

Forum:[Hack Forums](#)

Thread: [\[Tutorial\] Understanding Error-Based injections \[Detailed!\]](#)

How bout using your brain - September 27, 2014, 4:28 pm

This tutorial is somewhat advanced! You will probably need to have at least basic understanding of SQL. I will try to make it as simple as possible though, so everyone can follow.

Some pictures are posted as links as I can post max 15 images per post!

0. Motivation

Almost every 2-3 days I see threads here in which people ask how Error-Based SQL injection is actually happening. As I notice that a large part of people just learn the query to use but don't actually know what it does, I decided to do a tutorial for this.

I will talk about all three/four kinds of error based injections for MySQL databases that are currently "known".

1. Preliminaries

Some stuff to get cleared up before starting the tutorial. Skip this if you know about mysql command line and GROUP BY statements.

1.1 MySQL command line

Spoiler [\(Click to View\)](#)

I will be using the MySQL command line in this tutorial. For anybody who is not familiar with it, here is a simple query executed in it:

Code `SELECT user() as Username;`

This is how it looks like on the command line:

<https://i.imgur.com/aZPWIOf.png>

So on top you see the query, followed by a table in which the first row is the name of the columns and the following rows hold the requested data.

1.2 GROUP BY and aggregate functions

Spoiler [\(Click to View\)](#)

This tutorial makes heavy use of GROUP BY statements, so I wanted to make sure you understand them first.

What a GROUP BY is doing is taking the data from several rows via an aggregate function and then group the result by a column. What does that mean you ask? ?

Let's say we are interested in the amount of columns in each table. We can use the aggregate function `count(*)` to count:

Code `SELECT count(*) FROM information_schema.columns`

<https://i.imgur.com/OMSkWKp.png>

Now that gave us the total amount of columns in my database. BUT we want the count per table_name. We can do this like so:

Code `SELECT count(*), table_name FROM information_schema.columns GROUP BY table_name`
<https://i.imgur.com/HZnHOQ4.png>

Good job, that is what we wanted! So our GROUP BY has taken all the rows that had the same value for table_name and grouped them together. Then it invoked count(*) on those groups, which lead to count(*) not counting the total amount of rows, but only the amount of rows in that specific group.

Ok with this cleared up let's get to it!

2. Classic error-based injections

With classic error-based injections I mean the ones that are usually thought of when we talk about error-based injection. In colloquial language on HF they are often referred to as "error-based injection" and "double query injection". Now even though they seem to be two different things, they exploit actually the same weakness in the MySQL engine.

From this point on I will refer to the "error-based injection" version as "OR error-based" and the "double query injection" as "AND error-based". This is also the name that sqlmap is using and at the end of this tutorial you will understand that it makes total sense to use these names.

2.1 What are we trying to achieve?

In MySQL there is a bug concerning the GROUP BY statement. (see <http://bugs.mysql.com/bug.php?id=58081> and <http://bugs.mysql.com/bug.php?id=60808>).

Basically it will throw an error in certain scenarios when it shouldn't. One of those scenarios needs the following three requirements.

The Golden Rules of error-based injection:

- we use an aggregate function (anywhere) in our query like count(), min(), max(), avg(), etc.

- we GROUP BY a column that has two identical entries for two different rows

- the output of rand() function have to be part of that column (I could not reproduce error in any other way)

2.2 OR error-based

So taking the query that is usually used in error-based SQLi tutorials (this is injected in the WHERE clause of the query):

Code or 1 GROUP BY concat(version(),floor(rand(0)*2)) HAVING min(0)

Ok first let's take this apart:GROUP BY: already explained that above ?

floor(): Floor will round any floating point number down to the next lower integer:

Spoiler [\(Click to View\)](#)

rand(): This is a pseudorandom number generator which outputs a random number between 0 and 1. It is called pseudo-random because given a seed (in our query seed is 0), it will produce the same sequence of numbers every single time.

floor(rand(0)*2): We are multiplying the random number by 2, so now our random number is between 0 and 2. We then take the floor of that number, which can only be either 0 or 1. Take a look at the first 10 of those:

Spoiler [\(Click to View\)](#)

HAVING: it is like WHERE, just that we can specify conditions on aggregate functions.
min(): It is an aggregate function and in this case it is only used, because our "bug" is requiring the use of an aggregate function to be triggered

Why do we have a "or 1" before the GROUP BY? Glad you asked, let me explain.
Look at the Golden Rules, they are your friends ?

We already fulfilled number 1 (we are using min() as aggregate function) and number 3 (we are using rand() in the group by column). Requirement number 2 asks us to have two identical entries in that column! But we are appending random 0's and 1's to our column. Take a look again at the output of the first ten numbers of floor(rand(0)*2). Note that the second and third number are the same, so here the error will be thrown! But we do not know if the original query actually is selecting 3 rows. So adding a "or 1", all previous conditions from the query are removed and we get the maximum amount of rows (hopefully at least 3 ?).

Well that's pretty much all the magic there is to it! Fulfill the Golden Rules and you will get an error and eternal glory!

What I described so far was the "OR error-based" injection. Why OR? Well because we use an OR to start the injection, and we just learned the reasons for it.

2.3 AND error-based / "double query injection"

Now let's take a look at the "AND error-based" injection, better known as Double-Query injection.

