# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/21/2024** | **Stephanie Dandaraw** |  |

## 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

Stephanie Dandaraw

**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:

Artemis Financial is a company with high-value assets that must comply with multiple government regulations. A successful security breach can steal company information such as encryption keys, proprietary financial strategies, back-end system access, and application logs. Client information can also be stolen; including personal information, login credentials, financial information, and more. Additional consequences for a successful attack include regulatory penalties, disruption of operations, revenue loss, exposure of trade secrets, loss of public trust, and increased legal risk. Furthermore, this collection of valuable data is highly-motivating for attackers including hacktivists, malicious insiders, nation-state actors, and criminal organizations. Since Artemis Financial is a US-based financial company, it is governed by two compliance standards: PCI-DSS and The Gramm-Leach-Bliley Act (Manico & Detlefsen, 2015). PCI-DSS is the data security standard for the payment card industry. It dictates encryption requirements for storing and transferring credit card information, and it specifies the credit card data components that can be stored (Manico & Detlefsen, 2015). The Gramm-Leach-Bliley Act requires consumer financial service companies to protect consumer data and disclose information-sharing policies (Manico & Detlefsen, 2015). Thus, it is clear that secure communications are incredibly important for the continued success of Artemis Financial.

In order to modernize the company’s operations and implement top-tier security strategies, several requirements must be fulfilled. For example, open-source libraries can introduce vulnerabilities into the application that are leveraged by attackers. This threat is compounded by using out-of-date dependency versions (Oracle, 2023). Thus, the application’s third-party dependencies must be monitored for newly discovered vulnerabilities and updated immediately after new versions are released (Oracle, 2023). Furthermore, the role of open-source libraries within the application must be periodically re-evaluated to determine the best product, version, and configuration options to fit Artemis Financial’s software needs (Manico & Detlefsen, 2015). According to Manico and Detlefsen, two of the most serious external threats facing the application are malicious input and SQL injection (2015). Since software security is a constantly evolving goal, another modernization requirement is the implementation of secure design patterns that successfully counter new instances of these attacks (Manico & Detlefsen, 2015).

**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.

**APIs: Secure API Interactions**

“APIs: Secure API Interactions” from the Vulnerability Assessment Process Flow Diagram is an area of security applicable to Artemis Financial’s software application. According to Jim Manico and August Detlefsen, SQL injection is the most serious threat facing online applications today (2015). SQL injection attacks corrupt the structure and behavior of an application by adding untrusted data to database queries. Potential consequences of SQL injection include vertical privilege escalation, theft of sensitive information, DoS, remote code execution, cross-site scripting, and breach of the direct operating system shell access. Since the application must handle trusted and untrusted data, securing API interactions is absolutely essential (Manico & Detlefsen, 2015).

**Encapsulation: Secure Data Structures**

Prioritizing the “Encapsulation: Secure Data Structures” area of security will be key to fulfilling Artemis Financial’s security goals. For example, implementation details of an API should be strongly encapsulated by declaring a module that only exports non security-sensitive packages that include a published API, and skips the packages that provide implementation support (Oracle, 2023). Another example of strong encapsulation is separating the application code from the access control policy. Failing to separate these components can lead to an inconsistent access control policy and increase the difficulty and cost of customization tasks (Oracle, 2023). Lastly, a secure Spring REST API will utilize a higher-level library to protect the application code from SQL, such as java.sql.PreparedStatement (Oracle, 2023).

**Input validation: Secure Input and Representations**

“Input validation: Secure Input and Representations” is another applicable area of security for the API. The exploitation of malicious input is a gateway for a myriad of well-documented attacks (Oracle, 2023). While ensuring secure API interactions through query parameterization is most effective, input validation and formatting can further protect the application from malicious input (Manico & Detlefsen, 2015). For example, invalid HTTP header data should be rejected to avoid cross-site POST attacks, and input that requires excessive resource consumption should be rejected to avoid DoS Attacks (Oracle, 2023).

