# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/28/2024** | **Anthony Schissler** |  |

## 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

Anthony Schissler

**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?
  1. Being a financial planning organization, Artemis Financial ensuring secure communications with its customers is incredibly important. By utilizing RESTful web programming and exposing an API endpoint, financial data needs to be secured through all channels in which it’s communicated. Encrypting traffic through means such as HTTPS would provide massive benefits initial benefits for security.
* Are there any international transactions that the company produces?
  1. If the company conducts international transactions, the scrutiny for legality increases exponentially depending on the nation that Atermis’ customer is from. Data privacy laws in the European Union for instance are heavily regulated, and severe penalties may occur for improper usage or storage of data.
* Are there governmental restrictions on secure communications to consider?
  1. Depending on the client, the client nation’s government may have a heavier hand on company communication than here in the United States. Countries like might China require governmental access to all communications provided between the client and the Artermis.
* What external threats might be present now and in the immediate future?
  1. With financial data, organized crime will be the largest adversary that could be faced. Actors for organized crime will try and extradite sensitive financial data, and either sell it on dark markets or utilize it to try and phish information or funds from financial institutions such as banks.
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?
  1. Open source has brought about a massive increase in developmental productivity, but there’s always inherent risks. Due to its nature, open source code is available for all to see, meaning mal-actors could be trying to find vulnerabilities within the code that haven’t been caught. Open-source shouldn’t be discouraged, but it should be heavily scrutinized to make sure that known vulnerabilities aren’t present in Artemis’ code. In addition, web technology has increase, and we need to ensure that the legacy systems that are present within Artemis’ infrastructure can support the required updates. Updating things like TLS could break older systems, and need to be considered carefully when modernizing web technologies. Creating encapsulated code for API calls is also incredibly important, and we should never have calls that perform unintended functionality.

**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.

Looking at the process flow from an architectural perspective, there are a variety of considerations that need to be made.

APIs

Ensuring proper encapsulation and communication between endpoints within the API are important for Artemis to properly secure their sensitive financial data. Certificates should be utilized for all endpoints, all code should be signed by developers, and all API calls should be interacted with and scrutinized for correct implementation. Without securing their APIs, Artemis could allow openings for attackers to retrieve or manipulate data as its being transferred for calls users and systems make to the API.

Cryptography

Encrypting traffic utilizing the latest web security protocols is hugely important for Artemis, as they are utilizing a front end that could be accessing very sensitive data. Encrypting data in transit allows Artemis to have extra confidence in conducting its daily business operations without the threat of that data being exposed. Encrypting data-at-rest is also important as well, as if attackers gain access to where data has been stored, storing that information unencrypted means attackers immediately have access to plaintext sensitive information and can act on it immediately. Encrypting gives the company time after a potential attack to notify customers of the issue before attackers can crack what lies within the encrypted shell, giving customers the ability to move or secure their financial information.

Code Error

If Artemis’ web applications don’t utilize robust error handling, unintended consequences could occur. From an application perspective, if an integer overflow isn’t considered when dealing with massive financial projections, then it could cause a program to fail, and open up more security vulnerabilities within the application. With proper error handling, failures in edge case scenarios can be addressed before they become issues, and ensure that failing web services don’t cause a cascade of other issues, such as database corruptions tracking a company’s forecast, or access to reporting being unavailable due to mishandling of formatted data.

Encapsulation

Encapsulation was touched on with APIs, but it again is incredibly important when it comes to securing RESTful API services. Users should only have access to the security context in which they need, and this also is the case for services running the web application. Web services or general users shouldn’t be accessing privileged data that they aren’t meant to be accessing. Performing solid encapsulation enables systems to only access what they need and when they need it, and no other time. This also entails abstracting data structures, and obscuring what actual data structure implementation has been used. Artemis will require large data structures to crunch projections for its clients, and the systems that access and manage that data require solid encapsulation to make sure that data is protected.

**3. Manual Review**

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

Starting with the next section of the assessment process, I started looking through the Model aspect of Artemis’ MVC. Under the Customer class, there’s an integer named “account\_balance” which doesn’t utilize any data encapsulation whatsoever which is utilized by end users to host their account balance. A similar issue was present under myDateTime, in which there were three variables (mySecond, myMinute, myHour) that didn’t utilize any encapsulation to protect their data. Customer also doesn’t utilize any authentication methods for ensuring proper login, and with having account\_balance not encapsulated, means that there’s no verification on whose logging into a customer object, and printing or manipulating the deposit numbers. In addition with myDateTime, the two methods for that class, retrieveDateTime() and setMyDateTime didn’t utilize encapsulation either, and with myDateTime being a public class, encapsulation is extra important. Concerning data access, there’s a class named “DocData” that’s in production that utilizes a seemingly test connection string and user and password for its authentication. If this is a real database utilized by Artemis, the access can be easily read as it’s inside a public method within a public class. DocData also utilizes an ID variable for its class, but doesn’t have anywhere in the class to set that value. With it being private, it opens up null pointer exceptions. In the application.properties file, I also noticed that the port for the server was not secure – as it utilizes 8081, which is an adjacent port for HTTP. Instead, the port should be changed to 443 to utilize HTTPS. For plugins, an old dependency-check-maven artifact was being used for version 5.3.0, and this should be upgraded to the latest version to find the latest hits. For API calls, under CRUDController, the CRUD method mapping maps to all HTTP methods, which should investigated for if we should break up that controller into other mappings. Besides a GET request in the GreetingController class, no other API calls are available to be made as well, not allowing changes to be made by users when classes have functionality built in them to allow changes to be made.

