Reporte de Vulnerabilidad

Archivo: registro.php

Código Analizado:

```
<?php
if (!empty($_POST["btnregistrar"])) {
    if (
        !empty($_POST["nombre"]) &&
        !empty($_POST["apellido"]) &&
        !empty($_POST["dni"]) &&
        !empty($_POST["fecha"]) &&
        !empty($_POST["email"])
    ) {
        // Captura de los valores
        $nombre = $_POST["nombre"];
        $apellido = $_POST["apellido"];
        $dni = $_POST["dni"];
        $fecha = $_POST["fecha"];
        $email = $_POST["email"];

        // Utilizamos consultas preparadas para evitar inyecciones SQL
        $stmt = $conexion->prepare("INSERT INTO PERSONAS (nombre, apellido, cedula,
fecha_nacimiento, correo)
                                    VALUES (?, ?, ?, ?, ?)");

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

        // Ejecutamos la consulta
        if ($stmt->execute()) {
            echo '<div class="alert alert-success" role="alert">
                    Persona registrada correctamente.
                  </div>';
        } else {
            echo '<div class="alert alert-danger" role="alert">
                    Persona no registrada 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
Vulnerabilidades de Seguridad

Inyección SQL

Tipo: Potencial Inyección SQL. Aunque el código utiliza sentencias preparadas, el uso correcto depende de que `$conexion` sea un objeto PDO o mysqli con la configuración adecuada para evitar la emulación de sentencias preparadas. Si la emulación está habilitada (comportamiento predeterminado en algunas versiones de PHP), la protección es considerablemente menor. Además, no hay validación de tipos ni saneamiento de entradas.

Línea Aproximada: La vulnerabilidad reside en la forma en que se construye la consulta preparada. Si `$conexion` no está configurado correctamente, se podría explotar. Especialmente en:

```
$stmt = $conexion->prepare("INSERT INTO PERSONAS (nombre, apellido, cedula, fecha_nacimiento,
correo) VALUES (?, ?, ?, ?, ?)");
```

Mitigación:

1. **Asegurar sentencias preparadas reales:** Verificar la configuración de la conexión a la base de datos para desactivar la emulación de sentencias preparadas. Para PDO, usar: `$pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);`. Para mysqli, se desactiva por defecto a partir de PHP 8.1, pero verificar versiones anteriores.
2. **Validación y Saneamiento de Entradas:** Validar rigurosamente los datos de entrada. Verificar que `dni` sea un número, `fecha` tenga el formato correcto, y `email` sea una dirección de correo válida. Usar funciones como `filter_var` con `FILTER_VALIDATE_EMAIL` para validar el correo electrónico. Sanitizar las entradas para eliminar caracteres potencialmente peligrosos (e.g., HTML entities con `htmlspecialchars` o `strip_tags`).
3. **Principio del menor privilegio:** Asegurarse de que la cuenta de base de datos utilizada tenga los permisos mínimos necesarios para realizar las operaciones.

Cross-Site Scripting (XSS)

Tipo: Potencial XSS en los mensajes de éxito/error. Los datos que se muestran al usuario (e.g., "Persona registrada correctamente") no se escapan. Si el mensaje llegara a contener código HTML malicioso (poco probable en este caso, pero es una buena práctica), este se ejecutaría en el navegador del usuario.

Línea Aproximada: En todas las líneas donde se usa `echo` para mostrar mensajes al usuario. Por ejemplo:

```
echo '<div class="alert alert-success" role="alert"> Persona registrada correctamente.
</div>';
```

Mitigación:

1. Escapar la salida HTML usando `htmlspecialchars()` antes de mostrar cualquier dato al usuario. Por ejemplo: `echo '<div class="alert alert-success" role="alert">'.htmlspecialchars('Persona registrada correctamente').'</div>'`.

Métricas de Calidad del Código

Complejidad

Descripción: La complejidad ciclomática es baja. La lógica principal está contenida en una estructura `if/else`.

Mejora: En este caso, la complejidad no es un problema. Si el código creciera y tuviera más validaciones, se podría refactorizar en funciones separadas para mejorar la legibilidad.

Duplicación

Descripción: Hay una pequeña duplicación en la estructura de los mensajes de alerta (`div class="alert..."`).

Mejora: Se podría extraer la generación de la alerta a una función separada, pasándole el tipo de alerta y el mensaje como parámetros. Esto reduciría la duplicación y facilitaría el mantenimiento. Por ejemplo: `function showAlert($type, $message) { return '<div class="alert alert-'.$type.'" role="alert">'.htmlspecialchars($message).'</div>'; }` y usarlo como `echo showAlert('success', 'Persona registrada correctamente');`.

Legibilidad

Descripción: El código es relativamente legible. El uso de comentarios es adecuado.

Mejora: Se podrían añadir más comentarios explicativos, especialmente sobre la configuración de la conexión a la base de datos y el manejo de errores. Usar nombres de variables más descriptivos podría mejorar aún más la legibilidad.

Acoplamiento

Descripción: El código está fuertemente acoplado a la base de datos y a la estructura `$_POST`.

Mejora:

- **Inyección de Dependencias:** Se podría desacoplar la conexión a la base de datos inyectando el objeto `$conexion` en lugar de depender de una variable global.
- **Capa de Abstracción:** Crear una capa de abstracción para el acceso a la base de datos (un repositorio) para aislar la lógica de acceso a datos del resto del código.
- **Validación en otra capa:** Aislar la lógica de validación de `$_POST` en una función o clase separada.

Solución Propuesta

Este fragmento muestra una versión mejorada con algunas de las mitigaciones sugeridas. Asume que se usa PDO y se ha configurado correctamente (`ATTR_EMULATE_PREPARES => false`). También se añade validación básica de los datos.

```
<?php
// Función para generar mensajes de alerta HTML
function showAlert($type, $message) {
    return '<div class="alert alert-'.$type.'"
role="alert">'.htmlspecialchars($message).'</div>';
}

if (!empty($_POST["btnregistrar"])) {
    // Validar que los campos no estén vacíos
    if (
        !empty($_POST["nombre"]) &&
        !empty($_POST["apellido"]) &&
        !empty($_POST["dni"]) &&
        !empty($_POST["fecha"]) &&
        !empty($_POST["email"])
    ) {
        // Validar el formato del email
        if (!filter_var($_POST["email"], FILTER_VALIDATE_EMAIL)) {
            echo showAlert('danger', 'El email no es válido.');
            exit; // Detener la ejecución si la validación falla
        }

        // Saneamiento de entradas (ejemplo básico)
        $nombre = htmlspecialchars($_POST["nombre"]);
        $apellido = htmlspecialchars($_POST["apellido"]);
        $dni = htmlspecialchars($_POST["dni"]);
        $fecha = htmlspecialchars($_POST["fecha"]);
        $email = htmlspecialchars($_POST["email"]);

        try {
            // Utilizamos consultas preparadas para evitar inyecciones SQL
            $stmt = $conexion->prepare("INSERT INTO PERSONAS (nombre, apellido, cedula,
fecha_nacimiento, correo)
                                        VALUES (?, ?, ?, ?, ?)");

            // Vinculamos los parámetros
            $stmt->bindParam(1, $nombre, PDO::PARAM_STR);
            $stmt->bindParam(2, $apellido, PDO::PARAM_STR);
            $stmt->bindParam(3, $dni, PDO::PARAM_STR);
            $stmt->bindParam(4, $fecha, PDO::PARAM_STR);
            $stmt->bindParam(5, $email, PDO::PARAM_STR);

            // Ejecutamos la consulta
            if ($stmt->execute()) {
                echo showAlert('success', 'Persona registrada correctamente.');
            } else {
                echo showAlert('danger', 'Persona no registrada correctamente.');
            }

            // Cerramos el statement
            $stmt->closeCursor(); // Usar closeCursor() en PDO
        } catch (PDOException $e) {
            // Manejar errores de la base de datos
            error_log("Error en la base de datos: " . $e->getMessage()); // Log del error
```

```
            echo showAlert('danger', 'Error al registrar persona. Por favor, intente de nuevo
más tarde.'); // Mensaje genérico al usuario
        }

    } else {
        echo showAlert('danger', 'Alguno de los campos está vacío.');
    }
}
?>
```