

Descubriendo BurpSuite: Análisis y Mitigación de la Inyección SQL

Hamza Akdi

Servicios de red e internet

22/11/2025

Índice:

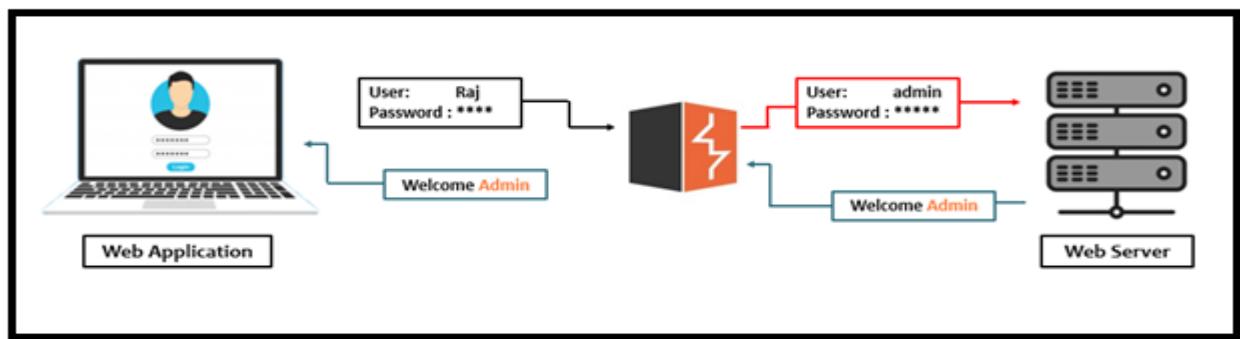
Índice:.....	2
Introducción.....	3
1. Despliegue de BurpSuite	4
1.1 Instalación de BurpSuite	4
1.2 Configuración del navegador con proxy	4
1.3 Captura de peticiones HTTP/HTTPS	5
1.4 Descarga de certificados.....	6
2. Entorno de pruebas	7
2.1 Creación y despliegue del entorno	7
2.2 Creación de los archivos del laboratorio.....	7
2.2.1 Creación del archivo setup.sql	8
2.2.2 Creación de archivo docker-compose.yml	8
2.2.4 Despliegue del entorno de pruebas.....	11
3. Ataque de inyección SQL	11
3.1 Captura de petición inicial	12
3.1.1 Captura con usuario y contraseña errónea	13
3.1.2 Captura con usuario y contraseña correcta	14
4. Ejecución de inyección SQL en BurpSuite/Repeater	14
4.1 Método desde el formulario (Payload)	15
4.2 Inyección Lógica (Igualdad de Cadenas)	16
5. Solución a la inyección SQL en el código PHP	17
5.1 Implementación segura	17
Comentarios	19
Problemas y soluciones	20
Conclusiones	20

Introducción

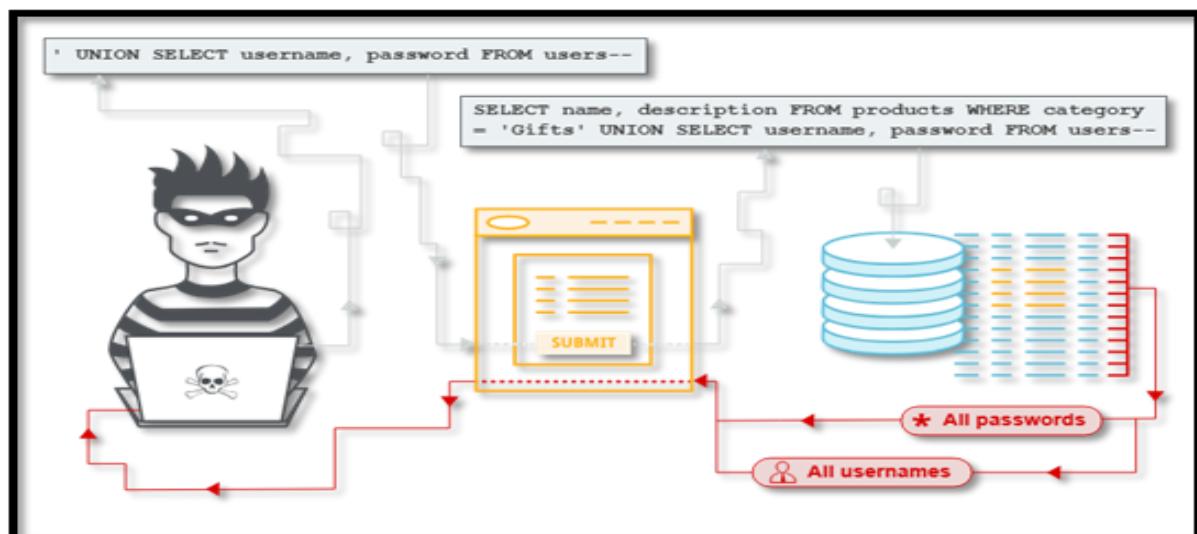
En esta práctica vamos a trabajar los siguientes aspectos:

- Instalación y configuración del servicio **BurpSuite** (edición Community).
- *Configuración del navegador con proxy.*
- Captura de peticiones **HTTP** mediante el interceptor de **BurpSuite**.
- Identificación de parámetros vulnerables a inyección **SQL**.
- Prueba de **payloads** básicos y comprensión de respuestas del servidor.

