# Reporte de Vulnerabilidad

**Archivo:** factura.php

**Código Analizado:**

```
<?php

require_once("../models/Factura.php");

if(isset($_GET['op'])){
    $op = $_GET['op'];

    switch($op){
        case "guardar":
            // Verifica si se recibió el parámetro 'factura' vía POST
            if(isset($_POST['factura'])){
                // Decodifica el JSON recibido
                $facturaData = json_decode($_POST['factura'], true);

                // Opcional: valida la estructura de $facturaData según tus necesidades

                // Instancia el modelo y llama al método para guardar la factura
                $facturaModel = new Factura();

                try {
                    $resultado = $facturaModel->guardarFactura($facturaData);

                    if($resultado){
                        // Responde con un mensaje de éxito
                        echo json_encode([
                            "status" => "success",
                            "message" => "Factura guardada correctamente."
                        ]);
                    } else {
                        echo json_encode([
                            "status" => "error",
                            "message" => "No se pudo guardar la factura."
                        ]);
                    }
                } catch(PDOException $e) {
                    // En caso de error, envía el mensaje de error
                    echo json_encode([
                        "status" => "error",
                        "message" => "Error al guardar la factura: " . $e->getMessage()
                    ]);
                }
            } else {
                echo json_encode([
                    "status" => "error",
                    "message" => "No se recibieron datos de la factura."
                ]);
            }
            break;

        default:
            echo json_encode([
                "status" => "error",
                "message" => "Operación no reconocida."
            ]);
            break;
    }
} else {
    echo json_encode([
        "status" => "error",
        "message" => "No se especificó ninguna operación."
    ]);
}
?>
```

**Análisis:** ```html
Análisis de Vulnerabilidades de Seguridad

#### Vulnerabilidad: Inyección SQL (Potencial)

**Línea Aproximada:** Dentro del método `guardarFactura` en el modelo `Factura.php` (no proporcionado). Si el método `guardarFactura` del modelo `Factura` no utiliza consultas preparadas o escaping adecuado, es vulnerable a inyección SQL. El código actual decodifica datos JSON y los pasa a un método, pero no muestra cómo se procesan esos datos *dentro* del modelo.

**Explicación:** Si los datos en `$facturaData` (especialmente cadenas) se insertan directamente en una consulta SQL sin validación o escaping, un atacante podría manipular los datos para ejecutar código SQL arbitrario.

**Mitigación:**

- **Consultas Preparadas:** Utilice consultas preparadas (prepared statements) con placeholders para todos los accesos a la base de datos dentro del modelo `Factura`. Esta es la forma más efectiva de prevenir la inyección SQL.
- **Validación y Saneamiento:** Valide rigurosamente los datos recibidos en `$facturaData` antes de usarlos en la consulta SQL. Saneé las entradas (por ejemplo, usando funciones de escaping específicas de la base de datos) para eliminar caracteres peligrosos.

Análisis de Calidad del Código

#### Métricas de Calidad

**Complejidad Ciclomática:** La complejidad ciclomática de este script es relativamente baja, debido a la simple estructura `switch`. Sin embargo, la complejidad aumenta significativamente si la lógica dentro del método `guardarFactura` es compleja.

**Acoplamiento:** El acoplamiento es moderado. El script depende del modelo `Factura`. Podría reducirse el acoplamiento utilizando una interfaz para el modelo `Factura`.

**Legibilidad:** El código es relativamente legible, con nombres de variables descriptivos y una estructura clara. Sin embargo, la falta de comentarios detallados sobre el propósito de cada sección dificulta la comprensión rápida.

**Duplicación:** Hay cierta duplicación en la forma en que se construyen las respuestas JSON de error.

**Mejoras:**

- **Comentarios:** Agregar comentarios para explicar la lógica detrás de cada bloque de código.
- **Refactorización:** Extraer la construcción de respuestas JSON en una función separada para evitar la duplicación de código. Esto también mejora la mantenibilidad.
- **Manejo de Errores:** Considerar el uso de un logger para registrar errores en lugar de simplemente enviarlos como respuesta JSON. Esto permite un mejor monitoreo y depuración.
- **Validación de Datos:** Implementar una validación robusta de los datos recibidos. Esto puede incluir validación de tipo, validación de rango y validación de formato. La validación debería realizarse antes de cualquier operación en la base de datos.

Solución Propuesta

#### Refactorización y Mitigación de Vulnerabilidades

Este es un ejemplo de cómo se podría refactorizar el código para mejorar la seguridad y la calidad:

```
$status, "message" => $message];
   if ($data !== null) {
       $response["data"] = $data;
   }
   return json_encode($response);
```

```
}


if(isset($_GET['op'])){
    $op = $_GET['op'];

    switch($op){
        case "guardar":
            // Verifica si se recibió el parámetro 'factura' vía POST
            if(isset($_POST['factura'])){
                // Decodifica el JSON recibido
                $facturaData = json_decode($_POST['factura'], true);

                // Validación de la estructura de $facturaData (ejemplo)
                if (!is_array($facturaData) || !isset($facturaData['cliente_id']) ||
!isset($facturaData['items'])) {
                    echo buildJsonResponse("error", "Estructura de datos de factura
inválida.");
                    break; // Salir del switch
                }

                // Instancia el modelo y llama al método para guardar la factura
                $facturaModel = new Factura();

                try {
                    $resultado = $facturaModel->guardarFactura($facturaData); //Asumiendo que
este metodo usa prepared statements

                    if($resultado){
                        // Responde con un mensaje de éxito
                        echo buildJsonResponse("success", "Factura guardada correctamente.");
                    } else {
                        echo buildJsonResponse("error", "No se pudo guardar la factura.");
                    }
                } catch(PDOException $e) {
                    // En caso de error, registra el mensaje y envía una respuesta genérica
                    error_log("Error al guardar la factura: " . $e->getMessage()); // Log del
error
                    echo buildJsonResponse("error", "Error al guardar la factura. Contacte al
administrador.");
                }
            } else {
                echo buildJsonResponse("error", "No se recibieron datos de la factura.");
            }
            break;

        default:
            echo buildJsonResponse("error", "Operación no reconocida.");
            break;
    }
} else {
    echo buildJsonResponse("error", "No se especificó ninguna operación.");
}
?>
```

**Mejoras Implementadas:**

- **Función `buildJsonResponse`:** Reducción de la duplicación de código para la creación de respuestas JSON.
- **Validación Básica:** Se añadió un ejemplo de validación de la estructura de los datos recibidos.
- **Manejo de Errores Mejorado:** Se reemplazó el envío directo del mensaje de error PDOException con un log del error y un mensaje genérico al cliente.

Recuerde que la implementación real de la seguridad SQL reside en el modelo `Factura.php`, el cual no se proporcionó en el código.

```
```