Reporte de Vulnerabilidad

Archivo: eliminar.php

Código Analizado:

```
<?php
    if (!empty($_GET["id"])) {
        # code...
        $id = $_GET["id"];

        $sql = $conexion->query("DELETE FROM PERSONAS WHERE ID = $id");
        if ($sql == 1) {
            # code...
            echo '<div class="alert alert-warning" role="alert">
            Persona no eliminada correctamente.
          </div>';
        }else{
            echo '<div class="alert alert-danger" role="alert">
            Persona no eliminada correctamente.
          </div>';
        }
    }
?>
```

Análisis: ```html
Vulnerabilidades de Seguridad

Inyección SQL

Tipo: Inyección SQL
Línea Aproximada: `$sql = $conexion->query("DELETE FROM PERSONAS WHERE ID = $id");`
Descripción: La variable `$id`, obtenida directamente de `$_GET["id"]`, se utiliza sin validación ni sanitización en la consulta SQL. Un atacante puede manipular el valor de `id` en la URL para inyectar código SQL malicioso. Por ejemplo, `?id=1 OR 1=1;--` eliminaría todos los registros de la tabla `PERSONAS`.

Métricas de Calidad del Código

Mejoras Necesarias

Seguridad: La principal preocupación es la vulnerabilidad de inyección SQL.
Legibilidad: El código es relativamente legible, pero la falta de manejo de errores específico (aparte de la simple comparación con `1`) dificulta la depuración. Los mensajes de alerta son confusos, ya que el `alert-warning` dice "no eliminada correctamente" y el `alert-danger` también. Deberían ser al revés y/o más descriptivos.
Robustez: El código no maneja casos donde `$conexion` sea nula o donde la consulta falle por otras razones (restricciones de la base de datos, etc.). Tampoco se valida si `$id` es un número antes de usarlo en la consulta.

Solución Propuesta

Código Mejorado

Mitigación de Inyección SQL: Usar consultas preparadas con parámetros enlazados.
Manejo de Errores: Implementar un manejo de errores más detallado.
Validación: Validar y sanitizar la entrada del usuario.

```
<?php
    if (!empty($_GET["id"])) {
        $id = $_GET["id"];
```

```
        // Validar que $id sea un entero
        if (!is_numeric($id)) {
            echo '<div class="alert alert-danger" role="alert">ID no válido.</div>';
            exit; // Salir del script para evitar la inyección SQL
        }

        // Utilizar consultas preparadas para prevenir la inyección SQL
        $stmt = $conexion->prepare("DELETE FROM PERSONAS WHERE ID = ?");
        $stmt->bind_param("i", $id); // "i" indica que $id es un entero

        if ($stmt->execute()) {
            if ($stmt->affected_rows > 0) {
                echo '<div class="alert alert-success" role="alert">Persona eliminada
correctamente.</div>';
            } else {
                echo '<div class="alert alert-warning" role="alert">No se encontró ninguna
persona con ese ID.</div>';
            }
        } else {
            echo '<div class="alert alert-danger" role="alert">Error al eliminar la persona: '
. $conexion->error . '</div>';
        }

        $stmt->close();
    }
?>
```

Explicación de la Solución:

- **Validación de Entrada:** Se asegura que `$id` sea un número antes de utilizarlo.
- **Consultas Preparadas:** Se utiliza `$conexion->prepare()` para preparar la consulta SQL. Luego, `$stmt->bind_param()` asocia el valor de `$id` como un entero (`"i"`). Esto previene la inyección SQL porque el valor se trata como datos, no como código SQL ejecutable.
- **Manejo de Errores Mejorado:** Se utiliza `$stmt->execute()` para ejecutar la consulta y se comprueba si se realizó correctamente. Se utiliza `$stmt->affected_rows` para determinar si se eliminó algún registro. Además, se muestra el mensaje de error específico de la base de datos si falla la consulta.

```
```