Dos ilustraciones interesantes para tener en cuenta son las siguientes:



(Vista de esquema de Burpsuite)



(Vista inyección SQL)

1. Despliegue de BurpSuite

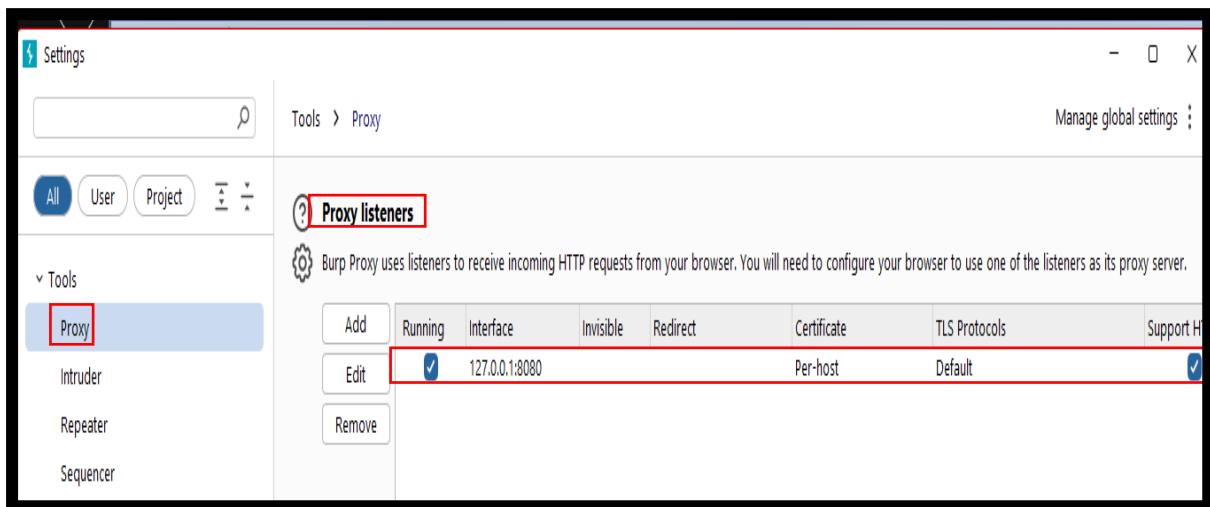
La edición **Community de BurpSuite** es una suite de herramientas de seguridad web esencial, desarrollada por **PortSwigger**.

1.1 Instalación de BurpSuite

Iremos al sitio web oficial de PortSwigger (<https://portswigger.net/burp>.), y descargaremos la versión Community.

1.2 Configuración del navegador con proxy

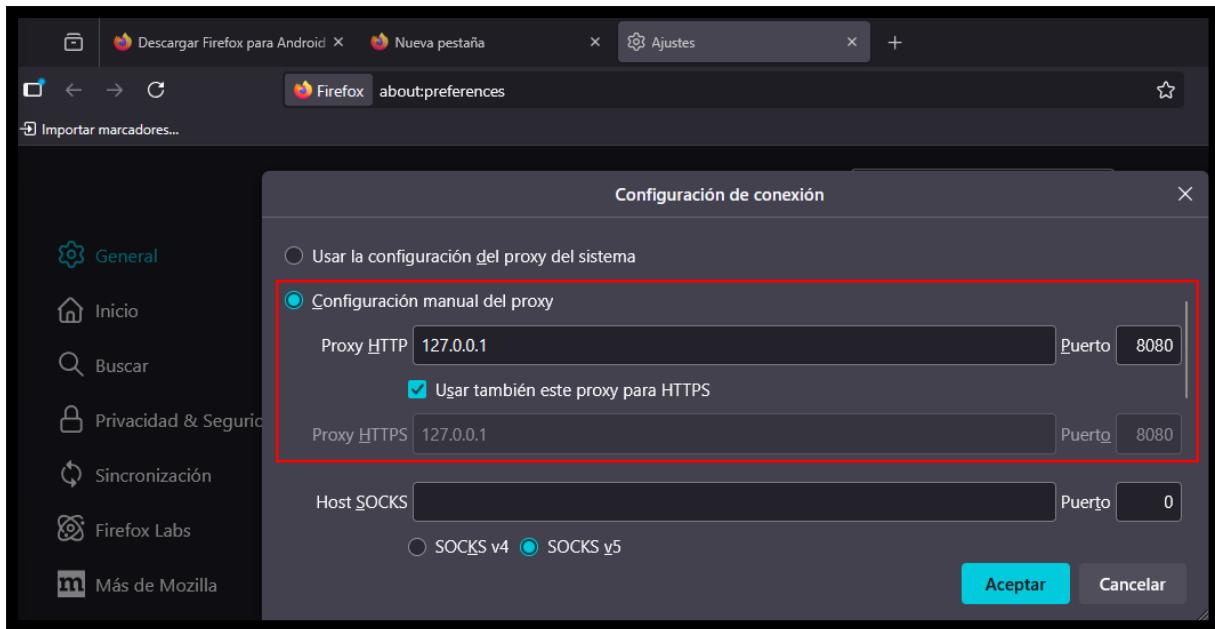
Para que BurpSuite pueda capturar e interceptar el tráfico entre nuestro navegador y el servidor web, debe actuar como un proxy intermediario. BurpSuite, por defecto utiliza la dirección 127.0.0.1 (localhost) en el puerto 8080 como se indica a continuación.



