Spring 2020

CSE-5382-001 – Secure Programming

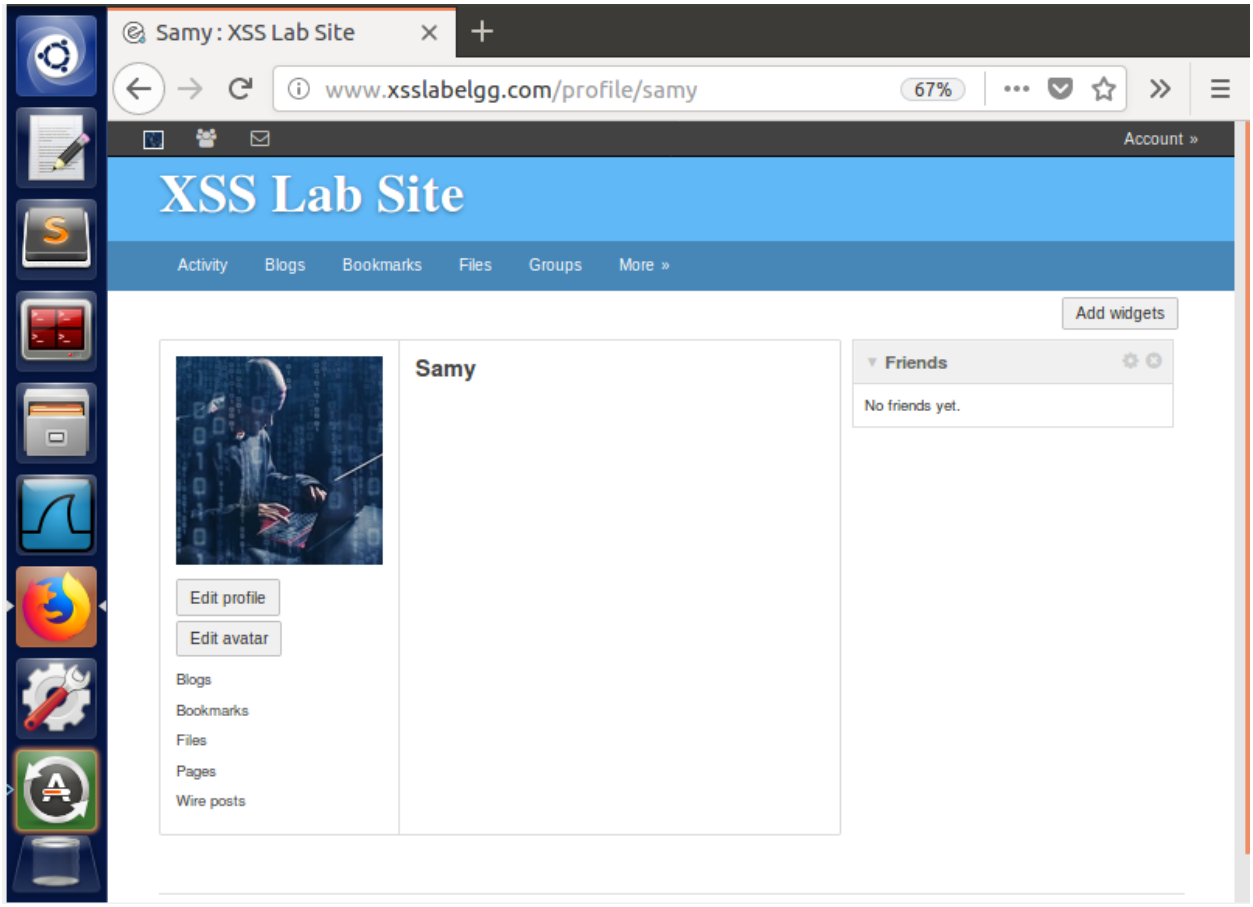
Homework Assignment 8 – Cross-Site Scripting (XSS) Attack

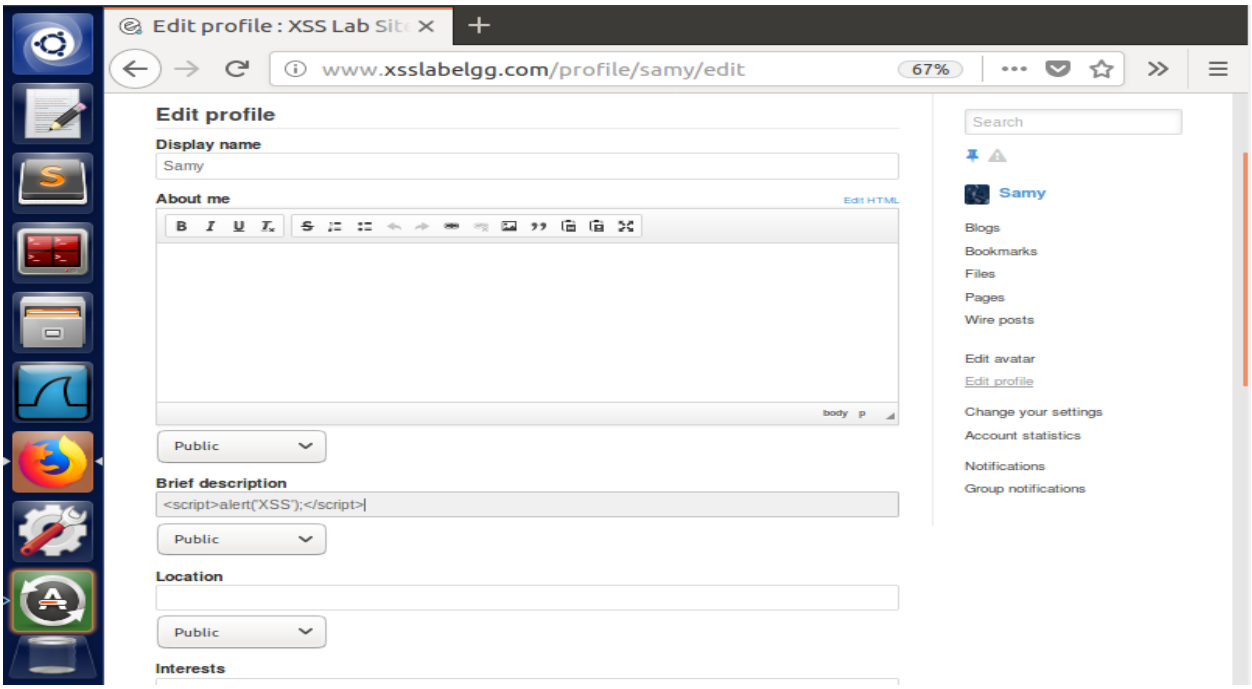
Name	UTA ID
Goutami Padmanabhan	1001669338

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

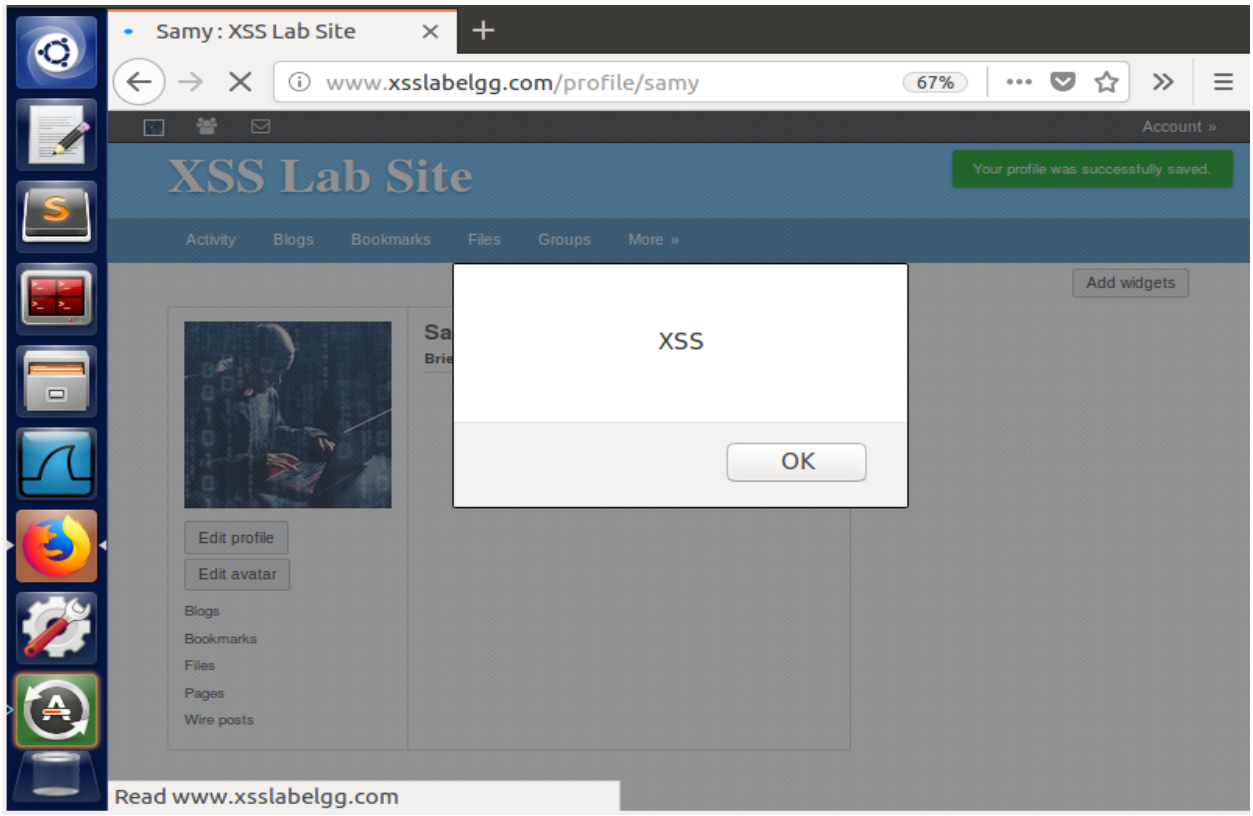
The purpose of this task is to embed a JavaScript in the attacker's profile so that, anyone who comes and visits this profile, the JavaScript will be executed, and an alert window will be displayed that says 'XSS'.

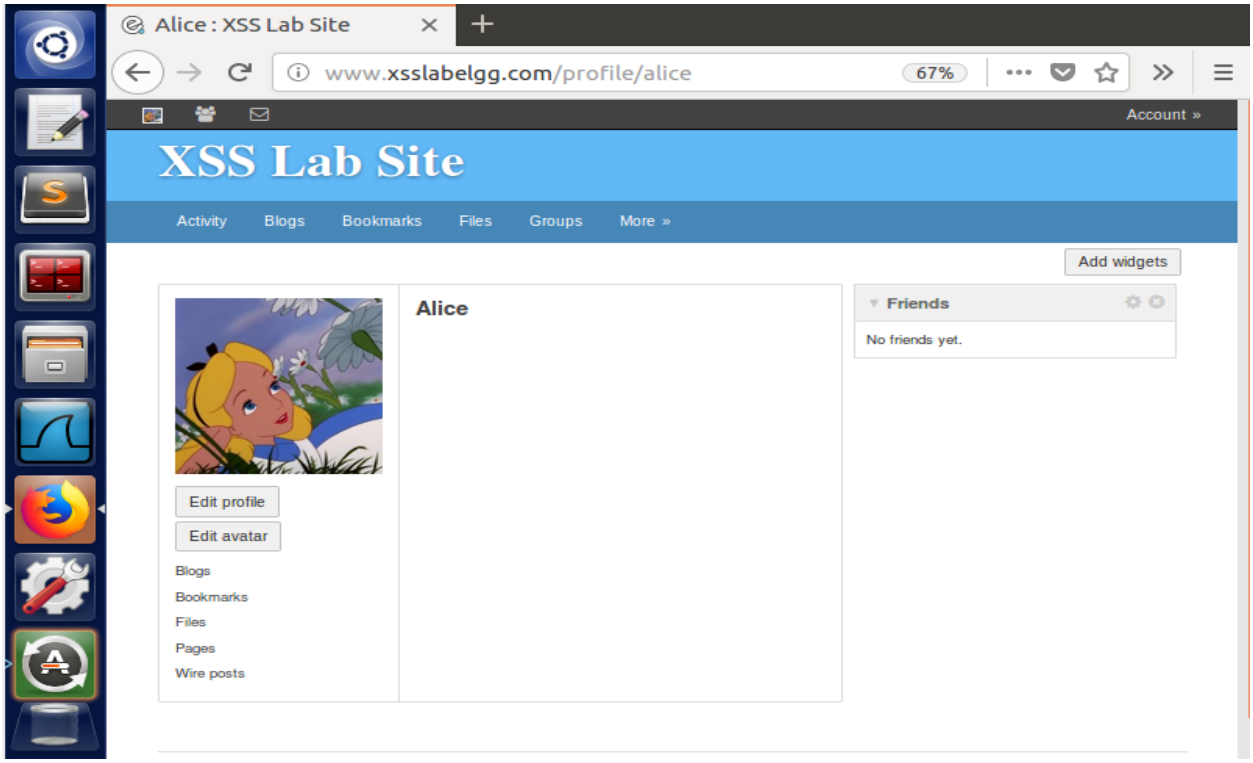
In this case, we go to the website www.xsslabelgg.com and use an attacker named Samy. The attacker Samy logs in with the credentials given in the assignment.



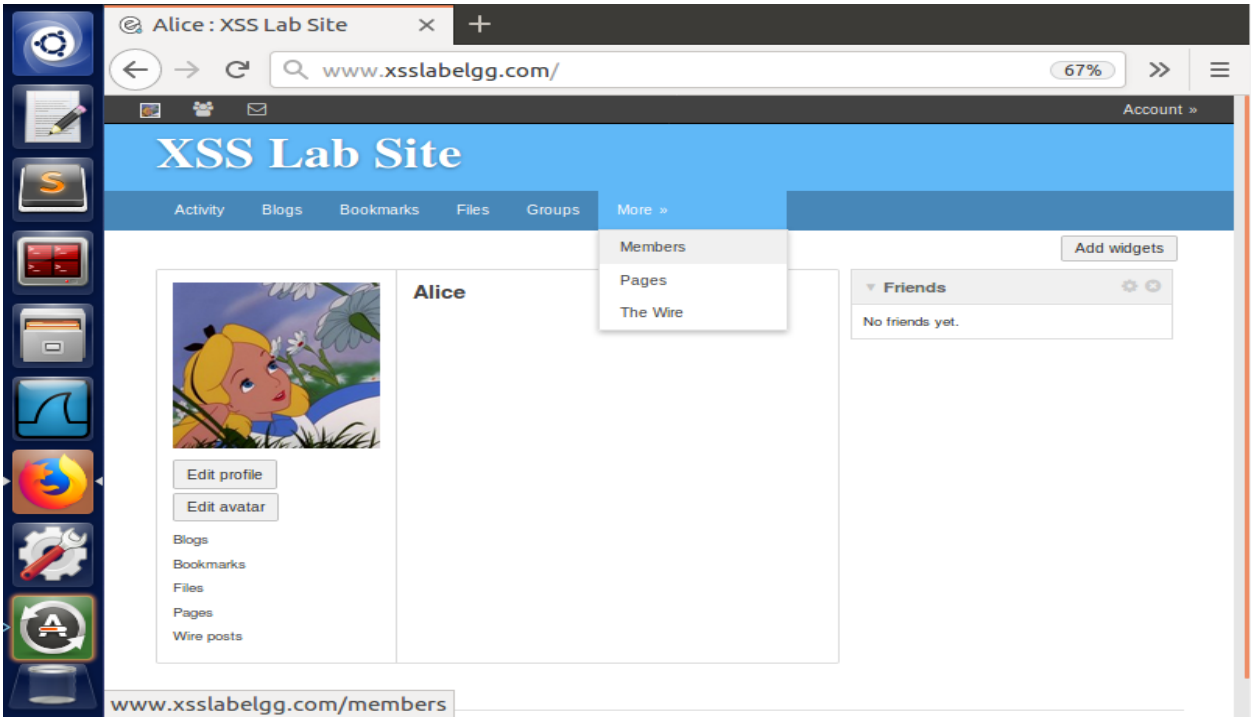


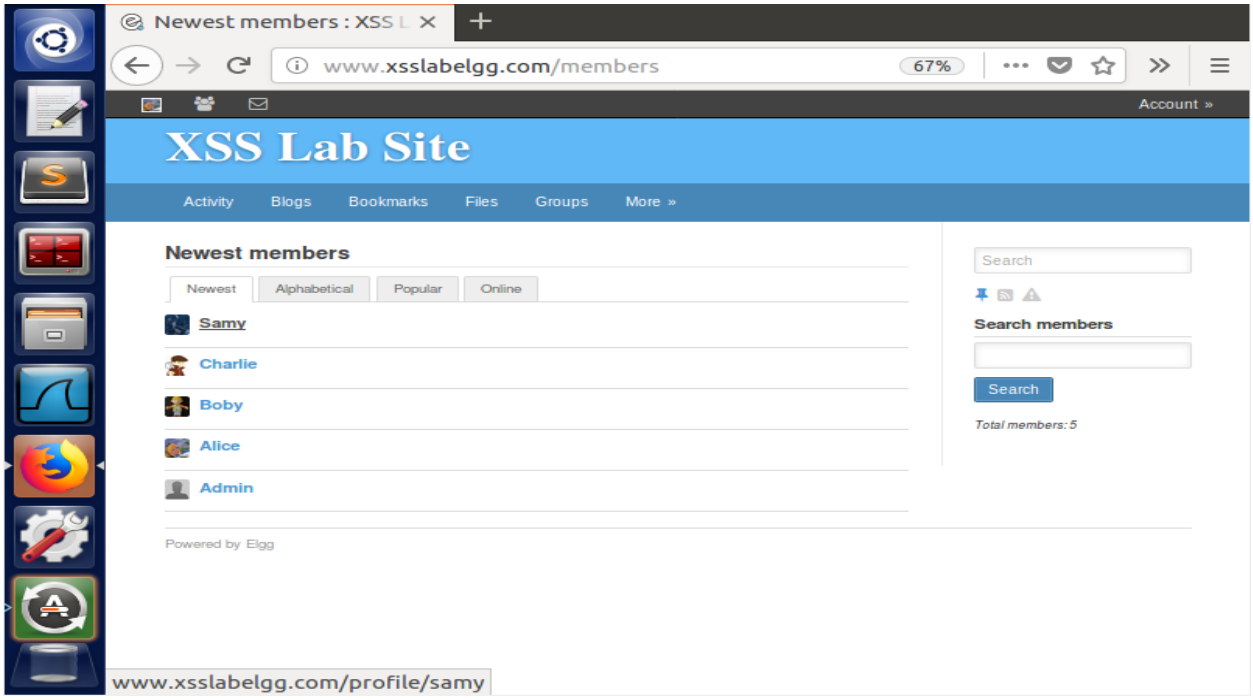
Samy goes and edits his profile to add a brief description field that has a JavaScript code to alert any user who visits his profile. Here the brief description is mall and so we write it directly in this field. Once the brief description is saved, he himself gets an alert that says 'XSS'.



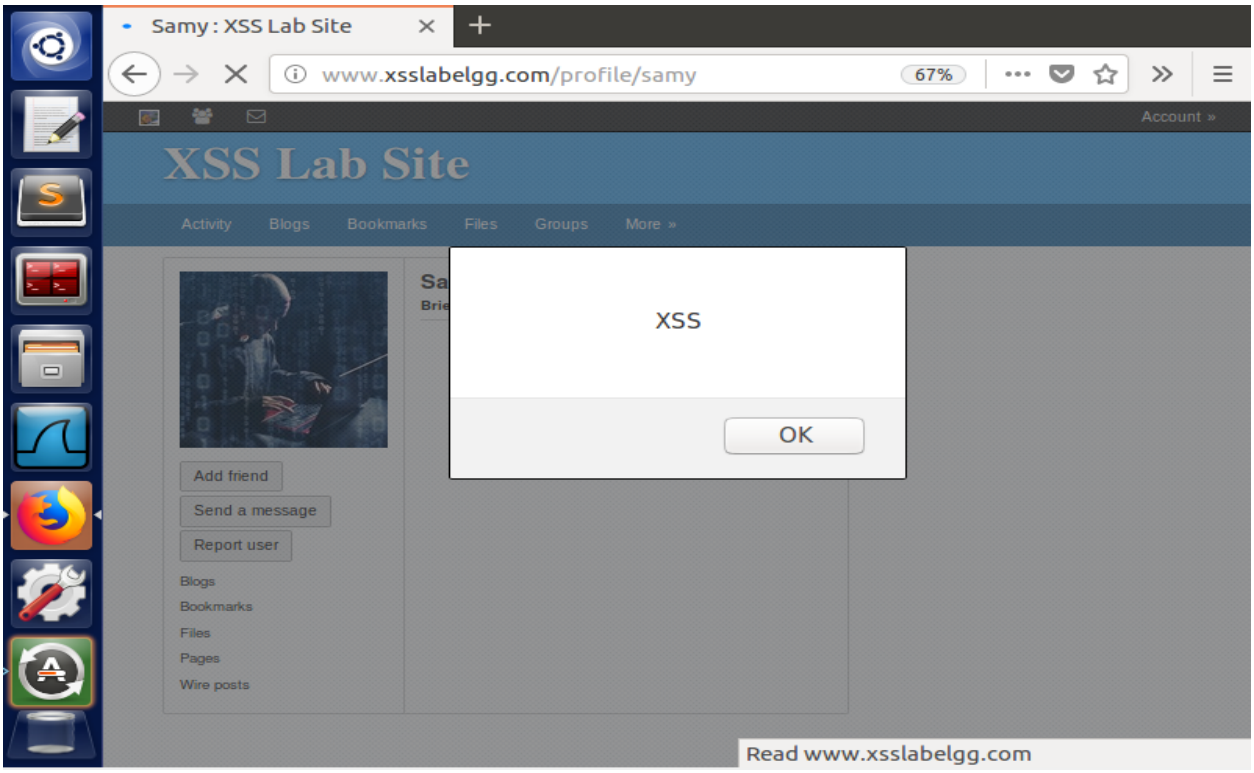


We then go and login as a different user, Alice, for example. Alice goes and checks the members in the Elgg website.





