HACK X CRACK

SQL INJECTION

By KENKEIRAS

Todo el tutorial puede ponerse en práctica en la siguiente dirección: http://talleres.hackxcrack.es/sql/practica/

hack X crack
WWW.HACKXCRACK.ES

Presentación

Antes de nada, me gustaría agradecer públicamente el labor y trabajo que han estado llevando a cabo últimamente varias personas por la comunidad. En primer lugar a Kenkeiras por este magnifico tutorial acompañado de la sección práctica en el apartado de talleres de la comunidad, pero también y no menos importante, la labor de Kmykc desde lo lejos en todo lo relacionado con los wargames, de Cam10n, Finaltime, los moderadores de sección y globales que permiten que la atención de otros se centre en nuevos proyectos y todos aquellos que habéis ayudado en algún momento, ya sea con trabajo o con donaciones.

Aprovecho para comunicaros, una vez más, y creedme que no me gusta nada tener que pedir y menos tan repetitivamente, que la comunidad está pasando por un momento MUY DELICADO ECONOMICAMENTE. Agradeceríamos a aquellos que puedan, que hicieran una donación por pequeña que sea. Podéis hacerla por paypal a la cuenta que consta en la web o pedirme los datos bancarios de Hack x Crack para hacer una transferencia o ingreso.

No os entretengo más, demos lugar a este estupendo tutorial, espero que lo disfrutéis y aprendáis mucho con el.

Un saludo a todos los Hack x Crackeros,

Neddih I

0. Antes de empezar...

0.1 ¿Que es una inyección SQL?

La vulnerabilidad a inyecciones SQL es un bug de seguridad que suele darse en sitios web aunque también puede darse en un programa normal. Se basa en un fallo en la comunicación entre el usuario final y la base de datos. Esta comunicación se hace a través de un lenguaje llamado SQL (Structured Query Language, Lenguaje Estructurado de Consultas), de ahí el nombre.

La forma más común de ese error es confiar en la entrada y permitir al usuario modificar la consulta a la base de datos para que haga algo diferente a su propósito original.

0.2 ¿Que es SQL?

SQL es un lenguaje diseñado para consultar y modificar bases de datos, es simple inglés, lo que permite aprenderlo rápidamente, por ejemplo, si queremos ver la información de una tabla *usuarios*, la consulta sería:

```
select * from bd_tuto;
```

Después veremos como funciona básicamente este lenguaje para poder usarlo en nuestro favor.

0.3 ¿Se puede evitar una inyección SQL?

Por supuesto, "simplemente" comprobando que los datos de entrada son seguros (normalmente los que provee el usuario), conseguiremos una página web/programa/script que no sea vulnerable a una inyección SQL.

Todo el tutorial puede ponerse en práctica en la siguiente dirección: http://talleres.hackxcrack.es/sql/practica/

1.Introducción a SQL

No tiene sentido explicar SQL a fondo, incluyendo como se crean tablas, bases de datos y demás, así procuraré centrarme en lo que puede ser útil en este caso y cada uno puede estudiar a fondo SQL por su cuenta si le engancha este mundillo de las bases de datos.

1.1 Tipos de datos

En este momento solo nos interesan dos tipos de datos, el resto no tienen importancia ya que son poco comunes, no son estándar, no son visibles desde el exterior o se obtienen a través de estos dos, estos son:

- **Alfanuméricos**: son cadenas de letras, números y símbolos, están delimitados por ' y ', por ejemplo: *'esto es un ejemplo'*
- **Numéricos**: son simples números, así que no necesitan estar delimitados, por ejemplo: *42*

1.2 Construcción de consultas

Todas consultas a la base de datos (peticiones y modificaciones de datos) tienen muchos elementos en común, si consideramos todas en conjunto, las partes que puede tener son:

- <nombre de la consulta><datos a los que se aplica>
- **from**<nombre de la tabla>
- **where**<condiciones para la operación>

IMPORTANTE:Y toda consulta acaba por punto y coma ";"

