## Competencies

In this project, you will demonstrate your mastery of the following competencies:

- Write secure communications through the application of current encryption technologies and techniques
- Design and implement code that complies with software security testing protocols

## Scenario

You work as a developer for a software engineering company, Global Rain, that specializes in custom software design and development. The software is for entrepreneurs, businesses, and government agencies around the world. Part of the company's mission is that "Security is everyone's responsibility." Global Rain has promoted you to their new agile scrum team.

At Global Rain, you work with a client, Artemis Financial, a consulting company that develops individualized financial plans for their customers. The financial plans include savings, retirement, investments, and insurance.

Artemis Financial wants to modernize their operations. As a crucial part of the success of their custom software, they also want to use the most current and effective software security. Artemis Financial has a public web interface. They are seeking Global Rain's expertise about how to protect their client data and financial information.

Specifically, Artemis Financial wants to add a file verification step to their web application to ensure secure communications. When the web application is used to transfer data, they will need a data verification step in the form of a checksum. You must take their current software application and add secure communication mechanisms to meet their software security requirements. You'll deliver a production quality integrated application that includes secure coding protocols.

## Directions

You must examine Artemis Financial's software to address any security vulnerabilities. This will require you to refactor the Project Two Code Base, linked in the Supporting Materials section, to add functionality to meet software security requirements for Artemis Financial's application. Specifically, you'll need to follow the steps outlined below to facilitate your findings, address and remedy all areas, and use the Project Two Template, linked in What to Submit, to document your work for your practices for secure software report. You will also submit zipped files that contain the refactored code.

1. **Algorithm Cipher:** Recommend an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justify your reasoning. Review the scenario and the supporting materials to support your recommendation. In your practices for secure software report, be sure to address the following:
    a. Provide a brief, high-level overview of the encryption algorithm cipher.

    b. Discuss the hash functions and bit levels of the cipher.
    c. Explain the use of random numbers, symmetric versus non-symmetric keys, and so on.
    d. Describe the history and current state of encryption algorithms.

2. **Certificate Generation:** Generate appropriate self-signed certificates using the Java Keytool in Eclipse.
    a. To demonstrate that the certificate was correctly generated:
        i. Export your certificates (CER file).
        ii. Submit a screenshot of the CER file in your practices for secure software report.
3. **Deploy Cipher:** Deploy and implement the cryptographic hash algorithm by refactoring code. Demonstrate functionality with a checksum verification.
    a. Submit a screenshot of the checksum verification in your practices for secure software report. The screenshot must show your name and a unique data string that has been created.
4. **Secure Communications**: Verify secure communication. In the application.properties file, refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code. Then once the server is running, type https://localhost:8443/hash in a new browser to demonstrate that the secure communication works successfully.
    a. Create a screenshot of the web browser that shows a secure webpage and include it in your practices for secure software report.
5. **Secondary Testing:** Run a secondary static testing of the refactored code using the OWASP Dependency-Check Maven (see Supporting Materials) to ensure code complies with software security enhancements. You need to focus on only the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities. Include the following in your practices for secure software report:
    a. A screenshot of the refactored code executed without errors
    b. A screenshot of the report of the output from the dependency-check static tester
6. **Functional Testing:** Identify the software application's syntactical, logical, and security vulnerabilities by manually reviewing code.
    a. Complete this functional testing and include a screenshot of the refactored code, executed without errors, in your practices for secure software report.

**What if I receive errors or new vulnerabilities?**
You will need to iterate on your design and refactored code, address vulnerabilities, and retest until no new vulnerabilities are found.

7. **Summary:** Discuss how the code has been refactored and how it complies with security testing protocols. In the summary of your practices for secure software report, be sure to address the following:
    a. Refer to the Vulnerability Assessment Process Flow Diagram. Highlight the areas of security that you addressed by refactoring the code.
    b. Discuss your process for adding layers of security to the software application.
8. **Industry Standard Best Practices:** Explain how you applied industry standard best practices for secure coding to mitigate against known security vulnerabilities. Be sure to address the following:
    a. Explain how you used industry standard best practices to maintain the software application's current security.
    b. Explain the value of applying industry standard best practices for secure coding to the company's overall wellbeing.

