# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/21/20234** | **Shannon Musgrave** |  |

## 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

Shannon Musgrave

**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?

Secure communications are of the utmost importance to Artemis Financial Services. Being a financial organization will mean that Artemis will handle sensitive financial information such as social security numbers, driver’s licenses, and banking account numbers. This requires the web application to authenticate, authorize, and encrypt user’s sessions and data.

Although Global Rain works on international projects, there is no information as to whether this specific client, Artemis Financial, does any international work. Since it also works in investment, it is likely that some international work would be done, and the developers of the system must be aware of cybersecurity rules in any jurisdiction that the company operates in.

There are many governmental laws on financial transactions created to protect customers from fraudulent activities and crime. These laws are different depending on the jurisdiction. Although states can have different laws, most generally, companies must comply with federal laws such as the FFIEC (Federal Financial Institutions Examination Council) which is “an interagency body that aims to prescribe uniform principles of best practices for financial institutions (Top 9 Cybersecurity Regulations for Financial Services | UpGuard, n.d.). Likewise, outside of the US, there are regulations governing financial transactions, the European Union, for example has the EU-GDPR (European General Data Protection Regulation) which “is a security framework by the European Union designed to protect its citizens from personal data compromise (Top 9 Cybersecurity Regulations for Financial Services | UpGuard, n.d.).

There are many types of external threats against a web application. The most dangerous are zero-day threats. Zero-day threats are threats that have not been used in attacks before therefore their first attack is on the “zero day”. These attacks are dangerous because cybersecurity professionals have not had time to detect and develop patches to stop these attacks from happening. Once an attack has been detected and studied, a patch can be created, and the threat is only then viable against unpatched systems. Some types of threats can be very technical such as viruses, worms, and trojans. Others can also be less technical such as, eavesdropping, phishing, and social engineering.

The modernization of the web applications is essential in providing a secure session for Artemis’s customers. For example, HTTPS communication must be used so that plain text is not transmitted over the open web. The web application must also enforce the principle of least privilege and use modern authorization and authentication principles to enforce this. The best example of this modernization would be the required use of multi-factor authentication. Regarding open-source libraries, a policy should be implemented that requires all libraries to be routinely checked for vulnerabilities and patched as needed. The source code written for this specific application may be sound, but ignoring any external libraries can result in an undefended attack vector. Other examples of modernization include using stronger encryption schemes, as computers continue to get faster, schemes that were secure years ago are now breakable and should be abandoned.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financials’ software application. Justify your reasoning for why each area is relevant to the software application.

Input Validation – Validation of all client inputs is essential. All data coming from clients should be considered untrusted data.

APIs – Application programming interfaces should be secure to remove the likelihood that improper calls could be made. Examples of this include handling all documented exceptions and passing proper arguments to API functions. For instance, checking the string length is a good practice to ensure a buffer overflow is not initiated.

Cryptography – There are two avenues of cryptography in this application. First, user passwords should be salted and hashed before being stored in the database. If the database was ever downloaded by a threat actor, the one-way hashed passwords would not be beneficial. Secondly, HTTPS is required to keep plain text communications from being broadcast over the internet.

Code Quality – Code should be written using best coding practices. Reducing bugs will remove attack possibilities because it removes exploitations. Issues such as string length, bounds checking on arrays, and enforcing rules on data mutation should be implemented to reduce the chances of vulnerabilities.

**3. Manual Review**

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

* CRUD class does not perform validation or rules on fields data.
* CRUD Controller does not validate the “business\_name” variable.
* Customer class does not validate parameters.
* Doc data uses the same user and password.
* Greeting class does not validate id and content variables.
* Greeting controller class passes in an atomic long variable into the greeting constructor which only takes a long parameter, since a long is smaller, information could be lost.
* Greeting controller class does not validate name variable.
* Add HTTPS redirection.
* Add Authentication and Authorization.
* @RequestMapping maps to all http methods and should be modified to a POST.

**4. Static Testing**

Run a dependency check on Artemis Financials’ 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

**bcprov-jdk15on-1.46.jar** Install newer update.

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7.

1. CVE-2024-34447
2. CVE-2016-1000338
3. CVE-2016-
4. CVE-2016-1000343
5. CVE-2024-29857
6. CVE-2016-1000344
7. CVE-2016-1000352
8. CVE-2024-30171
9. CVE-2016-1000341
10. CVE-2016-1000345
11. CVE-2017-13098
12. CVE-2020-15522
13. CVE-2020-0187
14. CVE-2023-33202
15. CVE-2020-26939
16. CVE-2023-33201
17. CVE-2016-1000339
18. CVE-2015-7940
19. CVE-2018-5382
20. CVE-2013-1624
21. CVE-2016-1000346
22. CVE-2015-6644