1.3 Consulta Select (recuperando datos de la base de datos)

Por ejemplo, volviendo a la consulta de ejemplo, veamos como funciona:

*select * from bd_tuto;*

- ***select ****, la operación **select** muestra la base de datos, los <datos a los que se aplica> son las columnas de la base de datos que se mostrarán, con ***** se mostrarán todas, si queremos que se muestren las dos primeras columnas, sería ***select 1, 2*** o llamando a las columnas por su nombre (aunque en lo que nos ocupa ahora no creo que sean conocidos).
- ***from bd_tuto*** , se especifica que la operación se realiza sobre la tabla "*bd_tuto*", si fuera sobre la tabla "otratabla", sería ***from otratabla***

- No hay un "***where***", no hay ninguna condición para que se muestre un usuario en concreto, pero suponiendo que solo queremos mostrar los usuarios que se llaman "*ejemplo*", y que ese dato está almacenado en una columna **usuario**, sería ***where usuario='ejemplo'*** , como se puede ver el nombre tiene que ser un dato alfanumérico (contiene letras), así que está delimitado por ' y ', esto es importante recordarlo para después.

1.4 Consulta Insert (introduciendo datos)

Esta consulta es ligeramente distinta a las demás, pongamos como ejemplo esta:

```
insert into  bd_tuto
values ( dato1, dato2, dato3 );
```

- ***insert into bd_tuto*** , el tipo de operación es ***insert*** (insertar), y se realiza sobre la tabla ***bd_tuto***.
- ***values (dato1, dato2, dato3)***, se especifican los datos a introducir entre paréntesis, hay que recordar que las variables alfanuméricas van entre ' y ' .

1.5 Consulta Delete (borrando datos)

Como su nombre indica, la consulta delete (borrar) sirve para eliminar datos, por ejemplo:

```
delete from bd_tuto
where usuario='ejemplo';
```

- ***delete from bd_tuto***, tipo de operación ***delete*** (borrar) sobre la tabla ***bd_tuto***.
- ***where usuario='ejemplo'***,se especifica la condición para borrar esa fila (en este caso que el nombre del usuario sea "***ejemplo***", explicare mejor esto después), esta parte no es necesaria, pero sin ella borra toda la tabla.

1.6 Consulta Update (actualizando datos)

```
update bd_tuto
set usuario='nuevo nombre'
where usuario='ejemplo';
```

- ***update bd_tuto***, es una operación de actualización (***update***) sobre la tabla ***bd_tuto***.

- ***set usuario='nuevo nombre'***, se indican que columnas se modificarán (aquí "***ejemplo***") y el nuevo valor (aquí "***nuevo nombre***").
- ***where usuario='ejemplo'*** , para no variar, la condición para que se aplique la actualización a esa fila (como siempre opcional), en este caso que el anterior "***usuario***" sea "***ejemplo***".

1.7 Consulta Describe (obteniendo información de una tabla)

La consulta **Describe** nos permite obtener información sobre una tabla y sus campos:

describe bd_tuto;

- ***Describe***, tipo de consulta, y ***bd_tuto***, tabla a la que se dirige.

1.8 Como funcionan los condicionales (where)

Los condicionales se suelen usar (sobre todo) en el "**where**" de las consultas para modificar solo lo que nos interesa de la tabla, su uso es bastante sencillo, si queremos que el valor de una columna sea igual a uno que decidamos haremos:

Si debe ser igual: *nombre_de_columna = variable*

Si debe ser distinto: *nombre_de_columna != variable*

Para que sea mayor: *nombre_de_columna> variable*

Para que sea mayor o igual: *nombre_de_columna>= variable*

... creo que ya se entiende, pero por si alguien necesita un poco de refuerzo aquí os dejo un link: https://encrypted.google.com/search?q=sql+where.