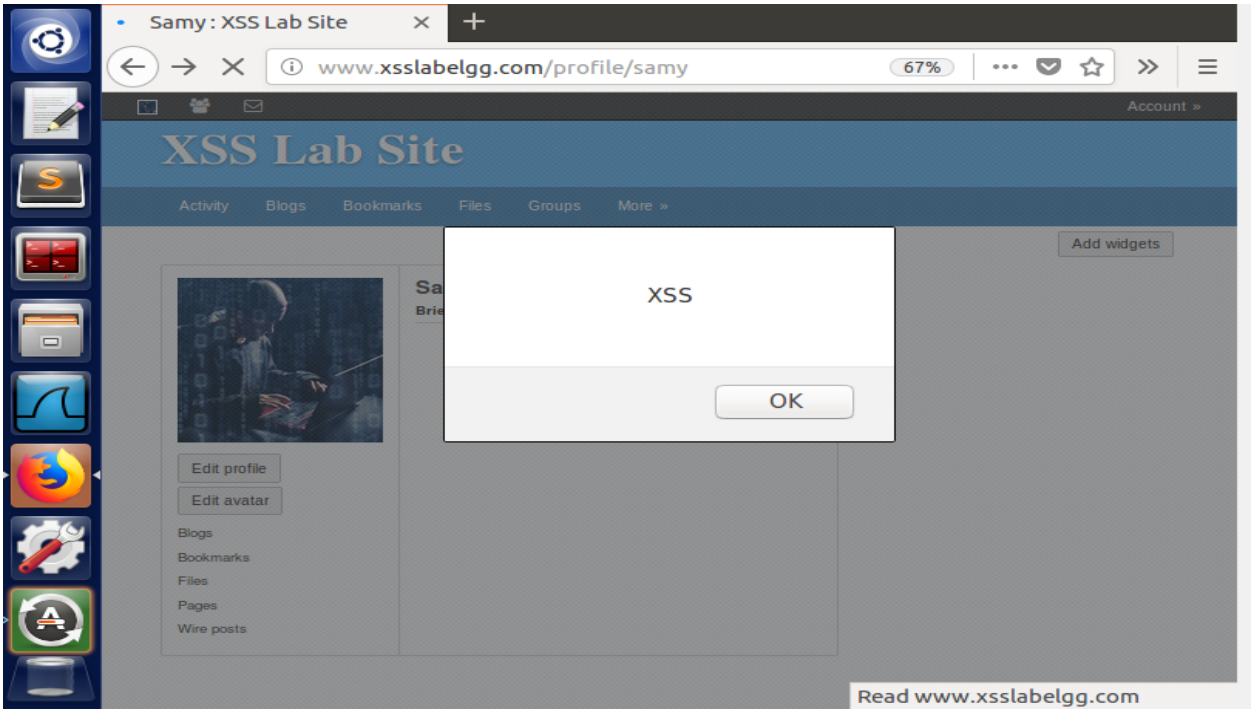
When Alice casually goes and checks the members of the Elgg website, she visits Samy's profile. When she clicks on Samy's profile she gets an alert that says 'XSS'.



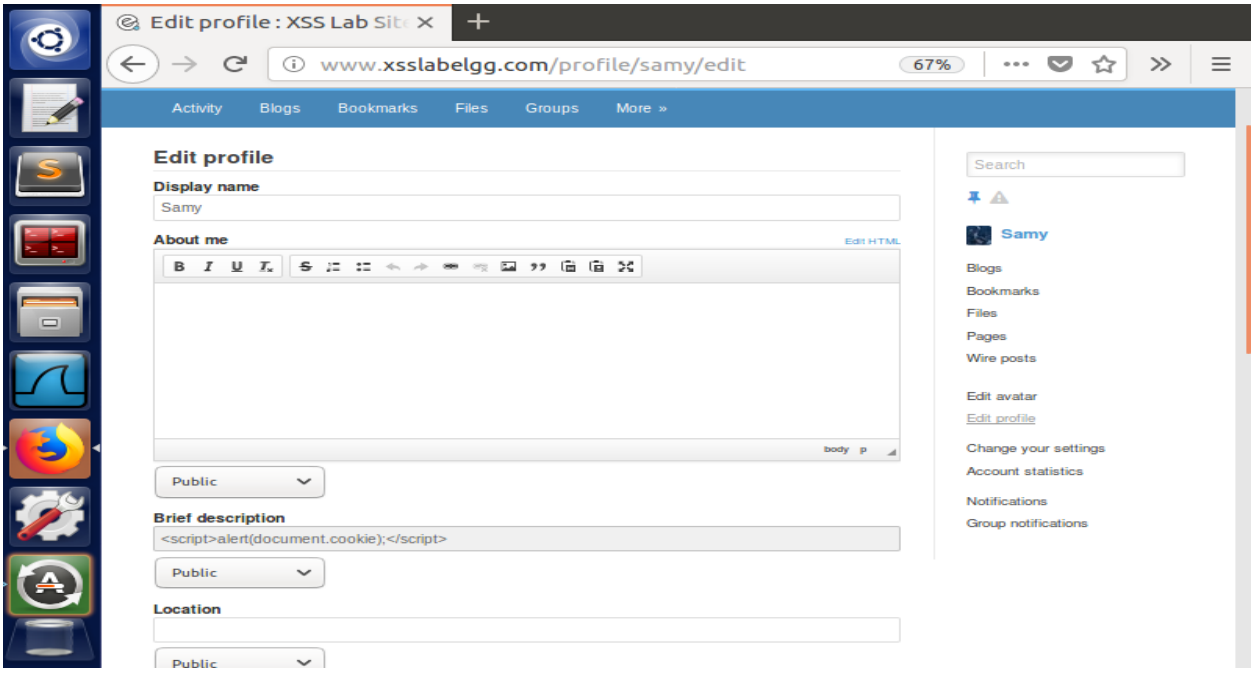
Conclusion: Samy has successfully embedded the JavaScript program in his profile's brief description field, such that any user who visits his profile gets an alert message that says 'XSS', including him.

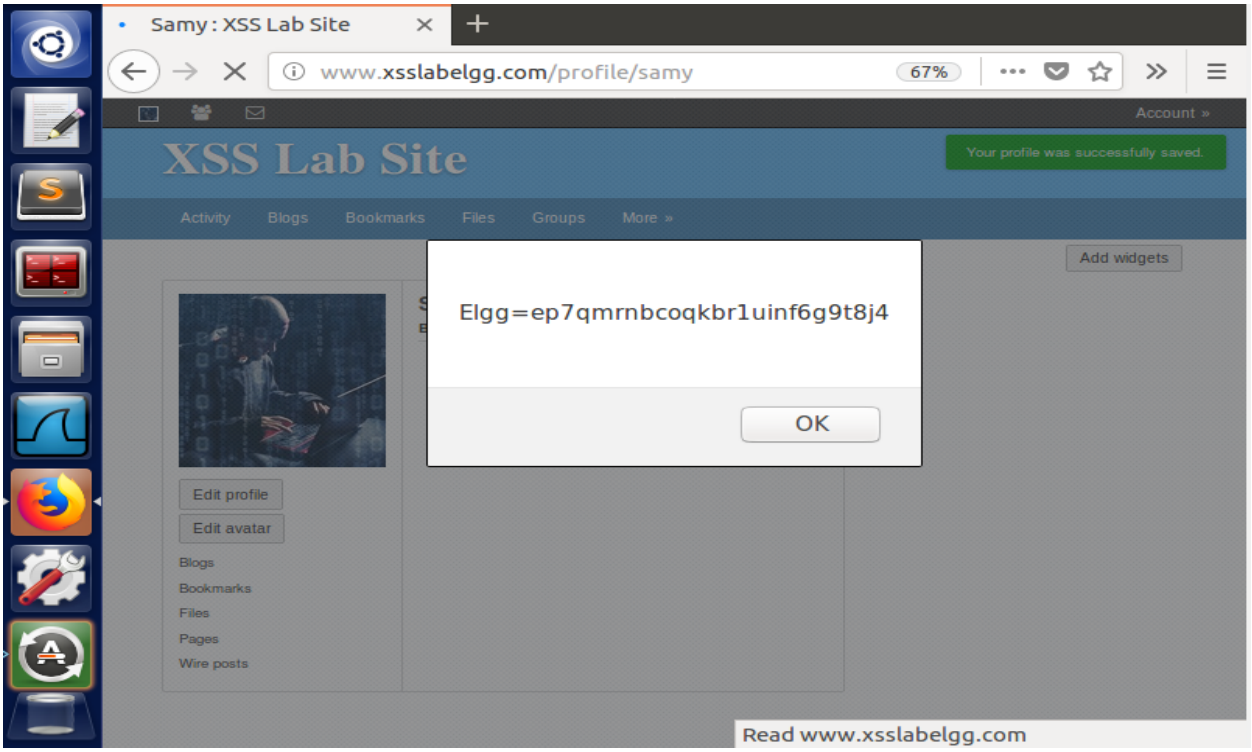
3.3 Task 2: Posting a Malicious Message to Display Cookies

The purpose of this task is to embed a JavaScript in the attacker's profile so that, anyone who comes and visits this profile, the JavaScript will be executed, and an alert window will be displayed that shows the cookies of the person who is viewing the attacker's profile.

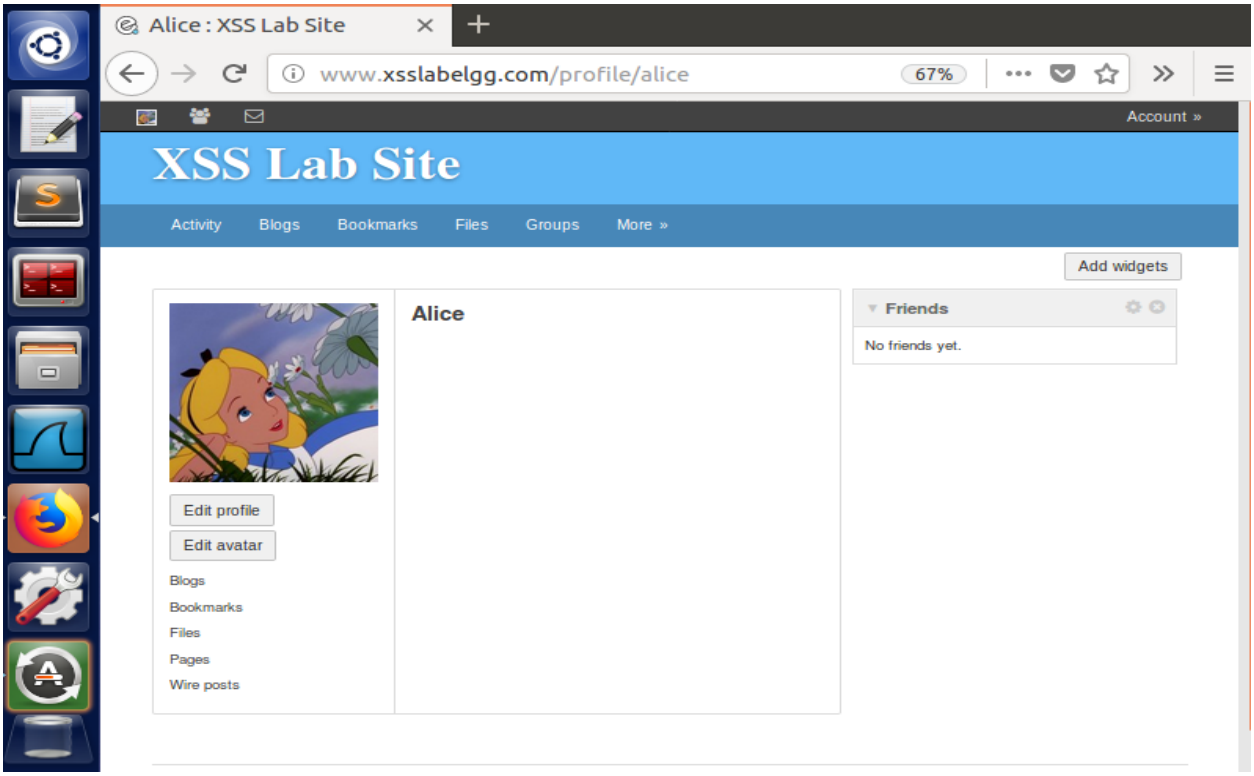


We login to Elgg website as Samy, the attacker. Samy goes and edits his profile to change the brief description to document.cookie.

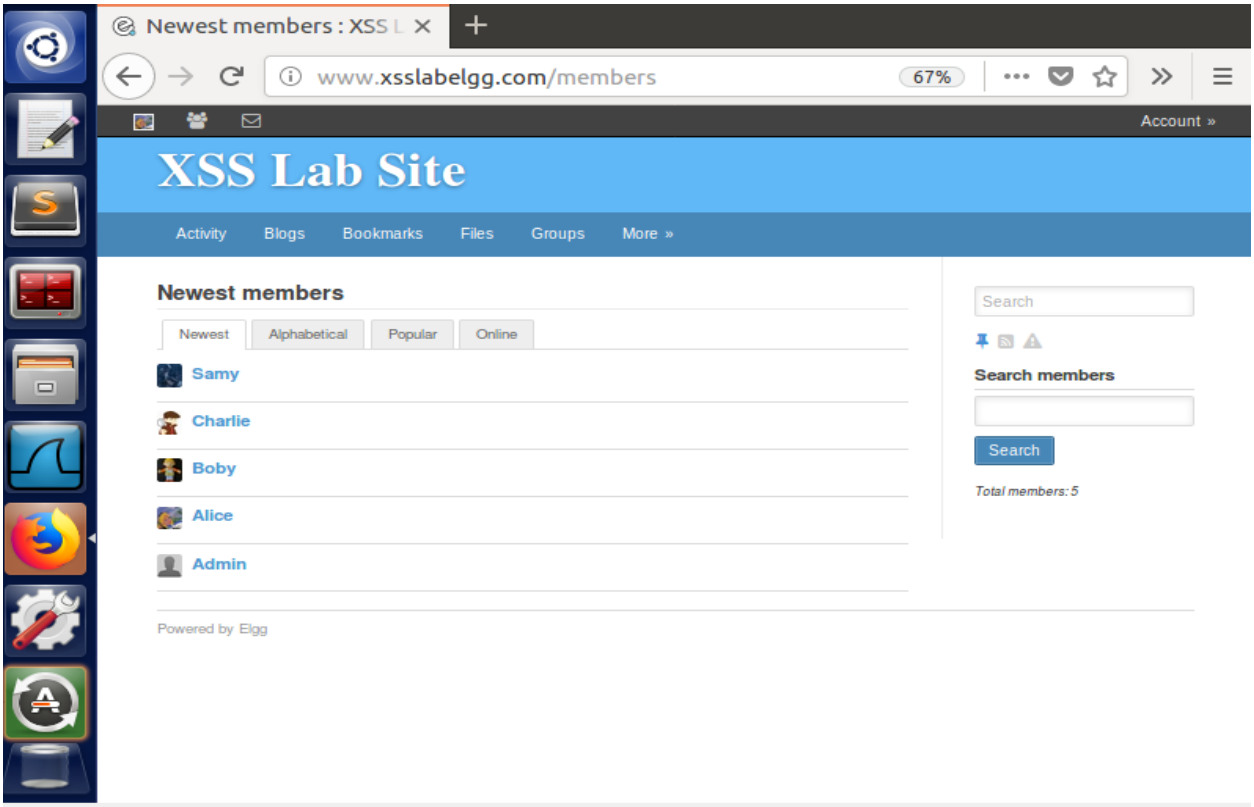




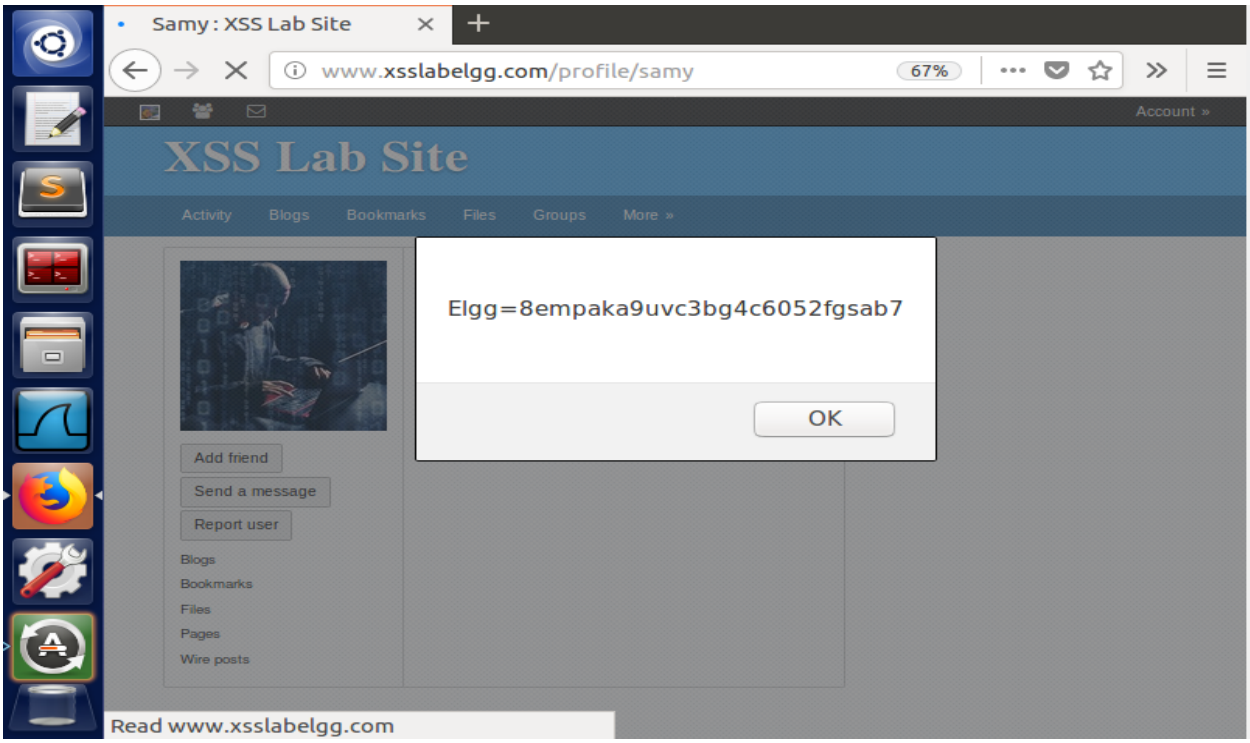
When the profile is saved, Samy himself sees an alert that shows his cookies as a message. We now login as a different user, Alice.



Alice casually goes and checks the members in the Elgg website and checks Samy's profile.



