OWASP: Injection and Insecure Design

**Vulnerabilities/Attacks**

**Index (Login)**

* **Detailed Error Messages**

Detailed error messages expose potential vulnerabilities to attackers by providing too much information about the system's inner workings.

* **Weak Password Encryption (SHA1)**

Usage of SHA1 for password encryption is considered weak due to its susceptibility to brute-force attacks.

* **Lack of input sanitization**

Unauthorized database access and manipulation.

SQL Injection Input Example: *daniela@example.com" --*

Note: SQLite in PHP doesn't support multiple queries in a single execution by default for security reasons. Therefore, in this case, it is not possible to perform, for example, a *DROP TABLE Customer* after the *SELECT*.

**Registration**

* Vale a pena mostrar sql injection?

**Book page**

* **No Maximum Quantity Limited to Stock**

Lack of enforcement for maximum quantity could lead to inventory manipulation or depletion.

**Ping Page**

* **Sanitization Absent in Ping**

Lack of sanitization in ping commands may expose the system to potential command injections or unauthorized system access.

Command Injection Input Example: *8.8.8.8; ls -la*

**Countermeasures**

**Injection**

* **Use of Prepared Statements (Parameterized Queries)**

Prepared statements ensure separation between SQL code and user input, mitigating SQL injection risks.

$stmt = $dbh->prepare('SELECT \* FROM Books WHERE title = ?');

$stmt->execute(array($title));

$books = $stmt->fetchAll();

* **Use of Stored Procedures**

Utilizing stored procedures further protects against SQL injection by encapsulating SQL logic within the database.

* **Escaping all User Supplied Input**

String escaping or input validation helps prevent unintended SQL commands from being executed.

* **Sanitized Hostname in Ping**

Implement sanitization in ping commands to prevent command injections.

**Insecure Design**

* **Enforcement of (Maximum) Stock Limits**

Set limits on quantity to prevent inventory manipulation.

* **Reduced Detailed Error Messages**

Minimizing error message details limits exposure of system information to potential attackers.

* **Stronger Password Hashing Functions**

Use more secure hashing functions like SHA256, SHA512 or bcrypt.

Generating and Validating Passwords:

* Prepend salt to the password and hash it using a standard cryptographic function like bcrypt.
* Store both salt and hash in the user's database record.
* Retrieve the user's salt and hash for password validation.

Hash and validate passwords in PHP:

string password\_hash (string $pwd, integer $algo [, array $opts])

boolean password\_verify (string $pwd, string $hash)

password\_hash: Generates a hash with its own salt.

password\_verify: Validates the hash against the password without requiring separate storage for salt or algorithm.