Vulnerability Report

Archivo: producto.php

Code Analyzed:

```
<?php
include "../../models/conexion.php"; // Asegúrate de que la ruta a este archivo
es correcta

$id = $_GET["id"];
// Realizamos la consulta a la base de datos
$sql = $conexion->query("SELECT * FROM productos where $id");

$producto = $sql->fetch_assoc();


// Devolvemos los datos en formato JSON
echo json_encode($producto);
?>
```

Analysis: ```html

Security Vulnerabilities

SQL Injection

Type: SQL Injection
Approximate Line: 6 (`$sql = $conexion->query("SELECT * FROM productos where $id");`)
Description: The code directly incorporates the `$id` parameter from the GET request into the SQL query without any sanitization or escaping. This allows an attacker to inject arbitrary SQL code, potentially compromising the entire database. For example, an attacker could provide an `id` value like `"1 OR 1=1; --"` to bypass authentication or access sensitive data.
Mitigation:

1. **Use Prepared Statements with Parameter Binding:** This is the most effective defense against SQL injection. Prepared statements separate the SQL code from the data, preventing user input from being interpreted as SQL code.
2. **Input Validation:** Validate that the `$id` parameter is of the expected type (e.g., an integer). Reject any input that doesn't match the expected format.
3. **Escaping:** If prepared statements cannot be used, properly escape the user input using a function like `mysqli_real_escape_string()` before including it in the query. However, prepared statements are strongly preferred.

4. **Principle of Least Privilege:** Ensure that the database user account used by the application has only the minimum necessary privileges.

Code Quality Metrics

Code Quality Issues

- **Readability:** The code is relatively short and straightforward, but lacks comments explaining the purpose of each section. Comments should be added to improve maintainability.
- **Coupling:** The code is tightly coupled to the `conexion.php` file. While using an include for a database connection is common, using a more abstract database access layer could reduce coupling.
- **Error Handling:** The code lacks error handling. If the database connection fails or the query returns an error, the script will likely fail silently, potentially leading to unexpected behavior. Add error handling to gracefully handle failures and log errors for debugging.
- **Input Validation:** The code lacks input validation. As mentioned in the security vulnerabilities section, this is a critical issue. Beyond SQL injection, lack of validation can cause unexpected behavior.
- **File Inclusion:** Relative pathing to include files can be dangerous. It's best practice to use absolute paths for including files to avoid potential security issues, or define a root directory constant.

Proposed Solution

Improved Code (using prepared statements):

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

$id = $_GET["id"];

// Validate that $id is an integer
if (!is_numeric($id)) {
    http_response_code(400); // Bad Request
    echo json_encode(["error" => "Invalid ID"]);
```

```
    exit;
}

// Use a prepared statement to prevent SQL injection
$stmt = $conexion->prepare("SELECT * FROM productos WHERE id = ?");
$stmt->bind_param("i", $id); // "i" indicates that $id is an integer

// Execute the statement
$stmt->execute();

// Get the result
$result = $stmt->get_result();
$producto = $result->fetch_assoc();

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

// Handle cases where no product is found
if ($producto === null) {
    http_response_code(404); // Not Found
    echo json_encode(["error" => "Product not found"]);
    exit;
}

// Return the data in JSON format
header('Content-Type: application/json'); // Set the Content-Type header
echo json_encode($producto);
?>
```

Improvements:

- **SQL Injection Prevention:** Uses prepared statements with parameter binding to eliminate the risk of SQL injection.
- **Input Validation:** Validates that `$id` is an integer.
- **Error Handling:** Includes error handling for invalid input and cases where the product is not found, and uses HTTP status codes for better client-side error management.
- **Content Type:** Sets the `Content-Type` header to `application/json` to correctly communicate the response format to the client.
- **Code Clarity:** Added comments to explain each section of the code.
- **Resource Management:** Properly closes the prepared statement to release resources.

```
```