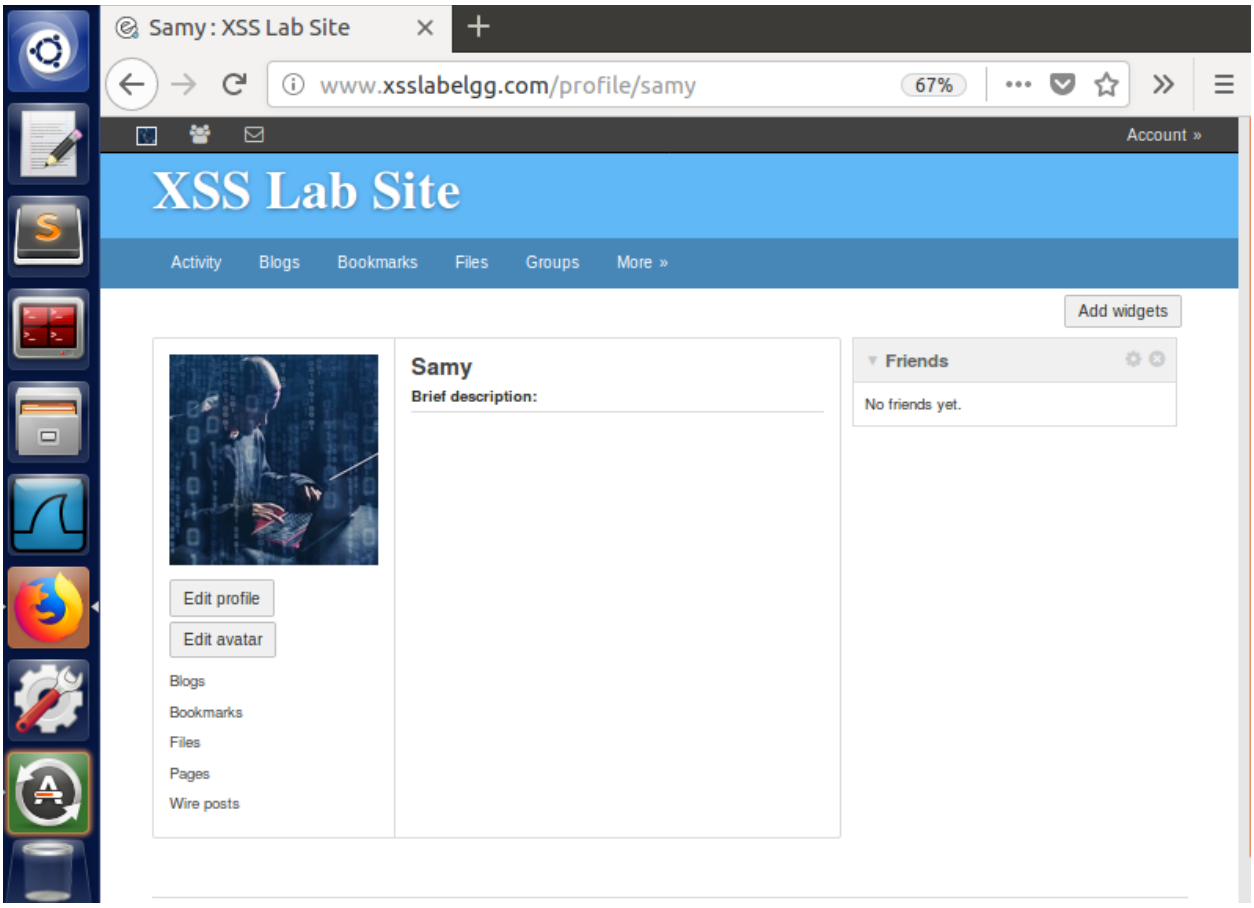
When Alice visits Samy's profile she sees her cookies being displayed as a message in an alert window.



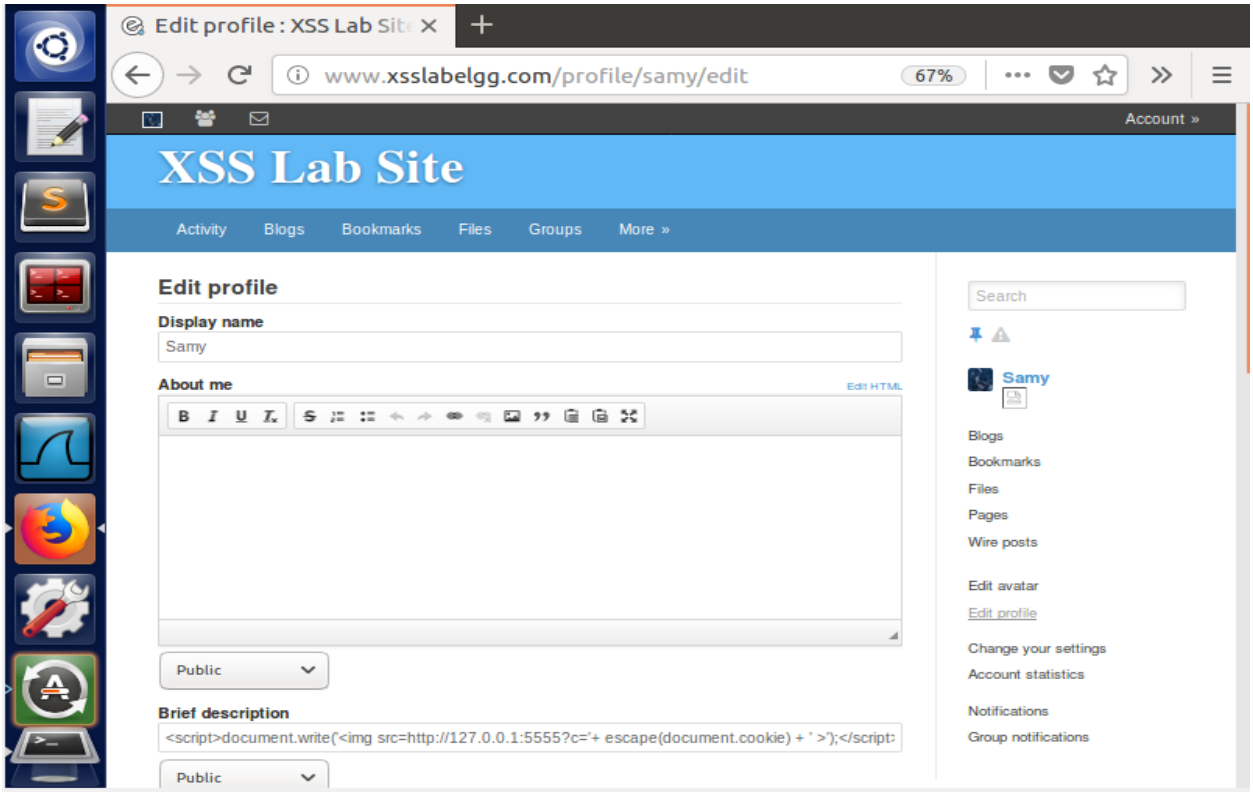
Conclusion: Here Samy uses the JavaScript code and use document.cookie to display any user's cookies created for that particular session. Here Samy doesn't come to know what the cookies of each user are who visits his profile. The cookies are displayed only to the user who visits Samy's profile. In this case, only Alice can see her user cookies for that session. Samy is not able to know Alice's cookies.

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

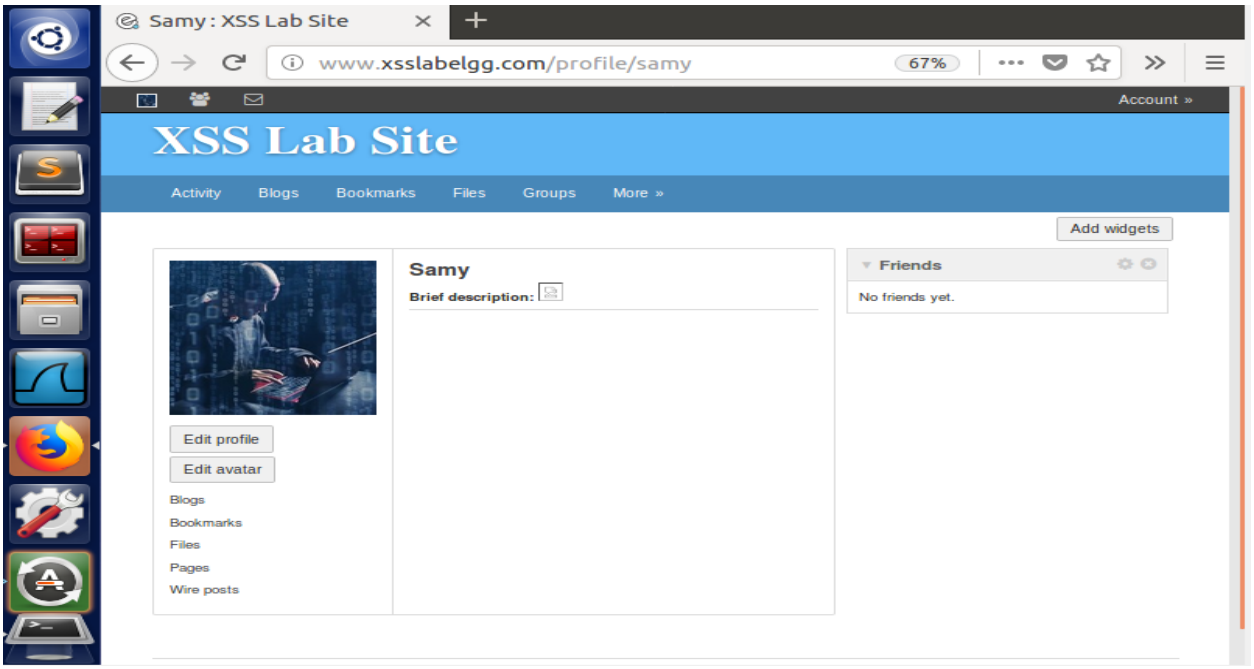
The purpose of this task is to steal the cookies of the victim from the victim's machine and send it to the attacker using a JavaScript code.



We login to Samy;s profile and act as an attacker. Samy edits his profile and changes his brief description. He makes use of the img tag and the src attribute in this tag for the attack. This is sused so that when 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. These results are listened by the attacker using the netcat -l command which makes the terminal into a TCP server listening port.

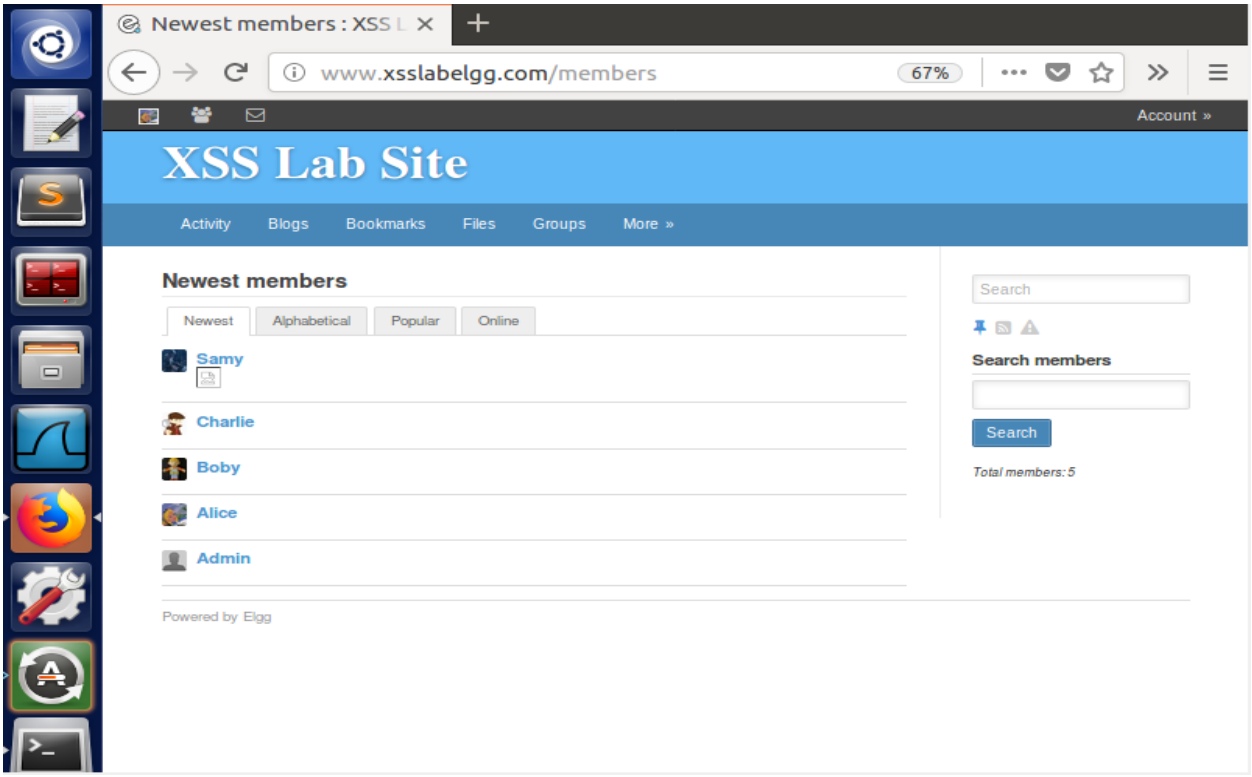


The JavaScript code contains the local host 12.0.0.1 which is the attacker's IP address. Here, document.cookie is used to get the victim user's cookies. Once the JavaScript is saved in the brief description, it appears like a JavaScript image as shown in the screenshot.



Samy then listens from a terminal that acts as a TCP server listening to the port 5555 by using the netcat -l command. Samy is just waiting for any user to just visit his profile while keeps listening in order to get the victim's cookies.

We then login as Alice. Alice casually goes and checks the members page of Elgg website.

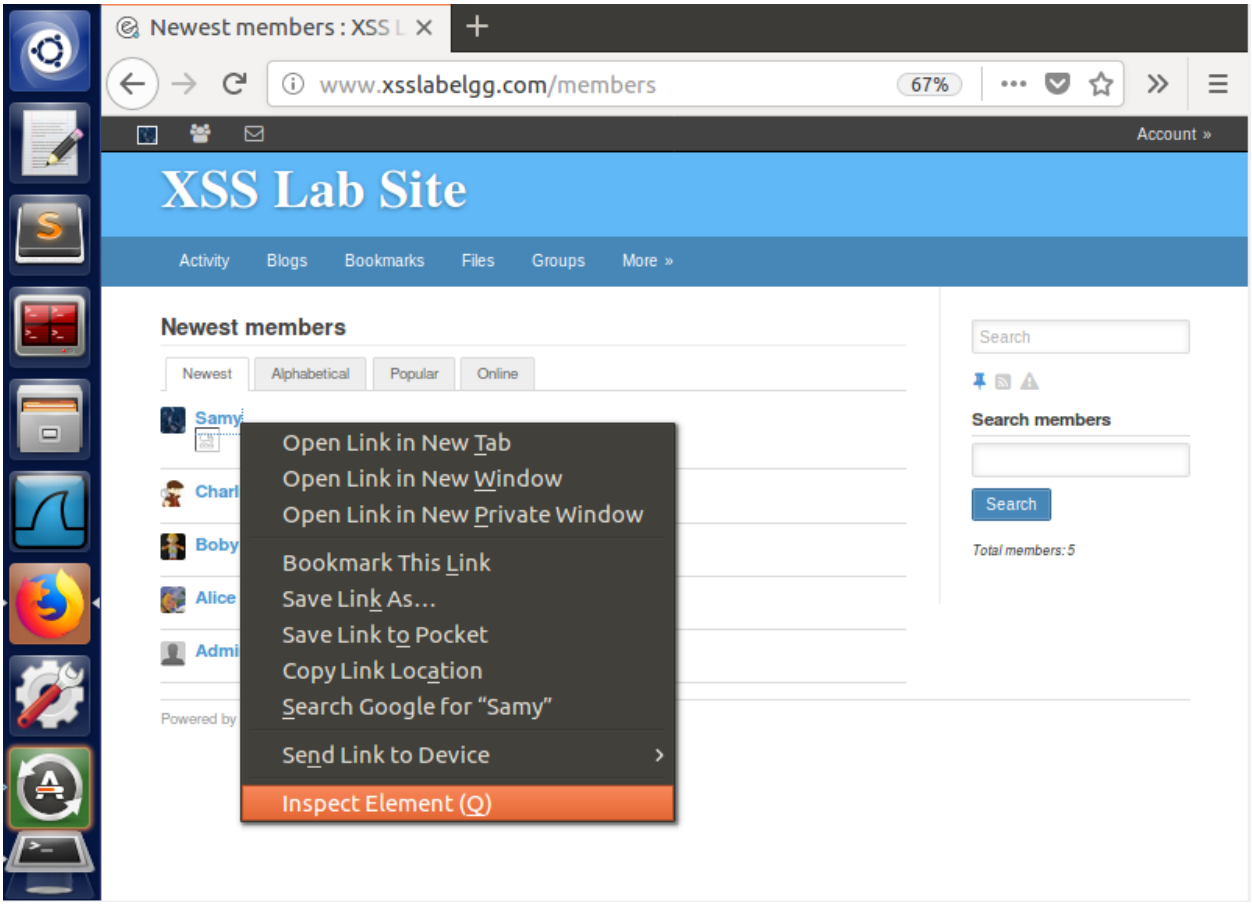


Once Alice visits Samy's profile, Samy gets her session cookies through the listening port 5555 through the HTTP GET request.

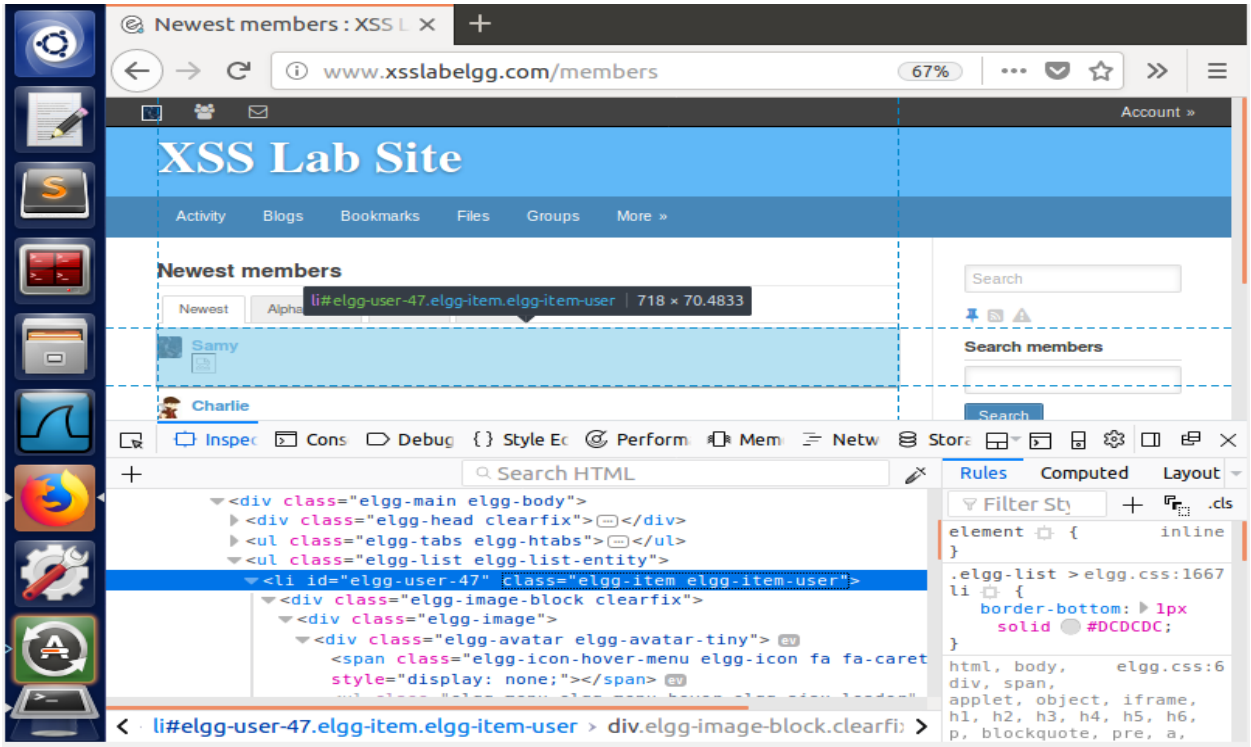
Conclusion: Samy wanted to get ny victim user's active session cookies. So he embedded an JavaScript code in this profile that gets the victim's cookies and sends it to the attacker's IP address port. He sets up a listening port 5555 from a terminal in his machine which acts as a TCP server and keeps on listening to this port so that whenever any user visits his profile, he can get their cookies. When Alice visits Samy's profile, her active session's cookies are sent to Samy.