**3. Manual Review**

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

* Identify all vulnerabilities in the Project One Code Base linked in the Supporting Materials section by manually inspecting the code.
* Document at least 7 to 10 findings in your vulnerability assessment report.
* Include a description that identifies where the vulnerabilities are found. Provide the specific class file, if applicable.

A manual review of the code has revealed several vulnerabilities.

1. Constructors in CRUD.java, DocData.java, Greeting.java, are public, which exposes the application to unauthorized object instantiation. Guideline 7-1 / OBJECT-1 of the Secure Coding Guidelines for Java SE states that exposing constructors of sensitive classes should be avoided, with static factory methods used instead (Oracle, 2023).
2. Input validation should be implemented in several areas of the application, including Customer.java’s deposit() method, Greeting.java’s constructor, and MyDateTime.java’s SetMyDateTime() method. According to Guideline 6-6 / MUTABLE-6 of the Secure Coding Guidelines for Java SE, input passed to an untrusted object should be copied and validated (Oracle, 2023).
3. Mutable output values are vulnerable in several areas of the application, including Greeting.java’s getId() and getContent() methods, DocData.java’s getId() method, Customer.java’s showInfo() method, and CRUD.java’s getContent() and getContent2() methods. According to Guideline 6-2 / MUTABLE-2, copies of mutable output values should be created and returned (Oracle, 2023).
4. Exposed native methods are included in the CRUDController class, GreetingController class, and RestServiceApplication class. Guideline 5-3 / INPUT-3 of the Secure Coding Guidelines for Java SE state that wrappers should be defined around native methods, and native methods should not be made public (Oracle, 2023).
5. The GreetingController class uses GetMapping, but PostMapping should be implemented for its additional security benefits (Manico & Detlefsen, 2015).
6. The application currently uses HTTP, but implementing HTTPS will help fulfill Artemis Financial’s security goals (Manico & Detlefsen, 2015).
7. Several object fields in the application are currently exposed, including Customer.java’s account\_balance, and MyDateTime.java’s attributes. Object fields should be declared private and final wherever possible, and interaction should be done through accessor and mutator methods (Manico & Detlefsen, 2015).
8. To prevent SQL injection and secure API interactions, query parameterization should be added to CRUDController.java, GreetingController.java, and DocData.java, in the form of prepared statements (Manico & Detlefsen, 2015).
9. Since these java files handle sensitive application and customer data, security control should be added to CRUD.java, customer.java, GreetingController.java, and DocData.java to verify authentication and authorization of requestors (Oracle, 2023). Then, secure error handling should be implemented to handle the rejected inputs.

**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

**bcprov-jdk15on-1.46.jar**

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

The Bouncy Castle Crypto package contained in bcprov-jdk15on-1.46.jar is vulnerable for several reasons. First, the Artemis Financial application is configured for JDK 1.8, but the crypto package is designed for JDK 1.5 to JDK 1.7. Furthermore, version 1.46 was released over 10 years ago, and it triggers 13 CVEs in the OWASP dependency check. Thus, a newer package version that is configured for Java 8 should be implemented, such as org/bouncycastle/bcprov-jdk15to18/1.78.1, before the dependency check is rerun (<https://repo1.maven.org/maven2/org/bouncycastle/bcprov-jdk15to18/1.78.1/> ).

**hibernate-validator-6.0.18.Final.jar**

**CVE-2020-10693**: CWE-20 Improper Input Validation

<https://nvd.nist.gov/vuln/detail/CVE-2020-10693>

A flaw in the Hibernate Validator message interpolation processor enables EL expressions to be evaluated although they are invalid. This flaw enables attackers to circumvent input sanitation controls that developers put in place to handle user-controlled data in error messages.

* **Solution**: Upgrade to hibernate-validator-6.0.20.Final due to CVE-2020-10693 and CVE-2019-10219

**jackson-databind-2.10.2.jar**

**CVE-2020-25649**: CWE-611 Improper Restriction of XML External Entity Reference

<https://nvd.nist.gov/vuln/detail/CVE-2020-25649>

Entity expansion was not secured properly in FasterXML Jackson Databind, enabling XML external entity (XXE) attacks. XXE attacks can threaten data integrity.

