# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **26MAY2024** | **Dan Peterson** |  |

## 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

Dan Peterson

**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 consults with clients and develops individualized financial plans. The financial plans include but are not limited to savings, retirement, investments, and insurance. This type of data is particularly vulnerable to attack because the attacker potentially gains the ability to steal identities or money from the client or their clients. Artemis Financial provides its services worldwide and must be able to adapt to different government and healthcare regulations. Potential clients include government agencies, entrepreneurs, businesses, hospitals, and medical offices. Artemis should follow guidelines presented by OWAPS secure coding practices. Dependencies of the REST API show that Artemis should be concerned with CBC padding, input validation, denial of service via large-depth nested objects, deserialization of untrusted data, XML Entity Expansion, Code Injection, resource exhaustion, improper handling of case sensitivity, allocation of resources without limits, and HTTP request smuggling.

**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.

The security of the application must be strictly considered during every step of the software. This application will be used worldwide. This means that it must utilize the internet to transfer data. The use of the internet to transfer data makes security even more important. There will likely be a multitude of users thus every aspect of the seven steps is paramount. development lifecycle.

Input Validation:

The dependencies of this application are particularly vulnerable to code injection and verification of untrusted data. Thus, steps should be taken to determine that all input is safe, and error-free. Input validation should be applied to all data input by the user. It should also use a specific set of characters and should be limited to maximum and minimum lengths.

APIs:

APIs are the way that applications communicate with each other. All APIs should be validated, and all dependencies must be checked for vulnerabilities. All vulnerabilities must be addressed in the best way possible. Developers must prevent the application from directly controlling the operating system. Developers must also develop security measures on their own or utilize previously established security measures that are not third-party. The use of third-party code can be effective, but it also opens the application up to known vulnerabilities that are not yet fixed or that were nefariously implanted by the developer of that code.

Cryptography:

Due to the sensitive nature of the data handled by the application encryption is a must. The developers should use in-house encryption rather than third-party encryption. This will prevent the data from being vulnerable to known ciphers.

Client/Server:

The nature of the application means that it will have a high rate of traffic between clients and the server. Multiple levels of access to the server should be implemented based on the company's determined need for everyone. All user access should be validated and authorized. The latest version of Transport Layer Protection (TLS) should be considered to implement the latest methods of authentication and authorization.

Code Error:

Error messages presented to the user should be generic enough to protect information. All errors (regardless of user) should be logged and reported to developers while also notifying the user of the error.

Code Quality:

The code should be clean and easy to read for all developers. It should refrain from using duplicate or unnecessary lines. The code should follow best practices and conventions established by Artemis Financial. The code will be extensively tested for errors and bugs. All errors and bugs will be fixed in the most secure way possible.

Encapsulation:

Encapsulation will limit data and functionality to a need-to-have basis. Giving variable levels of access allows users to only access needed functions or data while still allowing them the intended level of access.

**3. Manual Review**

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

Error spans all files.

I don’t see any lines of code that ask for user verification. This could mean that user verification is being done through a dependency. Third-party dependencies for input validation and user verification pose a huge threat. The other possibility is that there is zero user verification. This also would be a huge threat because anyone in the world would have free access to the data.

Error spans all files.

I see no in-house security measures taken. This application heavily relies on third-party code, and this is especially risky. Third-party code can be exploited by an external attacker especially when the application has a server, and everyone connects it through the internet. Overuse of third-party code will result in extra vulnerability as the third party code may not “play well with others”.

myDateTime.java

public class myDateTime {

int mySecond;

int myMinute;

int myHour;

int[] retrieveDateTime() {

/\* implement accessor method \*/

return new int[3];

}

void setMyDateTime(int seconds, int minutes, int hour) {

/\* implement accessor method \*/

}

}

**Input validation should be implemented so that the user can only submit the correct date/time format. This is especially important due to dependencies that make the application vulnerable through the manipulation of invalid inputs.**

