

**Instituto Politécnico
Nacional**



Escuela superior de computo

Tema:

Prevención de SQL Injection en Mysql.

Asignatura:

Gobierno de TI (Computer security)

Alumno:

Julio Cesar Hernández Reyes

Docente:

Alejandro Sigfrido Cifuentes Álvarez

Grupo:

6CM2

Fecha:

05/Enero/2022

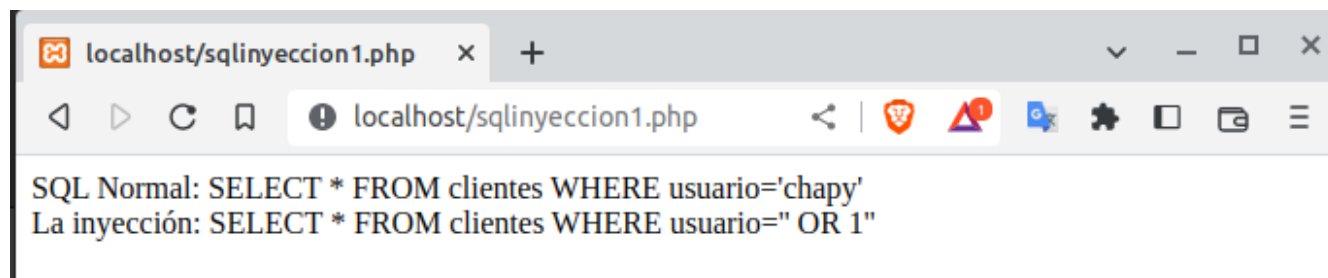
En algunos casos existe la posibilidad en la que se deja abierta una base de datos; ello ocasiona un problema de seguridad conocido como SQL Injection. Además, es muy grave si se ha tomado la entrada del usuario sin procesar y se ha insertado en la base de datos.

La inyección de SQL consiste en insertar una instrucción de MySQL para ejecutarla en la base de datos sin el conocimiento del administrador. La inyección ocurre generalmente cuando se pide a un usuario una entrada, por ejemplo el nombre, y en vez del nombre se ingresa una instrucción de MySQL que en forma impredecible se ejecutará en la base de datos.

Ejemplo de inyección SQL.

Enseguida se muestra una cadena de ejemplo que se ha tomado de un usuario normal y un usuario incorrecto que intenta utilizar SQL Injection. Se le solicita a un usuario su inicio de sesión, el cual se utilizará para ejecutar una instrucción SELECT y obtener su información.

```
sqlinyeccion1.php x sqlinyeccion2.php x
1 <?php
2     $nombre = "chapy";
3     // un nombre de usuario
4     $query = "SELECT * FROM clientes WHERE usuario='
5         $nombre'";
6     echo "SQL Normal: ".$query."<br />";
7     $inyeccion = "' OR 1'";
8     // entrada del usuario utiliza SQL Injection
9     $danio = "SELECT * FROM clientes WHERE usuario='
10         $inyeccion'"; // consulta no segura
11     echo "La inyección: ".$danio;
12     ?>
```



localhost/sqlinyeccion1.php x +

localhost/sqlinyeccion1.php

SQL Normal: SELECT * FROM clientes WHERE usuario='chapy'
La inyección: SELECT * FROM clientes WHERE usuario=" OR 1"

La consulta normal no es un problema, ya que la instrucción MySQL sólo seleccionará todos los clientes que tienen un nombre de usuario igual a **chapy**.

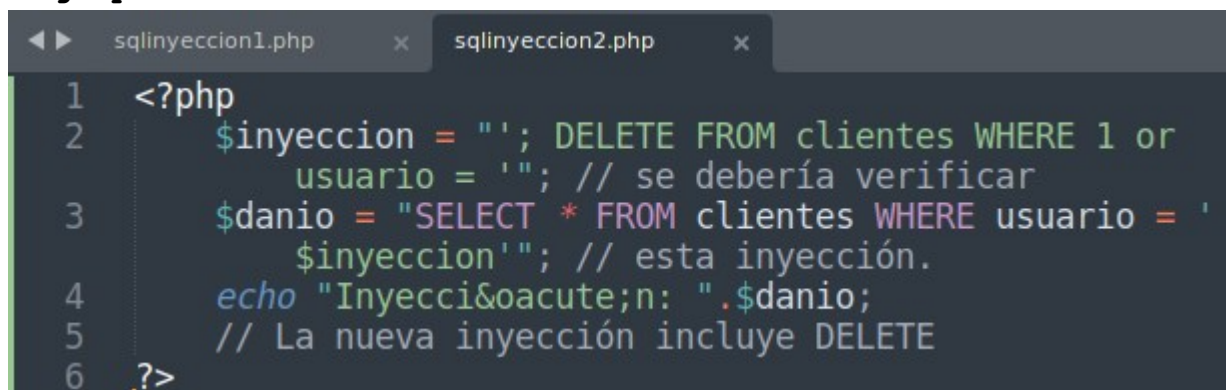
Sin embargo, el ataque de inyección hace que la consulta se comporte de manera diferente de lo que se pretendía. Mediante el uso de una comilla sencilla (') se ha cambiado y terminado la parte de la cadena de la consulta MySQL: **usuario = ''**.