**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 large amount of vulnerabilities were identified when running a dependency check – I’ll be listing them by CVE count per dependency as well as highest severity level:

Status: tomcat-embed-websocket-9.0.30.jar – Critical

Description: Core Tomcat implementation

CVE Count – 27 – Evidence Count – 30

Recommended action: update to at least 9.0.86

Status: tomcat-embed-core-9.0.30.jar – Critical

Description: Core Tomcat implementation

CVE Count – 26 – Evidence Count – 30

Recommended action: update to at least 9.0.90

Status: spring-webmvc-5.2.3.RELEASE.jar – Critical

Description: Spring Web MVC

CVE Count – 12 – Evidence Count – 36

Recommended action: update to latest at 6.1.13

Status: spring-web-5.2.3.RELEASE.jar – Critical

Description: Spring Web

CVE Count – 15 – Evidence Count – 34

Recommended action: update to at least 6.1.13 for parity with other Spring components

Status: spring-expression-5.2.3.RELEASE.jar – Critical

Description: Spring Expression Language (SpEL)

CVE Count – 12 – Evidence Count – 36

Recommended action: update to at least 6.1.13 for parity with other Spring components

Status: spring-core-5.2.3.RELEASE.jar – Critical

Description: Spring Core

CVE Count – 11 – Evidence Count – 36

Recommended action: update to at least 6.1.13 for parity with other Spring components

Status: spring-boot-starter-web-2.2.4.RELEASE.jar – Critical

Description: Starter for building web, including RESTful, applications using Spring MVC

CVE Count – 3 – Evidence Count – 35

Recommended action: update to at least 2.6.6

Status: spring-boot-2.2.4.RELEASE.jar – Critical

Description: Spring Boot

CVE Count – 3 – Evidence Count – 39

Recommended action: update to at least 3.1.6

Status: snakeyaml-1.25.jar – Critical

Description: YAML 1.1 parser and emitter for Java

CVE Count – 8 – Evidence Count – 44

Recommended action: update to at least 2.0

Status: bcprov-jdk15on-1.46.jar – High

Description: The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms

CVE Count – 22 – Evidence Count – 38

Recommended action: Remove this dependency in favor of another – No known clean updates

Status: logback-core-1.2.3.jar – High

Description: logback-core module

CVE Count – 2 – Evidence Count – 31

Recommended action: update to at least 1.4.14

Status: jackson-databind-2.10.2.jar – High

Description: General data-binding functionality for Jackson: works on core streaming API

CVE Count – 6 – Evidence Count – 39

Recommended action: update to at least 2.13.5

Status: hibernate-validator-6.0.18.Final.jar – High

Description: Hibernate's Bean Validation (JSR-380) reference implementation.

CVE Count – 1 – Evidence Count – 32

Recommended action: update to at least 6.1.0

**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.

From our manual review conducted, there were a variety of issues presented that need to be addressed. Encapsulation as a whole needs to be reconducted over all classes. Multiple issues were presented that degraded security for the application significantly just because encapsulation parameters were left off. No authentication methods seemed to be present either, and authentication needs to be addressed for customers accessing their accounts and data. One class hard coded either a test dummy database connection string into its code, or it included a production machine and a user name and password in order to login to its database instance. The login for that account (if real) needs to be changed from its default as well. The DocData class specifically has a variable that cannot be set, and opens itself up to null pointer exceptions as the variable is also private. HTTP was noticed to be in use in the config files instead of HTTPS – this needs to be remedied immediately. API calls were essentially non-existent except for one read entry, and one monolith RequestMapping entry. We should look into breaking up the RequestMapping entry into specific calls, to ensure that it only performs the actions that it needs to be taking. In addition, API calls need to be added elsewhere in the code to allow for users to make the changes they need to on their accounts. Finally, which leads into our next aspect, the pom.xml file needed to have its maven dependency checker updated – it was 5 full versions behind, going from 5 to 10.

For the static review, there were a variety of issues that need to be rectified. For almost all of the issues, I included “safe” versions to update to that are close to the current edition. I also took care to try and bunch together dependencies that might rely on each other, and place them onto the same update scheme. There was one dependency that I noticed severe issues for – Bouncy Castle’s crypto package on Maven demonstrated issues with every single release that it’s had. We should investigate for another package that could satisfy cryptographic requirements that doesn’t contain as many vulnerabilities in order to reduce our risk.

Overall, we need to go through the code again and make sure it’s done correctly to RESTful standards, and watch out for encapsulation of data. Our dependencies require multiple updates that should be handled according to their severity level, until our overall risk has been drastically reduced.