3.5 Task 4: Becoming the Victim's Friend

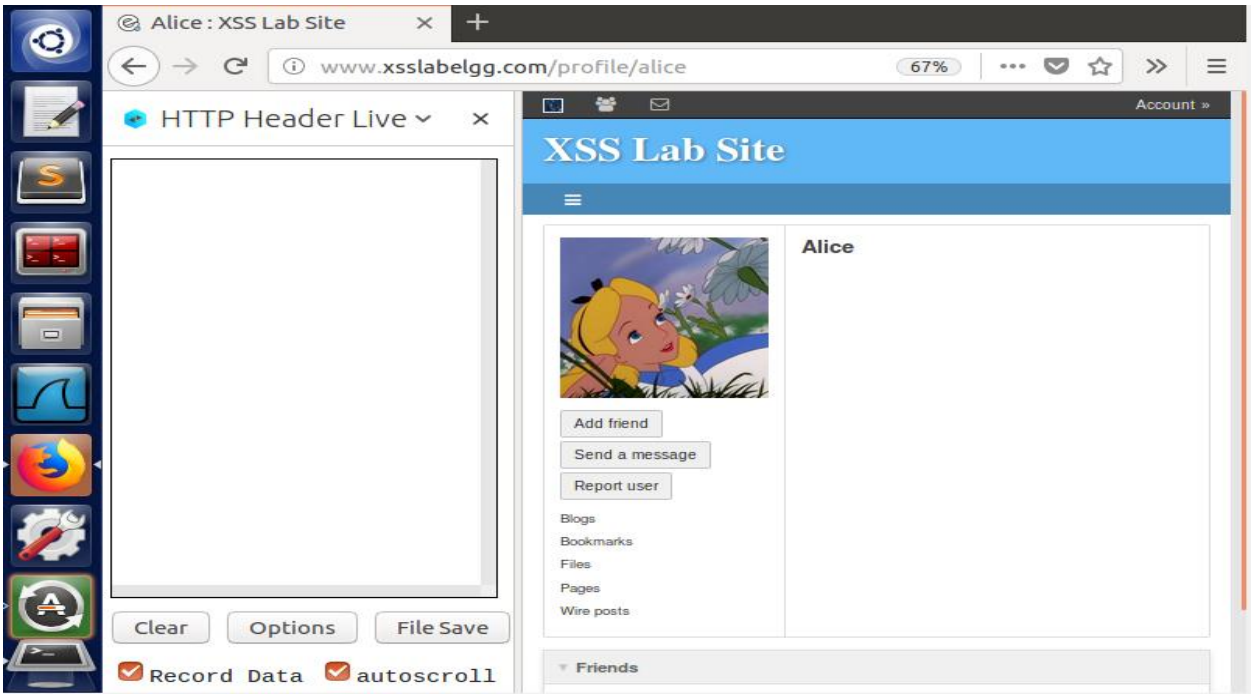
The purpose of this task is to add an XSS worm to the attacker's profile that adds the attacker as a friend to any victim who visits the attacker's profile. This worm does not self-propogate. Here, the attacker is Samy. We login as Samy.

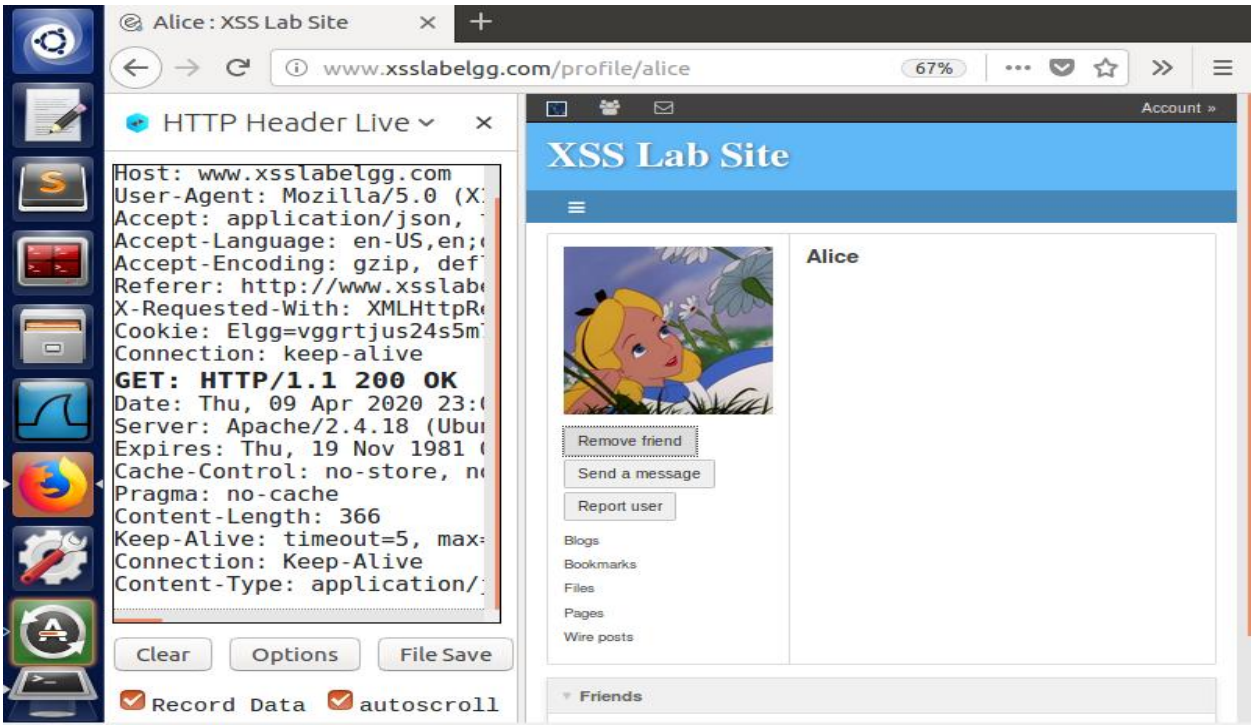


Samy goes to the Elgg website members page www.xsslabelgg.com/members. He makes use of the Inspect Element feature in the Firefox browser to find out his user id/user number so that he can use that in the XSS worm code.

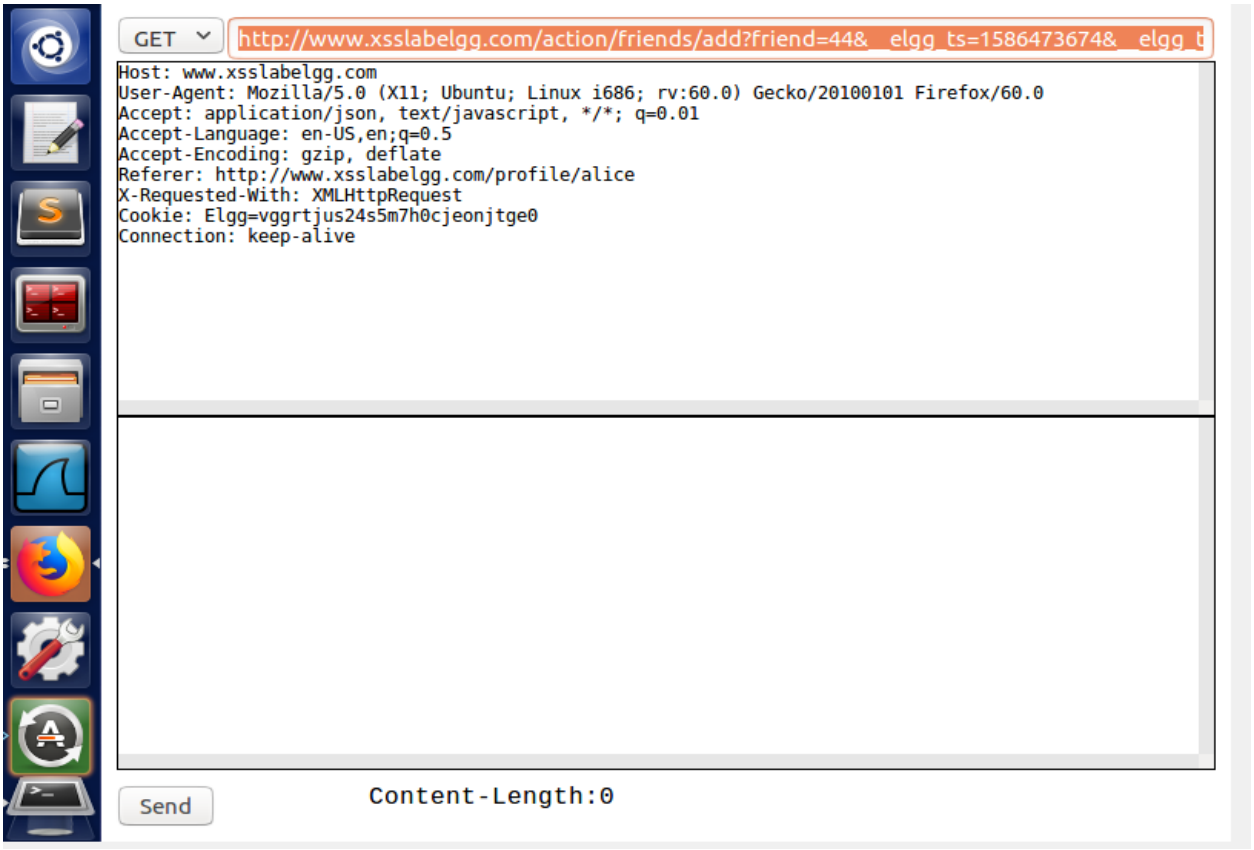


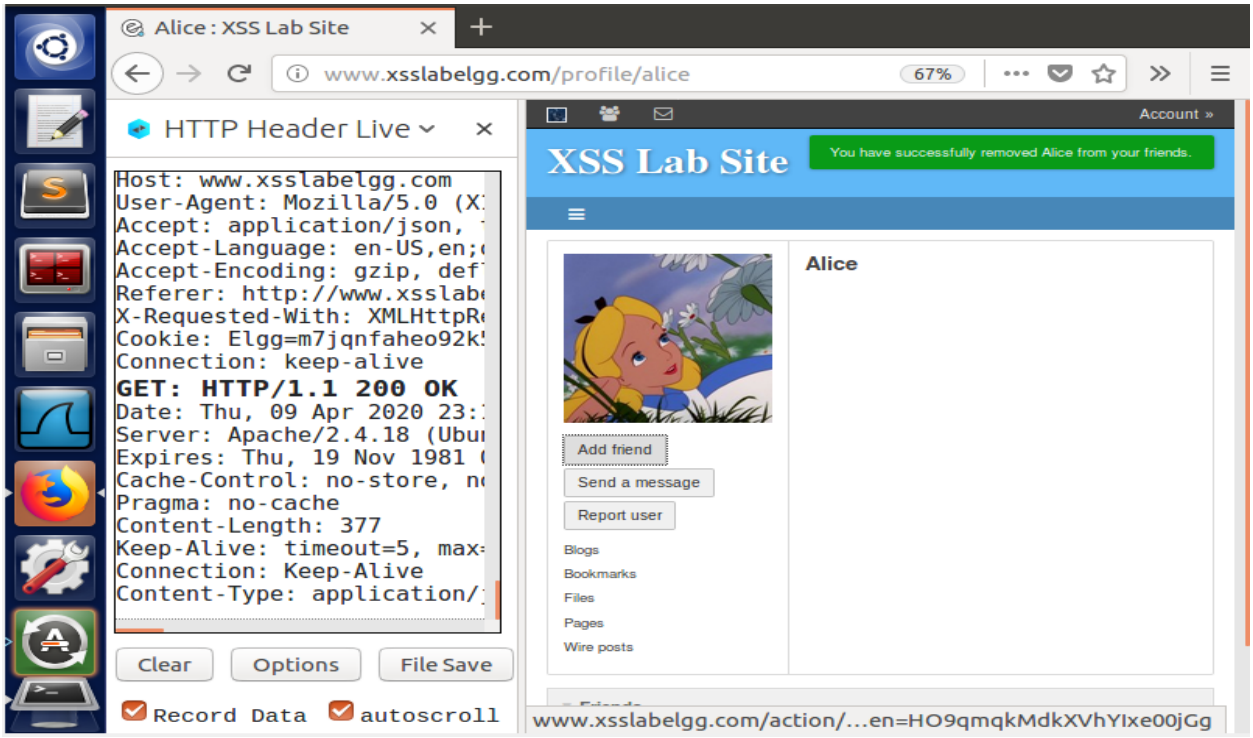
Samy finds out that his user id is 47. He then tries to find out the exact GET URL used to add him as a user. For this he uses the HTTP Header Live Add on and goes to Alice's profile.



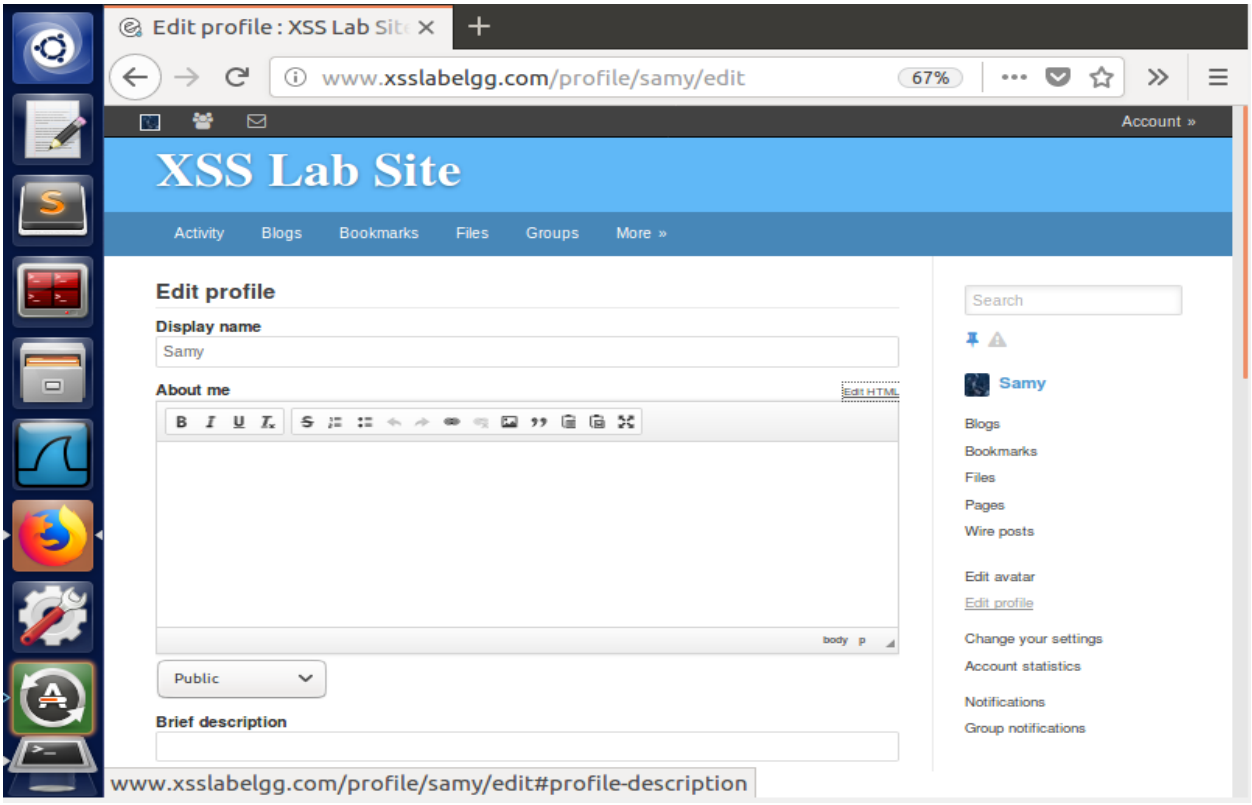


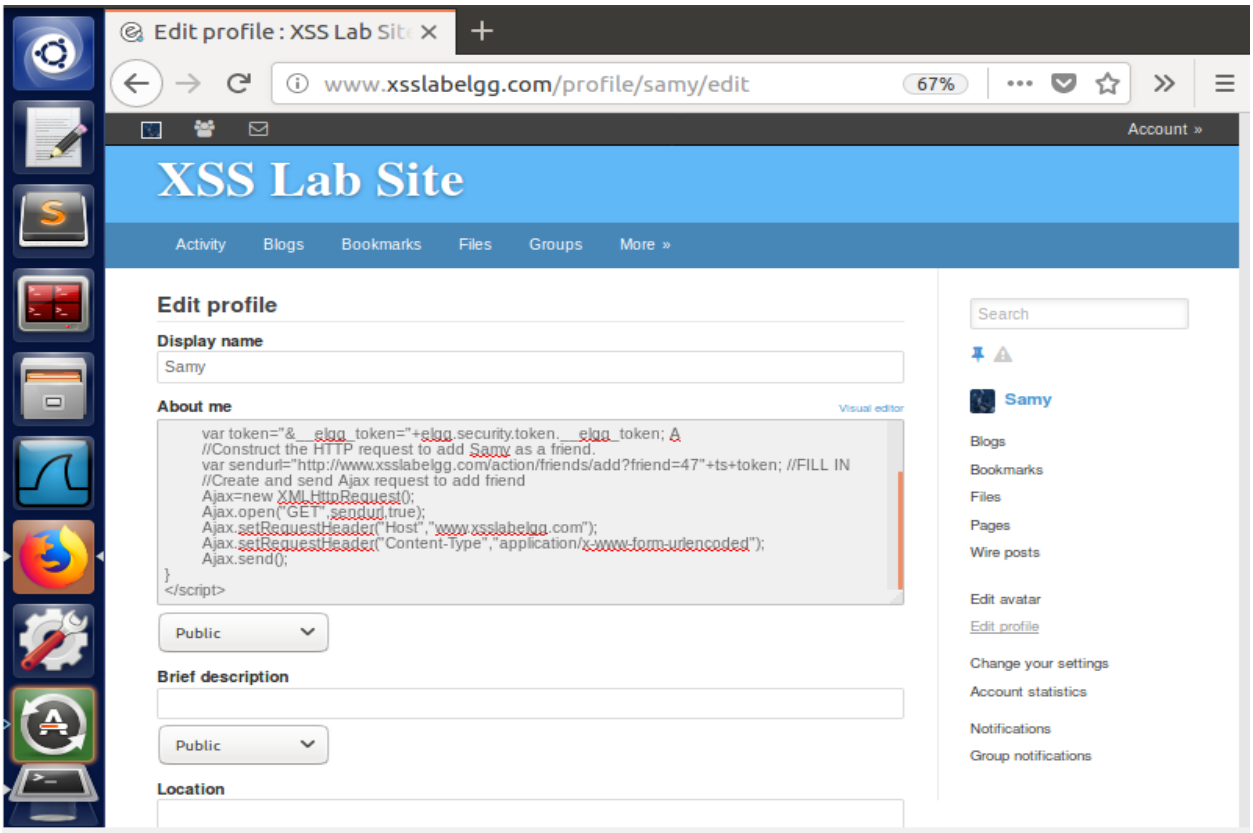
Samy adds Alice as his friend and finds out the HTTP Get request URL from the HTTP Header Live add on. He gets the exact URL and its parameters.