Y luego se ha añadido a la declaración WHERE con una cláusula OR de 1 (siempre es true): **usuario = '' OR 1**

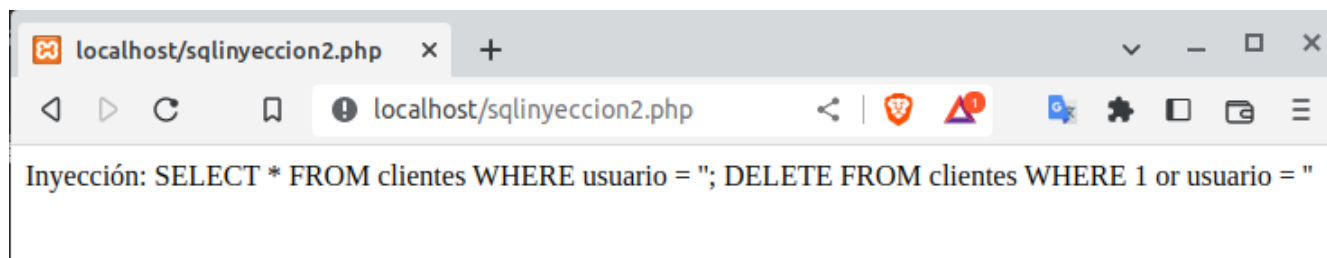
Esta cláusula **'OR 1'** siempre será verdadera y, por lo tanto, cada entrada en la tabla clientes sería seleccionada por esta declaración.

Algunos Ataques peligrosos de inyección **SQL**. Aunque el ejemplo anterior muestra una situación en la que un atacante podría tener acceso a mucha información que no debería tener, los ataques pueden ser peores. Por ejemplo, un atacante podría vaciar una tabla ejecutando una instrucción **DELETE**.

Por ejemplo:



```
1 <?php
2 $inyeccion = "'; DELETE FROM clientes WHERE 1 or
   usuario = '"; // se debería verificar
3 $danio = "SELECT * FROM clientes WHERE usuario = '
   $inyeccion'"; // esta inyección.
4 echo "Inyección: ". $danio;
5 // La nueva inyección incluye DELETE
6 ?>
```



localhost/sqlinyeccion2.php x +

localhost/sqlinyeccion2.php

Inyección: SELECT * FROM clientes WHERE usuario = "; DELETE FROM clientes WHERE 1 or usuario = "

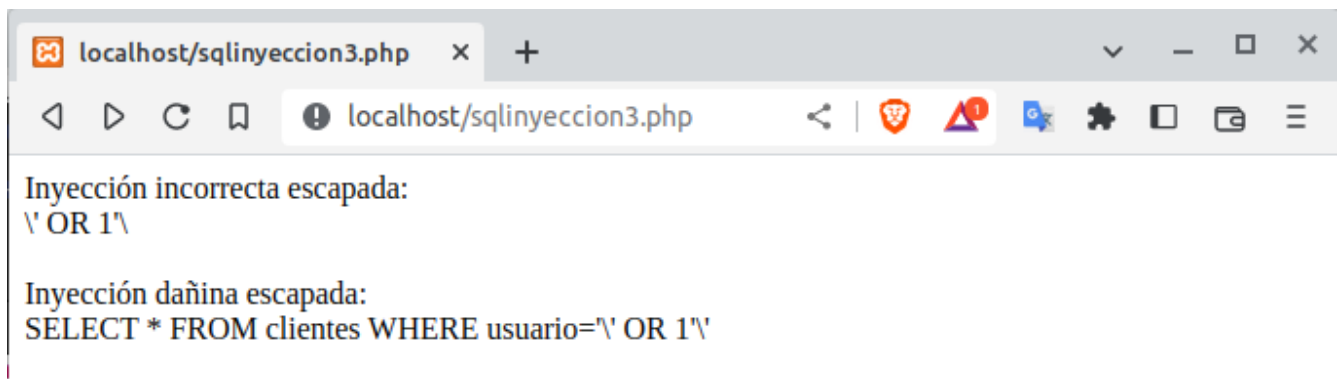
Si se ejecuta esta consulta, la instrucción **DELETE** inyectada vaciará completamente la tabla de **clientes**.

Prevención de la inyección con **mysql_real_escape_string()**.

PHP tiene una función **mysql_real_escape_string** para prevenir estos ataques. Lo que hace **mysql_real_escape_string** es tomar la cadena que será utilizada en la consulta MySQL y devolver la misma cadena con todos los intentos de inyección de SQL con escapes de seguridad. Básicamente, reemplazará aquellas comillas problemáticas (') que un usuario podría ingresar por un sustituto seguro de MySQL con una comilla y escape \'.

Ahora, se prueba esta función en los dos ataques de inyección anteriores para ver el funcionamiento. Por ejemplo:

```
sqlinyeccion3.php x
1 <?php //Nota: conectar a la base de datos MySQL para
  usar la funcion.
2 $mysqli = mysqli_connect("localhost","root","",""
  Tema_sql");
3 $inyeccion = "' OR 1'";
4 $inyeccion = $mysqli->real_escape_string($inyeccion
  );
5 $danio = "SELECT * FROM clientes WHERE usuario='
  $inyeccion'";
6 echo "Inyección incorrecta escapada: <br />"
  . $inyeccion . "<br /><br />";
7 $inyeccion = "'; DELETE FROM clientes WHERE 1 or
  usuario='";
8 $inyeccion = $mysqli->real_escape_string($inyeccion
  );
9 $danio = "SELECT * FROM clientes WHERE usuario='
  $inyeccion'";
10 echo "Inyección dañina escapada: <br
  />" . $danio;
11 ?>
12
```



Se debe observar que esas comillas dañinas se han escapado con una barra invertida \, evitando el ataque de inyección.

Ahora todas estas consultas tratarán de encontrar un nombre de usuario que es completamente desconocido:

Incorrecto: \ 'OR 1 \'

El daño: \'; DELETE FROM clientes WHERE 1 or usuario= \ '