# CS 305 Module Two Written Assignment

## Areas of Security

The application was queried for several security-critical aspects:

* Input Validation: Query parameter user input (/greeting) and path variable user input (/number/{id}) are neither validated nor sanitized. User input is evaluated as a Spring Expression, leaving scope for potential injection attacks.
* Secure Input and Representations: SpEL (Spring Expression Language) is used on raw user input with no validation or escaping, which is dangerous. The input is run as code during runtime.
* Application Architecture: Sticks to standard Spring Boot MVC organization but lacks secure layering and proper separation of responsibilities. Lacks service or repository layers, and all the logic is in the controller.
* Code Review: All application logic is centralized within controllers. Input validation, exception handling, and data access boundaries are weakly enforced or missing.
* Secure Error Handling: Application sends raw messages and exceptions directly to the console, exposing implementation details to logs. No global exception handler is established.
* Secure Coding Practices / Patterns: The use of dynamic evaluation (SpEL) on raw input is contrary to good security coding practices. Magic values (e.g., hardcoded array and index usage) generate logic errors and denial of service.
* Secure API: Interactions REST APIs are exposed to unauthenticated users. There is no rate limiting, no access controls, and no HTTPS enforcement.
* Cryptography: No encryption is implemented. Everything is plain HTTP. There is no hashing or encryption for potentially sensitive IDs or messages.
* Code Quality: Properly organized models with encapsulated properties and accessors. Low cohesion is evidenced by business logic in controllers.

## Areas of Security Justification

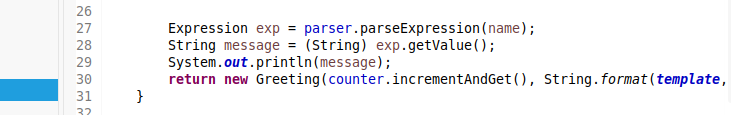
Each priority area was selected from OWASP Top Ten best practices and vulnerabilities in secure software development:

* Input validation and encoding to prevent injection attacks (SpEL, SQL, command injection).
* Secure API interactions for data integrity, confidentiality, and proper access control.
* Error handling to prevent leakage of inner details to potential attackers.
* Secure coding patterns reduce the attack surface and make code maintainable.
* Architecture review helps in validating the appropriate segregation of duties and modular architecture for scalability and security.

## Code Review Summary

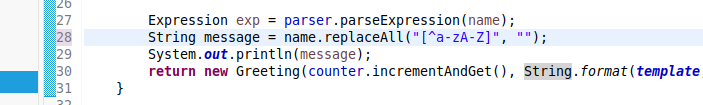
1. Controller Review: GreetingController.java

* Issue 1: SpEL Injection in the /greeting endpoint:



A user could pass malicious input like T(java.lang.Runtime).getRuntime().exec("rm -rf /").

* Fix: Avoid evaluating user input entirely as shown below:



* Issue 2: Unsafe array access in /number/{id}:

| String message = "Element in the given index is :: "+myArray[id]; |
| --- |

No bounds checking, which can cause an ArrayIndexOutOfBoundsException.

* To fix, add this to the code:

| if (id < 0 || id >= myArray.length) {  throw new IllegalArgumentException("Invalid array index.");  }  String message = "Element in the given index is: " + myArray[id]; |
| --- |

* Issue 3: No exception handling

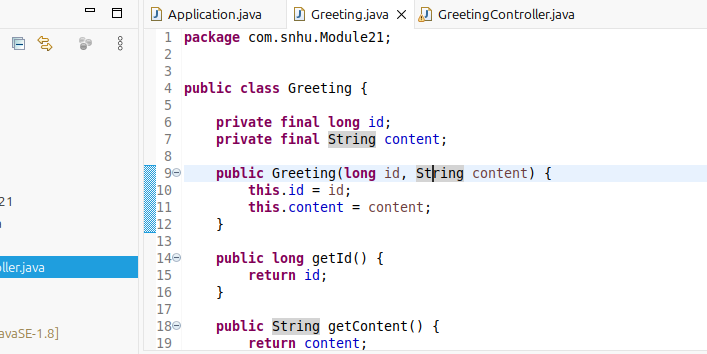
All exceptions are printed to the console.

To fix, we need to add global exception handling with @ControllerAdvice:

| @ControllerAdvice  public class GlobalExceptionHandler {  @ExceptionHandler(Exception.class)  public ResponseEntity<String> handle(Exception e) {  return ResponseEntity.badRequest().body("An error occurred: Invalid request.");  }  } |
| --- |

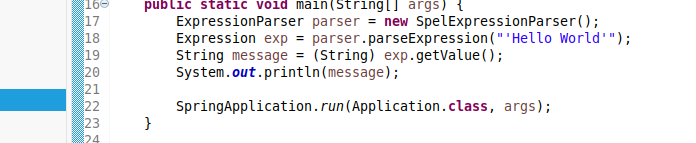
Model Review: Greeting.java

This class is a plain POJO and adheres to encapsulation principles. No change required:



Bootstrap Review: Application.java

Issue: SpEL expression usage in main() is redundant and unnecessary for production code:



To fix, we need to replace it with standard output:

| System.out.println("Hello World"); |
| --- |

Spring Boot bootstrapping is otherwise correct.

## Mitigation Plan

| **Issue** | **Risk** | **Mitigation** |
| --- | --- | --- |
| SpEL Injection | Code execution, remote command injection | Remove SpEL usage. Sanitize inputs using whitelist validation. |
| Array Index | App crashes due to an unhandled exception | Add bounds checking before accessing the array. |
| No Authentication | Unrestricted API usage | Use Spring Security to enforce basic authentication. |
| No Input Validation | Injection & logic issues | Apply input validation rules with @Valid or manual regex. |
| Poor Error Handling | Information disclosure | Implement a global exception handler. Avoid logging internal stack traces. |
| No HTTPS | Data sniffing | Enforce HTTPS via Spring Security and server configuration. |
| No Rate Limiting | DoS | Use an API gateway or Spring filters to limit excessive requests. |
| Hardcoded data | Not scalable or maintainable | Use properties files or a database for dynamic data like arrays. |