Vulnerability Report

Archivo: guardar.php

Code Analyzed:

```
<?php
include "../../models/conexion.php"; // Asegúrate de que la ruta a este archivo es correcta
$data = json_decode(file_get_contents('php://input'), true);

$nombre = $data['nombre'];
$cantidad = $data['cantidad'];
$precio = $data['precio'];

$sql = "INSERT INTO productos (nombre, cantidad, precio) VALUES ('$nombre', '$cantidad',
'$precio')";

if ($conexion->query($sql) == true) {
    echo json_encode(["message" => "Producto guardado exitosamente"]);
} else {
    echo json_encode(["message" => "Error al guardar producto"]);
}

?>
```

Analysis: ```html
Security Vulnerabilities

SQL Injection

Description: The code is vulnerable to SQL injection. The variables `$nombre`, `$cantidad`, and `$precio` are directly inserted into the SQL query without proper sanitization or escaping. An attacker could inject malicious SQL code through these parameters to read, modify, or delete data, or even execute arbitrary commands on the database server.

Approximate Line: Line 9: `$sql = "INSERT INTO productos (nombre, cantidad, precio) VALUES ('$nombre', '$cantidad', '$precio')";`

Mitigation: Use prepared statements with parameterized queries. Prepared statements separate the SQL code from the data, preventing the database from interpreting user-supplied data as SQL code. Use PDO or mysqli with prepared statements. Also validate the data types and lengths before inserting into the database.

Code Quality Metrics

Code Metrics Issues

- **Readability:** The code is relatively simple and easy to understand in its current form. However, the lack of error handling and security measures detracts from the overall readability, as it's not immediately clear that the code is unsafe.
- **Coupling:** The code has high coupling with the `conexion.php` file. This is acceptable if it is the designated location for database connection settings.
- **Error Handling:** The code lacks robust error handling. It only checks if the query was successful but doesn't handle potential errors during JSON decoding or database connection issues.
- **Validation:** The code doesn't validate or sanitize the input data. This is a major issue considering the SQL injection vulnerability. Data type and length validation are necessary.

Proposed Solution

Secure and Improved Code

The following code snippet demonstrates how to mitigate the SQL injection vulnerability and improve code quality using prepared statements and PDO:

```
<?php
try {
    include "../../models/conexion.php"; // Make sure the path is correct

    // Set PDO to throw exceptions on errors.
    $conexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    $data = json_decode(file_get_contents('php://input'), true);

    $nombre = $data['nombre'] ?? null; // Use null coalescing operator
    $cantidad = $data['cantidad'] ?? null;
    $precio = $data['precio'] ?? null;

    // Validate data (example)
    if (empty($nombre) || empty($cantidad) || empty($precio)) {
        throw new Exception("Missing data.");
    }

    if (!is_numeric($cantidad) || !is_numeric($precio)) {
      throw new Exception("Cantidad and Precio must be numeric.");
    }

    // Prepare the SQL statement
    $sql = "INSERT INTO productos (nombre, cantidad, precio) VALUES (:nombre, :cantidad,
:precio)";
    $stmt = $conexion->prepare($sql);

    // Bind the parameters
    $stmt->bindParam(':nombre', $nombre, PDO::PARAM_STR);
    $stmt->bindParam(':cantidad', $cantidad, PDO::PARAM_INT);
    $stmt->bindParam(':precio', $precio, PDO::PARAM_STR); //Or PDO::PARAM_INT/PDO::PARAM_FLOAT
depending on database schema

    // Execute the statement
    $stmt->execute();

    echo json_encode(["message" => "Producto guardado exitosamente"]);

} catch (PDOException $e) {
    echo json_encode(["message" => "Error al guardar producto: " . $e->getMessage()]);
} catch (Exception $e) {
    echo json_encode(["message" => "Error: " . $e->getMessage()]);
}
?>
```

Improvements:

- **SQL Injection Prevention:** Using prepared statements with parameter binding eliminates the SQL injection vulnerability.
- **Error Handling:** Implemented try-catch blocks to handle potential exceptions during database operations and JSON decoding, providing more informative error messages.
- **Data Validation:** Added basic data validation to check for missing or invalid input. More comprehensive validation should be performed based on specific requirements.
- **Readability:** The code is now more readable due to the explicit parameter binding and improved error handling.

```
```