Reporte de Vulnerabilidad

Archivo: guardar.php

Código Analizado:

```
<?php
include "../../models/conexion.php"; // Asegúrate de que la ruta a este archivo es correcta
$data = json_decode(file_get_contents('php://input'), true);

$nombre = $data['nombre'];
$cantidad = $data['cantidad'];
$precio = $data['precio'];

$sql = "INSERT INTO productos (nombre, cantidad, precio) VALUES ('$nombre', '$cantidad',
'$precio')";

if ($conexion->query($sql) == true) {
    echo json_encode(["message" => "Producto guardado exitosamente"]);
} else {
    echo json_encode(["message" => "Error al guardar producto"]);
}

?>
```

Análisis: ```html
Vulnerabilidades de Seguridad
Inyección SQL

Tipo: Inyección SQL

Línea Aproximada:

```
9: $sql = "INSERT INTO productos (nombre, cantidad, precio) VALUES ('$nombre', '$cantidad',
'$precio')";
```

Descripción: El código construye la consulta SQL directamente concatenando los valores proporcionados por el usuario. Si los valores de `$nombre`, `$cantidad`, o `$precio` contienen caracteres especiales (como comillas simples), pueden modificar la estructura de la consulta, permitiendo la ejecución de código SQL arbitrario.

Mitigación: Utilizar consultas preparadas (prepared statements) con parámetros enlazados. Esta técnica separa la consulta SQL de los datos, impidiendo que los datos sean interpretados como parte del código SQL. PDO o MySQLi son excelentes opciones para implementar esto en PHP. La función `addslashes()` NO es una mitigación adecuada, ya que es propensa a errores y puede ser eludida.

Ejemplo de Mitigación (usando PDO):

```


prepare($sql);
$stmt->bindParam(':nombre', $nombre);
$stmt->bindParam(':cantidad', $cantidad);
$stmt->bindParam(':precio', $precio);

if ($stmt->execute()) {
    echo json_encode(["message" => "Producto guardado exitosamente"]);
} else {
    echo json_encode(["message" => "Error al guardar producto"]);
}
?>
```

Ruta sensible expuesta

Tipo: Exposición de la ruta interna.

Línea Aproximada:

```
1: include "../../models/conexion.php";
```

Descripción: Si un atacante es capaz de ver el código fuente, o se genera un error que revele la ruta, el atacante conocerá la estructura de directorios del servidor. Aunque no es una vulnerabilidad directa, facilita la explotación de otras vulnerabilidades.

Mitigación: Evitar exponer información innecesaria de la estructura interna del proyecto. Al manejar errores, evitar mostrar rutas completas. Se recomienda mantener la estructura interna lo más opaca posible desde el exterior. Configurar adecuadamente el servidor web para evitar la exposición de archivos PHP sin procesar.

Métricas de Calidad del Código
Legibilidad y Mantenibilidad

Problema: El código es relativamente simple, pero la concatenación directa en la consulta SQL reduce la legibilidad, especialmente al considerar la necesidad de escapar caracteres (ahora resuelto con prepared statements).

Mejora: El uso de prepared statements mejora drásticamente la legibilidad, ya que la lógica de la consulta SQL se separa de los datos. Usar nombres de variables descriptivos y comentarios concisos también contribuye a la legibilidad.

Acoplamiento

Problema: El código está acoplado a la base de datos directamente en el mismo script.

Mejora: Considerar el uso de un patrón de diseño como MVC (Modelo-Vista-Controlador) para separar la lógica de acceso a datos (modelo) de la lógica de presentación (vista) y la lógica de control (controlador). Esto reduce el acoplamiento y facilita la mantenibilidad y las pruebas. El archivo `conexion.php` puede considerarse un punto único de acceso a la base de datos, lo cual es una buena práctica en este contexto.

Manejo de Errores

Problema: El manejo de errores es básico. Simplemente indica si la operación se realizó con éxito o no, sin proporcionar detalles específicos sobre el error.

Mejora: Registrar los errores en un archivo de registro o en un sistema de monitoreo. En el caso de PDO, puedes obtener información detallada del error utilizando `$stmt->errorInfo()`. Evitar mostrar mensajes de error detallados al usuario final por razones de seguridad (podrían revelar información sensible de la base de datos).

Solución Propuesta (Código Mitigado)

```
 "Faltan parámetros requeridos (nombre, cantidad, precio)."]);
        exit;
    }

    $nombre = $data['nombre'];
    $cantidad = $data['cantidad'];
    $precio = $data['precio'];

    $sql = "INSERT INTO productos (nombre, cantidad, precio) VALUES (:nombre, :cantidad,
:precio)";
    $stmt = $conexion->prepare($sql);

    $stmt->bindParam(':nombre', $nombre);
    $stmt->bindParam(':cantidad', $cantidad, PDO::PARAM_INT); // Especifica el tipo de dato
    $stmt->bindParam(':precio', $precio, PDO::PARAM_STR); //Especifica el tipo de dato
```

```
    if ($stmt->execute()) {
        http_response_code(201); // Created
        echo json_encode(["message" => "Producto guardado exitosamente"]);
    } else {
        http_response_code(500); // Internal Server Error
        error_log("Error al guardar producto: " . print_r($stmt->errorInfo(), true)); //Log
del error
        echo json_encode(["message" => "Error al guardar producto.  Por favor, contacte al
administrador."]);
    }

} catch (PDOException $e) {
    http_response_code(500); // Internal Server Error
    error_log("Error de conexión a la base de datos: " . $e->getMessage());
    echo json_encode(["message" => "Error de conexión a la base de datos.  Por favor, contacte
al administrador."]);
} catch (Exception $e) {
    http_response_code(500); // Internal Server Error
    error_log("Error general: " . $e->getMessage());
    echo json_encode(["message" => "Error inesperado. Por favor, contacte al
administrador."]);
}
?>
```

```
```