Reporte de Vulnerabilidad

Archivo: transaccion.php

Código Analizado:

```
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.2/css/all.min.css">
    <title>Hello, world!</title>
</head>

<body>

    <div class="container-fluid-row ">

        <div class="row px-4">
            <h1>Prueba Tecnica</h1>
            <div class="col">
                <div class="card">
                    <div class="card-body">
                        <form action="" method="POST">
                            <!-- Mostrar Error     -->
                            <?php
                            include "models/conexion.php";
                            include "controller/transaccion.php";

                            ?>
                            <div class="mb-3">
                                <label for="producto_id" class="form-label">ID del
Producto</label>
                                <input type="number" class="form-control" id="producto_id"
name="producto_id" required>
                            </div>
                            <div class="mb-3">
                                <label for="cantidad" class="form-label">Cantidad
Vendida</label>
                                <input type="number" class="form-control" id="cantidad"
name="cantidad" required>
                            </div>
                            <div class="mb-3">
                                <label for="precio" class="form-label">Precio</label>
                                <input type="number" class="form-control" id="precio"
name="precio" step="0.01" required>
                            </div>
                            <button type="submit" name="btntransaccion" class="btn btn-
success" value="ok">Registrar Venta</button>
                        </form>
                    </div>
                </div>
            </div>

        </div>
    </div>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
```

```
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
integrity="sha384-IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
integrity="sha384-cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
crossorigin="anonymous"></script>
    -->
</body>

</html>
```

Análisis: ```html
Vulnerabilidades de Seguridad

Inclusión de archivos sin validación

Tipo: Inclusión de archivos locales/remotos (LFI/RFI).
Línea Aproximada:

```
<?php
                            include "models/conexion.php";
                            include "controller/transaccion.php";
                            ?>
```

Descripción: Incluir archivos directamente basados en variables externas o sin validación puede permitir a un atacante incluir archivos arbitrarios del servidor (LFI) o incluso desde servidores remotos (RFI). Si `conexion.php` o `transaccion.php` contienen código malicioso, o si se puede manipular el sistema para que estos archivos contengan código malicioso, el atacante puede ejecutar ese código en el servidor.
Mitigación:

- **Validar y sanitizar las entradas:** Asegurarse de que cualquier variable que influya en la ruta del archivo incluido esté correctamente validada y sanitizada. En este caso, como las inclusiones son fijas, no hay entrada a validar directamente aquí. El riesgo pasa porque los archivos incluidos permitan entradas no validas y tengan vulnerabilidades.
- **Usar una lista blanca:** En lugar de permitir cualquier archivo, definir una lista blanca de archivos que pueden ser incluidos.
- **Deshabilitar la inclusión de URLs remotas:** En el archivo `php.ini`, establecer `allow_url_include = Off`.
- **Control de acceso:** Asegurar que solo los usuarios autorizados puedan modificar los archivos incluidos.

Mejora de Métricas: N/A (esta vulnerabilidad no está directamente relacionada con métricas de calidad de código tradicionales, pero sí con la seguridad).

Posible Inyección SQL (Dependiendo del Código en Archivos Incluidos)

Tipo: Inyección SQL.
Línea Aproximada: Depende del código dentro de `models/conexion.php` y `controller/transaccion.php`, específicamente en las consultas a la base de datos.
Descripción: Si los archivos incluidos (especialmente `controller/transaccion.php`) contienen consultas SQL construidas concatenando cadenas con datos proporcionados por el usuario (por ejemplo, `$_POST['producto_id']`, `$_POST['cantidad']`, `$_POST['precio']`), existe una alta probabilidad de vulnerabilidad de inyección SQL. Un atacante podría manipular estas entradas para ejecutar comandos SQL arbitrarios, comprometiendo la integridad de la base de datos.
Mitigación:

- **Usar sentencias preparadas (Prepared Statements) con PDO o MySQLi:** Las sentencias preparadas permiten separar la consulta SQL de los datos, evitando que los datos sean interpretados como parte de la consulta.
- **Escapar las entradas del usuario:** Si no es posible usar sentencias preparadas, escapar las entradas del usuario antes de usarlas en la consulta SQL. Usar funciones como `mysqli_real_escape_string()` (con MySQLi)

para escapar caracteres especiales.
- **Validar las entradas:** Validar que los datos proporcionados por el usuario tengan el formato esperado (por ejemplo, que `producto_id`, `cantidad` y `precio` sean números).
- **Principio de mínimo privilegio:** Asegurar que la cuenta de usuario de la base de datos utilizada por la aplicación tenga solo los permisos necesarios para realizar las operaciones requeridas.

Mejora de Métricas: N/A (esta vulnerabilidad no está directamente relacionada con métricas de calidad de código tradicionales, pero sí con la seguridad).
Métricas de Calidad del Código

Acoplamiento

Descripción: El código presenta un acoplamiento moderado debido a las inclusiones de archivos (`models/conexion.php` y `controller/transaccion.php`). Cualquier cambio en estos archivos podría afectar al script principal.
Mejora:

- **Inyección de dependencias:** Considerar el uso de un contenedor de inyección de dependencias para reducir el acoplamiento entre los componentes.
- **Interfaces:** Definir interfaces para los componentes y depender de las interfaces en lugar de las implementaciones concretas.

Legibilidad

Descripción: El código es relativamente legible, pero podría mejorarse con comentarios y una mejor organización.
Mejora:

- **Comentarios:** Agregar comentarios para explicar la funcionalidad de las diferentes secciones del código.
- **Formato:** Mantener un formato consistente y legible.
- **Nombres descriptivos:** Usar nombres descriptivos para las variables y funciones.

Complejidad Ciclomática

Descripción: La complejidad ciclomática es baja en este fragmento, pero podría aumentar significativamente en los archivos incluidos (`models/conexion.php` y `controller/transaccion.php`).
Mejora:

- **Refactorización:** Dividir funciones largas y complejas en funciones más pequeñas y manejables.
- **Patrones de diseño:** Utilizar patrones de diseño para simplificar la lógica y reducir la complejidad.

Solución Propuesta

Refactorización y Medidas de Seguridad

1. **Validación exhaustiva:** Implementar validación de entrada en el archivo `controller/transaccion.php` para los campos `producto_id`, `cantidad`, y `precio`. Asegurar que sean numéricos y estén dentro de rangos aceptables. Utilizar filtros de saneamiento de PHP (`filter_var`) para evitar entradas maliciosas.
2. **Sentencias preparadas:** Usar sentencias preparadas con PDO para interactuar con la base de datos en `controller/transaccion.php`. Esto previene la inyección SQL.
3. **Control de errores:** Implementar un manejo de errores adecuado en `controller/transaccion.php` y `models/conexion.php`. Registrar los errores en un archivo de log para su posterior análisis, en lugar de mostrarlos directamente al usuario.
4. **Estructura del proyecto:** Considerar una estructura de proyecto MVC más completa para separar la lógica de la presentación y mejorar la mantenibilidad.
5. **Aislamiento:** Implementar aislamiento en la base de datos, usando usuarios con privilegios mínimos necesarios.

```
```