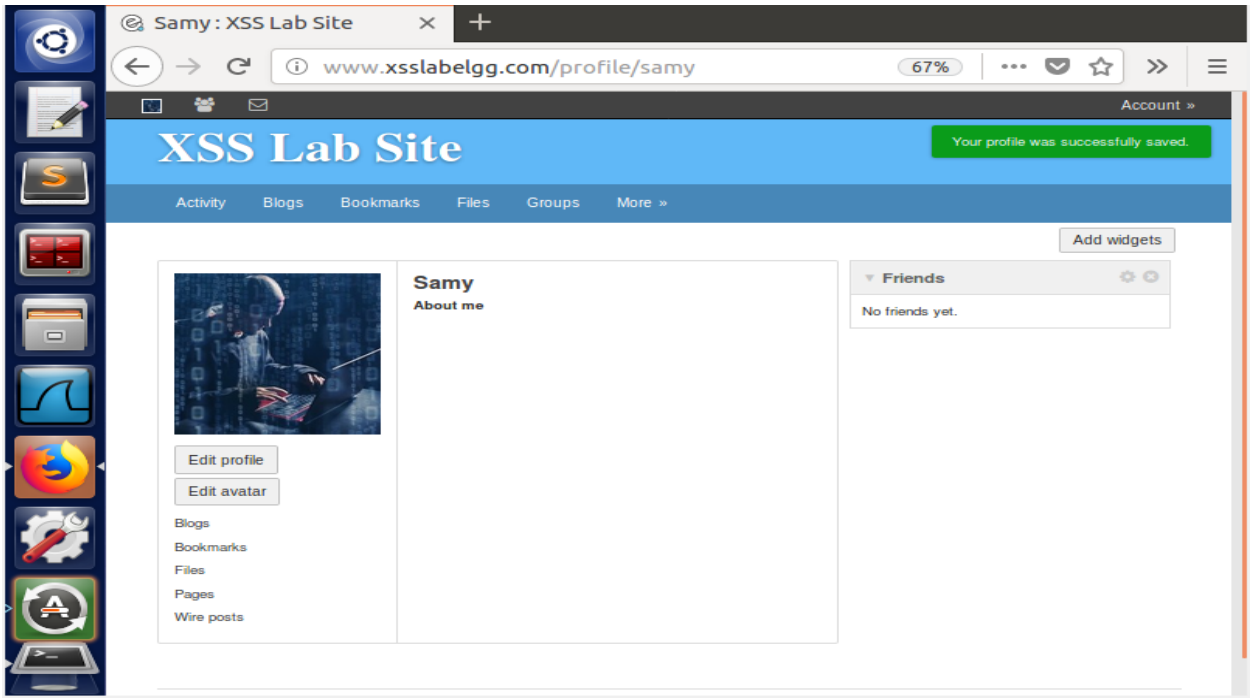


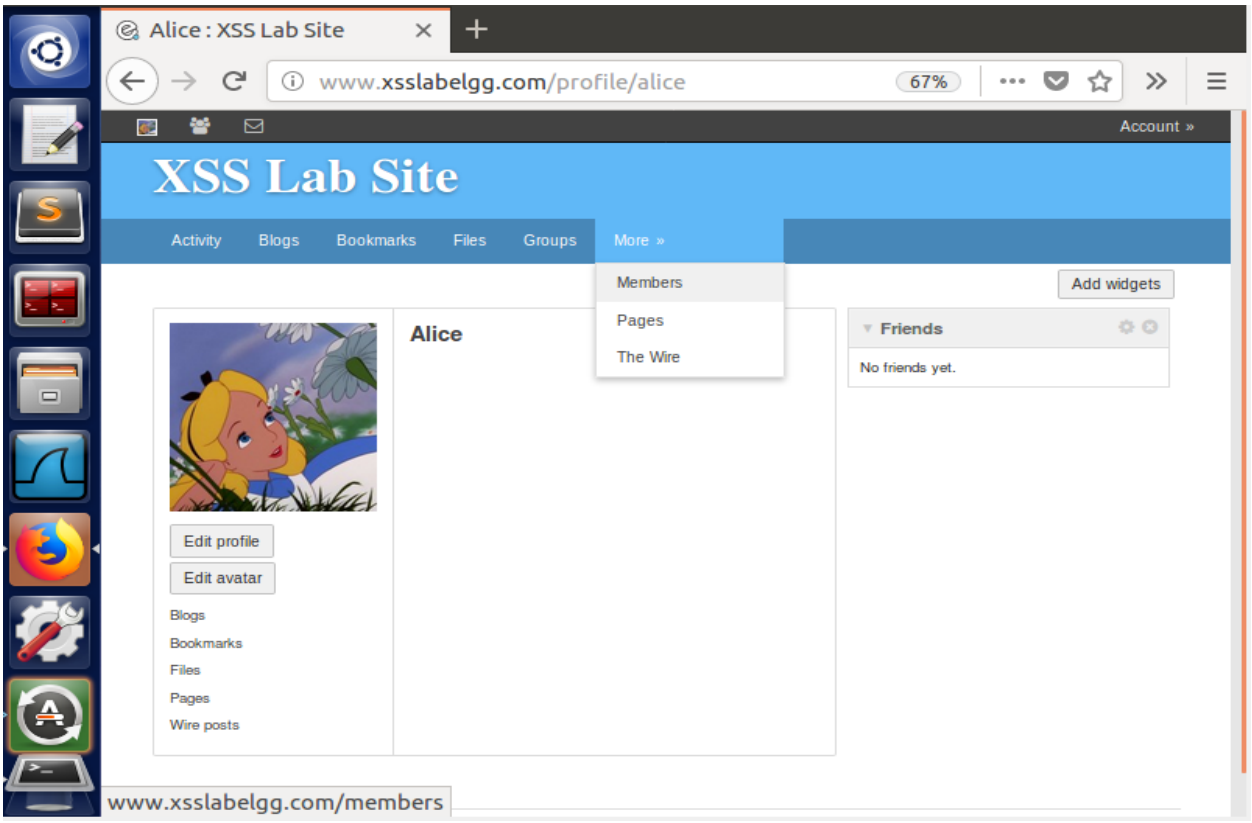
Samy then removes Alice as his friend in order to perform the attack on Alice unknowingly.



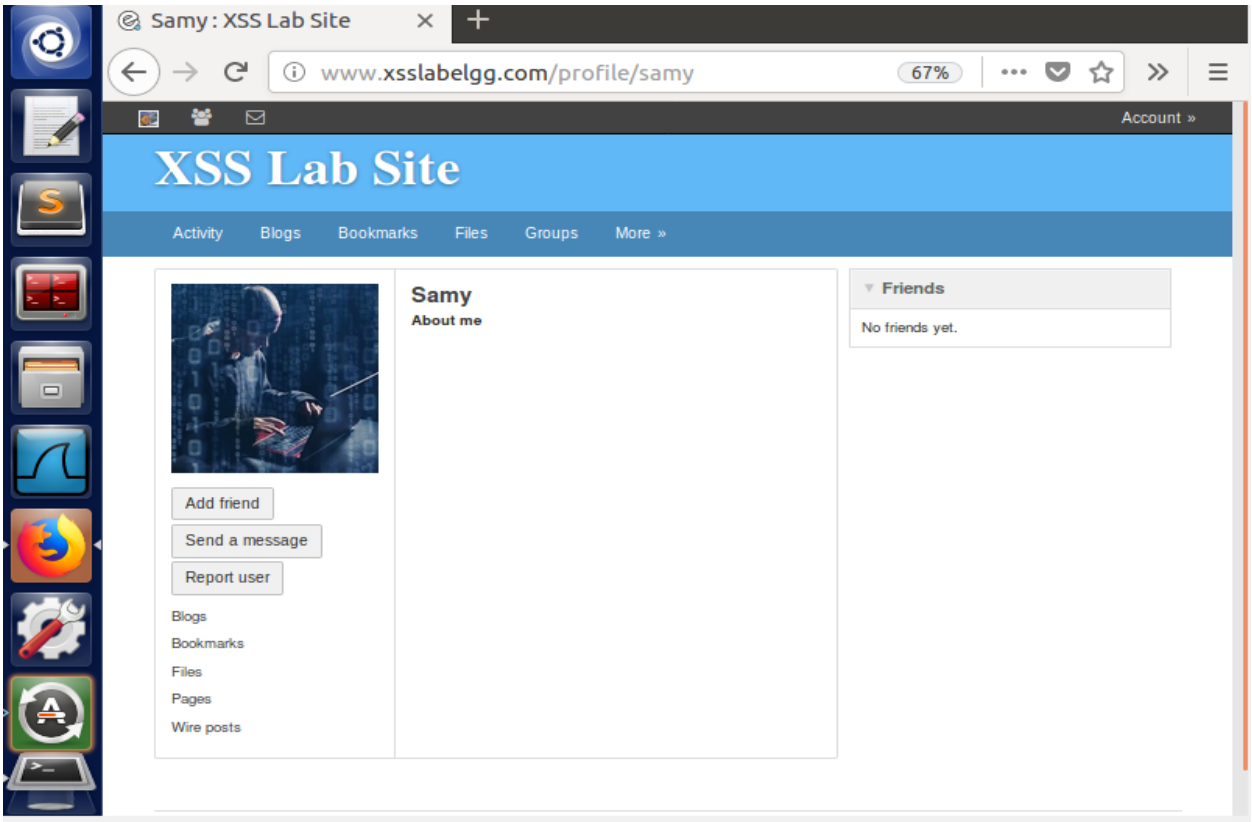


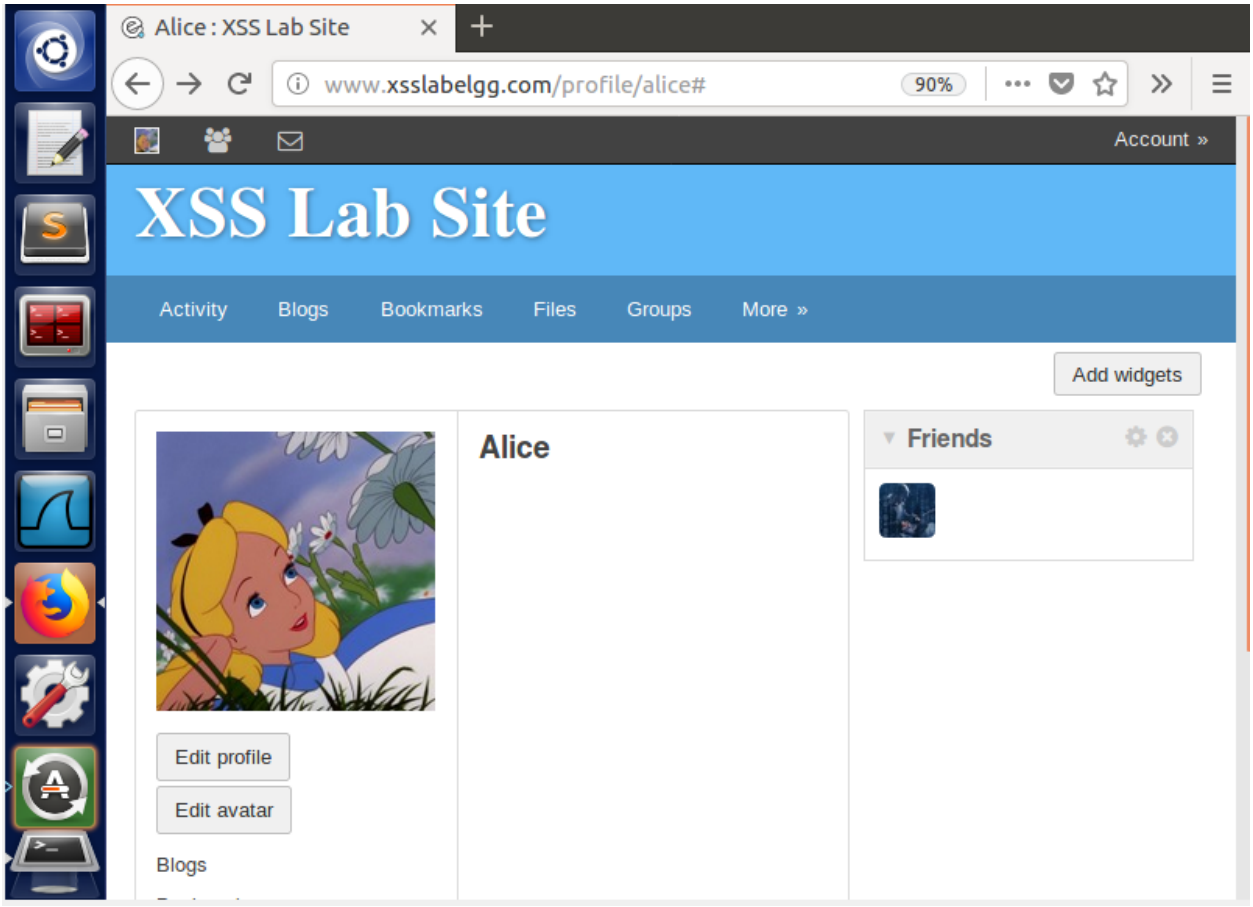
Samy logs in to his profile and use the 'Edit HTML' feature in the 'About Me' section. He writes a code that uses the HTTP GET request URL and adds his user id as the parameter and puts a JavaScript code there and saves his profile.





We login as Alice. Alice casually goes and checks the members in the Elgg website and visits Samy's profile.



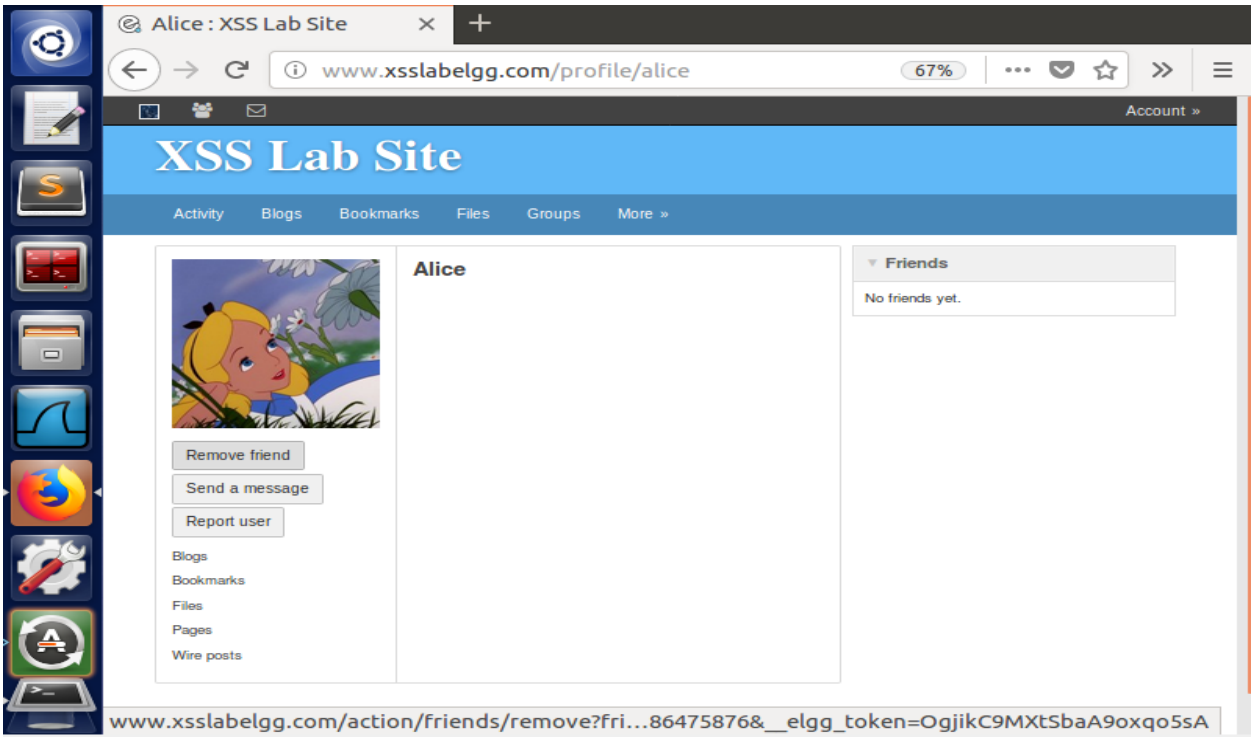


When Alice goes back and checks her profile, she sees that Samy is already added to her friend list without her knowledge. She never accepted Samy's friend request. But somehow, Samy is added to Alice's friends list.

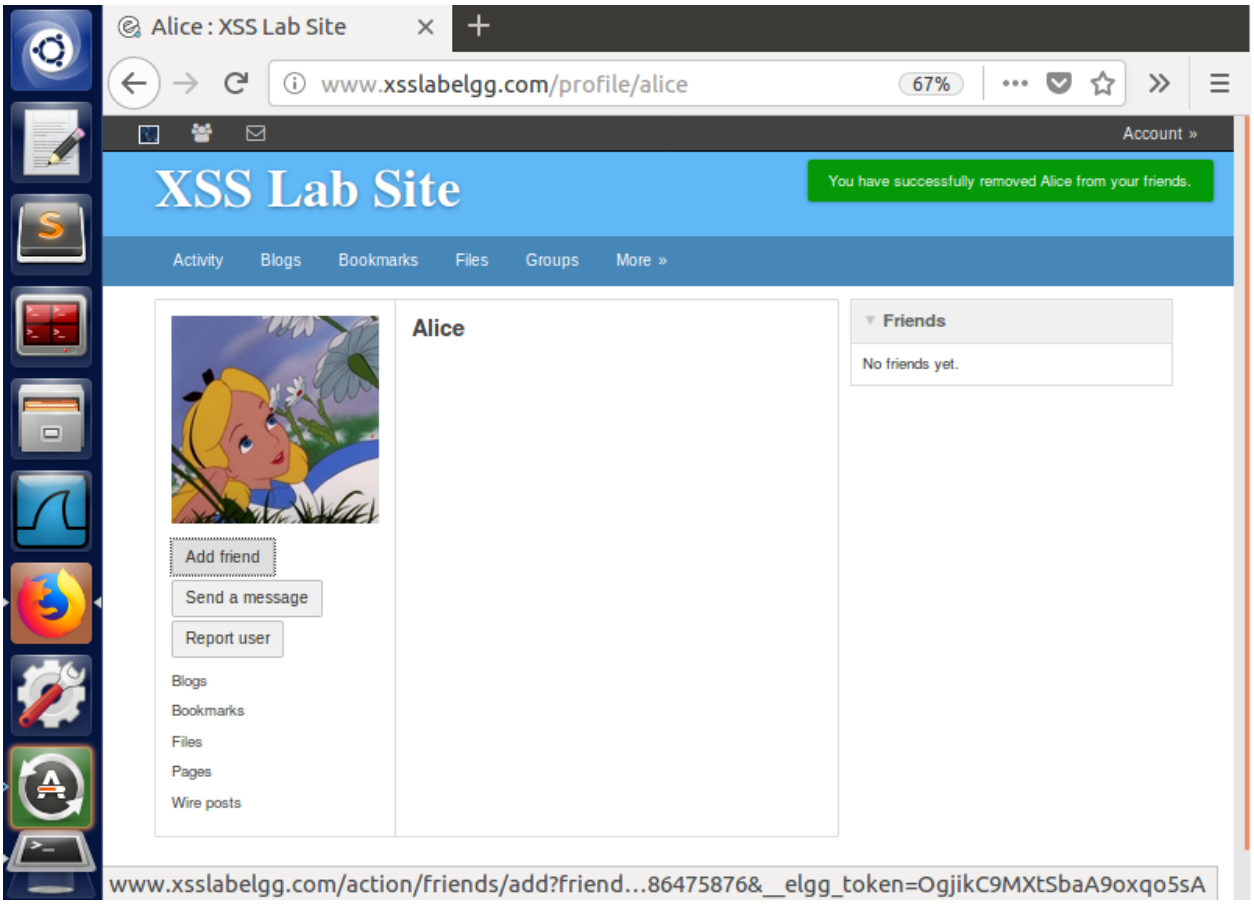
Conclusion: Samy wants him to be added as a friend to anyone who visits his profile without him sending the actual friend request and wants to be in the victim user's friends list without the victim's knowledge. Samy finds out the HTTP GET request URL used to add him as a friend and uses its parameters in the JavaScript code and adds them in his profile's 'About me' in order to add himself to the victim's friends list. This XSS worm is not self-propagating. This means that only if the victim goes and visits Samy's profile, he gets added in the friend list. The XSS worm does not spread to anyone else.

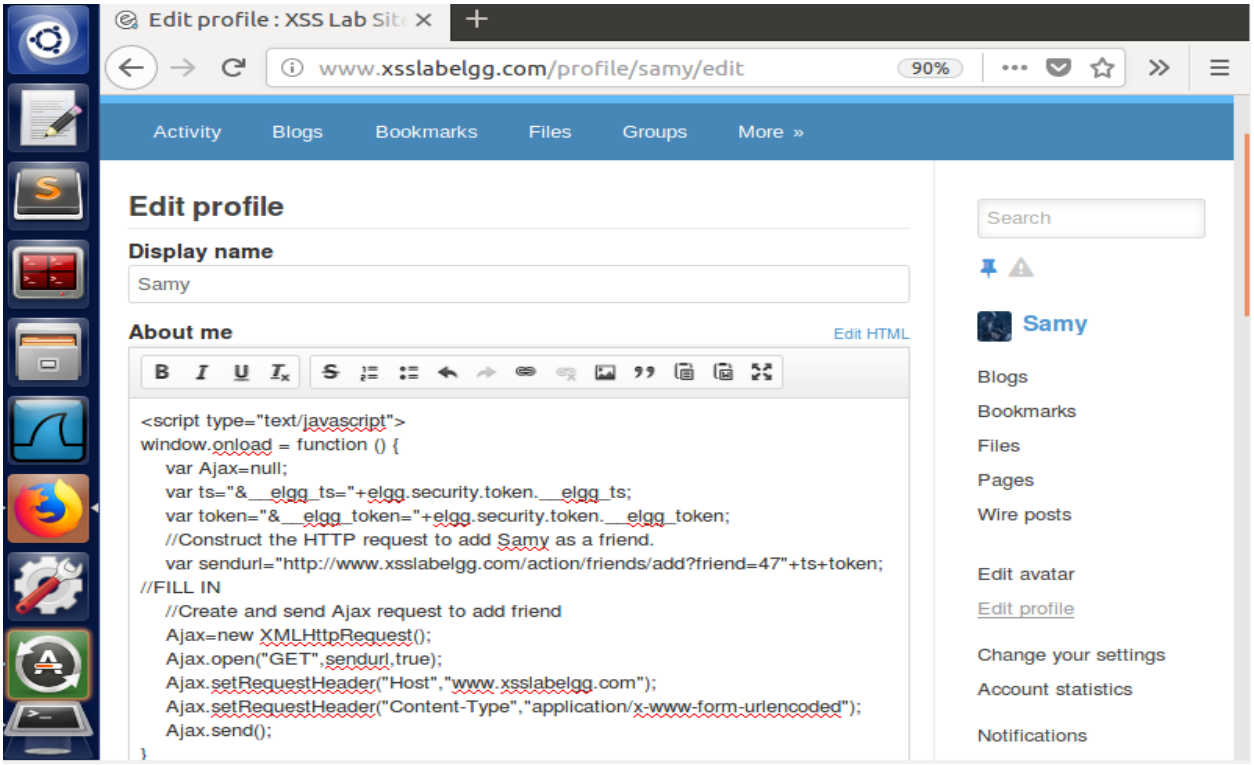
Answer to Question 1: Lines 1 and 2 are elgg_ts i.e. timestamp and elgg_token, the secret token uniquely created for each user. These two values are uniquely created for each active session of each user. Even though it is a forged request, we have written JavaScript code to get the active user elgg_ts and elgg_token values. We use these two secret values to get access to the user's current active session. Since we get the active user's session details and these two secret values, the validation gets through. By doing this we will be able to add Samy as a friend to anyone who visits his profile.