Además podemos encadenar cuantas comparaciones queramos, simplemente haciendo uso del **AND**:

comparación_1 ***and*** *comparación_2*

Para que sea "verdad" solo si las dos comparaciones son verdad (AND = Y), o

comparación 1 ***or*** *comparación 2*

Si es verdad cuando cualquiera de las dos lo es (OR = O no exclusivo). Por último, se pueden usar los paréntesis, como se hace en el álgebra común, por ejemplo

comparación_1 ***and*** *(comparación_2* ***or*** *comparación_3)*

Sería verdad, si lo es la primera comparación y además, lo fuera la segunda o la tercera.

2.Técnicas básicas de inyección SQL

2.1 Saliendo de la variable

Supongamos una consulta que comprueba que un par usuario / contraseña es correcto, haciendo

```
select *
from bd_tuto
where usuario = '$usuario' and (password = '$password');
```

(Para distinguir mejor, los datos variables tienen un $ al principio).

La idea es que si quien programó la parte que se conecta a la base de datos y hace la consulta no tuvo especial cuidado podemos "salirnos" de la variable y modificar la consulta, por ejemplo, si introducimos un usuario cualquiera ("ejemplo"), pero como contraseña introducimos una comilla simple ('), la consulta quedaría así:

```
select *
from bd_tuto
where usuario = 'ejemplo' and (password = ''');
```

Como se puede ver, la consulta es errónea, en la parte de la contraseña hay:

```
[...]password = '' ';
```

Se comprueba que "password" es igual a '' (esto no nos importa), pero después está el comienzo de un dato alfanumérico (del que ; forma parte), que nunca acaba, con lo que tampoco acaba la consulta, esto hace que se produzca un error, y la comunicación con la base de datos no continúe, pero ya sabemos que la aplicación es vulnerable a este tipo de ataques.

Si añadimos dos ', este error no se produciría, sino que simplemente quedaría un dato alfanumérico por el medio, normalmente las bases de datos simplemente ignoran ese dato . La situación entonces queda así:

```
select *
from bd_tuto
where usuario = 'ejemplo' and (password = '' '');
```

Por último, podemos apoyarnos en el operador ***or*** (si cualquiera de los dos componentes es "verdadero", el resultado lo es) así que poniendo como contraseña:
' or '1'='1 , la consulta resultante sería:

```
select *
from bd_tuto
where usuario = 'ejemplo' and (password = '' or '1'='1');
```

Lo que resultaría son todos los usuarios que se llamen "yoquese" y para los que la contraseña sea "" (vacía) o que "1" sea igual a "1", así que algún usuario saldrá (a menos que no haya ninguno con ese nombre). En este ejemplo, el fallo permitiría (por ejemplo) loguearse como otro usuario.

2.2 Inyecciones con datos numéricos

Es lo mismo que el ejemplo anterior, pero aplicado a datos numéricos, supongamos que se comprueba que usuario es a través de un ID (numérico)

```
select *
from bd_tuto
where id = $id;
```

(Estos ID se suelen ver en las URL's)
Lo interesante de estos casos es que (a menos que estén bien protegidos) dejan la consulta totalmente expuesta, supongamos que nuestra ID es 65535, si después de nuestra ID añadimos un ';' y una consulta adicional (sin el ';' , ya que está el de la consulta original), podemos hacer lo que queramos, por ejemplo, si usamos como una id ***65535; insert into bd_tuto values ('nuevoUsuario','nuevaContraseña',0)*** , el resultado será:

```
select *
from bd_tuto
where id = 65535; insert into bd_tuto values ('nuevoUsuario','nuevaContraseña',0) ;
```

Seguramente todo parezca que va bien en esta consulta, pero la tabla *bd_tuto* ahora tiene un usuario más, el que hemos inyectado.
Esto mismo se puede hacer con el otro tipo de datos, simplemente añadiendo las ' cuando sea necesario, además se puede hacer lo mismo que en el caso anterior (si hubiera una comprobación de ID), poniendo una id de ***65535 or 1=1*** :

```
select *
```

```
from bd_tuto
where id = 65535 or 1=1 ;
```

