



SWAP

INYECCIÓN SQL

Ginés Jesús Fuentes Sánchez

Índice

Descripción.....	3
Ejemplo.....	3
Blind SQL injection	4
Inyección basada en tiempo	4
Ejemplo Blind Injection	5
Protección contra Inyección SQL.....	7
Buenas técnicas.....	7
Sentencias preparadas	7

Descripción

Inyección SQL es un método de infiltración de código que se aprovecha de una vulnerabilidad informática en el sistema.

El origen de la vulnerabilidad radica en la incorrecta comprobación o uso de variables utilizadas en un programa a la hora de generar código SQL para realizar una instrucción.

Se dice que existe una inyección SQL cuando, de alguna manera, se inserta código SQL dentro del código SQL programado, a fin de alterar el funcionamiento normal del código y lograr así que ejecute la porción de código “invasor” incrustado en la base de datos.

Ejemplo

Todos los ejemplos se harán con PHP.

Si tenemos una típica consulta de búsqueda de texto simple como la que tenemos a continuación:

Tabla 1

[Index](#)[Tabla 1](#)

Consulta : `SELECT * FROM tabla1 WHERE text LIKE '%%'`

ID	Texto
1	texto1
2	texto2
5	texto13
6	texto14
7	texto21
9	texto22
10	texto22

Y tenemos este código que recoge el texto y realiza la consulta en BD.

```
$link = mysql_connect('localhost', 'inyeccion', 'inyeccion') or
die('No se pudo conectar: ' . mysql_error());
//echo 'Connected successfully';
mysql_select_db('inyeccion') or die('No se pudo seleccionar la
base de datos');

$query = "SELECT * FROM tabla1 WHERE text LIKE '%$texto%'";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());
```

Si ponemos nuestro texto normal para una consulta funcionará perfectamente:

Tabla 1

[Index](#) [Tabla 1](#)

Consulta : `SELECT * FROM tabla1 WHERE text LIKE '%1%'`

ID	Texto
1	texto1
5	texto13
6	texto14
7	texto21

Esto puede ser fácil de provocar una inyección SQL, simplemente con poner cualquier carácter especial en la cadena de entrada del input haremos fallar nuestra consulta, y el sistema quedará comprometido, sabiendo que no se controlan las cadenas de entrada y más importante, no se controlan los errores.

Tabla 1

[Index](#) [Tabla 1](#) Hola, nuevo. [Cerrar Sesión.](#)

Consulta : `SELECT * FROM tabla1 WHERE text LIKE '% ' %'`

Consulta fallida: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ' %' ' at line 1

Vemos que con una simple comilla ya hacemos que falle la consulta.

Blind SQL injection

Ataque a ciegas por inyección de SQL, se basan en realizar operaciones buscando siempre la respuesta correcta, que suele ser cuando la página no muestra mensajes o por error no tiene mensajes para esas respuestas, como: inicio de sesión, consulta SELECT, updates, etc.

Las sentencias condicionales tipo `'Or 1=1'` o `""=""` son consultas que siempre devuelve True por lo que sirven para realizar condiciones que la consulta siempre ejecutará.

Inyección basada en tiempo

La inyección a ciegas basada en tiempo hace que la base de datos se pause por un tiempo especificado, para que posteriormente devuelva los resultados.

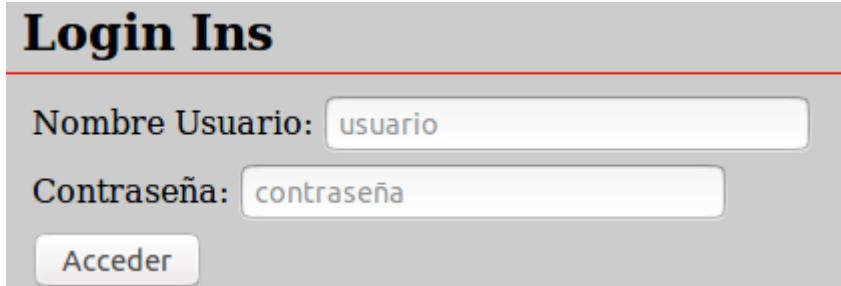
En base a esta inyección, un atacante puede llegar a saber el nombre de la base de datos, de una tabla, una contraseña, etc.