Answer to Question 2: The JavaScript code should be written in the 'Edit HTML' field of 'About me' section. If we write in the Text field or the 'Visual Editor', we will not be able to perform the attack. Demo is shown in the below screenshots.

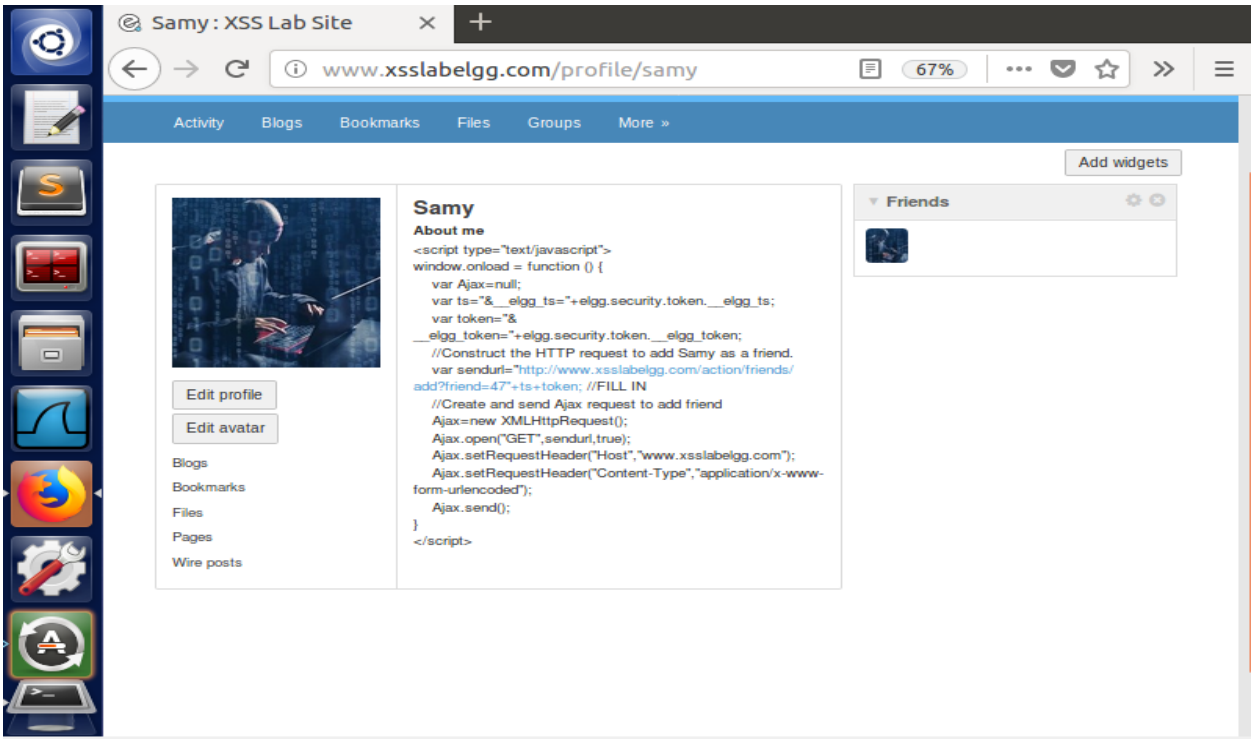


We first login as Samy and remove Alice as friend so that Samy can perform the attack again on Alice.

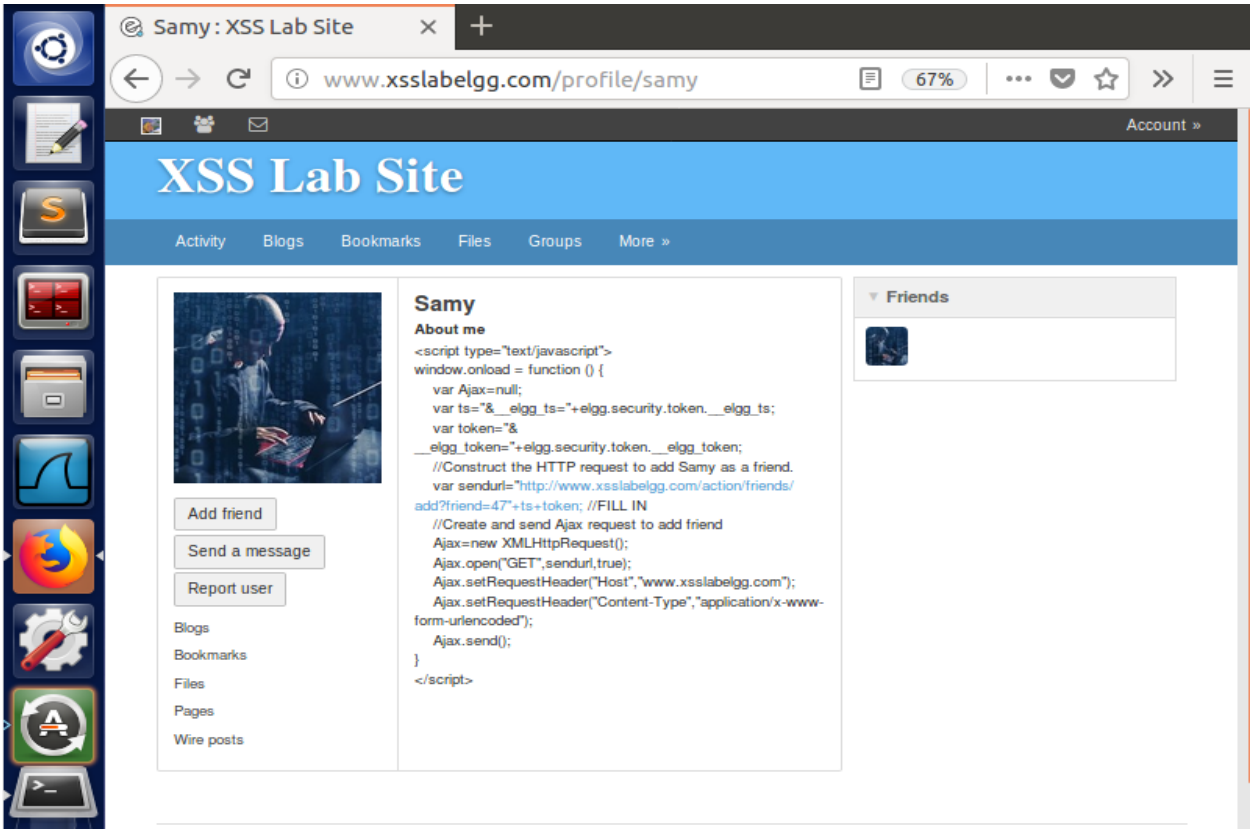


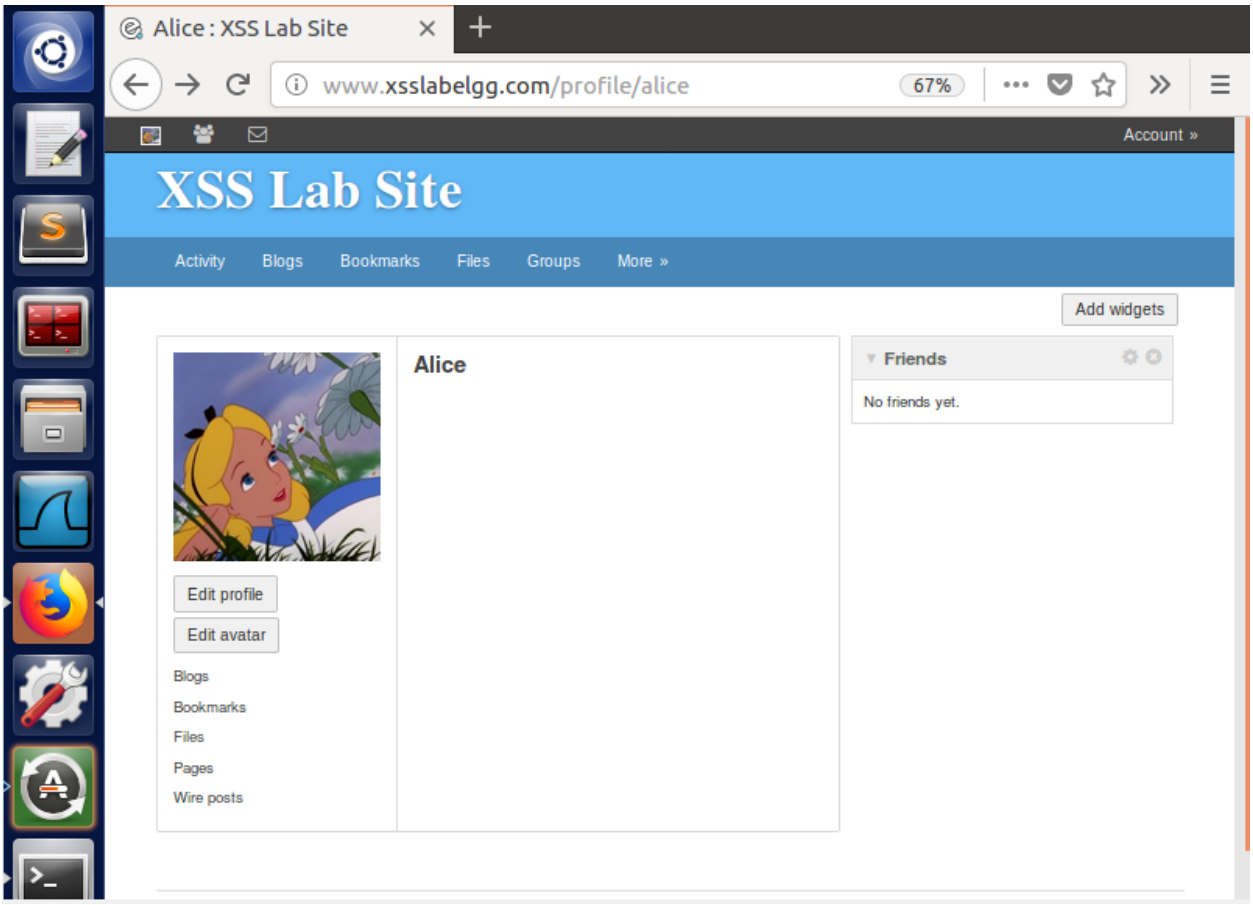


Samy, then writes the JavaScript code in the Visual Editor of the 'About me' section and saves his profile.



We then login as Alice. Alice casually goes and checks the members page of the Elgg website and visits Samy's profile.



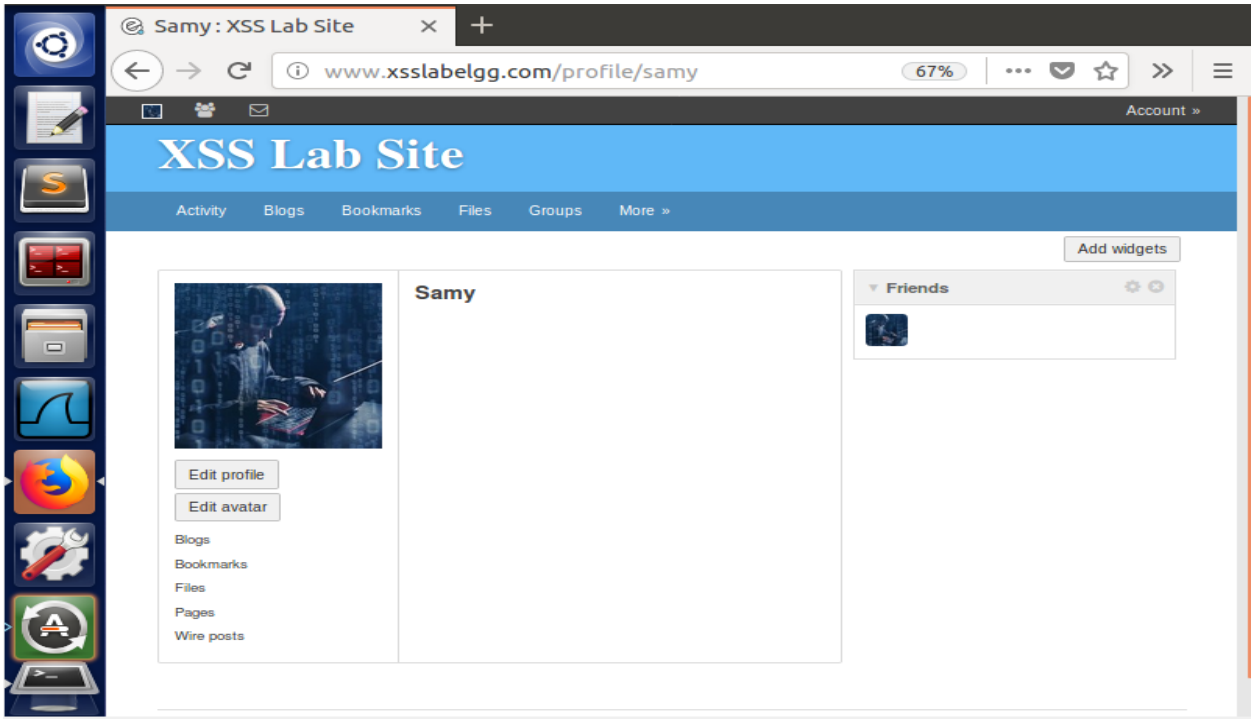


Alice, then returns to her profile and checks that her profile is unchanged.

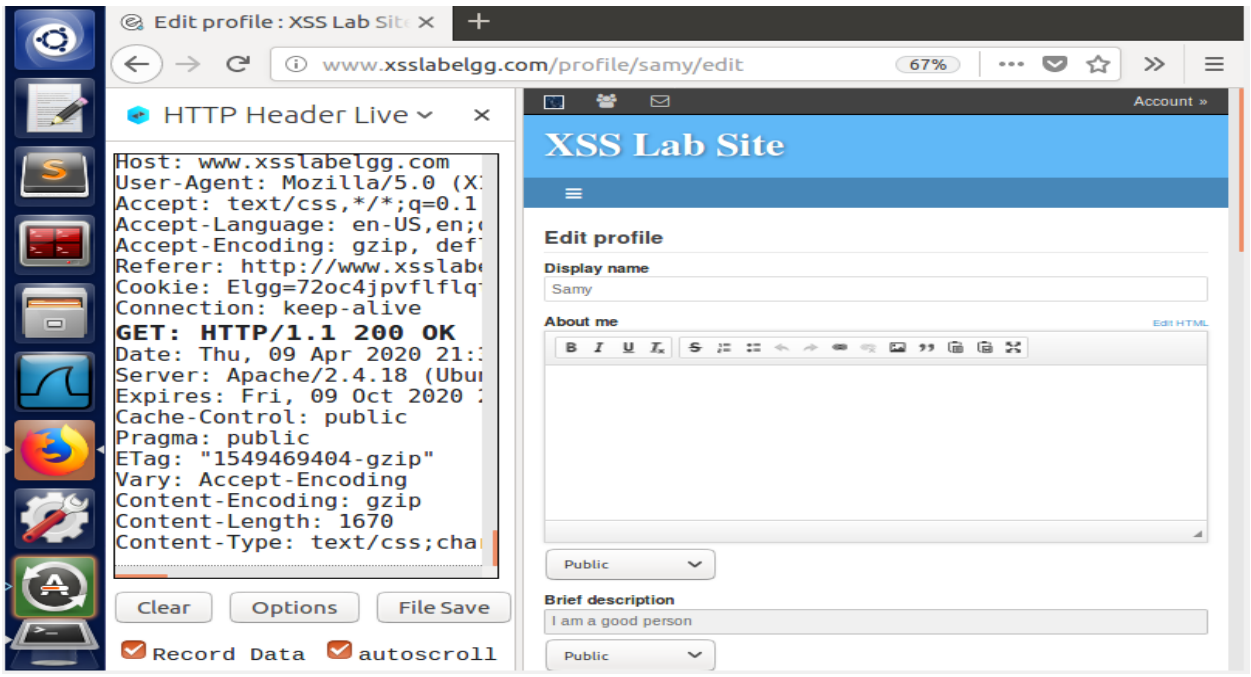
Conclusion: The JavaScript code should be written in the 'Edit HTML' field of 'About me' section so that the JavaScript gets added to the actual HTML code of that profile page. IF we add it in the 'Visual Editor' we will see that it will be added as a normal text like when a user describes about himself/herself. It acts as a normal profile rather than an attacker.

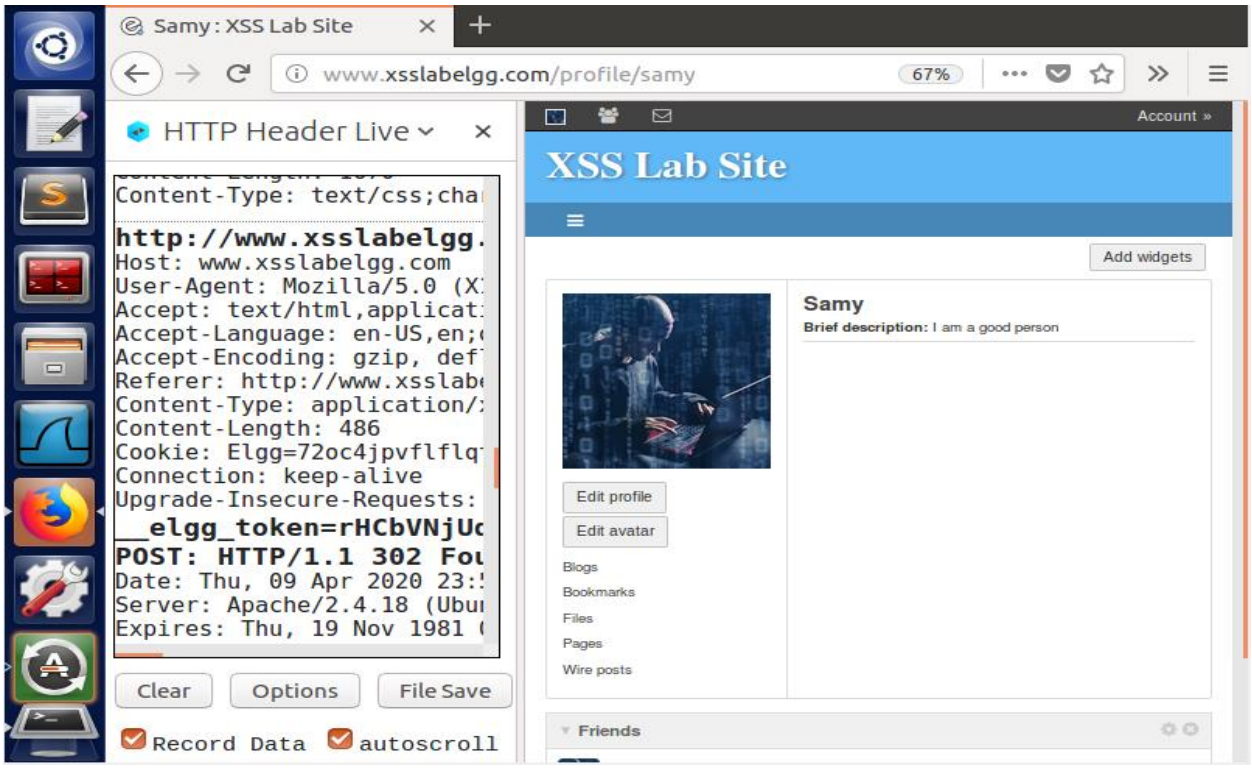
3.6 Task 5: Modifying the Victim's Profile

The purpose of this task is to modify the victim's profile when the victim visits attacker's page. We will write an XSS worm that will do this. This worm does not self-propagate.