3. Más inyecciones SQL

Aunque ya se conozcan las técnicas básicas de inyección SQL, no todo lo que se puede inyectar es ***' or '1' = '1***

3.1 Comentarios

Supongamos que por un motivo cualquiera queramos eliminar una porción completa de la consulta a la base de datos, esto es posible porque SQL permite comentarios dentro de las propias consultas, los dos tipos más frecuentes (y que se encuentran en los servidores mas usados) son:

- Comentario en una sola linea (**--**), a partir de esto el resto de la linea se ignorará, aunque parece que puede dar problemas según que bases de datos, por ejemplo MySQL requiere que después haya un espacio en blanco y un carácter (por eso aquí usaré ***-- a*** para estos comentarios, la *a* no tiene ningún significado especial).
  Volviendo a la consulta de usuario/contraseña...

```
select *
from bd_tuto
where usuario = '$usuario' and password = '$password';
```

Si al final del nombre del usuario se añade un **'** (para "escaparse" de la cadena alfanumérica), se marca que se acabó la consulta con un ***;*** y después se añade un **--** a , la consulta quedaría así:

```
select *
from bd_tuto
where usuario = 'ejemplo' ; -- a ' andpassword = '$password';
```

(En rojo lo que queda comentado, lo que no "lee" el SQL) Como se puede ver, no se hace ninguna comprobación sobre la contraseña, ya que queda comentada, esto puede ser especialmente útil cuando, por ejemplo, solo se comprueba que la parte de la contraseña no tiene caracteres maliciosos.

- Comentarios multilínea (/* y */), a partir del comienzo del comentario (**/***), y hasta su final (***/**), todo se ignorará, por ejemplo :

```
select *
from bd_tuto
where usuario = '$usuario'
and password = '$password'
```

```
and ID=$ID;
```

Si al nombre del usuario se añade un **'** (de nuevo para "escapar" de la cadena alfanumérica) y un **/*** para iniciar el comentario, y a la ID (que, recordemos que es numérica) un ***/** para finalizar el comentario, el resultado sería:

```
select *
from bd_tuto
where usuario = 'ejemplo' /* '
andpassword = 'blablabla'
and ID=65535 */;
```

(Lo rojo, es lo comentado) Como se puede ver, es muy útil cuando las comprobaciones se realizan en distintas líneas.

A veces, con un solo tipo de comentarios, no es suficiente, si en el caso anterior, la ID fuera alfanumérica:

```
select *
from bd_tuto
where usuario = 'ejemplo' /* '
andpassword = 'blablabla'
and ID='65535 */ ';
```

La consulta es errónea, así que no nos sirve, pero si la ID se cambia por un ***/**, para acabar el comentario anterior, un **;** para acabar la consulta y un -- ***a*** para comentar el resto de la línea:

```
select *
from bd_tuto
where usuario = 'ejemplo' /* '
andpassword = 'blablabla'
and ID='65535 */ ; -- a';
```

Aunque el **;** no sería necesario si en la consulta estuviera en una línea independiente, porque:

```
select *
```

```
from bd_tuto
where usuario = 'ejemplo' /* '
andpassword = 'blablabla'
and ID='65535 */ -- a'
;
```

Además comentar que el ***/**/*** se puede utilizar como sustituto de un espacio, esto lo hace especialmente útil cuando se filtran los espacios de lo que introduce el usuario, haciendo que se puedan seguir construyendo consultas que los requieran.

