Vulnerability Report

Archivo: actualizar.php

Code Analyzed:

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

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

$sql = "UPDATE productos SET nombre='$nombre', cantidad='$cantidad',
precio='$precio' WHERE id=$id";

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

?>
```

Analysis: ```html

Security Vulnerabilities

SQL Injection

Type: SQL Injection
Approximate Line: 9
Description: The code directly embeds user-provided data (`$nombre`, `$cantidad`, `$precio`, and `$id`) into the SQL query without proper sanitization or escaping. This allows an attacker to inject malicious SQL code, potentially compromising the entire database. A crafted `$nombre` value like `'; DROP TABLE productos; --` could lead to data loss or unauthorized access.

Code Quality Issues

Code Quality Metrics

- **Readability:** The code is relatively easy to read due to its small size and straightforward logic. However, embedding variables directly in the SQL query hurts readability and maintainability due to the SQL injection risk.

- **Coupling:** The code is tightly coupled with the database connection. A change in the database connection details requires modification of this code.
- **Security:** As mentioned, the major issue is SQL Injection.
- **Error Handling:** The code provides basic error handling but lacks detailed logging or error reporting.

Proposed Solution

Parameterized Queries (Prepared Statements)

Mitigation: Use parameterized queries (prepared statements) to prevent SQL injection. Prepared statements separate the SQL code from the data, ensuring that the data is treated as data, not as executable SQL code.

Improved Code:

```
<?php
include "../../models/conexion.php";

$data = json_decode(file_get_contents('php://input'), true);

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

// Use prepared statements to prevent SQL injection
$sql = "UPDATE productos SET nombre=?, cantidad=?, precio=? WHERE id=?";
$stmt = $conexion->prepare($sql);

// Check if prepare was successful
if ($stmt === false) {
    error_log("Prepare failed: " . $conexion->error);
    echo json_encode(["message" => "Error al preparar la consulta"]);
    exit; // Exit to prevent further execution with a broken statement
}

// Bind parameters
$stmt->bind_param("sidi", $nombre, $cantidad, $precio, $id);

// Execute the statement
if ($stmt->execute()) {
    echo json_encode(["message" => "Producto actualizado exitosamente"]);
} else {
```

```
    error_log("Execute failed: " . $stmt->error); // Log the specific error
    echo json_encode(["message" => "Error al actualizado producto: " . $stmt->error]);
}

// Close the statement
$stmt->close();
?>
```

Explanation:

- **Prepare statement:** `$conexion->prepare($sql)` prepares the SQL query with placeholders (`?`) for the variables.
- **Bind parameters:** `$stmt->bind_param("sidi", $nombre, $cantidad, $precio, $id)` binds the variables to the placeholders. The first argument `"sidi"` defines the data types of the variables: `s` for string, `i` for integer, and `d` for double. Ensure that these data types match the actual types of the database columns.
- **Execute statement:** `$stmt->execute()` executes the prepared statement with the bound parameters.
- **Error Handling:** Improved error handling with `error_log` to log the specific database errors. Includes checks for statement preparation errors.
- **Data Types:** Ensure that you use the correct data types in the `bind_param` function. Using the wrong type can still cause unexpected behavior.
- **Exit after error:** The code now exits if the prepare statement fails, preventing further execution with a broken query.

Improvements:

- **Security:** Eliminates SQL injection vulnerability.
- **Readability:** Slightly improves readability by separating the SQL code from the data.
- **Maintainability:** Improves maintainability as the code is less prone to errors due to SQL injection and easier to update the query.
- **Error Reporting:** Includes more specific error reporting using `error_log`.

```
```