When you see what kind of queries people post in double-query injection tutorials, you just wanna go kill yourself.

For example (no offense to that guy, I just randomly picked his thread):

```
Code and(select 1 from(select count(*),concat((select (select
concat(0x7e,0x27,cast(version() as char),0x27,0x7e)) from information_schema.tables limit
0,1),floor(rand(0)*2))x from
information_schema.tables group by x)a) and 1=1source:
```

<https://hackforums.net/showthread.php?tid=2085773>

Spoiler [\(Click to View\)](#)

Obviously this works, but who the fuck can actually read or write this shit? Let's get rid of most of this shit in the query:

and 1=1: ehm ok I don't see the use....

cast(): totally useless, just write version()

concat(): totally useless

select(select concat()): why simple if we can make it complicated...

select version() from information_schema.tables limit 0,1: useless to use this stuff when you can just use select version()

concat(select(version()), ...): nope, don't need select there neither

So the reduced, readable code looks like this:

```
Code and(select 1 from (select count(*),concat(version(),floor(rand(0)*2))x from
information_schema.tables group by x)a)
```

Bit better, isn't it? It also of course still works:

Spoiler [\(Click to View\)](#)

Let me save us some more selects by moving that concat() to the group by clause and we can get rid of that select 1:

Code and(select count(*) from information_schema.tables group by concat(version(),floor(rand(0)*2)))Spoiler [\(Click to View\)](#)

Do you see the similarities to our query from OR error-based? It is basically the same query, just as a subquery.

There is some slight modification though:

The biggest one is using select count(*) from information_schema.tables. Why do this? Look at the Golden Rules! We need to have at least three rows (information_schema.tables has more than 3 for sure.) and we need to use an aggregate function, in this case count(*).

But we might as well change it to having min(0) as in OR error-based:

Code and(select 1 from information_schema.tables group by concat(version(),floor(rand(0)*2)) having min(0))Spoiler [\(Click to View\)](#)

Still works perfectly. Once you understand how it works you can play around with it as you can see and adjust it to your needs.

The other difference to OR error-based is that here our query is actually a subquery in an AND condition which is injected into the WHERE clause. Hence the name "AND error-based" ?

The name double query comes from the fact, that we are using a subquery which is a query inside a query, so now you got a "double query" ?

So as you can see, AND and OR error-based are basically almost the same ?

3. XML error based

For the last paragraph I quickly talk about the other two error-based injection methods which are related to XML functions.

This shit is simple, basically whenever an XPath value is invalid it throws an error. There are two MySQL functions that accept XPath as input:

updatexml:

Code select updatexml(null, concat(0x0a, version()), null)Spoiler [\(Click to View\)](#)
<https://i.imgur.com/nhVHfqR.png>

extractvalue:

Code select extractvalue(null, concat(0x3a, version()))Spoiler [\(Click to View\)](#)

We put the 0x0a before the version() so that the whole output is displayed. On actual injection we would of course inject that as subquery or just as condition into WHERE clause.

I barely ever use xpath error based injection, but it is quite simple and straight forward to use.

That's it for today guys. Hope this was useful to some of you as it took me quite some time to write it up ?

Nicely explained. I wish I could rep you more to +3 as you are really a smart cat ?

r0ml4h - September 29, 2014, 10:54 am

awesome tutorial, very detailed explanation

sweetoow - September 29, 2014, 11:49 am

Awesome tutorial my friend...

it will really help people to actually understand the error based injections..

also.. why use `floor(rand(0)*2)` when we can use `hex(rand(0))`

-zERAI - September 29, 2014, 12:09 pm

Well written tutorial, nice contribution!

How bout using your brain - September 29, 2014, 12:51 pm

Thanks for the feedback guys!

(Sep 29, 2014 11:49 AM)sweetoow Wrote: [\[CITE\]](#) also.. why use `floor(rand(0)*2)` when we can use `hex(rand(0))`

Ha great! Never thought about doing that! Thanks for the idea ?

ridhisidhi12 - September 29, 2014, 7:01 pm

Its a very great tutorial an awesome tutorial Sir.

Sir one thing i want to know.

Tell me the exact detail use of

1. `floor(rand()*2)`
2. or 1
3. `min(0)`

I didn't understand that above part. So clear my doubts on above

Thanks again for the great tutorial

hacbach - October 1, 2014, 6:50 pm

SirRootALot:

Very good and detailed. But some do not see the image. Please correct the. Thanks

How bout using your brain - October 3, 2014, 3:18 am

(Oct 1, 2014 06:50 PM)hacbach Wrote: [\[CITE\]](#) SirRootALot:

Very good and detailed. But some do not see the image. Please correct the. Thanks

There is a maximum amount of images you can put into one post, so I had to remove some. But I posted the links so if you click on it you should be able to see them ?

iHacker-n00b - October 5, 2014, 8:54 am

Nice One Really HQ Post ? Couldn't be Explained Better

Aakash Choudhary - November 29, 2014, 12:14 am

Very nice tutorial SIR. Really a HQ post. ?

m_a_s_h - November 29, 2014, 2:50 am

Excellent explanation. You know your job :)

___**Spear**___ - *March 10, 2015, 7:02 am*

Awesome tutorial,
you are verrryyy talented ?

Disparity - *March 18, 2017, 1:43 am*

I know this tutorial is old, but it deserves way more attention than it ever got.