**DocData.java**

public void read\_document(String key, String value)

{

/\* implement read method \*/

//Class.forName("com.mysql.jdbc.Driver");

try {

Connection con=DriverManager.*getConnection*(

"jdbc:mysql://localhost:3306/test","root","root");

} catch (SQLException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

//here test is database name, root is username and password

**There is still a TODO left to complete. The TODO is in the try-catch of the SQL exception. The TODO is likely related to the lack of error messages. A generic error message that doesn’t give away vital information should be utilized. Also, I don’t like leaving TODOs in the code so be sure to remove the inline comment once the task is complete. Also, eclipse tells me that the value of the local variable con is not used at all in the code. This variable should be researched, and a determination should be made on what the corrective action should be. Remove the variable if it is not necessary or utilize the variable if it is.**

**Customer.java**

public class customer {

private int account\_number;

int account\_balance;

public int showInfo() {

//code to show customer information

return this.account\_number;

}

public void deposit(int a) {

account\_balance = account\_balance + a;

}

}

**There is no input validation. This leaves the code especially vulnerable to manipulation of input for nefarious means. Also, account\_ balance should be a double limited to two decimals instead of an int as it represents a financial value and would use percentages in all numbers. It’s fine for account\_number to be an int as it will not ever need to utilize decimals.**

**GreetingController.java**

**import java.util.concurrent.atomic.AtomicLong;**

**import org.springframework.web.bind.annotation.GetMapping;**

**import org.springframework.web.bind.annotation.RequestParam;**

**import org.springframework.web.bind.annotation.RestController;**

**@RestController**

**public class GreetingController {**

**private static final String template = "Hello, %s!";**

**private final AtomicLong counter = new AtomicLong();**

**@GetMapping("/greeting")**

**public Greeting greeting(@RequestParam(value = "name", defaultValue = "World") String name) {**

**return new Greeting(counter.incrementAndGet(), String.format(template, name));**

**}**

**Greeting Controller asks the user for their name. Input validation should be used to prevent nefarious manipulation of input.**

**CrudController.java**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestParam;**

**import org.springframework.web.bind.annotation.RestController;**

**@RestController**

**public class CRUDController {**

**@RequestMapping("/read")**

**public CRUD CRUD(@RequestParam(value="business\_name") String name) {**

**DocData doc = new DocData();**

**return new CRUD(doc.toString());**

**}**

**Crud Controller requests user input to determine the business name. I see that the code will take user input and convert it all into a string. This can cause some problems, but it is a step in the right direction. This tells me that the intended use of the stored variable requires it to be a string. The user input still needs to be validated though. I can’t stress enough how important input validation is with the current dependencies/vulnerabilities that the code has.**

**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

A screenshot of a computer

Description automatically generated

**Attached above is the result of my dependency check. There are 4 CRITICAL vulnerabilities, 5 HIGH vulnerabilities, and 4 MEDIUM vulnerabilities. I recommend working on the vulnerabilities from the CRITICAL to MEDIUM ones. Tomcat and SPRINGBOOT don’t particularly play well together. They can be exploited in tandem in order to exploit in more ways than either can do alone.**

**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.

I recommend working from CRITICAL to MEDIUM to resolve vulnerabilities caused by dependencies. I would start by looking at the dependency upon the dependency tracker and seeing if there is any known fix. At this point, I would begin to write more code to involve heavier input and user validation. Plus I would use encapsulation to isolate various files that are a risk.

Citation

NVD. (n.d.). https://nvd.nist.gov/vuln/search/results?form\_type=Advanced&results\_type=overview&search\_type=all&cpe\_vendor=cpe%3A%2F%3Avmware&cpe\_product=cpe%3A%2F%3Avmware%3Aspring\_boot&cpe\_version=cpe%3A%2F%3Avmware%3Aspring\_boot%3A2.2.4

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

Secure coding guidelines for java SE. (n.d.). https://www.oracle.com/java/technologies/javase/seccodeguide.html