Relaciona tiempo con letras, por ejemplo, si la primera letra es una "A" esperar 5 segundos, si es una "B" esperar 6 segundos, etc. Con esto puedes, por el mismo proceso, ver la siguiente letra poniendo de nuevo relaciones con letras y tiempos hasta tener lo que se desea.

Ejemplo Blind Injection

Ahora vamos a ver el potencial de estos ataques. Vamos a ver un ejemplo típico de inicio de sesión de usuario en una página.

Tenemos nuestro formulario de login para introducir el usuario y la contraseña.



Login Ins

Nombre Usuario:

Contraseña:

Y como antes tenemos una consulta sencilla para ver si existe la pareja en la base de datos.

```
$nombre = $_POST["usuario"];
$pass = $_POST["password"];

// Conectando, seleccionando la base de datos
$link = mysql_connect('localhost', 'inyeccion', 'inyeccion') or
die('No se pudo conectar: ' . mysql_error());
mysql_select_db('inyeccion') or die('No se pudo seleccionar la base
de datos');

$query = "SELECT id FROM usuario WHERE nombre = '$nombre' AND
password = '$pass'";
$result = mysql_query($query) or die('Consulta fallida: ' .
mysql_error());

if (mysql_num_rows($result) > 0) {
    $_SESSION["nombreUsuario"] = $nombre;
    header('Location: index.php');
} else {
    header('Location: login.php?e=1');
}
```

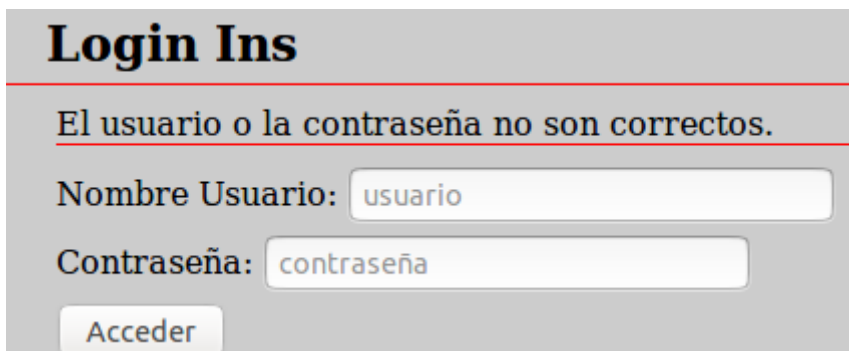
Con esto vemos que si introducimos el usuario y la contraseña correctamente, inicia sesión.



Inyeccion SQL

[Index](#) [Tabla 1](#) Hola, nuevo. [Cerrar Sesion.](#)

Al igual que si no la ponemos correctamente dará el error.



Login Ins

El usuario o la contraseña no son correctos.

Nombre Usuario:

Contraseña:

Al igual que en el ejemplo anterior, también se puede hacer una inyección SQL normal poniendo una comilla o algún carácter que no se espere y rompa la consulta, pero este tipo de inyección llegan más lejos.

Como sabemos, en una gran cantidad de los sistemas que se hacen existen usuarios con privilegios especiales, que suelen tener nombres como admin, administrador, root, etc... Vamos a ver como al tener un inicio de sesión inseguro rompemos la seguridad de una página fácilmente.

Ya que estos ataques se basan en las operaciones que siempre devuelven TRUE, vamos a proponer como nombre de usuario y contraseña algo que de siempre positivo.

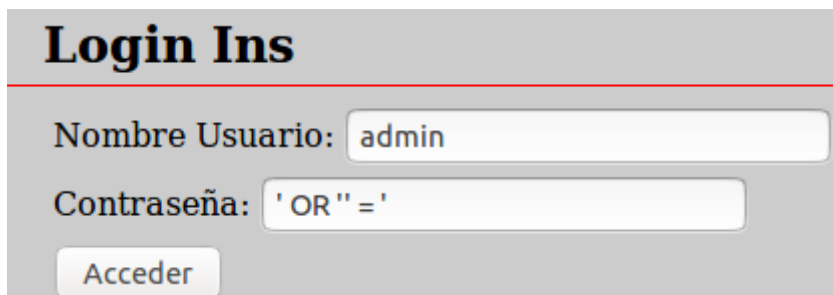
Si introducimos la cadena "' OR " = '" como nombre de usuario y como password, se ejecutará

```
$query = "SELECT id FROM usuario WHERE nombre = '$nombre' OR ' ' = ' ' AND password = '$pass' OR ' ' = ' '";
```

Como vemos esta consulta devolverá SIEMPRE true ahora, simplemente, tenemos que probar el nombre de usuario hasta que acertemos en el nombre de usuario.