**hibernate-validator-6.0.18.Final.jar** Install newer update.

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

1. CVE-2020-10693

**jackson-databind-2.10.2.jar** Install newer update.

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

1. CVE-2020-25649
2. CVE-2020-36518
3. CVE-2021-46877
4. CVE-2022-42003
5. CVE-2022-42004
6. CVE-2023-35116

**log4j-api-2.12.1.jar** Install newer update.

The Apache Log4j API

1. CVE-2020-9488

**logback-core-1.2.3.jar** Install newer update.

logback-core module

1. CVE-2023-6378
2. CVE-2021-42550

**snakeyaml-1.25.jar** Install newer update.

YAML 1.1 parser and emitter for Java

1. CVE-2022-1471
2. CVE-2017-18640
3. CVE-2022-25857
4. CVE-2022-38749
5. CVE-2022-38751
6. CVE-2022-38752
7. CVE-2022-41854
8. CVE-2022-38750

**spring-boot-2.2.4.RELEASE.jar** Install newer update.

Spring Boot

1. CVE-2023-20873
2. CVE-2022-27772
3. CVE-2023-20883

**spring-boot-starter-web-2.2.4.RELEASE.jar** Install newer update.

Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container

1. CVE-2023-20873
2. CVE-2022-27772
3. CVE-2023-20883

**spring-core-5.2.3.RELEASE.jar** Install newer update.

Spring Core

1. CVE-2022-22965
2. CVE-2021-22118
3. CVE-2020-5421
4. CVE-2022-22950
5. CVE-2022-22971
6. CVE-2023-20861
7. CVE-2023-20863
8. CVE-2022-22968
9. CVE-2022-22970
10. CVE-2021-22060
11. CVE-2021-22096

**spring-web-5.2.3.RELEASE.jar** Install newer update.

Spring Web

1. CVE-2016-1000027
2. CVE-2022-22965
3. CVE-2024-22243
4. CVE-2024-22262
5. CVE-2021-22118
6. CVE-2020-5421
7. CVE-2022-22950
8. CVE-2022-22971
9. CVE-2023-20861
10. CVE-2023-20863
11. CVE-2022-22968
12. CVE-2022-22970
13. CVE-2021-22060
14. CVE-2021-22096

**spring-webmvc-5.2.3.RELEASE.jar** Install newer update.

Spring Web MVC

1. CVE-2022-22965
2. CVE-2021-22118
3. CVE-2020-5421
4. CVE-2022-22950
5. CVE-2022-22971
6. CVE-2023-20861
7. CVE-2023-20863
8. CVE-2022-22968
9. CVE-2022-22970
10. CVE-2021-22060
11. CVE-2021-22096

**tomcat-embed-core-9.0.30.jar** Install newer update.

Core Tomcat implementation

1. CVE-2020-1938
2. CVE-2020-11996
3. CVE-2020-13934
4. CVE-2020-13935
5. CVE-2020-17527
6. CVE-2021-25122
7. CVE-2021-41079
8. CVE-2022-29885
9. CVE-2022-42252
10. CVE-2023-44487
11. CVE-2023-46589
12. CVE-2020-9484
13. CVE-2021-25329
14. CVE-2021-30640
15. CVE-2022-34305
16. CVE-2023-41080
17. CVE-2021-24122
18. CVE-2021-33037
19. CVE-2023-42795
20. CVE-2023-45648
21. CVE-2024-21733
22. CVE-2019-17569
23. CVE-2020-1935
24. CVE-2020-13943
25. CVE-2023-28708
26. CVE-2021-43980

**tomcat-embed-websocket-9.0.30.**jar Install newer update.

Core Tomcat implementation

1. CVE-2020-1938
2. CVE-2020-8022
3. CVE-2020-11996
4. CVE-2020-13934
5. CVE-2020-13935
6. CVE-2020-17527
7. CVE-2021-25122
8. CVE-2021-41079
9. CVE-2022-29885
10. CVE-2022-42252
11. CVE-2023-44487
12. CVE-2023-46589
13. CVE-2020-9484
14. CVE-2021-25329
15. CVE-2021-30640
16. CVE-2022-34305
17. CVE-2023-41080
18. CVE-2021-24122
19. CVE-2021-33037
20. CVE-2023-42795
21. CVE-2023-45648
22. CVE-2024-21733
23. CVE-2019-17569
24. CVE-2020-1935
25. CVE-2020-13943
26. CVE-2023-28708
27. CVE-2021-43980

**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 Financials’ software application.

The steps to mitigate the threats in the Artemis Financials’ application is to begin by updating all 3rd party libraries to use the newest software versions. This will reduce the number of vulnerabilities that need to be remediated because many of the vulnerabilities have been fixed in newer releases. Next, all bullet points in the manual review must be evaluated. Adjustments to the code should be applied using business rules and good coding practices to improve code and remove exploitations such as buffer overflows and SQL injections. Finally, a second static test should be run to uncover any leftover vulnerabilities, this list should be prioritized for the most critical and likely failures. Starting at the most critical, each vulnerability should be eliminated. This can be a slow process since each vulnerability must be understood and fixed or reduced. For instance, if a library does not check the string length of a passed in parameter, a fix can be implemented to check the length of all strings before being passed in, therefore eliminating a vulnerability by fixing the Artemis’s code base, not the libraries.

**References**

*Top 9 Cybersecurity Regulations for Financial Services | UpGuard*. (n.d.). https://www.upguard.com/blog/cybersecurity-regulations-financial-industry#toc-9