(Configuración de proxy en Burpsuite)

Necesitamos decirle al navegador que envíe todo su tráfico HTTP y HTTPS a esta dirección. Utilizaremos el navegador MozillaFirefox, ya que podemos configurar el proxy independientemente del sistema operativo.

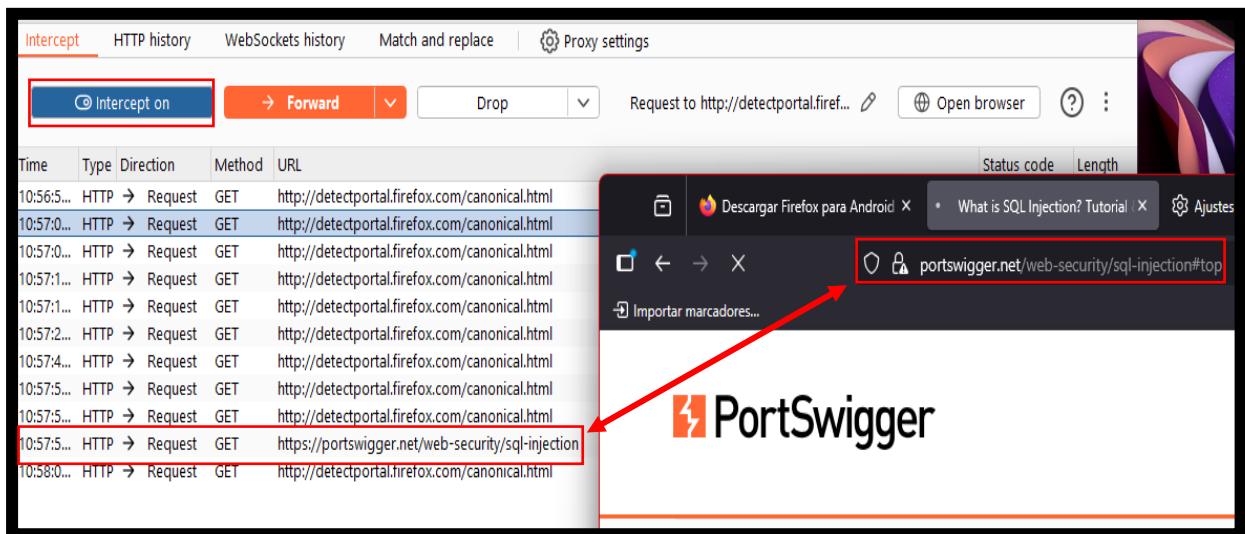
Para ello, iremos a la configuración de red del navegador, donde haremos la siguiente configuración:



(Configuración proxy en navegador)

1.3 Captura de peticiones HTTP/HTTPS

Una vez configurado el proxy, el tráfico de nuestro navegador pasará por BurpSuite. Para hacer esta captura de peticiones HTTP/HTTPS, activaremos la interceptación (intercept on) en la pestaña “proxy” y haremos una prueba de captura, navegando a la url <https://portswigger.net/web-security/sql-injection>

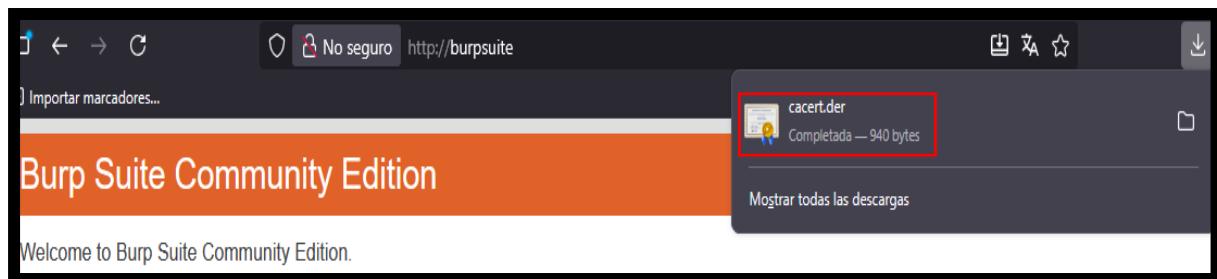


(Captura de peticiones HTTP/HTTPS en url de prueba)

1.4 Descarga de certificados

Cuando accedemos a una web **HTTPS**, el navegador ve que el certificado de BurpSuite no es de confianza. Para capturar tráfico **HTTPS**, necesitamos instalar el **certificado**

CA de BurpSuite en nuestro navegador. Esto le dice a tu navegador que confíe en BurpSuite como intermediario seguro.