En este momento podemos decidir si ponerlo en el nombre de usuario o no, ya que con un script y un algoritmo de fuerza bruta, sería fácil adivinar los nombres de los usuario de la base de datos. Vamos a ver como se puede hacer login fácilmente.

Vamos a intentar hacer login con admin, en el nombre de usuario ponemos admin, administrador, root, etc. Y en la contraseña la cadena que hemos nombrado anteriormente.



Login Ins

Nombre Usuario:

Contraseña:

Y vemos que acabamos de entrar como admin.



Inyeccion SQL

[Index](#) [Tabla 1](#) Hola, admin. [Cerrar Sesion.](#)

Esto nos permite también entrar como cualquier usuario que no sea admin.

Protección contra Inyección SQL

Vamos a ver algunas formas de evitar la inyección SQL con PHP.

Buenas técnicas

Existen algunas técnicas para mejorar la seguridad, aparte de controlar la entrada como pueden ser:

- El usuario de la base de datos no debe ser superusuario, se deben crear usuario específicos para la base de datos o tablas.
- Tener en cuenta todos los resultados de una consulta, por ejemplo, para el login se sabe que todo lo que no devuelva una sola línea con el usuario debe de ser errónea.
- Cuando llega un dato comprobar que es del tipo esperado y que concuerda con la base de datos.
- Cifrar/ofuscar los ids numéricos para mayor seguridad.

Sentencias preparadas

Se definen como un tipo de plantillas compiladas para SQL que las aplicaciones quieren ejecutar, pudiendo ser personalizadas utilizando parámetros variables.

Vamos a utilizar la librería PDO que tiene sql:

```
$nombre = $_POST["usuario"];
$pass = $_POST["password"];

try {
    $pdo = new PDO('mysql:host=localhost;dbname=inyeccion',
"inyeccion", "inyeccion");
} catch ( PDOException $e ) {
    die( "Conexión fallida: " . $e->getMessage() );
}

$query = $pdo->prepare("SELECT id FROM usuario WHERE nombre =
:nombre AND password = :pass");
// $query = $pdo->prepare("SELECT * FROM tabla1");
$query->bindParam(':nombre', $nombre);
$query->bindParam(':pass', $pass);

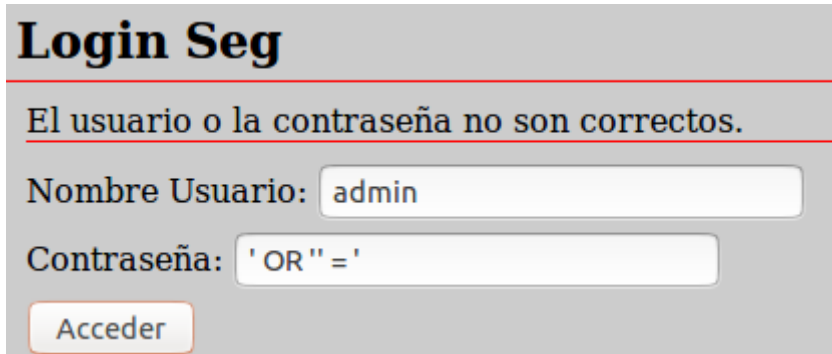
$query->execute();

echo $query->rowCount() . "<br>";
if ($query->rowCount() > 0) {

    $_SESSION["nombreUsuario"] = $nombre;

    header('Location: index.php');
} else {
    header('Location: login2.php?e=1');
    //echo "No se han encontrado resultados.";
}
```

Como vemos se crea la clase PDO con la conexión a la base de datos y a la tabla. Se utilizan parámetros para pasar los valores, de esta manera se evita la inyección SQL.



Login Seg

El usuario o la contraseña no son correctos.

Nombre Usuario:

Contraseña:

Ahora al poner el código con el que anteriormente hemos conseguido entrar en el sistema, da error, ya que lo que hace realmente esta librería es escapar todo lo que va dentro de la variable, por lo que toma todo literal. En este caso estaría ejecutando la consulta:

```
$query = "SELECT id FROM usuario WHERE nombre = 'admin' AND password =  
'\ ' OR '\ ' \= '\ '";
```

De esta manera evitamos la inyección sql en nuestro sistema.