3.2 Concatenar consultas y nombres de tablas.

El problema hasta ahora era que no conocíamos el nombre de las otras tablas, lo que nos impedía hacer nuestras propias consultas, esto se puede solucionar leyendo la tabla ***information_schema.tables*** , leyendo la columna ***table_name***, podemos conocer los nombres de las tablas, si pudiésemos hacer la consulta directamente, con algo así bastaría:

```
select table_name
from information_schema.tables;
```

El problema ahora está en como inyectar esta consulta, no podemos simplemente finalizar la anterior y añadir esta después, pues puede causar problemas en la capa de la aplicación, con lo que no recibiríamos los datos.
La solución es unir dos consultas ***select*** , ahora el problema es que el número de tablas que devuelven las dos consultas (la original y la inyectada) debe ser el mismo. Se puede controlar el número de tablas de la consulta inyectada simplemente añadiendo columnas inútiles, por ejemplo, si la consulta original tiene 4 columnas, la inyectada podría ser:

```
select table_name,2,3,4
from information_schema.tables;
```

Las columnas 2, 3 y 4 son inútiles (son columnas con los números 2, 3 y 4 respectivamente), pero cumplen la función de regular el número de columnas en la consulta inyectada, ahora es necesario conocer el número de columnas en la consulta original, esto se puede hacer, por ejemplo, con la instrucción ***order by*** (va después del where) que indica que las consultas se ordenen según una columna en concreto, en este caso se puede indicar las columnas por su número, esto nos permite conocer su cantidad, ya que mientras se ordene por una columna existente, la consulta será correcta, pero si se ordena por una que no existe, habrá un error, por ejemplo, si la tabla usuarios tiene 3 columnas, al hacer

order by 1 El resultado será correcto
order by 2 El resultado será correcto
order by 3 El resultado será correcto
order by 4 Habrá un error (esa columna no existe)

Así sabemos que hay solo hay 3 columnas.
Por último, para unir dos consultas ***select*** se escribe la primera consulta (sin acabar con un ***;***), y se sigue con un ***union*** y la segunda consulta ***select***.
Volviendo al caso de la comprobación de la ID:

```
select *
from bd_tuto
where id = $id;
```

Añadiendo a la ID un ***union*** para incluir la otra consulta, y la consulta inyectada adaptada para el número de columnas:

```
select table_name, 2, 3
from information_schema.tables
```

(el ***;*** no es necesario, ya que está el de la otra consulta)
El resultado sería:

```
select *
from bd_tuto
where id = 65535 union select table_name, 2, 3
from information_schema.tables ;
```

3.3 Limitando los resultados

En algunos casos será necesario limitar el número de resultados, entonces habrá que utilizar la instrucción ***limit*** , su uso es bastante sencillo, se añade a un ***select*** para decidir que resultados se mostrarán, la sintaxis es

```
limit <primer caso>,<nº de casos>
```

Si por ejemplo tiene que salir solo el resultado número 31, se añadiría ***limit 30,1*** ,si interesa tener los resultados 31 y 32, ***limit 30,2*** (notar que el primer resultado es el número 0).
Así, si en la comprobación de usuario/contraseña esta falla cuando hay más de un resultado, se podría añadir al nombre de usuario ***'limit 0,1*** , para que solo muestre el primer resultado, y eliminar el resto de la consulta:

```
select *
from bd_tuto
where usuario = 'ejemplo' limit 0,1 ; -- a ' and password = '$password';
```

4. Defensa

La defensa no es demasiado complicada, consiste "simplemente" en asegurarse de que los datos de entrada son seguros. Como el lenguaje que se usa en estos casos suele ser PHP, es el que usaré, pero los conceptos son los mismos.

La forma de filtrar la entrada cambia según el tipo de datos, si es un número, simplemente hay que interpretarlo como tal y desechar el resto, por ejemplo,de "65535 or 1=1" pasaría a ser "65535".
Esto se puede hacer con ***$variable_segura = intval($variable_insegura);***
O, si es un número con coma flotante ***$variable_segura = floatval($variable_insegura);***

Si es un dato alfanumérico, solo hay que usar la función ***mysql_real_escape_string(cadena)*** , que sanitiza la cadena.

Lo que hace la función ***mysql_real_escape_string*** es escapar los caracteres, haciendo que no tengan ningún significado especial, por ejemplo, un ***'*** significa el final de una variable alfanumérica, pero con un ***₩*** antes (escapado), no significa el final, es solo una comilla en la variable, así, con ***$variable_segura = mysql_real_escape_string($variable_insegura);***

Si ***$variable_insegura*** es ***' or '1'='1***, la variable segura sería ***₩' or ₩'1₩'=₩'1*** , como se puede ver, todo lo que tiene algún significado en SQL tiene un ***₩*** antes, así que se convierte en un carácter normal.

