Actividad 1: Explotación y Mitigación de SQL Injection (SQLi)

Tema: *Ataque y protección contra inyección SQL*

Objetivo: *Configurar un servidor web con Apache + PHP + MySQL, alojar una página vulnerable a SQL Injection, explotarla y luego corregirla*

¿Qué es SQL Injection (SQLi)?

SQL Injection (SQLi) es un tipo de ataque en el que un atacante inserta código SQL malicioso en una consulta a la base de datos, con el objetivo de manipular, robar o eliminar información sensible.

Este ataque ocurre cuando una aplicación no valida correctamente la entrada del usuario y ejecuta consultas SQL dinámicas sin medidas de seguridad.

Instalar Apache, PHP y MySQL en Ubuntu (Linux)

Actualizar paquetes e instalar Apache, PHP y MySQL

```
sudo apt update && sudo apt install apache2 php libapache2-mod-php mariadb-server mariadb-client -y
```

Verificar que Apache está funcionando:

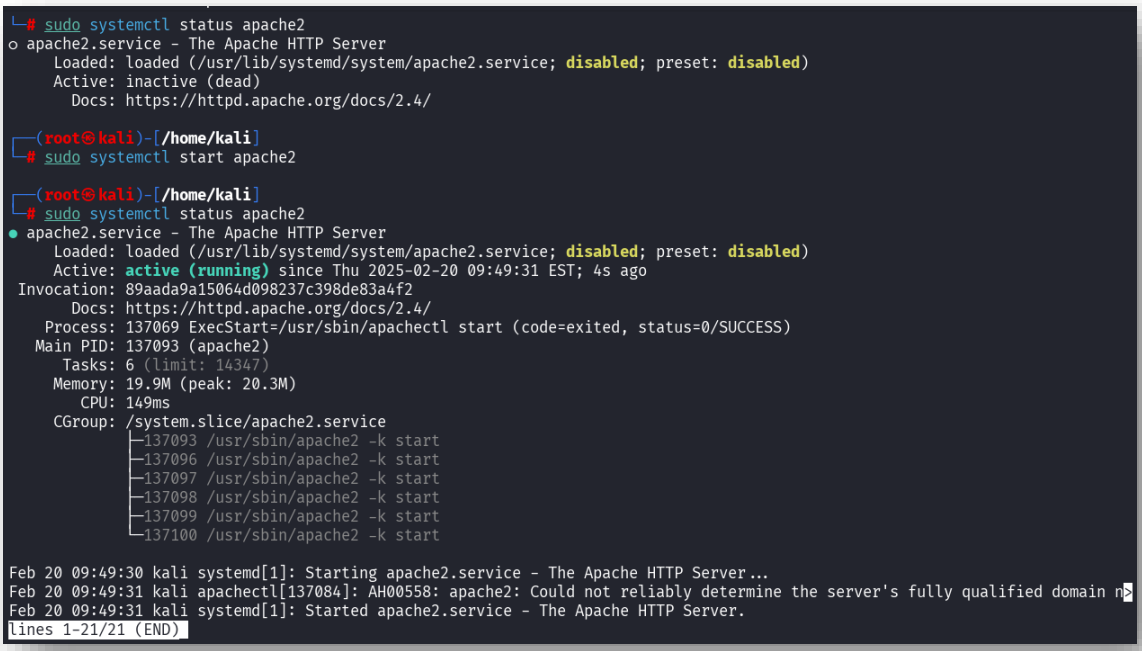
```
sudo systemctl status apache2
```

Si está detenido, iniciar con:

```
sudo systemctl start apache2
```

Configurar Apache para que inicie automáticamente:

```
sudo systemctl enable apache2
```



Iniciar MySQL y Configurar la Base de Datos

Ejecutar MySQL si está detenido:

```
sudo systemctl start mysql
```

Verificar que MySQL está activo:

```
sudo systemctl status mysql
```

Acceder a MySQL con el usuario root:

```
sudo mysql -u root -p
```

Si no se tiene contraseña, presiona Enter.