* **Solution**: Add patch that disallows the problematic doc type. <https://github.com/FasterXML/jackson-databind/issues/2589>

**CVE-2020-36518**: CWE-787 Out-of-bounds Write

<https://nvd.nist.gov/vuln/detail/CVE-2020-36518>

jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects.

* **Solution**: Update version to 2.17.2

<https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core/2.17.2>

**CVE-2022-42003**: CWE-502 Deserialization of Untrusted Data

<https://nvd.nist.gov/vuln/detail/CVE-2022-42003>

FasterXML Jackson Databind before 2.13.4.1 and 2.12.17.1 does not check primitive value deserializers to prevent deep wrapper array nesting, which can lead to resource exhaustion when the UNWRAP\_SINGLE\_VALUE\_ARRAYS feature is enabled.

* **Solution**: Update version or implement patch.

<https://github.com/FasterXML/jackson-databind/commit/d78d00ee7b5245b93103fef3187f70543d67ca33>

**CVE-2022-42004**: CWE-502 Deserialization of Untrusted Data

<https://nvd.nist.gov/vuln/detail/CVE-2022-42004>

FasterXML Jackson Databind before 2.13.4 does not check BeanDeserializer.\_deserializeFromArray to prevent deeply nested arrays, which can cause resource exhaustion for applications with certain customized choices for deserialization.

* **Solution**: Update version or implement patch <https://github.com/FasterXML/jackson-databind/commit/063183589218fec19a9293ed2f17ec53ea80ba88>

**logback-core-1.2.3.jar**

**CVE-2023-6378**: CWE-502 Deserialization of Untrusted Data

<https://nvd.nist.gov/vuln/detail/CVE-2023-6378>

Attackers can send poisoned data to cause a Denial-of-Service attack due to a serialization vulnerability in logback versions before 1.4.11.

* **Solution**: update to 1.2.13, 1.3.12, or 1.4.12 <https://logback.qos.ch/news.html#1.3.12>

**CVE-2021-42550**: CWE-502 Deserialization of Untrusted Data

<https://ossindex.sonatype.org/vulnerability/CVE-2021-42550?component-type=maven&component-name=ch.qos.logback%2Flogback-core&utm_source=dependency-check&utm_medium=integration&utm_content=10.0.3>

In logback version 1.2.7 and prior versions, an attacker with the required privileges to edit configuration files could craft a malicious configuration allowing to execute arbitrary code loaded from LDAP servers.

* **Solution**: This CVE was debated amongst security experts because any attacker that can edit configuration files can cause RCE in a myriad of ways. Ensure robust authentication and authorization to ensure attackers cannot edit configuration files.

**snakeyaml-1.25.jar**

**CVE-2022-1471**: CWE-502 Deserialization of Untrusted Data, CWE-20 Improper Input Validation

<https://nvd.nist.gov/vuln/detail/CVE-2022-1471>

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.

* **Solution**: Use SnakeYaml’s SafeConstructor when parsing untrusted content to restrict deserialization. Upgrade to version 2.0 or beyond.

(<https://bitbucket.org/snakeyaml/snakeyaml/issues/561/cve-2022-1471-vulnerability-in#comment-64581479>)

**CVE-2022-25857**: CWE-776 Improper Restriction of Recursive Entity References in DTDs ('XML Entity Expansion')

<https://nvd.nist.gov/vuln/detail/CVE-2022-25857>

The package org.yaml:snakeyaml from 0 and before 1.31 are vulnerable to Denial of Service (DoS) due to missing nested depth limitation for collections.

