CIS 643 Computer Security

Lab 10 SQL Injection Attack Lab

**Instructor: Wenliang (Kevin) Du**

Contents

Preparation: Turn off build-in SQL injection protection 2

Task 1: SQL Injection Attack on SELECT Statements 2

Task 1.1: Log into another person’s account without knowing the password 2

Task 1.2: Find a way to modify the database 3

Task 2: SQL Injection on UPDATE Statements 4

Task 3: Countermeasures 6

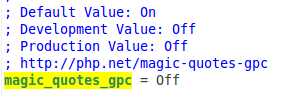
Task 3.1: Escaping Special Characters using magic\_quotes\_gpc 6

Task 3.2: Escaping Special Characters using mysql real escape string 6

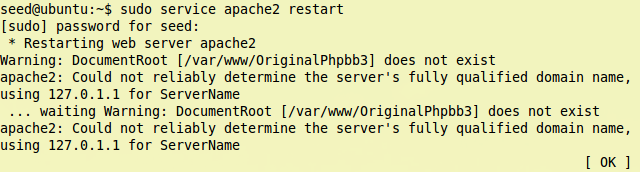
Task 3.3: Prepare Statement 7

## Preparation: Turn off build-in SQL injection protection

Open /etc/php5/apache2/php.ini file, set magic\_quotes\_gpc value to ‘Off’ to turn it off:



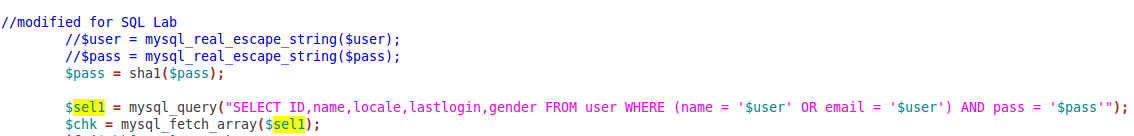
Restart apache server to active new setting.



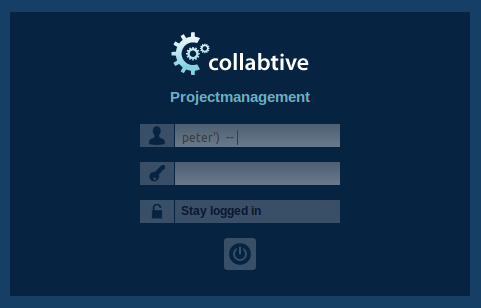
## Task 1: SQL Injection Attack on SELECT Statements

### Task 1.1: Log into another person’s account without knowing the password

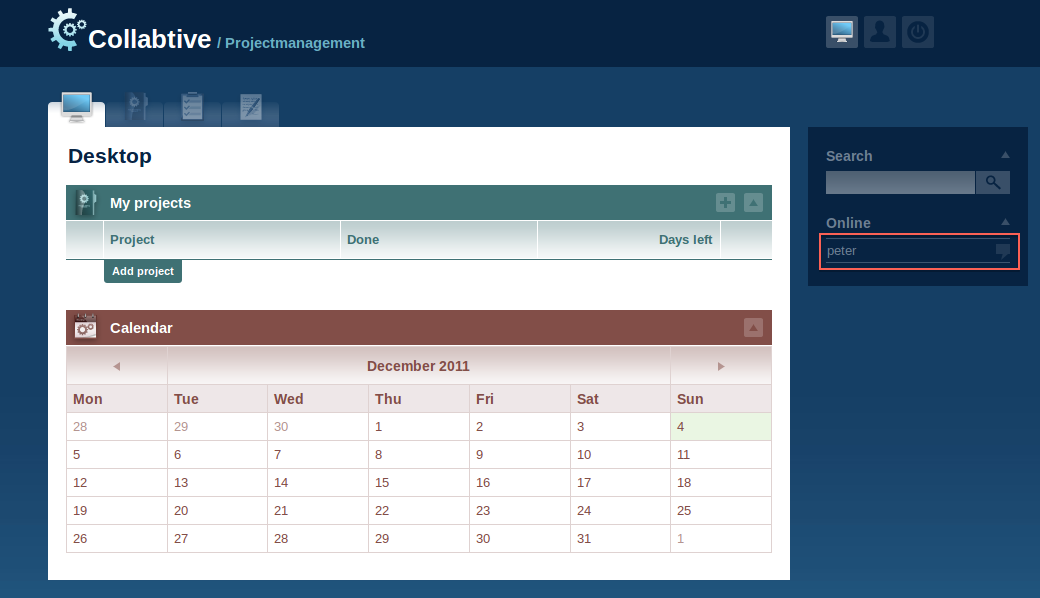
Following is PHP codes run at server side. In practice, we may not know how server send query to Database, but we still can guess.