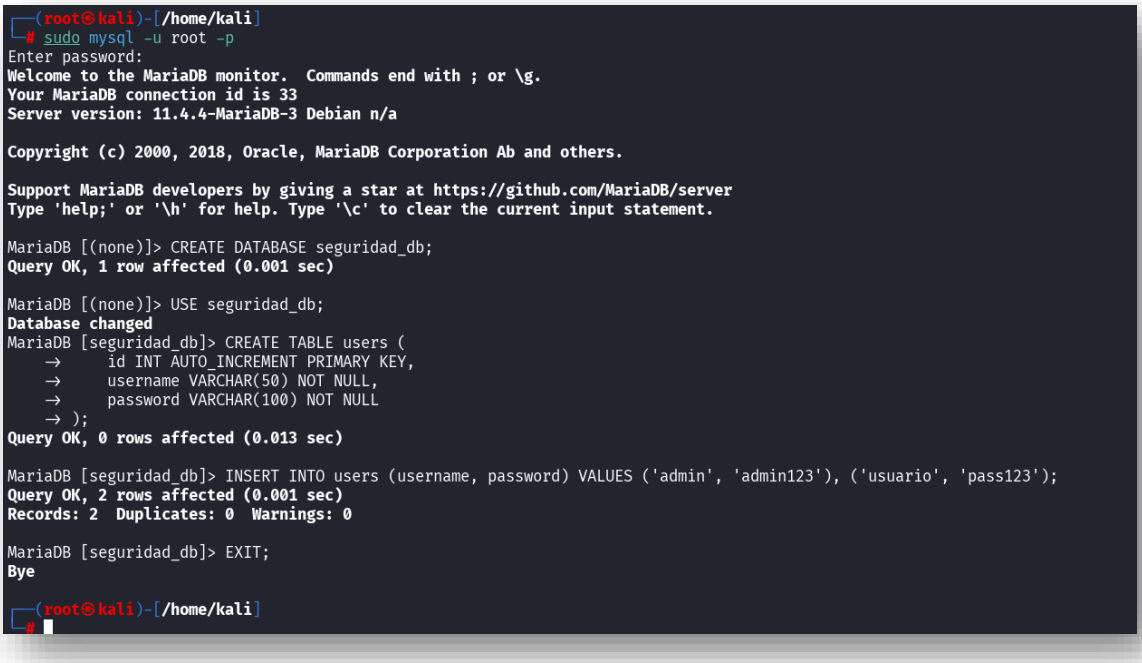
Crear la base de datos y la tabla de usuarios:

```
CREATE DATABASE seguridad_db;

USE seguridad_db;

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

```
INSERT INTO users (username, password) VALUES ('admin', 'admin123'), ('usuario', 'pass123');

SET PASSWORD FOR 'root'@'localhost' = PASSWORD('root');
```

Salir de MySQL:

```
EXIT;
```



Subir el Archivo PHP a Apache

Ubicación donde guarda el servidor Apache los archivos:

```
cd /var/www/html
```

Crear el archivo *login.php* en Apache:

```
sudo mousepad /var/www/html/login.php
```

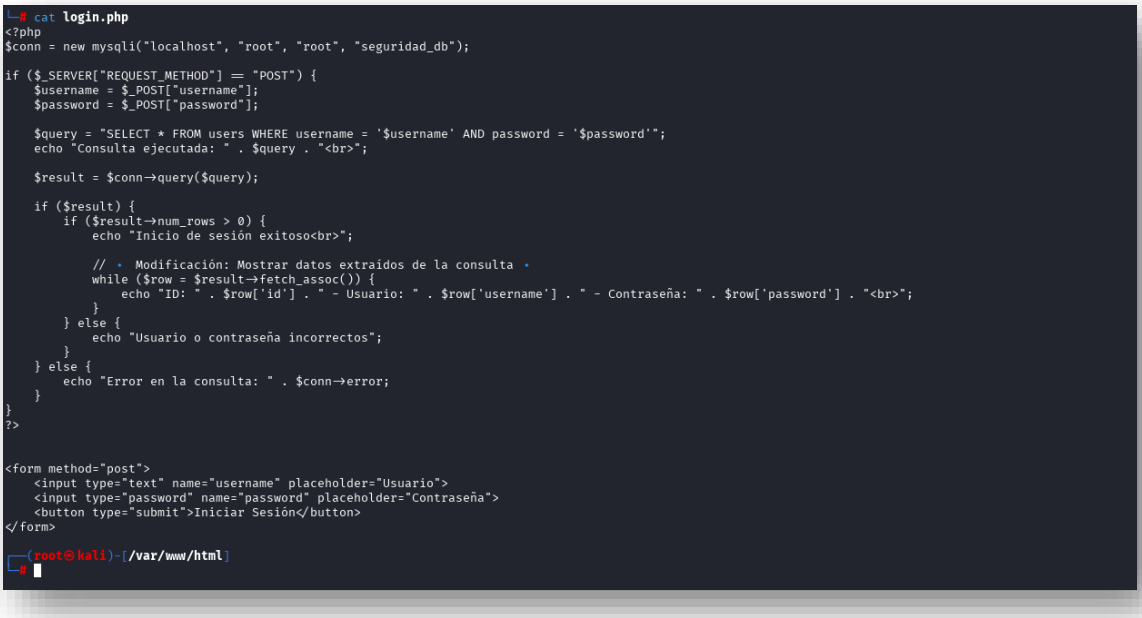
Copiar el siguiente **código vulnerable**:

```
<?php
$conn = new mysqli("localhost", "root", "root", "seguridad_db");

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST["username"];
    $password = $_POST["password"];

    $query = "SELECT * FROM users WHERE username = '$username' AND password = '$password'";
    echo "Consulta ejecutada: " . $query . "<br>";
```

```
    $result = $conn->query($query);

    if ($result) {
        if ($result->num_rows > 0) {
            echo "Inicio de sesión exitoso<br>";

            // Modificación: Mostrar datos extraídos de la consulta
            while ($row = $result->fetch_assoc()) {
                echo "ID: " . $row['id'] . " - Usuario: " . $row['username'] . " -
Contraseña: " . $row['password'] . "<br>";
            }
        } else {
            echo "Usuario o contraseña incorrectos";
        }
    } else {
        echo "Error en la consulta: " . $conn->error;
    }
}
?>

<form method="post">
    <input type="text" name="username" placeholder="Usuario">
    <input type="password" name="password" placeholder="Contraseña">
    <button type="submit">Iniciar Sesión</button>
</form>
```

Guardar el archivo y establecer permisos adecuados:

```
sudo chmod 755 /var/www/html/login.php

cat /var/www/html/login.php
```



Iniciar el Servidor Web y Acceder a la Página

Reiniciar Apache para aplicar los cambios:

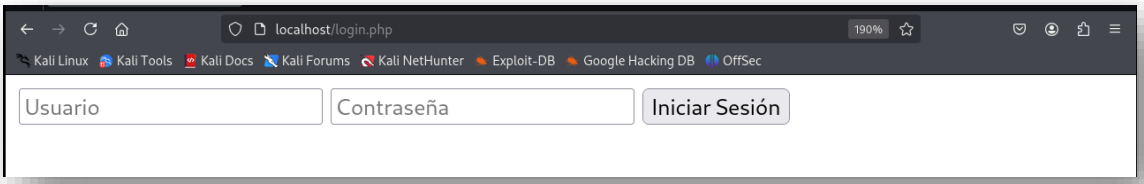
```
sudo systemctl restart apache2
```

Verificar que MySQL está ejecutándose antes de acceder a la página:

```
sudo systemctl status mysql
```

Abrir el navegador y acceder a la aplicación:

```
http://localhost/login.php
```



El código implementa un sistema de autenticación en PHP, conectándose a una base de datos MySQL para verificar credenciales enviadas a través de un formulario. Sin embargo, presenta **vulnerabilidades críticas**.

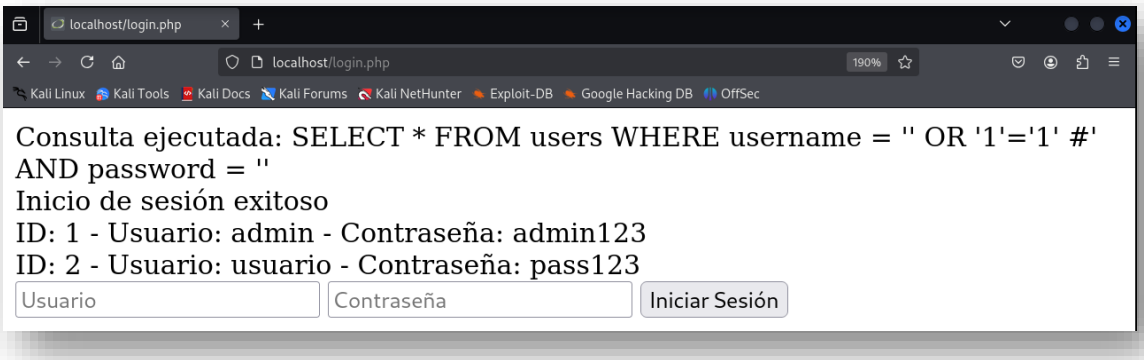
Exploitar SQL Injection

Bypass de autenticación

Para realizar la explotación, en el campo "*Usuario*" ingresar:

```
' OR '1'='1' #
```

Resultado esperado: Inicia sesión sin credenciales válidas.

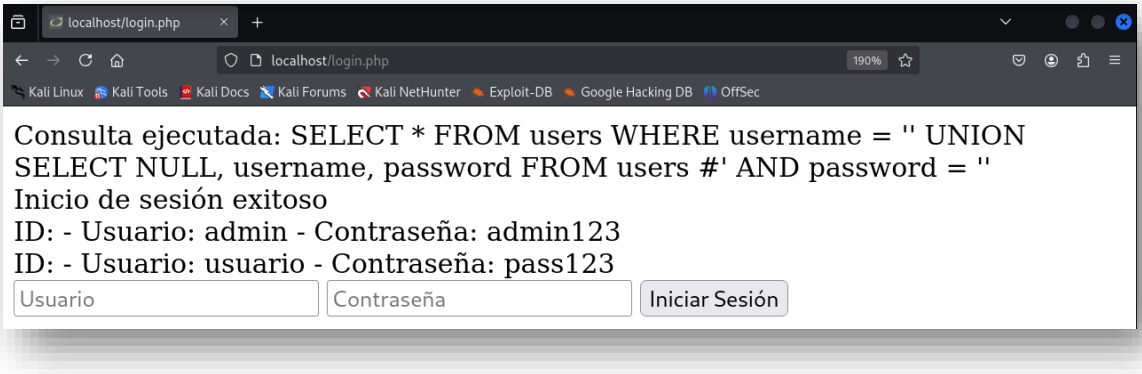


Obtener credenciales de la base de datos

Para realizar la explotación, en el campo "*Usuario*" ingresar:

```
' UNION SELECT NULL, username, password FROM users #
```

Resultado esperado: Se muestran todos los usuarios y contraseñas.



Mitigar SQL Injection

Abrir **login.php** y modificarlo para usar *consultas preparadas*:

```
sudo mousepad /var/www/html/login.php
```

Reemplazar el código por esta **versión segura**:

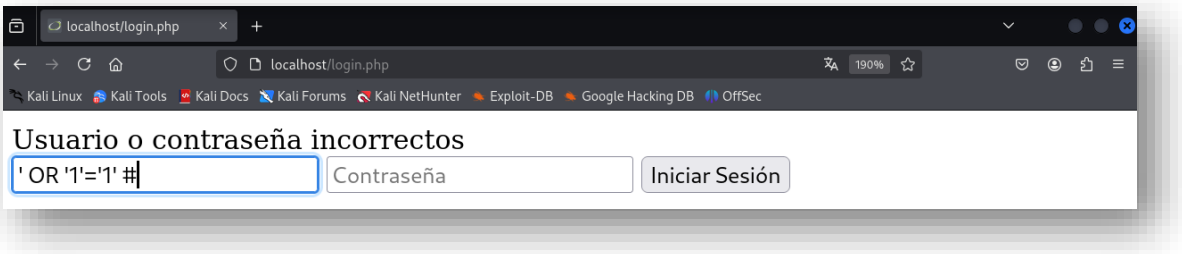
```
<?php

$conn = new mysqli("localhost", "root", "root", "seguridad_db");

if ($conn->connect_error) {
    die("Error de conexión: " . $conn->connect_error);
}

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = trim($_POST["username"]);
    $password = trim($_POST["password"]);

    $stmt = $conn->prepare("SELECT id, username, password FROM users WHERE username = ? AND
password = ?");
    $stmt->bind_param("ss", $username, $password);
    $stmt->execute();