* **Solution**: Upgrade to version 2.0 or beyond. (<https://bitbucket.org/snakeyaml/snakeyaml/issues/561/cve-2022-1471-vulnerability-in#comment-64581479>)

**Spring Framework**

**CVE-2023-20883**: CWE-400 Uncontrolled Resource Consumption

<https://spring.io/security/cve-2023-20883>

In Spring Boot versions 3.0.0 - 3.0.6, 2.7.0 - 2.7.11, 2.6.0 - 2.6.14, 2.5.0 - 2.5.14 and older unsupported versions, there is potential for a denial-of-service (DoS) attack if Spring MVC is used together with a reverse proxy cache.

* **Solution**: Users of older, unsupported versions should upgrade to 3.0.7+ or 2.7.12+. As a workaround, configure the reverse proxy not to cache 404 responses and/or not to cache responses to requests to the root (/) of the application. <https://spring.io/security/cve-2023-20883>

**CVE-2022-22968**: CWE-178 Improper Handling of Case Sensitivity

<https://nvd.nist.gov/vuln/detail/CVE-2022-22968>

In Spring Framework versions 5.3.0 - 5.3.18, 5.2.0 - 5.2.20, and older unsupported versions, the patterns for disallowedFields on a DataBinder are case sensitive. Thus, a field is not protected unless it is listed with both upper and lower case for the first character of the field.

* **Solution**: To continue using disallowed field patterns, update to Spring Framework 5.3.19 and 5.2.21 or greater. Data binding issues can occur without registering disallowed field patterns. Review the application’s DataBinder configuration and its approach to data binding. See the Data Binding Model Design section in the Spring Framework reference manual. Spring recommends using a dedicated model object that exposes properties relevant for supported use cases. Spring’s alternative approach is to switch to allowed field patterns, rather than disallowed field patterns. Spring does not recommend using types from the domain model such as Hibernate entities as the model object (Spring Blog, 2022).

**CVE-2022-22950 (OSSINDEX)**: Spring Expression DoS Vulnerability

CWE-770 Allocation of Resources Without Limits or Throttling

<https://spring.io/security/cve-2022-22950>

Specially crafted SpEL expressions can cause DoS in Spring Framework versions before 5.3.16.

* **Solution**: Users of affected versions should apply the following mitigation: 5.3.x users should upgrade to 5.3.17+. 5.2.x users should upgrade to 5.2.20+. No other steps are necessary. Releases that have fixed this issue include: Spring Framework 5.3.17+; 5.2.20+

**CVE-2023-20861**: Spring Expression DoS Vulnerability

<https://spring.io/security/cve-2023-20861>

Specially crafted SpEL expressions can cause DoS in Spring Framework versions 6.0.0 - 6.0.6, 5.3.0 - 5.3.25, 5.2.0.RELEASE - 5.2.22.RELEASE, and older unsupported versions.

* **Solution**: 6.0.x users should upgrade to 6.0.7+. 5.3.x users should upgrade to 5.3.26+. 5.2.x users should upgrade to 5.2.23.RELEASE+. Users of older, unsupported versions should upgrade to 6.0.7+ or 5.3.26+. No other steps are necessary. Releases that have fixed this issue include: Spring Framework 6.0.7+; 5.3.26+; 5.2.23.RELEASE+

**CVE-2023-20863**: Spring Expression DoS Vulnerability

CWE-400 Uncontrolled Resource Consumption, CWE-917 Improper Neutralization of Special Elements used in an Expression Language Statement ('Expression Language Injection')

<https://spring.io/security/cve-2023-20863>

Specially crafted SpEL expressions can cause DoS in Spring Framework versions prior to 5.2.24 release+ ,5.3.27+ and 6.0.8+.

* **Solution**: 6.0.x users should upgrade to 6.0.8+. 5.3.x users should upgrade to 5.3.27+. 5.2.x users should upgrade to 5.2.24.RELEASE+. Users of older, unsupported versions should upgrade to 6.0.8+ or 5.3.27+. No other steps are necessary. Releases that have fixed this issue include: Spring Framework 6.0.8+; 5.3.27+; 5.2.24.RELEASE+

**CVE-2024-22243 (OSSINDEX)**: Spring Framework URL Parsing with Host Validation CWE-20 Improper Input Validation

<https://spring.io/security/cve-2024-22243>

Applications that parse an externally provided URL through a query parameter and perform validation checks on the host are vulnerable to SSRF and open redirect attacks, if the URL is used after passing validation checks.