Eso es todo, tomando estas medidas de seguridad, el código está seguro contra las inyecciones SQL.

Ejercicios resueltos

Aquí van 4 ejercicios resueltos sobre como evadir filtros mal programados para que podáis practicar un poco:

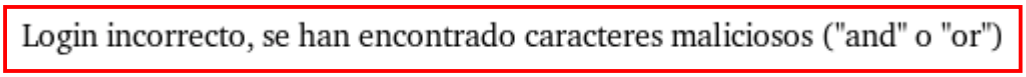
NOTA: RECORDAD QUE TODO ESTO SE PUEDE PONER EN PRACTICA EN: http://talleres.hackxcrack.es/sql/practica

Filtro 1

Empecemos con el primer filtro, se rellena el formulario con los valores típicos para una inyección SQL:

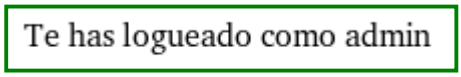
(Lo que aparece a la derecha del recuadro de password es lo que se introdujo en este)

Y produce un error:



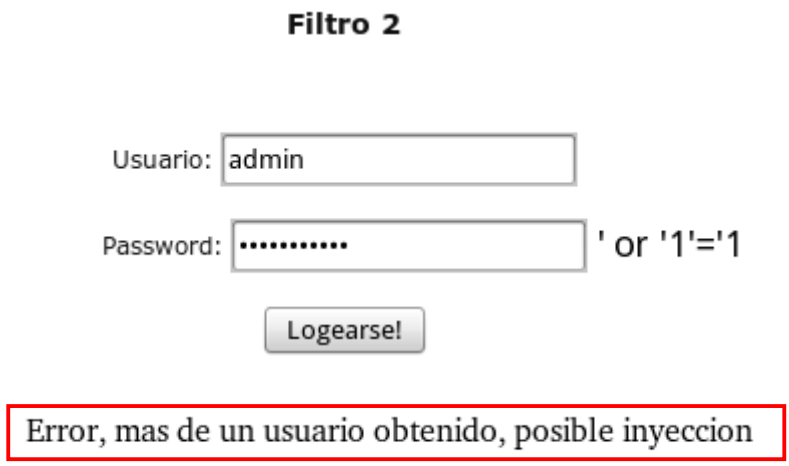
El filtro intenta evitar la inyección detectando los "and" y "or", la solución es sencilla, cambiar el "or" que alternativamente se puede representar como **||** (y el "and" como **&&**), esto hace que lo que enviaremos ahora sea así:

Y como se elimina el "or" ya no hay ningún problema:

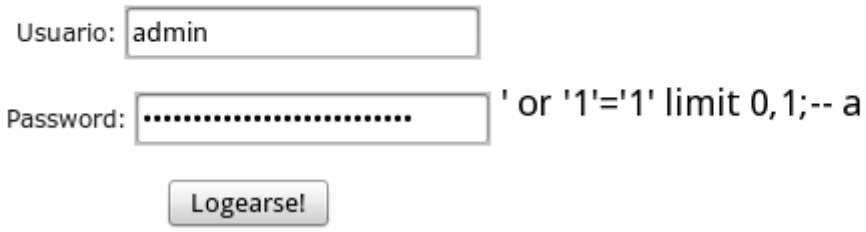


Filtro 2

Sigamos con el segundo filtro, probando con los valores habituales: **También produce un error:**

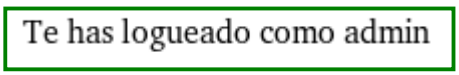


Es decir, **el filtro comprueba si se obtuvo más de un usuario**, recordemos que esto se puede superar delimitando lo que se obtiene de la base de datos usando **limit** ():



