Reporte de Vulnerabilidad

Archivo: Empresa.php

Código Analizado:

```
<?php
class Empresas extends Conectar
{
   /*  public function listarEmpresas()
    {
        $conectar = parent::Conexion();
        $sql = "SP_L_Empresa_01";
        $query = $conectar->prepare($sql);

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

    public function listarEmpresas()
    {
        $conectar = parent::Conexion();
        $sql = "CALL SP_L_EMPRESA_01()"; // Corrección: Usar CALL
        $query = $conectar->prepare($sql);
        $query->execute();
        return $query->fetchAll(PDO::FETCH_ASSOC);
    }


    public function Empresa_id($id)
    {
        $conectar = parent::Conexion();
        $sql = "CALL SP_L_EMPRESA_02(?)";
        $query = $conectar->prepare($sql);
        $query->bindValue(1, $id);
        $query->execute();
        return $query->fetchAll(PDO::FETCH_ASSOC);
    }


}
```

Análisis: ```html
Vulnerabilidades de Seguridad

Inyección SQL (Potencial)

Aunque el código utiliza prepared statements, la vulnerabilidad depende de la implementación de las stored procedures. Si las stored procedures `SP_L_EMPRESA_01` y `SP_L_EMPRESA_02` no sanitizan correctamente las entradas o concatenan directamente datos en sus consultas internas, aún existe un riesgo de inyección SQL.

Línea Afectada (Potencial): Todas las llamadas a stored procedures, especialmente la de `Empresa_id`

- **`CALL SP_L_EMPRESA_01()`**
- **`CALL SP_L_EMPRESA_02(?)`**

Mitigación:

- Revisar y asegurar que las stored procedures `SP_L_EMPRESA_01` y `SP_L_EMPRESA_02` usen parámetros y sanitización interna apropiada para prevenir la inyección SQL. Esto es crítico.
- Aplicar el principio de mínimo privilegio a las credenciales de la base de datos.

Métricas de Calidad del Código

Análisis de Calidad

- **Complejidad Ciclomática:** Baja. Las funciones son simples y directas.
- **Duplicación de Código:** Hay una pequeña duplicación en la forma en que se establece la conexión a la base de datos en cada función.
- **Legibilidad:** Buena. El código es fácil de entender.
- **Acoplamiento:** El código está acoplado a la clase `Conectar` y a las stored procedures específicas de la base de datos.

Mejoras Sugeridas

- **Eliminar Duplicación:** Mover la lógica de conexión a la base de datos a una función reutilizable (posiblemente en la clase `Conectar`) para evitar la duplicación.
- **Abstracción:** Considerar el uso de una capa de abstracción de base de datos (DBAL) para reducir el acoplamiento a la implementación específica de PDO y las stored procedures. Esto facilitaría el cambio a otra base de datos o la realización de pruebas unitarias.

Solución Propuesta

La solución principal es asegurar que las stored procedures estén correctamente implementadas para prevenir inyecciones SQL. Además, se sugiere refactorizar el código para mejorar la mantenibilidad y reducir la duplicación.

Ejemplo de Refactorización (Extrayendo la conexión):

```
            getConnection();
                    $sql = "CALL SP_L_EMPRESA_01()";
                    $query = $conectar->prepare($sql);
                    $query->execute();
                    return $query->fetchAll(PDO::FETCH_ASSOC);
                }

                public function Empresa_id($id)
                {
                    $conectar = $this->getConnection();
                    $sql = "CALL SP_L_EMPRESA_02(?)";
                    $query = $conectar->prepare($sql);
                    $query->bindValue(1, $id);
                    $query->execute();
                    return $query->fetchAll(PDO::FETCH_ASSOC);
                }
            }
            ?>
```

```
```