Vulnerability Report

Archivo: eliminar.php

Code Analyzed:

```
<?php
    if (!empty($_GET["id"])) {
        # code...
        $id = $_GET["id"];

        $sql = $conexion->query("DELETE FROM PERSONAS WHERE ID = $id");
        if ($sql == 1) {
            # code...
            echo '<div class="alert alert-warning" role="alert">
            Persona no eliminada correctamente.
          </div>';
        }else{
            echo '<div class="alert alert-danger" role="alert">
            Persona no eliminada correctamente.
          </div>';
        }
    }
?>
```

Analysis: ```html
Security Vulnerabilities

SQL Injection

Type: SQL Injection
Line: `$sql = $conexion->query("DELETE FROM PERSONAS WHERE ID = $id");`
Description: The code directly incorporates user-supplied input (`$_GET["id"]`) into an SQL query without proper sanitization or parameterization. This allows an attacker to inject malicious SQL code, potentially leading to data breaches, data manipulation, or unauthorized access.
Mitigation: Use prepared statements with parameterized queries. This allows the database to treat the user input as data, not as part of the SQL command.
Example: ```php $id = $_GET["id"]; $stmt = $conexion->prepare("DELETE FROM PERSONAS WHERE ID = ?"); $stmt->bind_param("i", $id); // 'i' indicates that $id is an integer $stmt->execute(); ```

Code Quality Metrics

Metrics Analysis

- **Readability:** The code is relatively readable due to its simplicity. However, the lack of proper error handling and input validation makes it harder to understand the complete execution flow and potential issues. Consider adding comments to explain the purpose of each section.
- **Coupling:** The code is tightly coupled to the `$conexion` object, making it difficult to reuse or test independently. Dependency Injection or a Data Access Object (DAO) pattern could be used to reduce coupling.
- **Error Handling:** The error handling is poor. The `if ($sql == 1)` condition is incorrect for checking the success of a `DELETE` query. The `query()` method of a mysqli object typically returns a `mysqli_result` object for `SELECT`, `SHOW`, `DESCRIBE` or `EXPLAIN` queries and `TRUE` or `FALSE` for other successful queries. It may also return `FALSE` if the query fails. A more robust error handling mechanism should be implemented to provide more informative feedback and prevent unexpected behavior. `mysqli_error($conexion)` should be used to log or display the actual error.
- **Input Validation:** There is no input validation. The code directly uses `$_GET["id"]` without checking if it is a valid integer or if it exists in the database. Validation should be implemented to prevent invalid or malicious data from being processed.

Proposed Solution

Secure and Improved Code

This solution addresses the SQL injection vulnerability, improves error handling, and adds input validation. It uses prepared statements, checks if the 'id' parameter is set and is an integer, and provides more specific error messages.

```
<?php
    if (isset($_GET["id"]) && is_numeric($_GET["id"])) {
        $id = (int)$_GET["id"]; // Cast to integer for added security and type safety

        // Use a prepared statement to prevent SQL injection
        $stmt = $conexion->prepare("DELETE FROM PERSONAS WHERE ID = ?");

        if ($stmt) {
            $stmt->bind_param("i", $id); // "i" specifies the parameter type as integer

            if ($stmt->execute()) {
                if ($stmt->affected_rows > 0) {
                    echo '<div class="alert alert-success" role="alert">Persona eliminada
correctamente.</div>';
                } else {
                    echo '<div class="alert alert-warning" role="alert">No se encontró la
persona con el ID especificado.</div>';
                }
            } else {
                echo '<div class="alert alert-danger" role="alert">Error al ejecutar la
consulta: ' . htmlspecialchars($stmt->error) . '</div>';
            }

            $stmt->close();
        } else {
            echo '<div class="alert alert-danger" role="alert">Error al preparar la consulta:
' . htmlspecialchars($conexion->error) . '</div>';
        }
    } else {
        echo '<div class="alert alert-warning" role="alert">ID inválido o no
especificado.</div>';
    }
?>
```

Key Improvements:

- SQL injection prevention with prepared statements and parameter binding.
- Input validation ensures that the 'id' parameter is set and is a number.
- Integer casting for type safety.
- More informative error messages, including database errors.

```
```