(Descarga de certificado de confianza)

2. Entorno de pruebas

Crearemos un **entorno de laboratorio estándar y seguro** para **inyección SQL** usando **Docker y Docker Compose**. Para poder ejecutar la arquitectura Docker-compose, necesitas tener instalado **Docker** y **Docker Compose** en nuestro sistema.

2.1 Creación y despliegue del entorno

Primero, instalaremos **Docker Desktop** en nuestro equipo desde la siguiente url:
<https://www.docker.com/products/docker-desktop/>



(Instalación de Docker Desktop)

2.2 Creación de los archivos del laboratorio

Vamos a crear un laboratorio básico de inyección SQL (una aplicación PHP vulnerable y una base de datos MySQL).

Comenzaremos creando dos archivos esenciales:

- Archivo **setup.sql**: Este archivo es esencial, es un **script de inicialización SQL** que se ejecuta automáticamente cuando el contenedor de MySQL se inicia por primera vez. Inicializa la base de datos con una tabla y algunos datos de prueba que atacaremos.
- Archivo **docker-compose.yml**: Este archivo define los servicios (contenedores) que formarán nuestra arquitectura de pruebas.

2.2.1 Creación del archivo setup.sql

Este archivo inicializa la base de datos con los usuarios y claves de prueba, con el contenido de la siguiente imagen.

Contenido del archivo **setup.sql**:

```
CREATE DATABASE IF NOT EXISTS sql_injection_lab;
USE sql_injection_lab;

CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50),
    password VARCHAR(50)
);

INSERT INTO users (username, password) VALUES
('admin', 'admin123'),
('isaac', 'amapolas'),
('amapola', 'david21');
```

(Contenido del archivo setup.sql)

2.2.2 Creación de archivo docker-compose.yml

A continuación, crearemos el archivo **docker-compose.yml**. Este archivo define contenedores de nuestra arquitectura de dos servicios (web y base de datos).

Hemos creado este archivo (**Dockerfile**) con la siguiente configuración, **siendo crucial en nuestro código PHP** para que la aplicación **sea compatible** con versiones modernas de PHP y para que la vulnerabilidad de inyección SQL persista.

Nombre	Fecha de modificación	Tipo	Tamaño
Dockerfile	25/11/2025 20:31	Archivo	1 KB
docker-compose.yml	25/11/2025 16:23	Archivo de origen ...	1 KB
setup.sql	25/11/2025 16:23	Archivo de origen ...	1 KB
web	25/11/2025 16:23	Carpeta de archivos	

(Creación del archivo Dockerfile)

Configuración del archivo Dockerfile:

```
FROM php:8.1-apache

RUN docker-php-ext-install mysqli && docker-php-ext-enable mysqli
```

Se identificó que el **código PHP** de la aplicación vulnerable utilizaba la **API obsoleta `mysql_connect()`**, la cual fue eliminada en PHP 7.0 (el contenedor utiliza **PHP 8.1**). Para solucionarlo, se migraron las funciones de conexión y consulta a la extensión **mysqli (`mysqli_connect` y `mysqli_query`)**. Esta modificación soluciona el error de compatibilidad y permite que la aplicación muestre el formulario, mientras que la vulnerabilidad de Inyección SQL se mantiene intacta al seguir utilizando la concatenación directa de variables en la consulta.

Hemos actualizado el archivo **docker-compose.yml** para usar un **Dockerfile** para construir la imagen de la aplicación web, lo cual **es la solución correcta** para asegurar la compatibilidad con **MySQLi**.

```

services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: vulnerable_php
    ports:
      - "8081:80"
    volumes:
      - ./web:/var/www/html
    depends_on:
      - db
    environment:
      - MYSQL_HOST=db
      - MYSQL_USER=root
      - MYSQL_PASSWORD=root
      - MYSQL_DATABASE=mysql_injection_lab

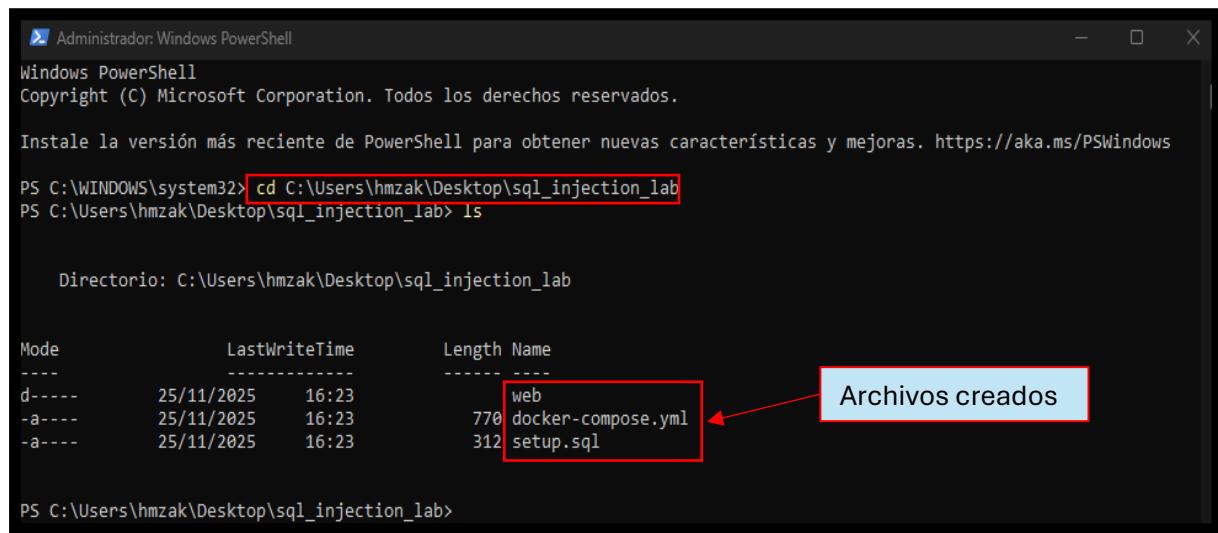
  db:
    image: mysql:5.7
    container_name: vulnerable_db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: mysql_injection_lab
    volumes:
      - ./setup.sql:/docker-entrypoint-initdb.d/setup.sql

  phpmyadmin:
    image: phpmyadmin/phpmyadmin
    container_name: vulnerable_phpmyadmin
    restart: always
    ports:
      - "8082:80"
    environment:
      PMA_HOST: db
      PMA_USER: root
      PMA_PASSWORD: root