Use flowing input to do attack, using ‘--’ to comment statement, make ‘name’ be the only condition for ‘where’

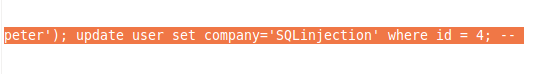


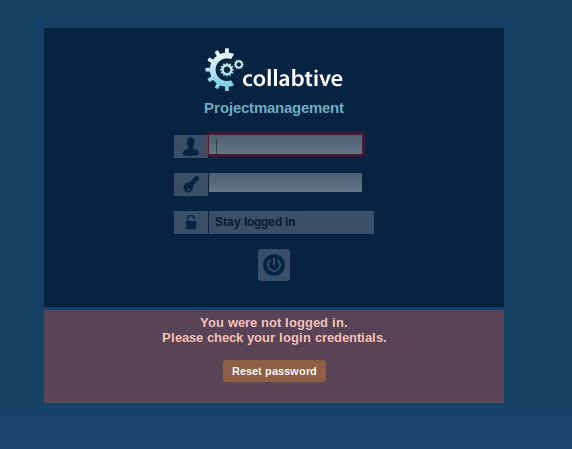
Then, we could log in successfully:



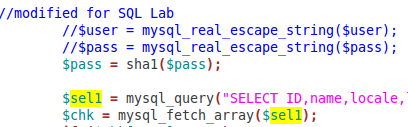
### Task 1.2: Find a way to modify the database

The basic way to modify the database by SQL injection is use semicolon to finish a SQL statement, and then append update/insert/delete statement right after semicolon. Hence it would send two statements to database. So I try using following input when log in:



However, we get failed

Check the PHP code again:



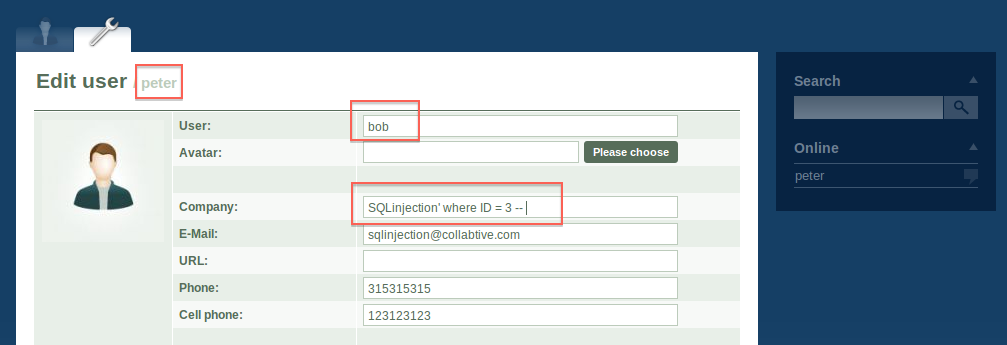
We could see that it use mysql\_query to send statement to database.

The reason that we could not succeed in injection is that mysql\_query could send one and only one query to database to run. Multiple queries are not supported.

So, it not allow to do select and update in the same time when using mysql\_query.

## Task 2: SQL Injection on UPDATE Statements

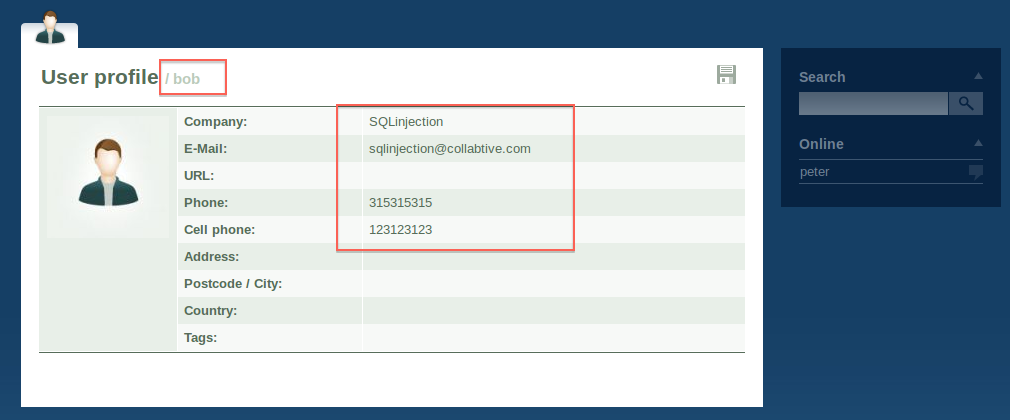
The principle of Task 2 would be the same as Task 1, just use ‘--’ to comment statement, and add our own where condition:



After send this form to server, we could see that peter’s profile is still the same as before:

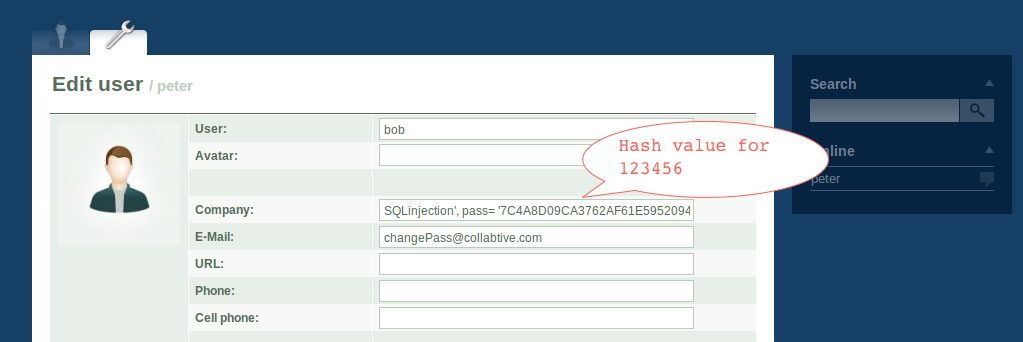


But if we view bob’s profile, we would find it has been modified with our inputs:

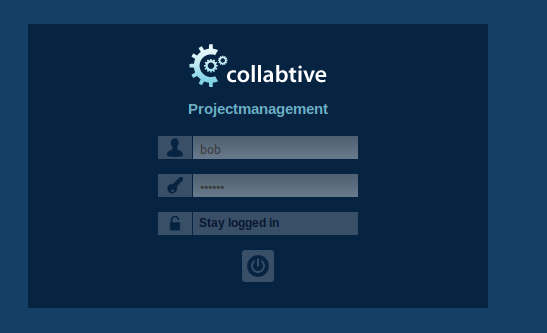


No this time, we try to modify the password of bob.

Since we want to change the password without knowing the old password, we could add ‘pass’ field in our injection. The value should be hashed:



Now log in bob with the our password:

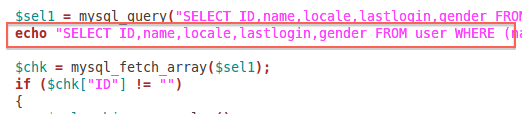


We could successfully log in:



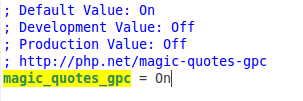
## Task 3: Countermeasures

To see what happen with following countermeasures, I add debug output information in PHP file:



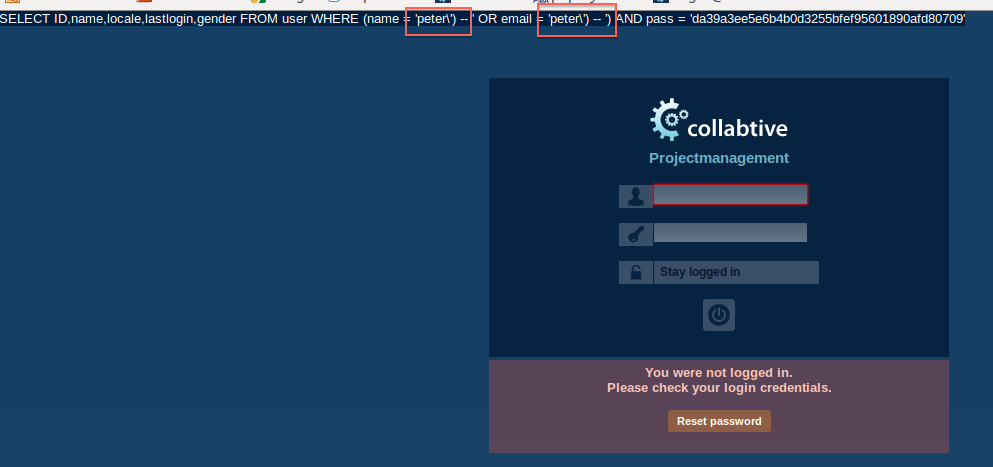
### Task 3.1: Escaping Special Characters using magic\_quotes\_gpc

Set magic\_quiotes\_gpc to On, and restart server



Use the same input that used for attack in Task1, but this time, we could not log into the system.

