Vulnerability Report

Archivo: edit.php

Code Analyzed:

```
<?php
include "models/conexion.php";
$id = $_GET["id"];
echo $id;

$sql = $conexion->query("SELECT * FROM PERSONAS WHERE id = $id");
?>
<!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 "controller/modificar.php";
                            // Suponiendo que la consulta ya fue ejecutada y $sql contiene un
solo registro
                            if ($a = $sql->fetch_object()) { ?>
                                <h3>Actudalizar Registro de Persona</h3>
                                <input type="hidden" name="id" value="<?= htmlspecialchars($a-
>id) ?>">
                                <div class="mb-3">
                                    <label for="nombre" class="form-label">Nombre</label>
                                    <input type="text" class="form-control" id="nombre"
name="nombre" value="<?= htmlspecialchars($a->nombre) ?>" required>
                                </div>
                                <div class="mb-3">
                                    <label for="apellido" class="form-label">Apellido</label>
                                    <input type="text" class="form-control" id="apellido"
name="apellido" value="<?= htmlspecialchars($a->apellido) ?>" required>
                                </div>
                                <div class="mb-3">
                                    <label for="dni" class="form-label">Cedula</label>
                                    <input type="text" class="form-control" id="dni"
name="dni" value="<?= htmlspecialchars($a->cedula) ?>" required>
                                </div>
                                <div class="mb-3">
                                    <label for="email" class="form-label">Email</label>
                                    <input type="email" class="form-control" id="email"
name="email" value="<?= htmlspecialchars($a->correo) ?>" required>
                                </div>
                                <button type="submit" name="btnactualizar" class="btn btn-
warning" value="ok">Actualizar</button>
                                <a href="index.php" class="btn btn-secondary">Cancelar</a>
                            <?php } else { ?>
```

```
                                <p>No se encontró el registro</p>
                            <?php } ?>

                        </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>
```

Analysis: ```html
Security Vulnerabilities

SQL Injection

Type: SQL Injection
Line: 3 & 5
Description: The code directly includes user-supplied input (`$_GET["id"]`) into the SQL query without proper sanitization or parameterization. This allows an attacker to inject arbitrary SQL code, potentially leading to data breaches, data manipulation, or even complete database takeover. The echoed value of $id on line 4 could also reveal information about your application if an attacker injects code that outputs database names etc.
Mitigation: Use prepared statements with parameterized queries (PDO or MySQLi) to separate the SQL code from the user-supplied data. This ensures that the input is treated as data and not as part of the SQL command.

Code Quality Metrics

Code Quality Issues

Complexity: The code is relatively simple for a small application, but the lack of error handling and the direct integration of database interaction within the presentation layer increase complexity in the long run due to maintainability challenges.
Duplication: While not immediately evident, including the database connection and controller directly within the view might lead to duplication if this pattern is repeated in other files.
Readability: Readability is fair, but could be improved by separating database access into a dedicated data access layer and employing consistent coding standards. Mixing PHP code with HTML can make it harder to follow.
Coupling: High coupling exists between the presentation layer (HTML) and the data access layer (direct SQL query). Changes in the database structure or access method will necessitate modifications throughout the code. The inclusion of "controller/modificar.php" directly inside the view also increases coupling.
Error Handling: The code lacks proper error handling. If the database connection fails or the query returns an error, the script may terminate abruptly or expose sensitive information.

Proposed Solution

Secure and Improve Code

1. Parameterized Queries (PDO Example): Replace the vulnerable SQL query with a parameterized query using PDO. This is the most critical security improvement.

```
<?php
include "models/conexion.php";
$id = $_GET["id"];

try {
    $stmt = $conexion->prepare("SELECT * FROM PERSONAS WHERE id = :id");
    $stmt->bindParam(':id', $id, PDO::PARAM_INT); // Ensure $id is treated as an integer

    if ($stmt->execute()) {
        $a = $stmt->fetch(PDO::FETCH_OBJ);

    } else {
        // Handle query execution error
        echo "<p>Error executing query.</p>";
        exit(); // Or redirect to an error page
    }


} catch (PDOException $e) {
    // Handle database connection or query errors
    echo "<p>Database error: " . htmlspecialchars($e->getMessage()) . "</p>";
    exit(); // Or redirect to an error page
}
?>
```

2. Input Validation: Before using `$_GET["id"]`, validate that it's an integer. This can prevent non-numeric values from being passed to the database. While PDO::PARAM_INT provides some protection, explicitly validating the input is best practice.

```
<?php
$id = $_GET["id"] ?? null; // Use null coalescing operator to handle missing ID

if (!is_numeric($id)) {
    echo "<p>Invalid ID format.</p>";
    exit(); // Or redirect to an error page
}

$id = (int)$id; // Cast to integer
?>
```

3. Data Sanitization (Output Encoding): The code already uses `htmlspecialchars()` for output encoding, which is good. Ensure this is applied consistently throughout the application. **4. Separation of Concerns (MVC):** Move the database interaction logic (the query) out of the view and into a model or a dedicated data access object. Move the logic of "controller/modificar.php" into a controller class and invoke it from the view. This reduces coupling and improves maintainability. **5. Error Handling:** Wrap database operations in `try...catch` blocks to handle potential exceptions. Log errors for debugging purposes and display user-friendly error messages to avoid exposing sensitive information. **6. Avoid Direct Includes in Views:** Consider creating functions or methods in a separate file and including *that* file, rather than directly including entire controllers. This can improve organization. **7. Code Formatting and Comments:** Use consistent code formatting and add comments to explain complex logic. Tools like linters and formatters can help maintain code consistency. **Impact on Code Quality Metrics:** * **Complexity:** Reduced by separating concerns and using more modular code. * **Duplication:** Minimized by centralizing database access logic. * **Readability:** Improved by using consistent formatting, comments, and meaningful variable names. * **Coupling:** Significantly reduced by separating the presentation layer from the data access layer.
```
```