```

(Contenido de configuración archivo docker-compose.yml)

Verificaremos la correcta creación de estos ficheros:



```

Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\WINDOWS\system32> cd C:\Users\hmzak\Desktop\sql_injection_lab
PS C:\Users\hmzak\Desktop\sql_injection_lab> ls

Directorio: C:\Users\hmzak\Desktop\sql_injection_lab

Mode                LastWriteTime         Length Name
----                -              -          -
d----       25/11/2025     16:23            770  web
-a---       25/11/2025     16:23           312  docker-compose.yml
-a---       25/11/2025     16:23            312  setup.sql

```

(Verificación creación de archivos en PowerShell)

2.2.4 Despliegue del entorno de pruebas

Ahora que los archivos están listos, desplegaremos la arquitectura. Primero, desplegaremos los contenedores con el siguiente comando:

The screenshot shows a terminal window with the following text:

```
PS C:\Users\hmzak\Desktop\sql_injection_lab> docker compose up -d
[+] Running 4/4
  ✓ web Pulled
  ✓ phpmyadmin Pulled
  ✓ db Pulled
  ✓ Container vulnerable_phpmyadmin Started
  ✓ Container vulnerable_db Started
  ✓ Container vulnerable_php Started
```

A red box highlights the command `docker compose up -d`. A red arrow points from this box to a callout box containing the text "Comando para desplegar contenedores en Dockers". Another red arrow points from the bottom of the callout box to the start of the command output.

(Despliegue del laboratorio en Dockers)

3. Ataque de inyección SQL

SEGÚN WIKIPEDIA:

INYECCIÓN SQL ES UN MÉTODO DE INFILTRACIÓN DE CÓDIGO INTRUSO QUE SE VALE DE UNA VULNERABILIDAD INFORMÁTICA PRESENTE EN UNA APLICACIÓN EN EL NIVEL DE VALIDACIÓN DE LAS ENTRADAS PARA REALIZAR OPERACIONES SOBRE UNA BASE DE DATOS.

EL ORIGEN DE LA VULNERABILIDAD RADICA EN LA INCORRECTA COMPROBACIÓN O FILTRADO DE LAS VARIABLES UTILIZADAS EN UN PROGRAMA QUE CONTIENE, O BIEN GENERA, CÓDIGO SQL. ES, DE HECHO, UN ERROR DE UNA CLASE MÁS GENERAL DE VULNERABILIDADES QUE PUEDE OCURRIR EN CUALQUIER LENGUAJE DE PROGRAMACIÓN O SCRIPT QUE ESTÉ INCRUSTADO EN OTRO.

SE CONOCE COMO INYECCIÓN SQL, INDISTINTAMENTE, AL TIPO DE VULNERABILIDAD, AL MÉTODO DE INFILTRACIÓN, AL HECHO DE INCRUSTAR CÓDIGO SQL INTRUSO Y A LA PORCIÓN DE CÓDIGO INCRUSTADO.

3.1 Captura de petición inicial

Interceptaremos el envío de un formulario de *login* para obtener la estructura de la **petición POST** con los parámetros *username* y *password*.

Comenzaremos asegurándonos que el botón “intercept on” este activo, dando a ver que Burpsuite está listo para detener el tráfico del navegador.

Una vez realizado este paso, iremos al navegador e introduciremos la url “localhost:8081” para acceder al formulario *login*.

Podemos verificar en la siguiente imagen que se hace la detección correctamente.

The screenshot shows the Burp Suite interface. At the top, there's a browser-like header with tabs for "Práctica SQL Injection" and "Nueva pestaña". The address bar shows "http://localhost:8081". Below the browser view, there's a "Laboratorio SQL Injection" page with fields for "Usuario:" and "Contraseña:", and a "Entrar" button. The Burp Suite navigation bar at the top has "Burp", "Project", "Intruder", "Repeater", "View", and "Help". The "Proxy" tab is selected and highlighted in red. The main workspace shows a list of network requests. A specific request from "http://localhost:8081" is highlighted with a red box and labeled "Request to http://localhost:8081 [127.0.0.1:8081]". The "Intercept" button in the toolbar is also highlighted in red, indicating it is active.

