unvalidated, allowing attackers to spoof the time and “cause transmissions to bump into each other and break the system” (Winder, 2022). This is a new discovery, so there is not a known way to mitigate this potential threat.

**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

1. **bcprov-jdk15on-1.46.jar**
   1. Improper Validation of Certificate with Host Mismatch.
      1. “In Bouncy Castle JCE Provider version 1.55 and earlier the DSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure.”
   2. **CVE-2024-34447**
      1. <https://www.cve.org/CVERecord?id=CVE-2024-34447>
2. **hibernate-validator-6.0.18.Final.jar**
   1. Bug in Hibernate’s Bean Validation.
      1. “A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.”
   2. [**CVE-2020-10693**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-10693)
      1. <https://www.cve.org/CVERecord?id=CVE-2020-10693>
3. **jackson-databind-2.10.2.jar**
   1. Entity expansion is not secured properly.
      1. “A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity.”
   2. [**CVE-2020-25649**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649)
      1. <https://www.cve.org/CVERecord?id=CVE-2020-25649>
4. **log4j-api-2.12.1.jar**
   1. Improper Validation of Certificate with Host Mismatch.
      1. “Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender.”
   2. **CVE-2020-9488**
      1. <https://www.cve.org/CVERecord?id=CVE-2020-9488>
5. **logback-core-1.2.3.jar**
   1. Attacker can send poisoned data and initiate a Denial-of-Service attack.
      1. “A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data.”
   2. **CVE-2023-6378**
      1. <https://www.cve.org/CVERecord?id=CVE-2023-6378>
6. **snakeyaml-1.25.jar**
   1. SnakeYaml’s Constructor() class does not restrict types which can be instantiated during deserialization.
      1. “SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization… Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond. CWE-502 Deserialization of Untrusted Data, CWE-20 Improper Input Validation”
   2. **CVE-2022-1471**
      1. <https://www.cve.org/CVERecord?id=CVE-2022-1471>
7. **spring-boot-2.2.4.RELEASE.jar** & **spring-boot-starter-web-2.2.4.RELEASE.jar**
   1. Applications deployed to Cloud Foundry could be susceptible to a security bypass.
      1. “In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+. NVD-CWE-noinfo”
   2. **CVE-2023-20873**
      1. <https://www.cve.org/CVERecord?id=CVE-2023-20873>
8. **spring-core-5.2.3.RELEASE.jar** & **spring-webmvc-5.2.3.RELEASE.jar**
   1. CISA Known Exploited Vulnerability: vulnerable to remote code execution via data binding.
      1. “A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it. CWE-94 Improper Control of Generation of Code ('Code Injection')”
   2. **CVE-2022-22965**
      1. <https://www.cve.org/CVERecord?id=CVE-2022-22965>
9. **spring-web-5.2.3.RELEASE.jar**
   1. Potential remote code execution if used for Java deserialization of untrusted data.
      1. “Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or not occur, and authentication may be required. NOTE: the vendor's position is that untrusted data is not an intended use case. The product's behavior will not be changed because some users rely on deserialization of trusted data. CWE-502 Deserialization of Untrusted Data”
   2. **CVE-2016-1000027**
      1. <https://www.cve.org/CVERecord?id=CVE-2016-1000027>
10. **tomcat-embeded-core-9.0.30.jar** & **tomcat-embed-websocket-9.0.30.jar**
    1. CISA Known Exploited Vulnerability: Improper privilege management.
       1. “When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users.”
    2. **CVE-2020-1938**
       1. <https://www.cve.org/CVERecord?id=CVE-2020-1938>

**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.

The security risks discovered during this report can be mitigated by ensuring that external libraries or software used are up to date, implementing effective input validation and sanitation techniques, encrypting sensitive data, utilizing access control, and avoiding sensitive data logging to avoid data leaks. There were some vulnerabilities which have been known to be exploited, so these should be given special attention to avoid exploitation.

# References

Akamai. (n.d.). *What are API Security Risks*. Retrieved from Akamai: https://www.akamai.com/glossary/what-are-api-security-risks#:~:text=The%20five%20common%20security%20risks,and%20exploited%20by%20malicious%20actors.

Benzel, T. V., Irvine, C. E., Levin, T. E., Bhaskara, G., Nguyen, T. D., & Clark, P. C. (n.d.). *Design Principles for Security.*

Busbee, K. L., & Braunschweig, D. (n.d.). *Programming Fundamentals – A Modular Structured Approach, 2nd Edition.*

FTC. (n.d.). *The Gramm–Leach–Bliley Act Privacy of Consumer Financial Information*. Retrieved 5 30, 2024, from FTC: http://www.ftc.gov/privacy/glbact/glboutline.htm

Intersoft Consulting. (n.d.). *General Data Protection Regulation*. Retrieved from info.eu: https://gdpr-info.eu/

Winder, D. (2022, April 21). *Can Time Be Hacked? Here’s How One Hacker Demonstrated It Can*. Retrieved from Forbes: https://www.forbes.com/sites/daveywinder/2021/11/20/can-time-be-hacked-heres-how-one-hacker-demonstrated-it-can/?sh=76049bc44ad5