Passos para a Demonstração Simplificada

1. Configuração do Banco de Dados

Crie um banco de dados e uma tabela simples:

Execute os seguintes comandos SQL no phpMyAdmin ou na linha de comando do MySQL:

sql
Copiar código

```sql
CREATE DATABASE test_db;

USE test_db;

CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(100) NOT NULL
);

INSERT INTO users (username, email) VALUES
('user1', 'user1@example.com'),
('user2', 'user2@example.com'),
('user3', 'user3@example.com');
```

2. Criação do Arquivo PHP

Crie um arquivo PHP chamado users.php com o seguinte código. Este código é intencionalmente vulnerável para demonstrar a injeção SQL:

php
Copiar código

```php
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Usuários</title>
</head>
<body>
    <h1>Usuários</h1>

    <?php
    // Configuração de exibição de erros
    ini_set('display_errors', 1);
    ini_set('display_startup_errors', 1);
    error_reporting(E_ALL);

    // Conexão com o banco de dados
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "test_db";
```

```
    $conn = new mysqli($servername, $username, $password, $dbname);

    if ($conn->connect_error) {
        die("Conexão falhou: " . $conn->connect_error);
    }

    // Obter o parâmetro 'id' da URL
    $id = isset($_GET['id']) ? $_GET['id'] : '';

    // Construir a consulta SQL vulnerável
    $sql = "SELECT * FROM users WHERE id = $id";

    // Exibir a consulta SQL para depuração
    echo "<p>Consulta SQL: $sql</p>";

    // Executar a consulta
    $result = $conn->query($sql);

    if ($result === false) {
        echo "Erro na consulta: " . $conn->error;
    } elseif ($result->num_rows > 0) {
        // Exibir os usuários encontrados
        while ($row = $result->fetch_assoc()) {
            echo "<h2>Usuário: " . htmlspecialchars($row['username']) . "</h2>";
            echo "<p>Email: " . htmlspecialchars($row['email']) . "</p>";
        }
    } else {
        echo "Nenhum usuário encontrado.";
    }

    $conn->close();
    ?>
</body>
</html>
```

3. Testar a Injeção SQL via URL
Use os seguintes exemplos de URL para testar a injeção SQL:

Injeção Básica:

http://localhost/users.php?id=1%20OR%201=1

Explicação: O valor 1 OR 1=1 sempre é verdadeiro, então a consulta deve retornar todos os registros da tabela users.

Para resolver a vulnerabilidade de SQL Injection no código PHP fornecido, você deve utilizar práticas seguras para evitar que entradas maliciosas comprometam a integridade do banco de dados. A abordagem recomendada é usar prepared statements com parametrização. Isso ajuda a garantir que os parâmetros da consulta não sejam interpretados como código SQL.

Aqui está como você pode modificar o código PHP para utilizar prepared statements e eliminar a vulnerabilidade:

Código PHP Seguro
Substitua o código PHP vulnerável com o seguinte código, que utiliza prepared statements para prevenir SQL Injection:

php
Copiar código

```
<!DOCTYPE html>
<html lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Usuários</title>
</head>
<body>
    <h1>Usuários</h1>

    <?php
    // Configuração de exibição de erros
    ini_set('display_errors', 1);
    ini_set('display_startup_errors', 1);
    error_reporting(E_ALL);

    // Conexão com o banco de dados
    $servername = "localhost";
    $username = "root";
    $password = "";
    $dbname = "test_db";

    $conn = new mysqli($servername, $username, $password, $dbname);

    if ($conn->connect_error) {
        die("Conexão falhou: " . $conn->connect_error);
    }

    // Obter o parâmetro 'id' da URL
    $id = isset($_GET['id']) ? $_GET['id'] : '';

    // Verificar se o id é um número inteiro para evitar erros
    if (filter_var($id, FILTER_VALIDATE_INT) === false) {
        echo "ID inválido.";
        $conn->close();
        exit();
    }

    // Construir a consulta SQL segura usando prepared statements
    $stmt = $conn->prepare("SELECT * FROM users WHERE id = ?");
    $stmt->bind_param("i", $id);  // 'i' indica que o parâmetro é um inteiro
```

```
    // Executar a consulta
    $stmt->execute();

    // Obter o resultado
    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        // Exibir os usuários encontrados
        while ($row = $result->fetch_assoc()) {
            echo "<h2>Usuário: " . htmlspecialchars($row['username']) . "</h2>";
            echo "<p>Email: " . htmlspecialchars($row['email']) . "</p>";
        }
    } else {
        echo "Nenhum usuário encontrado.";
    }

    // Fechar a conexão
    $stmt->close();
    $conn->close();
    ?>
</body>
</html>
```

Explicação das Alterações

Uso de Prepared Statements:

Utilizamos $conn->prepare("SELECT * FROM users WHERE id = ?") para criar uma consulta SQL com um placeholder (?) onde o parâmetro será inserido.

Binding dos Parâmetros:

bind_param("i", $id) vincula o parâmetro $id à consulta preparada. O "i" indica que o parâmetro é um inteiro. Isso garante que o valor será tratado como um número inteiro e não como parte do código SQL.

Validação do Parâmetro:

Usamos filter_var($id, FILTER_VALIDATE_INT) para verificar se o parâmetro id é um inteiro válido. Isso ajuda a prevenir erros e garantir que o valor seja apropriado.

Uso de htmlspecialchars:

A função htmlspecialchars() é usada para escapar qualquer saída HTML e evitar Cross-Site Scripting (XSS).