Reporte de Vulnerabilidad

Archivo: modificar.php

Código Analizado:

```
<?php
if (!empty($_POST["btnactualizar"])) {
    if (
        !empty($_POST["id"]) &&
        !empty($_POST["nombre"]) &&
        !empty($_POST["apellido"]) &&
        !empty($_POST["dni"]) &&

        !empty($_POST["email"])
    ) {
        // Captura de los valores
        $nombre = $_POST["nombre"];
        $apellido = $_POST["apellido"];
        $dni = $_POST["dni"];

        $email = $_POST["email"];

        // Utilizamos consultas preparadas para evitar inyecciones SQL
        $stmt = $conexion->prepare("UPDATE PERSONAS
        SET
        nombre = ?,
        apellido = ?,
        cedula = ?,
        fecha_nacimiento = NOW(),
        correo = ?
        WHERE id = ?");

        // Vinculamos los parámetros
        $stmt->bind_param("sssss", $nombre, $apellido, $dni, $email, $id);

        // Ejecutamos la consulta
        if ($stmt->execute()) {
            // Almacenamos el mensaje en la sesión
            session_start();
            $_SESSION['mensaje'] = 'Persona Actualizada correctamente.';

            // Redirigimos al index.php
            header("Location: index.php");
            exit;  // Es importante llamar a exit después de la redirección
        } else {
            echo '<div class="alert alert-danger" role="alert">
            Persona no Actualizada correctamente.
          </div>';
        }

        // Cerramos el statement
        $stmt->close();
    } else {
        echo '<div class="alert alert-danger" role="alert">
                Alguno de los campos está vacío.
              </div>';
    }
}
```

Análisis: ```html
Análisis de Vulnerabilidades

Vulnerabilidad: Inyección SQL (Potencial) y Falta de Saneamiento de Datos.

Línea Aproximada: La vulnerabilidad está en el uso de `$_POST["id"]` en la consulta UPDATE sin validar, aun cuando se preparan consultas. Aunque las otras variables son parametrizadas, el `id` se usa sin filtrar ni parametrizar.

Descripción: Aunque el código usa sentencias preparadas, existe una vulnerabilidad latente. El valor de `$_POST["id"]` no está siendo filtrado ni sanitizado antes de ser usado en la cláusula `WHERE id = ?`. Si `$_POST["id"]` contiene código SQL malicioso, podría eludir la protección de la sentencia preparada e inyectar código SQL. Además, aunque los datos son sanitizados al usar sentencias preparadas, no se hace validación para tipos de datos, longitudes máximas, o formatos esperados. Esto puede causar errores de base de datos o comportamientos inesperados.

Mitigación:

- **Validar y Sanitizar `id`:** Antes de usar `$_POST["id"]`, convertirlo a un entero usando `intval($_POST["id"])`. Esto asegura que sea un número entero, previniendo inyección SQL vía ese parámetro.**
- **Validar Tipos de Datos y Longitudes:** Implementar validación para cada campo para asegurar que coinciden con el tipo de dato esperado (ej: email válido, DNI con formato correcto) y que no exceden la longitud máxima permitida en la base de datos. Usar funciones como `filter_var` para validación de email.**
- **Escapar Salida:** Aunque no directamente relacionado con este fragmento, siempre escapar la salida al HTML para prevenir XSS.**

Métricas de Calidad del Código

Complejidad Ciclomática: Baja. El código tiene una estructura condicional sencilla (if/else). Sin embargo, se puede mejorar aún más dividiendo la lógica en funciones más pequeñas.

Duplicación: Baja. No hay duplicación significativa en este fragmento.

Legibilidad: Moderada. El código es relativamente fácil de entender, pero se puede mejorar añadiendo comentarios más descriptivos, especialmente sobre la validación y sanitización de datos.

Acoplamiento: Moderado. El código está acoplado a las variables `$_POST` y a la variable de conexión `$conexion` (no mostrada, pero implícita). Se puede reducir el acoplamiento usando inyección de dependencias para la conexión a la base de datos.

Solución Propuesta

El siguiente código incluye la mitigación para la inyección SQL en el ID y validaciones adicionales:

```
<?php
if (!empty($_POST["btnactualizar"])) {
    if (
        !empty($_POST["id"]) &&
        !empty($_POST["nombre"]) &&
        !empty($_POST["apellido"]) &&
        !empty($_POST["dni"]) &&
        !empty($_POST["email"])
    ) {
        // Captura y validación de los valores
        $id = intval($_POST["id"]); // Sanitize ID
        $nombre = htmlspecialchars(trim($_POST["nombre"])); // Escape HTML
        $apellido = htmlspecialchars(trim($_POST["apellido"])); // Escape HTML
        $dni = htmlspecialchars(trim($_POST["dni"])); // Escape HTML

        $email = filter_var($_POST["email"], FILTER_SANITIZE_EMAIL);  // Sanitize email
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            echo '<div class="alert alert-danger" role="alert">Email no válido.</div>';
            exit;
        }


        // Utilizamos consultas preparadas para evitar inyecciones SQL
        $stmt = $conexion->prepare("UPDATE PERSONAS
        SET
        nombre = ?,
```

```
        apellido = ?,
        cedula = ?,
        fecha_nacimiento = NOW(),
        correo = ?
        WHERE id = ?");

        // Vinculamos los parámetros
        $stmt->bind_param("ssssi", $nombre, $apellido, $dni, $email, $id);

        // Ejecutamos la consulta
        if ($stmt->execute()) {
            // Almacenamos el mensaje en la sesión
            session_start();
            $_SESSION['mensaje'] = 'Persona Actualizada correctamente.';

            // Redirigimos al index.php
            header("Location: index.php");
            exit;  // Es importante llamar a exit después de la redirección
        } else {
            echo '<div class="alert alert-danger" role="alert">
            Persona no Actualizada correctamente.
          </div>';
        }

        // Cerramos el statement
        $stmt->close();
    } else {
        echo '<div class="alert alert-danger" role="alert">
                Alguno de los campos está vacío.
              </div>';
    }
}
?>
```

Mejoras:

- Se sanitiza el ID con `intval()` para prevenir inyección SQL.
- Se implementa validación de email con `filter_var()`.
- Se escapa HTML en los campos nombre, apellido, y DNI para evitar XSS.
- Se actualiza el tipo de dato del bind_param para que coincida con el nuevo tipo de la variable id, usando "i" para integer.

```
```