* **Solution**: 6.1.x users should upgrade to 6.1.4; 6.0.x users should upgrade to 6.0.17; 5.3.x users should upgrade to 5.3.32

**CVE-2024-22262**: Spring Framework URL Parsing with Host Validation (3rd report) CWE-20 Improper Input Validation

<https://spring.io/security/cve-2024-22262>

Applications that parse an externally provided URL through a query parameter and perform validation checks on the host are vulnerable to SSRF and open redirect attacks, if the URL is used after passing validation checks. This is the same as CVE-2024-22259 and CVE-2024-22243, but with different input.

* **Solution**: 6.1.x users should upgrade to version 6.1.6; 6.0.x users should upgrade to version 6.0.19; and 5.3.x users should upgrade to version 5.3.34.

**CVE-2021-22060**: Additional Log Injection in Spring Framework (follow-up to CVE-2021-22096) – <https://spring.io/security/cve-2021-22060>

In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries.

* **Solution**: 5.3.x users should upgrade to 5.3.14+. 5.2.x users should upgrade to 5.2.19+. No other steps are necessary. Releases that have fixed this issue include: Spring Framework 5.3.14+; 5.2.19+

**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 results from the manual review reveal opportunities to improve secure API interactions, including the implementation of HTTPS and the POST method. Furthermore, there are areas in the application where encapsulation can be added to increase the security of the data structure. The static testing report indicated that the application has multiple dependencies operating unsupported versions. Updating all dependencies to current versions is recommended, in addition to the creation of a policy in which third-party security updates are tracked and promptly implemented.

* **Secure API Interactions**
  + DocData.java: lines 21->32
  + GreetingController.java: lines 15->18
  + Query Parameterization: Use prepared statements to protect queries from SQL injection.
  + GreetingController.java: line 5
  + Use @PostMapping for additional security measures
  + pom.xml
  + Implement HTTPS over HTTP
* **Encapsulation**
  + myDateTime.java: lines 5-> 7
  + customer.java: line 5
  + Private object fields: Fields of objects should be kept private.
  + myDateTime.java lines 9->11
  + customer.java: lines 7->10
  + Greeting.java: lines 12->14
  + Greeting.java: lines 15->18
  + Copy mutable objects: Return copies of mutable objects to prevent client code from modifying the instance state.
* **Input Validation**
  + myDateTime.java: lines 14->16
  + customer.java: lines 12->14
  + GreetingController.java: lines 15->18
  + Validate inputs: Ensure input format and value are accepted
  + Copy input: Create copies of input and add copies to object fields.
* **Secure Error Handling**
  + CRUD.java
  + customer.java: lines 7->10
  + GreetingController.java: lines 15->18
  + DocData.java: lines 21->32
  + Security Control: verify authentication and authorization of requestor and handle exceptions
* **Dependency Check for Files**
  + pom.xml, RestServiceApplication.java, CRUDController.java
  + Update Spring framework, Jackson, snakeyaml, and logback to current versions, see CVE and NVD databases.

**References**

Manico, J., & Detlefsen, A. (2015). *Iron-Clad Java : Building Secure Web Applications*. McGraw-Hill

Education. https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/title.html

Oracle. (2023, May). *Secure Coding Guidelines for Java SE*. Oracle.

<https://www.oracle.com/java/technologies/javase/seccodeguide.html>

Spring Blog. (2022, April 13). *Spring Framework Data Binding Rules Vulnerability (CVE-2022-22968)*. Spring; VMware Tanzu. https://spring.io/blog/2022/04/13/spring-framework-data-binding-rules-vulnerability-cve-2022-22968