Reporte de Vulnerabilidad

Archivo: Cliente.php

Código Analizado:

```
<?php
class Cliente extends Conectar
{
   /*  public function listarCliente()
    {
        $conectar = parent::Conexion();
        $sql = "SP_L_Cliente_01";
        $query = $conectar->prepare($sql);

        $query->execute();
        return $query->fetchAll(PDO::FETCH_ASSOC);
    } */

    public function listarCliente()
    {
        $conectar = parent::Conexion();
        $sql = "CALL SP_L_Cliente_01()"; // Corrección: Usar CALL
        $query = $conectar->prepare($sql);
        $query->execute();
        return $query->fetchAll(PDO::FETCH_ASSOC);
    }


    public function Cliente_id($id)
    {
        $conectar = parent::Conexion();
        $sql = "CALL SP_L_Cliente_02(?)";
        $query = $conectar->prepare($sql);
        $query->bindValue(1, $id);
        $query->execute();
        return $query->fetchAll(PDO::FETCH_ASSOC);
    }


}
```

Análisis: ```html
Análisis de Seguridad

Posible Vulnerabilidad: Inyección SQL

Tipo: Inyección SQL (SQLi)

Línea Aproximada: `$sql = "CALL SP_L_Cliente_02(?)";` dentro de la función `Cliente_id($id)`

Descripción: La función `Cliente_id($id)` utiliza un procedimiento almacenado (SP_L_Cliente_02) pero pasa el valor de `$id` como parámetro usando `bindValue`, lo cual es seguro contra inyección SQL directamente en la consulta. Sin embargo, la seguridad final depende de la implementación del procedimiento almacenado `SP_L_Cliente_02`. Si dentro del procedimiento almacenado se concatena el parámetro `$id` directamente en una consulta SQL, la aplicación sigue siendo vulnerable.

Mitigación:

- **Revisar el código del procedimiento almacenado `SP_L_Cliente_02`**: Asegurarse de que NO concatene el parámetro `$id` directamente en una consulta SQL dentro del procedimiento almacenado. Debe usar parámetros en el procedimiento almacenado también.
- **Validación del input:** Antes de llamar a la función `Cliente_id`, validar y limpiar el valor de `$id`. Asegurarse de que sea del tipo esperado (por ejemplo, un entero) y que no contenga caracteres inesperados.

Métricas de Calidad del Código

Análisis de Calidad

Complejidad: Baja. El código es relativamente simple y directo.

Duplicación: Baja. La estructura de las funciones `listarCliente` y `Cliente_id` es similar, pero la duplicación no es excesiva.

Legibilidad: Buena. El código es fácil de entender, con nombres de variables y funciones descriptivos. Los comentarios, aunque escasos, ayudan.

Acoplamiento: Medio. La clase `Cliente` depende de la clase `Conectar` (herencia). El acoplamiento a la base de datos también es inherente al propósito de la clase.

Mejoras:

- **Manejo de Errores:** No hay manejo de errores explícito. Agregar bloques `try...catch` para manejar excepciones que puedan ocurrir durante la conexión a la base de datos o la ejecución de las consultas. Registrar los errores en un archivo de log.
- **Abstracción de la capa de datos:** Considerar el uso de un patrón de repositorio para abstraer aún más la capa de acceso a datos, lo que facilitaría las pruebas unitarias y el cambio de la implementación de la base de datos en el futuro.
- **Comentarios/Documentación:** Agregar comentarios JSDoc (o equivalentes de PHP) para documentar las funciones y sus parámetros.

Solución Propuesta

El siguiente código muestra una propuesta de solución que incluye validación de entrada y manejo de errores:

```
<?php
class Cliente extends Conectar
{
    /**
     * Lista todos los clientes.
     *
     * @return array|false Un array asociativo con la información de los clientes, o false en
caso de error.
     */
    public function listarCliente()
    {
        try {
            $conectar = parent::Conexion();
            $sql = "CALL SP_L_Cliente_01()";
            $query = $conectar->prepare($sql);
            $query->execute();
            return $query->fetchAll(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            error_log("Error en listarCliente: " . $e->getMessage()); // Log del error
            return false; // Indicar que hubo un error
        }
    }

    /**
     * Obtiene un cliente por su ID.
     *
     * @param int $id El ID del cliente a buscar.
     * @return array|false Un array asociativo con la información del cliente, o false en caso
de error.
     * @throws InvalidArgumentException Si el ID no es un entero válido.
     */
    public function Cliente_id($id)
    {
        // Validar que $id sea un entero
        if (!is_numeric($id) || intval($id) != $id) {
            throw new InvalidArgumentException("El ID del cliente debe ser un entero
válido.");
        }
```

```
        try {
            $conectar = parent::Conexion();
            $sql = "CALL SP_L_Cliente_02(?)";
            $query = $conectar->prepare($sql);
            $query->bindValue(1, $id, PDO::PARAM_INT); // Especificar el tipo de dato
            $query->execute();
            return $query->fetchAll(PDO::FETCH_ASSOC);
        } catch (PDOException $e) {
            error_log("Error en Cliente_id: " . $e->getMessage()); // Log del error
            return false; // Indicar que hubo un error
        }
    }
}
```

Explicación de las mejoras:

- Se añadió manejo de excepciones (`try...catch`) para capturar errores de PDO y registrarlos en un archivo de log (error_log).
- Se agregó validación de entrada en la función `Cliente_id` para asegurar que el parámetro `$id` sea un entero válido. Se lanza una excepción `InvalidArgumentException` si la validación falla.
- Se especificó el tipo de dato en `bindValue` (PDO::PARAM_INT) para mayor seguridad.

```
```