Time	Type	Direction	Method	URL
7:47:01 25 ...	HTTP	→ Request	GET	http://detectportal.firefox.comcanonical.html
7:47:06 25 ...	HTTP	→ Request	GET	http://detectportal.firefox.comcanonical.html
7:47:11 25 ...	HTTP	→ Request	GET	http://detectportal.firefox.comcanonical.html
7:47:16 25 ...	HTTP	→ Request	GET	http://detectportal.firefox.comcanonical.html
7:47:21 25 ...	HTTP	→ Request	GET	http://detectportal.firefox.comcanonical.html
7:47:26 25 ...	HTTP	→ Request	GET	http://detectportal.firefox.comcanonical.html
7:47:48 25 ...	HTTP	→ Request	GET	http://localhost:8081/

(Verificación de acceso al formulario y detección en Burpsuite)

3.1.1 Captura con usuario y contraseña errónea

En el formulario de *login* de la aplicación, introduciremos credenciales **erróneas** como en el ejemplo de la siguiente imagen. Donde podemos ver los datos que se han dado para el *login*.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. The 'Intercept' button is active. A list of network requests is displayed, with the last one highlighted. The URL for this request is `http://192.168.56.1:8081/login.php?username=hamza&password=akdi`. Below the proxy interface is a Firefox browser window titled 'Práctica SQL Injection'. The address bar shows the URL `http://192.168.56.1:8081`. The page content is a login form with fields for 'Usuario:' containing 'hamza' and 'Contraseña:' containing 'akdi'. A red box highlights the URL in the browser's address bar.

(Intento de acceso con login erróneo)

Mensaje de *login* erróneo:

The screenshot shows a browser window with the URL `192.168.56.1:8081/login.php?username=hamza&password=akdi`. The page displays an error message: 'Consulta ejecutada: SELECT * FROM users WHERE username='SDSD' AND password='admiDSDS''. Below this, another message reads 'Usuario o contraseña incorrectos' (Incorrect user or password).

3.1.2 Captura con usuario y contraseña correcta

Esta vez accederemos con credenciales correctas como ejemplo, dándonos el siguiente mensaje de bienvenida.

The screenshot shows the Burp Suite interface with a list of captured HTTP requests and a browser window below it. The requests list shows multiple GET requests to 'http://detectportal.firefox.comcanonical.html' at various times between 20:38:30 and 20:43:26. The browser window shows a login page with the URL 'http://192.168.56.1:8081/login.php?username=admin&password=admin123'. The status bar indicates 'No seguro'. Below the URL, a message box displays the SQL query 'Consulta ejecutada: SELECT * FROM users WHERE username='admin' AND password='admin123''. The main content area of the browser shows the welcome message 'Bienvenido, admin'.

(Intento de acceso con credenciales correctas)

4. Ejecución de inyección SQL en BurpSuite/Repeater

La inyección que vamos a realizar es un "Bypass de Autenticación" (Authentication Bypass), aprovechando la forma en que el código concatena los datos de `username` y `password` en una consulta SQL.

Este es el método más común y eficiente, ya que anula la verificación de la contraseña.

El código vulnerable en el servidor toma la entrada del usuario y la inserta directamente en la consulta. El atacante utiliza un *payload* diseñado para cambiar la lógica del `WHERE` de la consulta, asumiendo el siguiente formato vulnerable:

SELECT * FROM users WHERE username = ' [INPUT] ' AND password = '[INPUT]'

Utilizamos BurpSuite Repeater para automatizar el envío del payload y analizar la respuesta del servidor de pruebas (<http://192.168.56.1:8081>). La petición **POST** de `login` capturada se carga en la pestaña **Repeater**.

Se edita el cuerpo de la petición (Request) para injectar el código malicioso en el parámetro username (**username= ' OR 1=1 --**).

Por último, se envía la solicitud al servidor haciendo clic en "**Send**" y el servidor procesa la solicitud con éxito (código **200 OK**), indicando que el *login* ha sido concedido.

The screenshot shows the Burpsuite interface with the Repeater tab selected. In the Request pane, a GET request is shown with the URL `/login.php?username=%27%OR+1=1--%27&password=`. In the Response pane, the server returns a `HTTP/1.1 200 OK` response with the following headers and body:

```
1 HTTP/1.1 200 OK
2 Date: Wed, 26 Nov 2025 15:24:24 GMT
3 Server: Apache/2.4.65 (Debian)
4 X-Powered-By: PHP/8.1.33
5 Vary: Accept-Encoding
6 Content-Length: 165
7 Keep-Alive: timeout=5, max=100
8 Connection: Keep-Alive
9 Content-Type: text/html; charset=UTF-8
10
11 <b>
      Consulta ejecutada:
</b>
SELECT * FROM users WHERE username=' OR 1=1 --
' AND password='
<br>
<br>
<h3>
      Bienvenido, ' OR 1=1 -- '
```

(Login correcto por inyección SQL en Burpsuite/Repeater)

4.1 Método desde el formulario (Payload)

La siguiente imagen muestra el formulario de la aplicación web y el *payload* introducido por el atacante. (' **OR 1=1 --**') este es el **payload de inyección SQL** completo. Se introduce directamente en el formulario: **La comilla simple (')** abre la consulta del código vulnerable. **OR 1=1** añade una condición que siempre es **TRUE**. (--) seguido de un espacio, que comenta y anula el resto de la consulta incluyendo la verificación de la contraseña. La contraseña no se requiere, ya que el *payload* anula esa parte de la consulta.