From the output:

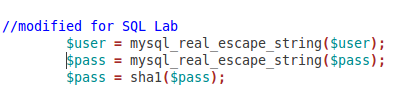


We could see there is backslash before single quote that we input in as user name.

This setting escapes the quote automatically, so that we could not match the quote, and use ‘--’ to comment the statement.

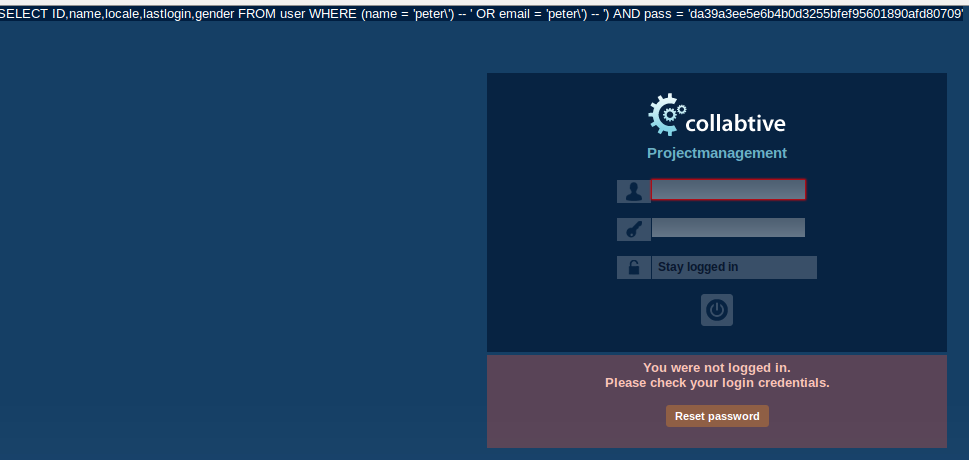
### Task 3.2: Escaping Special Characters using mysql real escape string

Set magic\_quiotes\_gpc to Off, but uncomment mysql\_real\_escape\_string in PHP file for login section.



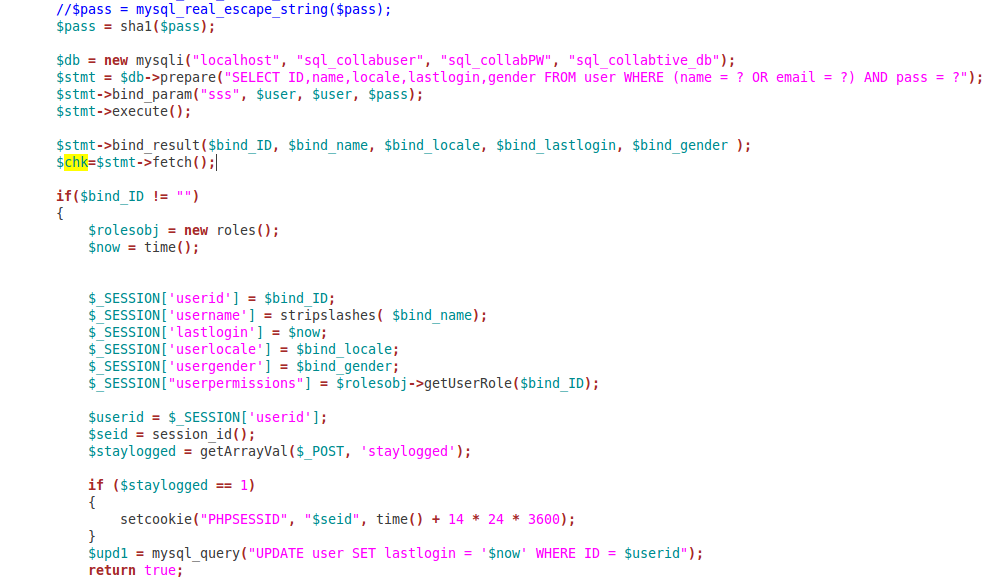
If we login with the input we used in Task 1, we still could not log in.

The output is the same as we got in previous task



### Task 3.3: Prepare Statement

The prepare statements for login section:



This time, neither could we login a user without password, as this mechanism separates the values from the statement.

However, we would still log in with propert

It sends values by parameters, and database treats each of them just as data, it would not be part of statement codes.