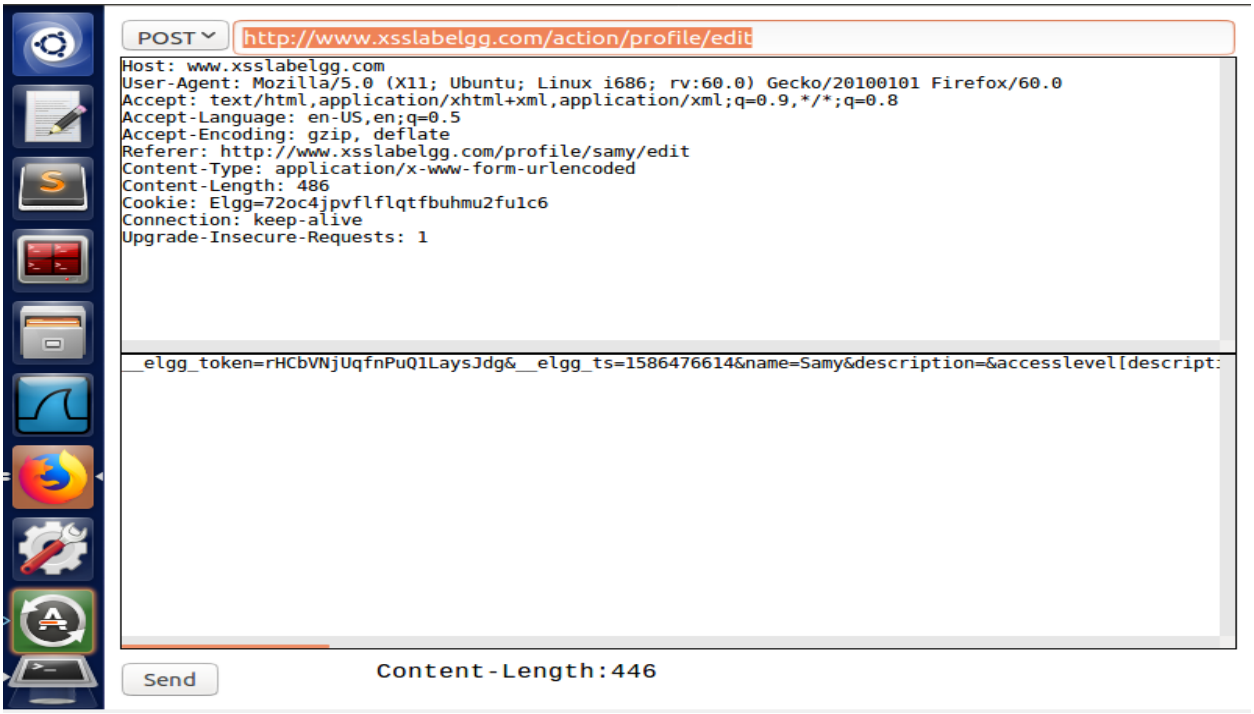


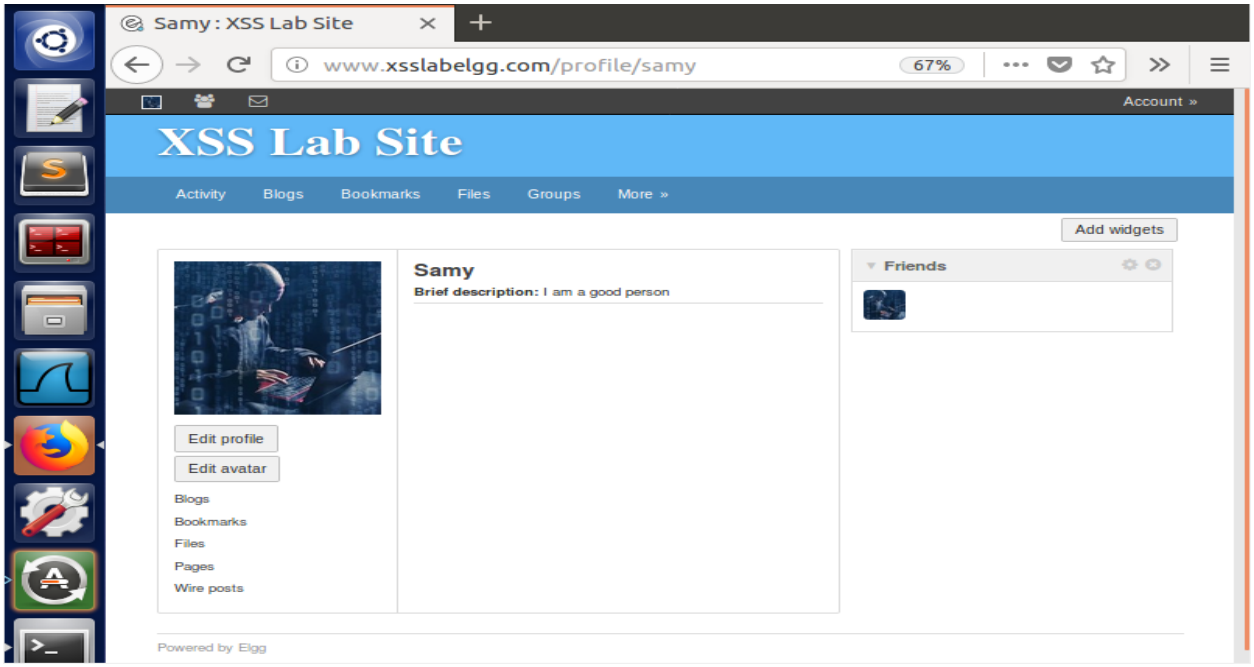
In order to modify user's profile, we must first know the exact HTTP POST request URL. To know this, Samy edits his own profile and adds a Brief description field.



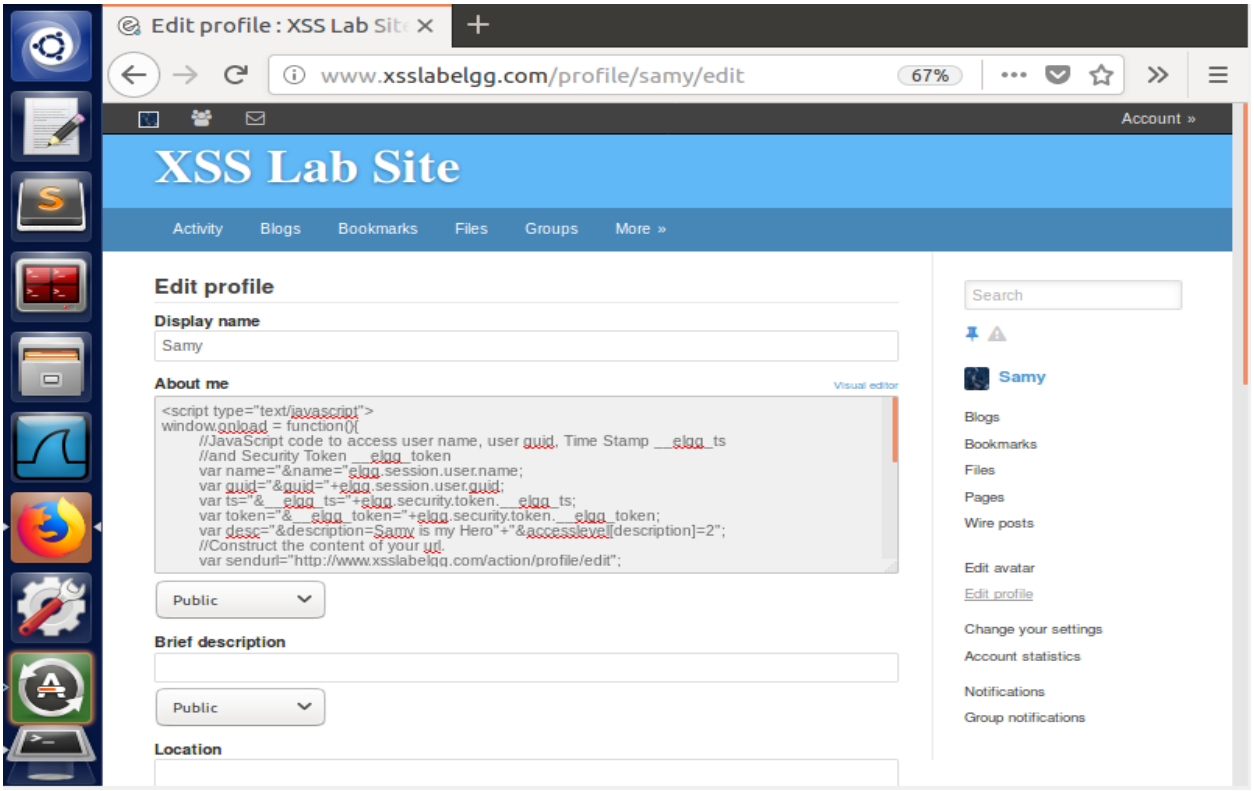


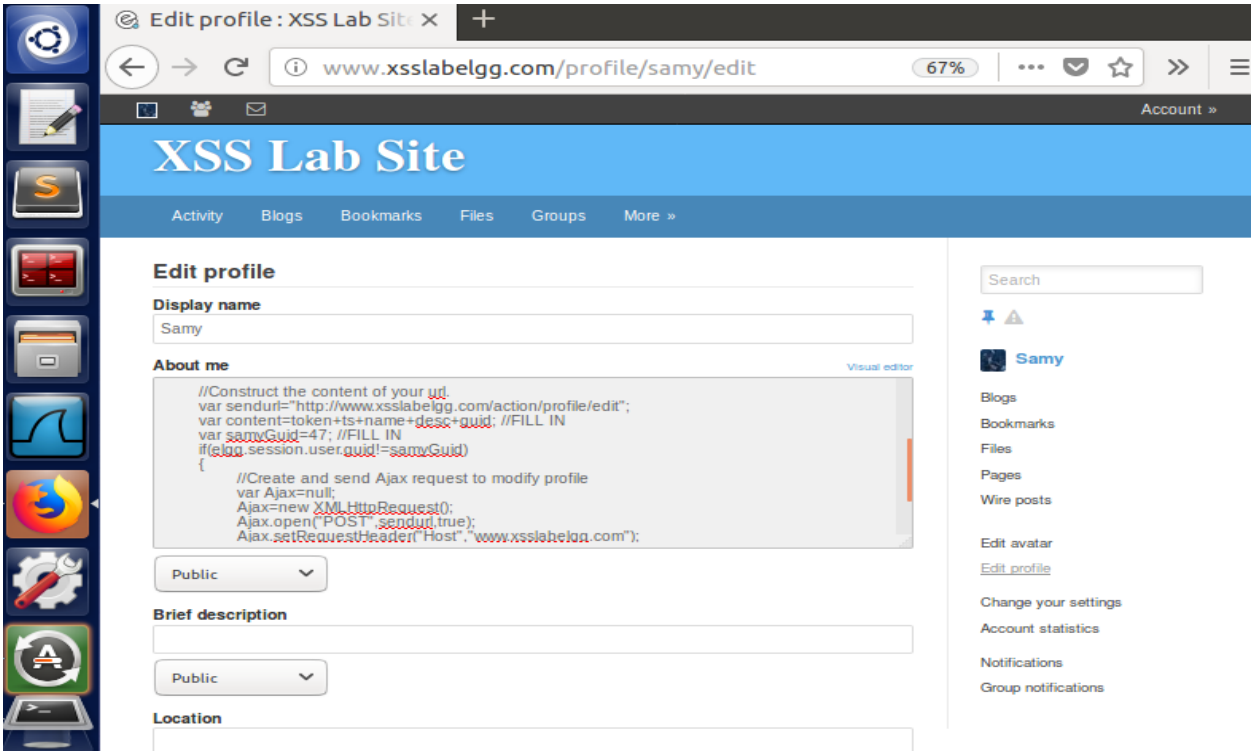
Samy uses the HTTP Header Live add on in Firefox to note the POST request URL. He then saves his profile and clicks on the POST request in the add on. The below screenshot shows the exact POST request URL needed along with the format of the parameters required for this request like the elgg_ts, elgg_token, accesslevel, description, etc. Samy makes use of these parameters in his JavaScript Code used as XSS worm.

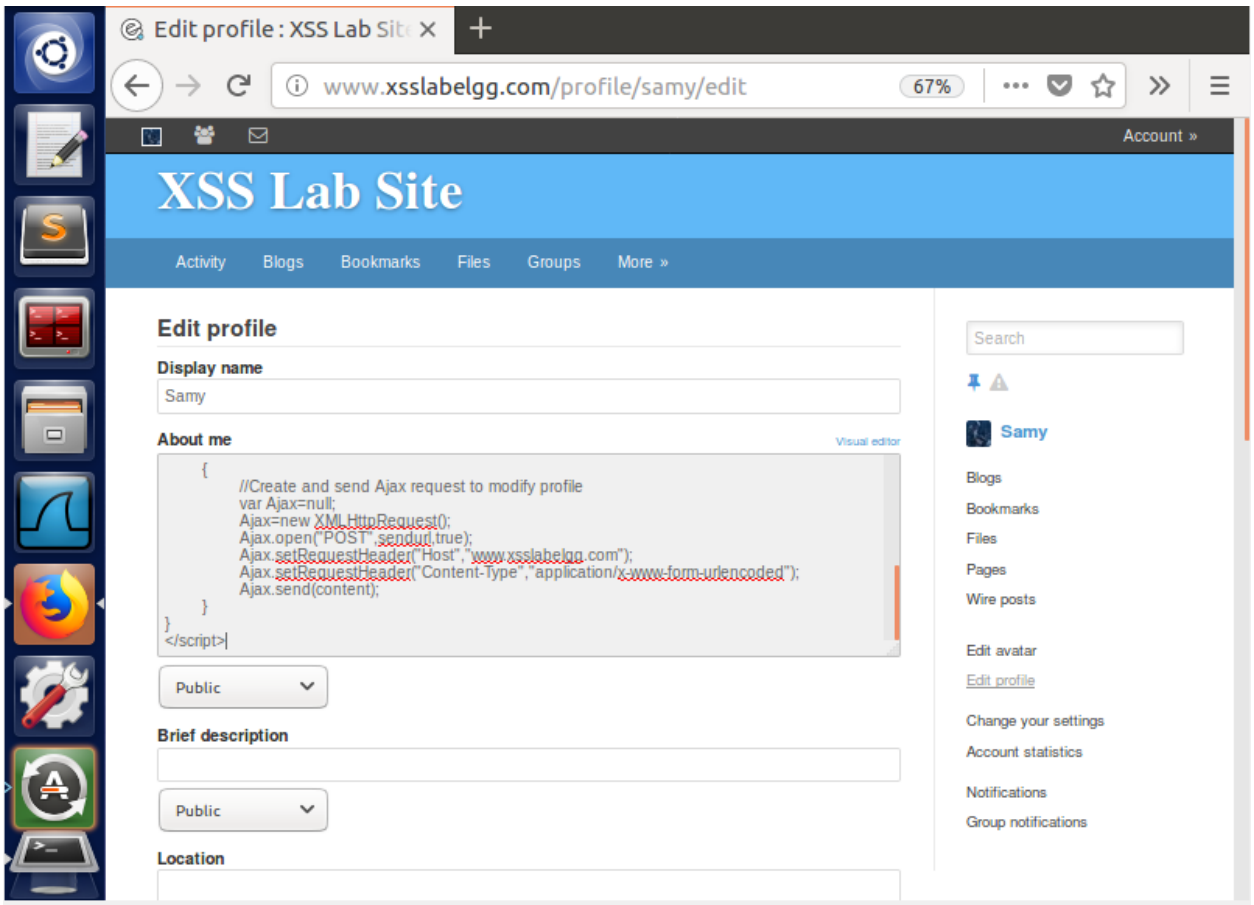


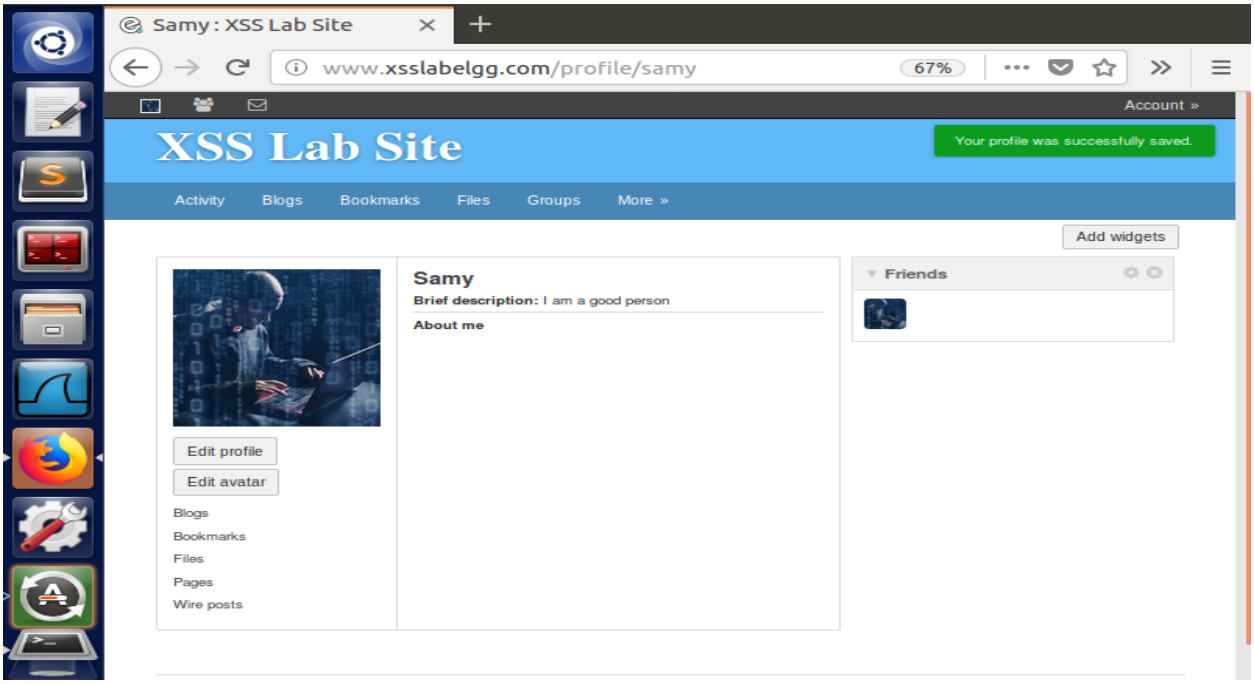


Samy creates a JavaScript code that contains the paramets he noted down in the post request. He then adds the desired description he wants to change in all the users who visit his profile. Here, the description changes to 'Samy is my Hero'. He puts this XSS worm in his 'Edit HTML' section of 'About me' in his profile so that the code gets added to the actual HTML code of his profile page.