    $result = $stmt->get_result();

    if ($result->num_rows > 0) {
        echo "Inicio de sesión exitoso<br>";
```

```
        // Mostrar datos extraídos de la consulta
        while ($row = $result->fetch_assoc()) {
            echo "ID: " . $row['id'] . " - Usuario: " . $row['username'] . " - Contraseña: "
. $row['password'] . "<br>";
        }
    } else {
        echo "Usuario o contraseña incorrectos";
    }

    $stmt->close();
}

$conn->close();
?>

<form method="post">
    <input type="text" name="username" placeholder="Usuario">
    <input type="password" name="password" placeholder="Contraseña">
    <button type="submit">Iniciar Sesión</button>
</form>
```

Reiniciar Apache para aplicar los cambios:

```
sudo systemctl restart apache2
```

Volver a probar **SQL Injection** y verificar que ya **no** funciona.



Problemas del primer código (Inseguro)

1. **Vulnerabilidad a inyección SQL**
   - La consulta SQL inserta directamente los valores del usuario (*'$username' AND password = '$password'*).
   - No se usan consultas preparadas**.**
2. **Contraseñas almacenadas en texto plano**
   - La base de datos parece almacenar las contraseñas en texto sin cifrar.
   - Esto es una **mala práctica**, ya que si la base de datos es comprometida, todas las contraseñas quedan expuestas.
3. **Falta de validación y sanitización de entrada**
   - No hay ningún tipo de limpieza en los valores obtenidos de *$_POST*, lo que facilita ataques como XSS o inyecciones maliciosas.
4. **No se maneja la conexión a la base de datos adecuadamente**
   - No se verifica si la conexión es exitosa.
   - No se cierra la conexión después de ejecutar la consulta.

5. **Exposición de información sensible**
   - Se imprime la consulta SQL ejecutada (*echo "Consulta ejecutada: " . $query . "
";*).
   - Se muestran las credenciales de los usuarios en la pantalla, lo que puede ser aprovechado por un atacante.

Mejoras en el segundo código (Más seguro, pero aún con problemas)

1. **Uso de consultas preparadas**
   - Se usa *$stmt->prepare()* y *bind_param()*, lo que previene **inyección SQL**.
   - **Ventaja:** No importa qué ingrese el usuario, la consulta tratará los valores como datos, no como código ejecutable.
2. **Se valida la conexión a la base de datos**
   - Se verifica si *connect_error* devuelve un error antes de continuar.
   - Si hay un fallo, el script termina con *die(),* lo que evita que se ejecuten consultas en una conexión fallida.
3. **Se limpia la entrada del usuario con trim()**
   - Se eliminan espacios en blanco al inicio y al final del username y password, evitando problemas con caracteres innecesarios.
4. **Manejo de la conexión a la base de datos**
   - Se cierra la consulta (*$stmt->close())* y la conexión (*$conn->close())* correctamente.

Problemas que aún tiene el segundo código

1. **Las contraseñas siguen almacenándose en texto plano**
   - Aunque se evita la inyección SQL, el código sigue comparando contraseñas directamente en la base de datos.
   - **Solución correcta:** Almacenar las contraseñas con *password_hash()* y verificar con *password_verify().*
2. **Mensajes de error genéricos**
   - Se sigue mostrando información detallada sobre los usuarios si la consulta es exitosa.
   - Lo correcto sería iniciar una sesión en lugar de mostrar información del usuario.
3. **No hay control de sesiones**
   - A pesar de corregir varios problemas de seguridad, no se establece una sesión segura (session_start()) después de una autenticación exitosa.

Medidas de Seguridad Adicionales

- No concatenar datos en consultas SQL, siempre **usar consultas preparadas.**
- **Validar y sanitizar** entradas de usuario:

```
$username = htmlspecialchars($_POST["username"], ENT_QUOTES, 'UTF-8');
```

La función **htmlspecialchars()** convierte ciertos caracteres especiales en sus entidades HTML correspondientes, lo que evita la ejecución de código HTML o JavaScript malicioso.

ENT_QUOTES → Escapa comillas dobles (") y comillas simples (').
UTF-8 → Define la codificación de caracteres para evitar problemas con caracteres especiales.

- **Configurar permisos mínimos** en la Base de Datos.
- **Implementar firewall** y bloquear IPs sospechosas.