## What to Submit

To complete this project, you must submit the following:

**Practices for Secure Software Report**
Use the Project Two Template to create your report. Submit one comprehensive report of the steps you have taken to increase the layers of security in Artemis Financial's software application. Also include details about the code files (see below) being "attachments" to the completed report.

**CS 305 Project Two Refactored Code Base.zip**
Refactor the code provided in the Supporting Materials section. Be certain to zip the refactored code into one zipped project folder that contains all files associated with Artemis

Financial's software application. Submit the zipped project folder.

## Supporting Materials

The following resources support your work on the project:

**Code Base:** Project Two Code Base
Refactor the code to meet the software security needs of your customer. You must submit your refactored code.

**Website:** Java Security Standard Algorithm Names
Refer to the standard list of algorithm ciphers provided by Oracle for recommending an appropriate encryption algorithm cipher to use in this project.

**Website:** How to Create a Self Signed Certificate Using Java Keytool
Access the Java Keytool using the command line for generating certificates. For more information, you may want to refer to this website and the following one.

**Website**: Oracle Key and Certificate Management Tool
This website provides more details about Java Keytool commands.

**Tutorial:** Integrating the Maven Dependency-Check Plug-in Tutorial
Follow the instructions in this tutorial to learn how to integrate the dependency-check plug-in into Maven. You'll need to edit the pom.xml file to add the dependency-check plug-in to Artemis Financial's software application. When you compile your code, Eclipse will run the Maven plug-in.

**Tool:** OWASP Dependency-Check Maven
OWASP Dependency-Check Maven is an open-source solution. It's used to scan Java applications to identify the use of known vulnerabilities. You'll use this to run a dependency check to statically test the code and produce an HTML report.

**Diagram:** Vulnerability Assessment Process Flow Diagram
Reference this process flow diagram during the project to determine which of the seven areas of security to assess for Artemis Financial's software application.

A text-only version is available: Vulnerability Assessment Process Flow Diagram Text-Only Version.

**Project Two Rubric**

| Criteria | Exemplary (100%) | Proficient (85%) | Needs Improvement (55%) | Not Evident (0%) | Value |
|---|---|---|---|---|---|
| **Algorithm Cipher** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Recommends an appropriate encryption algorithm cipher to deploy, given the security vulnerabilities, and justifies reasoning | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 12 |
| **Certificate Generation** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Generates appropriate self-signed certificates using the Java Keytool in Eclipse | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 11 |
| **Deploy Cipher** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Deploys and implements cryptographic hash algorithm by refactoring code; demonstrates functionality with a checksum verification; submits a screenshot of the checksum verification | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 15 |

| Criteria | Exemplary (100%) | Proficient (85%) | Needs Improvement (55%) | Not Evident (0%) | Value |
|---|---|---|---|---|---|
| **Secure Communications** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Verifies secure communication and demonstrates that the secure communication works successfully | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 15 |
| **Secondary Testing** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Runs a secondary static testing of the refactored code using the dependency-check tool | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 11 |
| **Functional Testing** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Identifies the software application's syntactical, logical, and security vulnerabilities by manually reviewing code | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 12 |
| **Summary** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Discusses how the code has been refactored and how it complies with security testing protocols | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 12 |
| **Industry Standard Best Practices** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Explains how industry standard best practices for secure coding were applied to mitigate against known security vulnerabilities | Shows progress toward proficiency, but with errors or omissions | Does not attempt criterion | 7 |
| **Articulation of Response** | Exceeds proficiency in an exceptionally clear, insightful, sophisticated, or creative manner | Clearly conveys meaning with correct grammar, sentence structure, and spelling, demonstrating an understanding of audience and purpose | Shows progress toward proficiency, but with errors in grammar, sentence structure, and spelling, negatively impacting readability | Submission has critical errors in grammar, sentence structure, and spelling, preventing understanding of ideas | 5 |
| | | | | **Total:** | 100% |