(El comentario al final permite que el **limit** sea el ultimo argumento que se envía al servidor SQL)

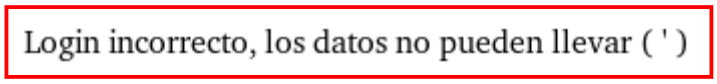
Con esto, se soluciona el problema del exceso de usuarios



Filtro 3

El tercer filtro es ligeramente más complejo, pero sigue sin ser imposible, para no variar, una consulta normal no funciona:

Porque el **filtro no deja pasar consultas con ' (comilla simple)**, carácter que se usa para envolver las cadenas alfanuméricas:



Supongamos que la consulta es algo del estilo (no tiene porque ser exactamente, pero servir para el funcionamiento general, después se podría refinar):

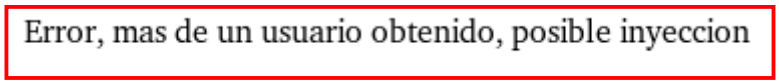
```
select * from usuarios
where usuario='$username' and password='$password';
```

Y que lo único que hace el filtro es detectar los (**'**), una forma de salir de la cadena alfanumérica es escapar con un **** el **'** del usuario, para que la cadena dure hasta el comienzo de la contraseña, donde acaba, y donde la contraseña estará fuera de una cadena, de una forma más gráfica (las cadenas en verde), quedaría algo así:

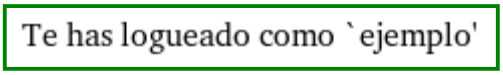
```
select * from usuarios
where usuario='\' and password='$password';
```

Lo que permite usar la contraseña para modificar la petición al servidor:

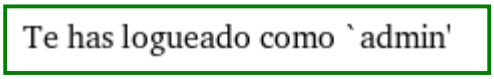
Pero esto vuelve a generar el error de exceso de usuarios



De nuevo, solo habría que añadir un **limit** para evitarlo:



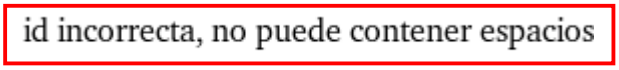
Es ir variando los valores del **limit** (0, 1 1, 2 2, 3) hasta encontrar el usuario deseado:



Filtro 4

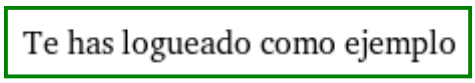
Por último, un filtro para valores numéricos, se prueba lo habitual:

Y el filtro avisa de que las id no llevan espacios:



La solución es sencilla, y ya se expuso al hablar de los comentarios, al abrir y cerrar un comentario multilínea se produce un espacio, algo que se puede aprovechar para construir la consulta:

Una vez superado eso, ya está:



5. Ejercicios

Algunos ejercicios propuestos para practicar lo aprendido en los talleres:

- ***5.1:*** Loguearse como administrador, por la tabla ***ej1*** .
- ***5.2:*** Conseguir la lista de usuarios y contraseñas de la tabla ***ej1*** .
- ***5.3:*** Modificar una contraseña de la tabla ***ej1*** .
- ***5.4:*** Introducir un nuevo usuario en la tabla ***ej2*** .
- ***5.5:*** Borrar los usuarios de la tabla ***ej2*** .

6. Referencias

Para ampliar la información o como referencia, estas páginas pueden servir como punto de partida:

http://dev.mysql.com/doc/refman/5.0/es/index.html [MySQL]

http://msdn.microsoft.com/en-us/library/ms189826(SQL.90).aspx [Transact-SQL]

https://secure.wikimedia.org/wikipedia/es/wiki/SQL [SQL–Wikipedia]

https://secure.wikimedia.org/wikipedia/es/wiki/Inyección_SQL [Inyección SQL–Wikipedia]

http://es2.php.net/manual/en/book.mysql.php [MySQL–PHP]

https://secure.wikimedia.org/wikipedia/es/wiki/SQLite [SQLite–Wikipedia]