Este es uno de los métodos de inyección sin necesidad de acceder a Burpsuite.

Laboratorio SQL Injection

Usuario:
'' OR 1=1 -- '

Contraseña:

Entrar

(Payload desde el formulario)

La siguiente imagen muestra la respuesta del servidor después de procesar el payload, verificando que la inyección funcionó.

localhost8081/login.php?username=X +

localhost8081/login.php?username=' OR 1%3D1+-- +'&password=

Consulta ejecutada: SELECT * FROM users WHERE username='' OR 1=1 -- '' AND password=''

Bienvenido, '' OR 1=1 -- ''

(Verificación de acceso con inyección SQL desde el formulario)

4.2 Inyección Lógica (Igualdad de Cadenas)

Este método logra el mismo resultado (**Bypass de Autenticación**) sin utilizar el comentario (--), sino forzando una condición **siempre verdadera** tanto en el campo de usuario como en el de contraseña.

Se utiliza el mismo *payload* en la contraseña para asegurar que el servidor no anula el campo contraseña, esta condición también sea verdadera, garantizando el éxito de toda la cláusula **WHERE**.

The screenshot shows a web browser window with the URL `http://localhost:8081`. The page title is "Laboratorio SQL Injection". There are two input fields: "Usuario" containing "' OR 'a'='a" and "Contraseña" also containing "' OR 'a'='a". Below the inputs is a "Entrar" button.

(Inyección lógica SQL-Igualdad de cadenas)

En la siguiente imagen podemos ver y verificar la inyección SQL.

The screenshot shows a web browser window with the URL `http://localhost:8081/login.php?username=' OR 'a'%3D'a&password=' OR 'a'%3D'a`. The page displays the query executed: "Consulta ejecutada: SELECT * FROM users WHERE username=' OR 'a'='a AND password=' OR 'a'='a'" and the welcome message "Bienvenido, ' OR 'a'='a".

(Verificación de acceso con inyección lógica SQL desde el formulario)

5. Solución a la inyección SQL en el código PHP

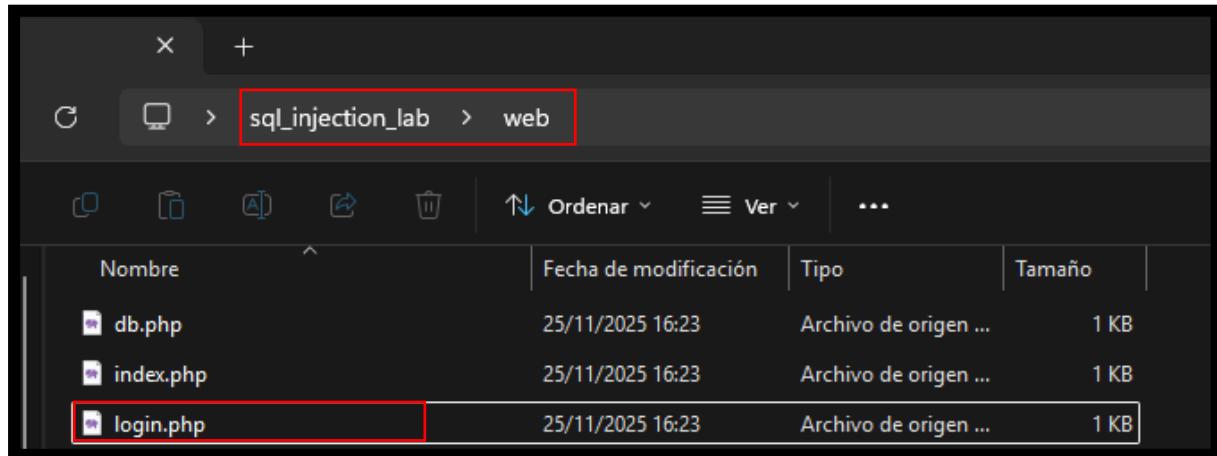
La única solución robusta y correcta para prevenir esta clase de ataque es el uso de **sentencias preparadas (Prepared Statements)**.

- **Separación estricta:** Las sentencias preparadas separan el **código SQL** de los **datos del usuario**. Esto se logra utilizando **marcadores de posición (?)** en la consulta.
- **Tratamiento de datos:** La base de datos es instruida para tratar la entrada del usuario (' OR 1=1 --) **como un valor de string**, sin interpretarlo nunca como una instrucción de comando SQL.

5.1 Implementación segura

Debemos reemplazar el código inseguro de la consulta SQL por la versión de sentencias preparadas.

Para ello, debemos primero detener los contenedores y localizar el archivo “login.php” donde está la estructura a modificar o remplazar por el código de sentencias separadas.



(Ruta del archivo login.php a modificar)

Una vez en el archivo, remplazaremos el código vulnerable antiguo, por el código de sentencias separadas que vemos en la siguiente imagen.

Código vulnerable:

```
<?php
include("db.php");

$username = $_GET['username'] ?? '';
$password = $_GET['password'] ?? '';

$sql = "SELECT * FROM users WHERE username='$username' AND password='$password'";

echo "<b>Consulta ejecutada:</b> " . htmlspecialchars($sql) . "<br><br>";

$result = mysqli_query($conn, $sql);

if ($result && mysqli_num_rows($result) > 0) {
    echo "<h3>Bienvenido, $username</h3>";
} else {
    echo "<h3>Usuario o contraseña incorrectos</h3>";
}
?>
```

Este código corrige la vulnerabilidad atacada:

```
// CÓDIGO SEGURO FINAL
$user_input = $_POST['username'];
$pass_input = $_POST['password'];

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

$result = $stmt->get_result();

if ($result->num_rows > 0) {
    echo "<h1>¡Bienvenido! El login fue seguro.</h1>";
} else {
    echo "Login fallido: Usuario o contraseña incorrectos.";
}
$stmt->close();
```

Guardaremos el archivo PHP modificado y volveremos a desplegar Docker-Compose.

Iremos a la pestaña "**Repeater**" donde tenemos el ataque guardado (' OR 1=1 --) y ejecutaremos la prueba/ataque. Nos aseguraremos de que el *payload* de inyección todavía siga en el campo *username* y enviamos (*send*).

The screenshot shows the Burp Suite interface with the Repeater tab selected. In the Request pane, a GET request is shown with the URL `/login.php?username='+OR+1=1--+&password=` highlighted in red. The Response pane displays the server's response, which includes a PHP script. The script uses prepared statements to handle user input, specifically for the 'username' field. It checks if the number of rows returned from the database is greater than zero, indicating a successful login attempt. The response body contains the message "¡Bienvenido! El login fue seguro." (Welcome! The login was successful.)

```
1 HTTP/1.1 200 OK
2 Date: Wed, 26 Nov 2025 18:20:28 GMT
3 Server: Apache/2.4.65 (Debian)
4 X-Powered-By: PHP/8.1.33
5 Vary: Accept-Encoding
6 Content-Length: 497
7 Keep-Alive: timeout=5, max=100
8 Connection: Keep-Alive
9 Content-Type: text/html; charset=UTF-8
10
11 // CÓDIGO SEGURO FINAL
12 $user_input = $_POST['username'];
13 $pass_input = $_POST['password'];
14
15 $stmt = $conn->prepare("SELECT * FROM users WHERE
16 username = ? AND password = ?");
17 $stmt->bind_param("ss", $user_input, $pass_input);
18
19 $stmt->execute();
20
21 $result = $stmt->get_result();
22
23 if ($result->num_rows > 0) {
24 // Si hay filas, el login es exitoso
25 echo "<h1>
26         ;Bienvenido! El login fue seguro.
27     </h1>
28 ";
29 } else {
30 echo "Login fallido: Usuario o contraseña incorrectos.";
31 }
32 $stmt->close();
```

La prueba de mitigación fue exitosa. Al ejecutar el *payload* malicioso (' OR 1=1 #) contra el servidor con las **sentencias preparadas** implementadas, el sistema respondió con un '**Login fallido**'. Esto confirma que la vulnerabilidad de Inyección SQL ha sido **corregida** mediante la correcta separación de la lógica de la consulta y los datos del usuario.

¡Bienvenido! El login fue seguro.

```
"; } else { echo "Login fallido: Usuario o contraseña incorrectos."; } $stmt->close();
```

Este método garantiza que los payloads probados en Repeater serán tratados como nombres de usuario literales y el login fallará.

Comentarios

Esta práctica de descubrimiento de **BurpSuite** y de la vulnerabilidad de Inyección SQL fue altamente instructiva, aunque presentó varios **desafíos técnicos** en la fase de configuración del entorno que requirieron una solución profunda y metódica.

Problemas y soluciones

Error de compatibilidad de PHP (Función Obsoleta):

Tras el despliegue, la aplicación web falló con el error *Call to undefined function mysql_connect()*. Se identificó que la **imagen base de PHP 8.1** no soportaba el código antiguo de la aplicación.

Solución:

Se tuvo que crear un **Dockerfile** personalizado para el servicio *web* que instalara la extensión moderna **mysqli** (`RUN docker-php-ext-install mysqli`). Esto permitió que el código heredado funcionara, manteniendo la vulnerabilidad de la consulta intacta para el laboratorio.

Conclusiones

Lo más sorprendente fue la **simplicidad del payload de Inyección SQL** utilizado para el *Bypass de Autenticación* (' OR 1=1 --). ¿Como una cadena tan corta y básica puede alterar la lógica de una consulta en la base de datos y conceder acceso administrativo si la validación de la entrada es deficiente? Interesante.

La herramienta **Repeater** de BurpSuite fue esencial. Nos ha permitido manipular una única petición HTTP/HTTPS repetidamente con alta velocidad, reduciendo drásticamente el tiempo para probar múltiples payloads.