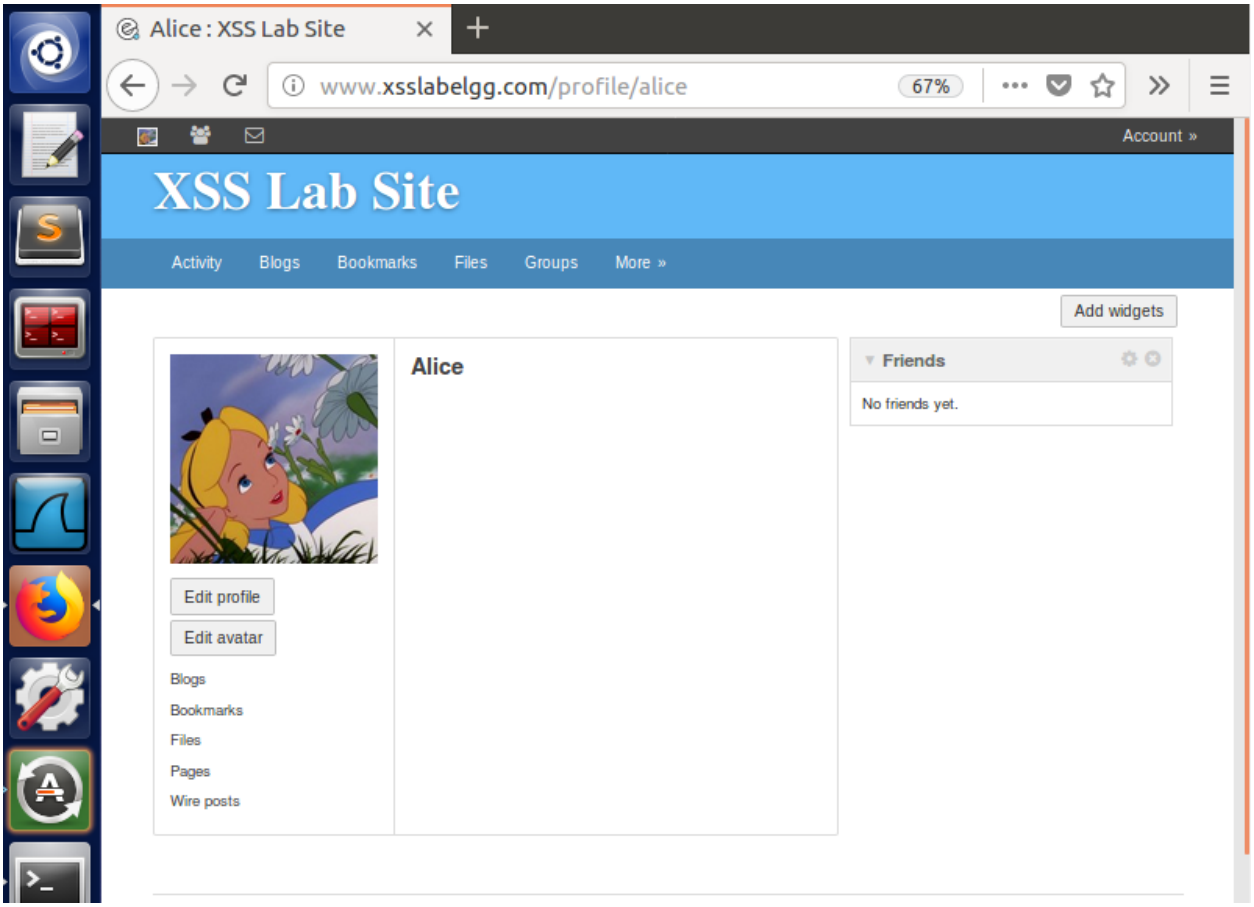


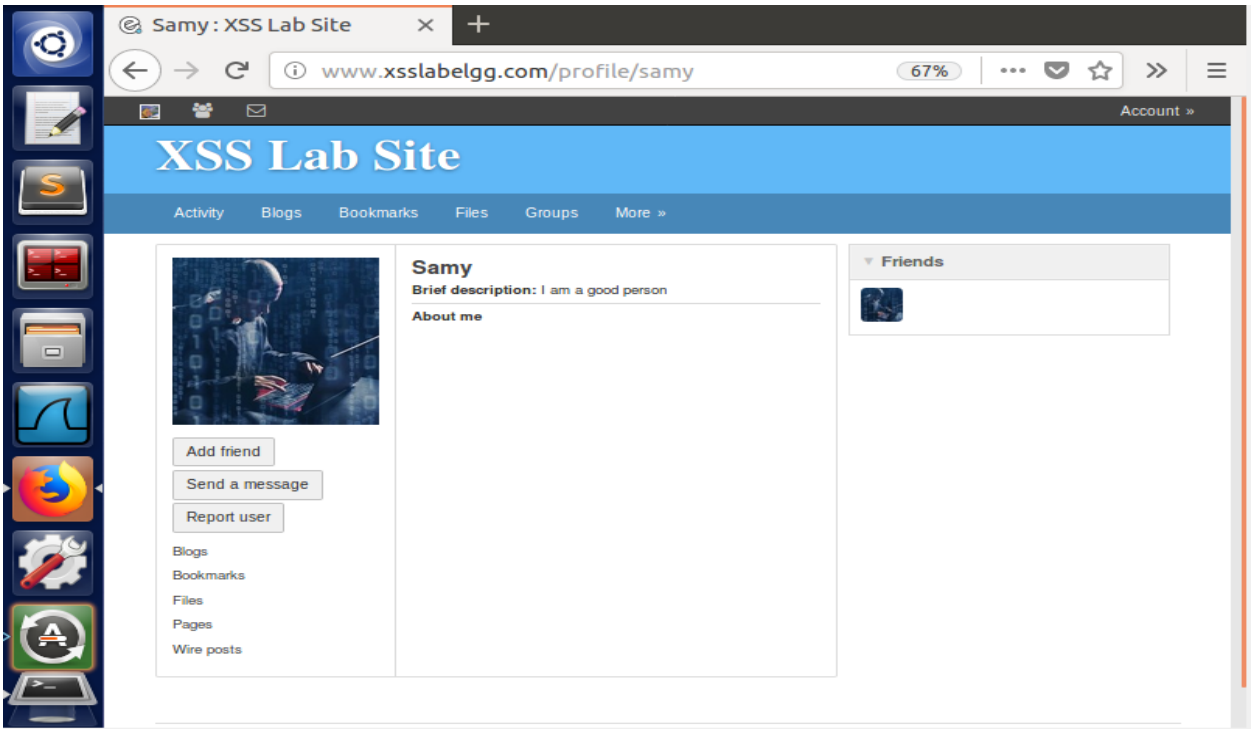

Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/samy/edit
67%
Account »
XSS Lab Site
Activity
Blogs
Bookmarks
Files
Groups
More »
Edit profile
Display name
Samy
About me
Visual editor
//Construct the content of your url.
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid; //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("Host","www.xsslabelgg.com");
Public
Brief description
Public
Location
Search
Samy
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Change your settings
Account statistics
Notifications
Group notifications


Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/samy/edit
67%
Account »
XSS Lab Site
Activity
Blogs
Bookmarks
Files
Groups
More »
Edit profile
Display name
Samy
About me
Visual editor
{
//Create and send Ajax request to modify profile
var Ajax=null;
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>
Public
Brief description
Public
Location
Search
Samy
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Change your settings
Account statistics
Notifications
Group notifications

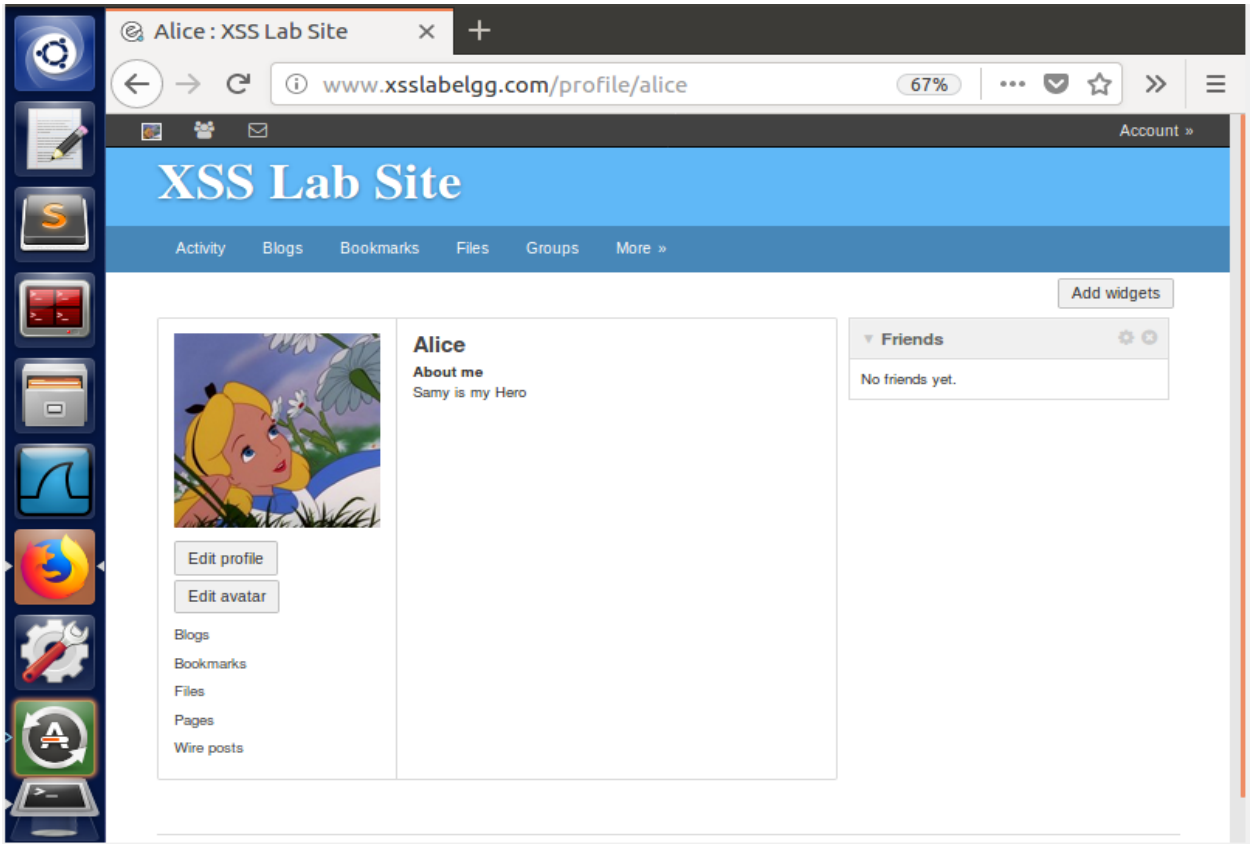


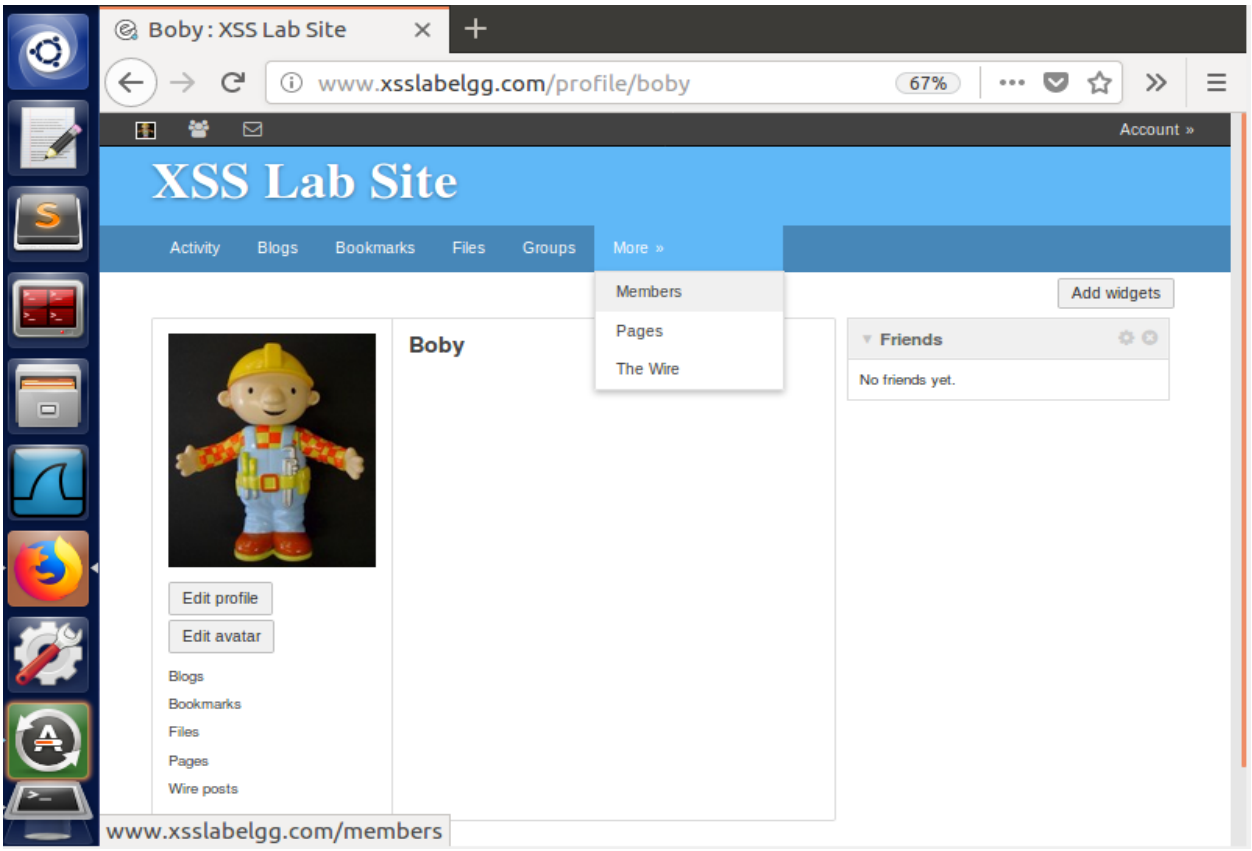
Samy's profile is saved. We then login as Alice and see that she has no description in her profile initially.



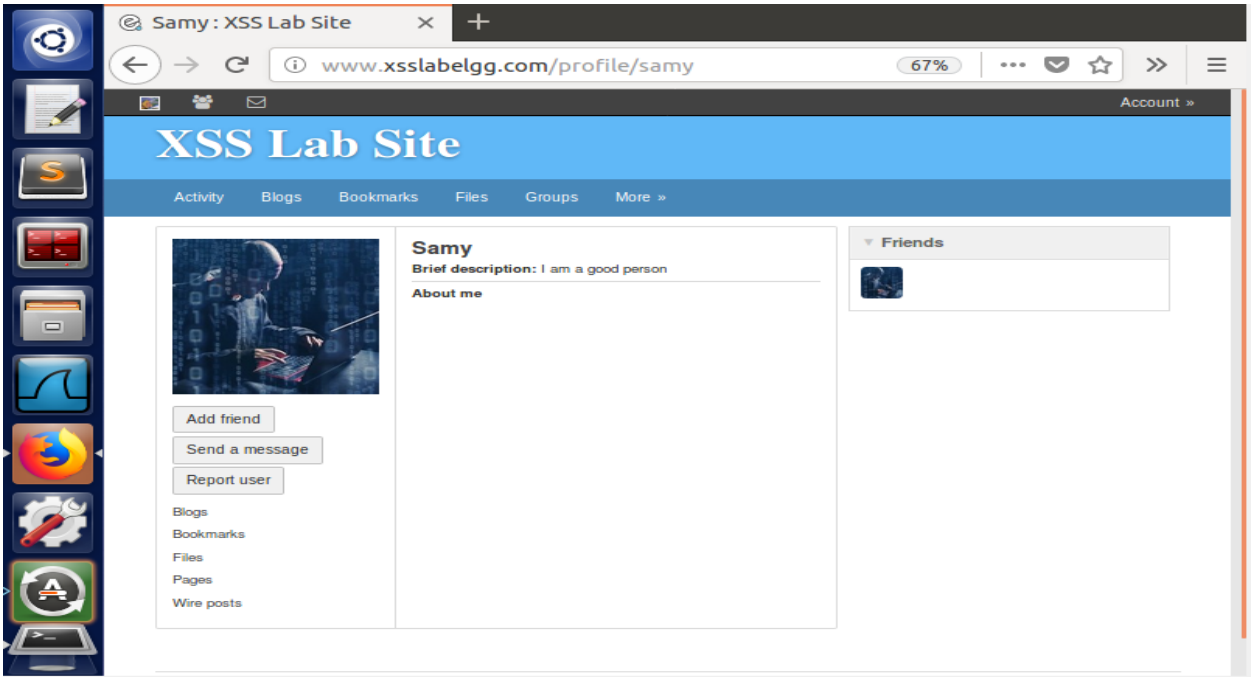


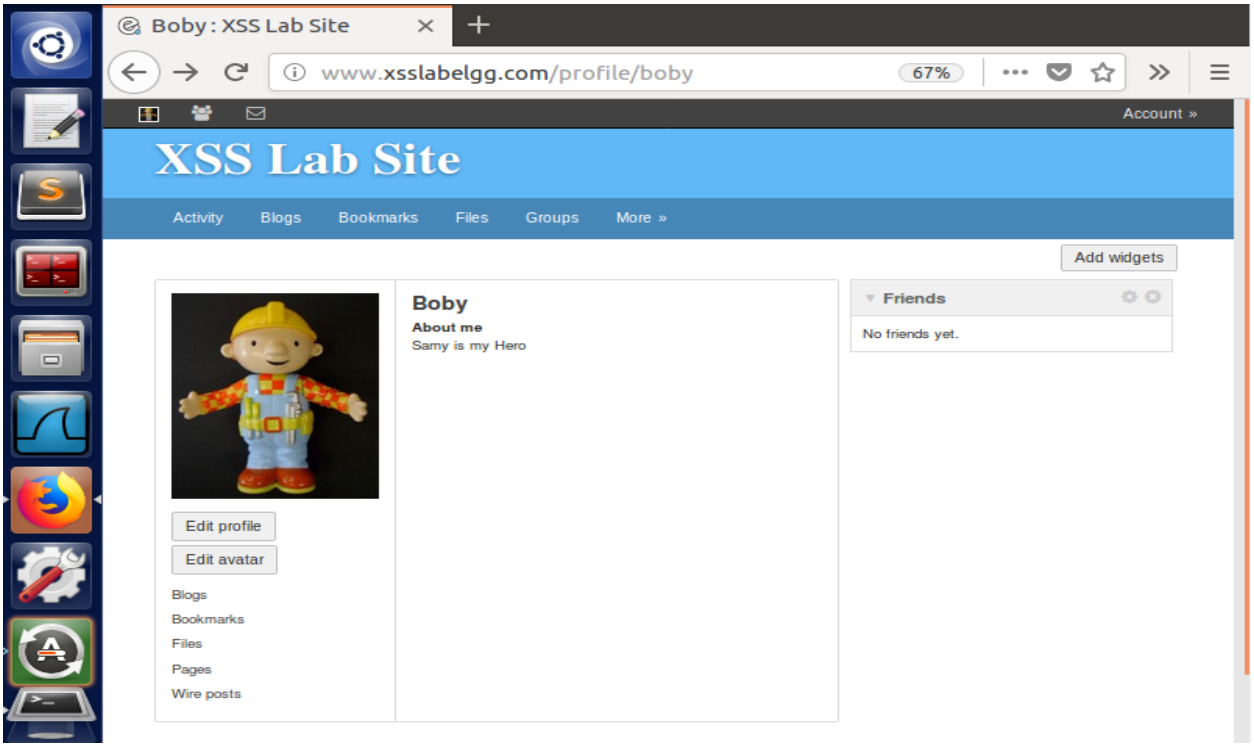
Alice casually goes and checks the members in Elgg website and visits Samy's profile. She then returns to her profile to see that her 'About me' section is changed to 'Samy is my Hero'.





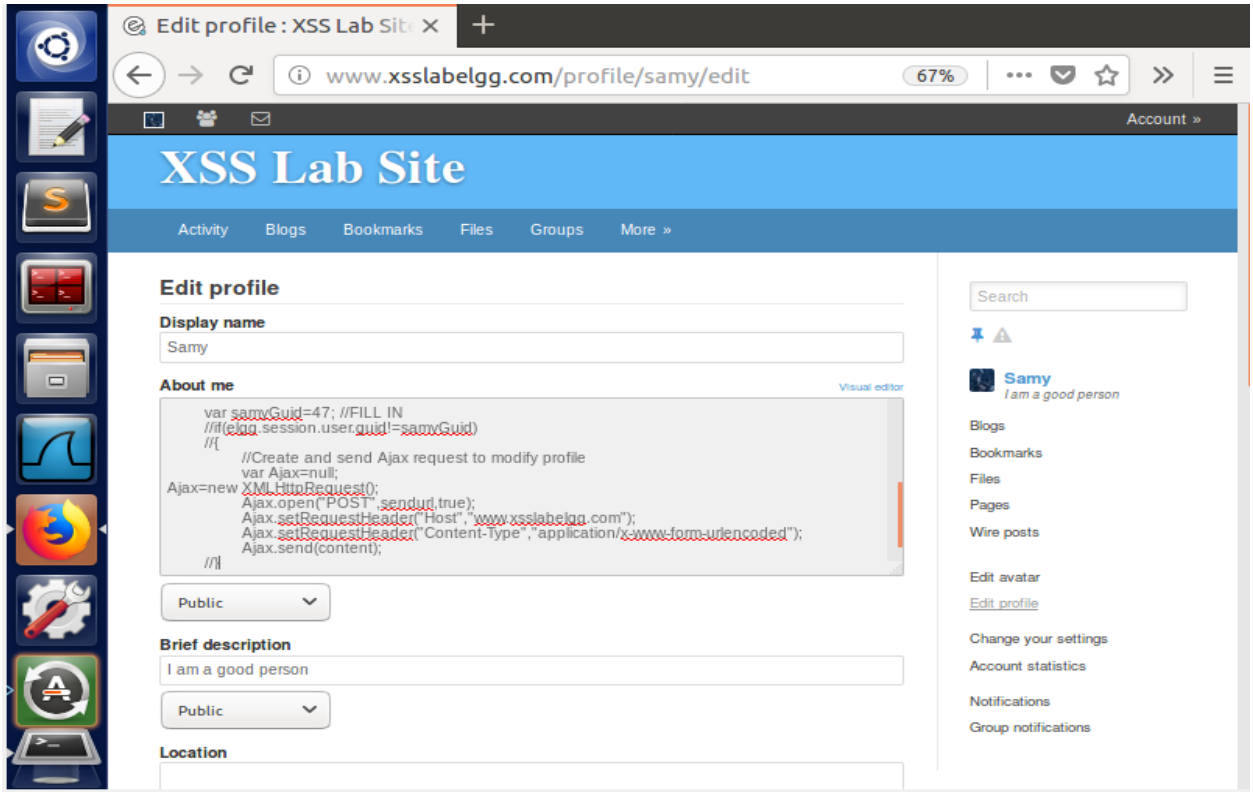
We repeat the same procedure with a different user named Boby. We sign in as Boby. Boby, then goes and checks Samy's profile casually and returns to his page to see his 'About me' changed to 'Samy is my Hero'.

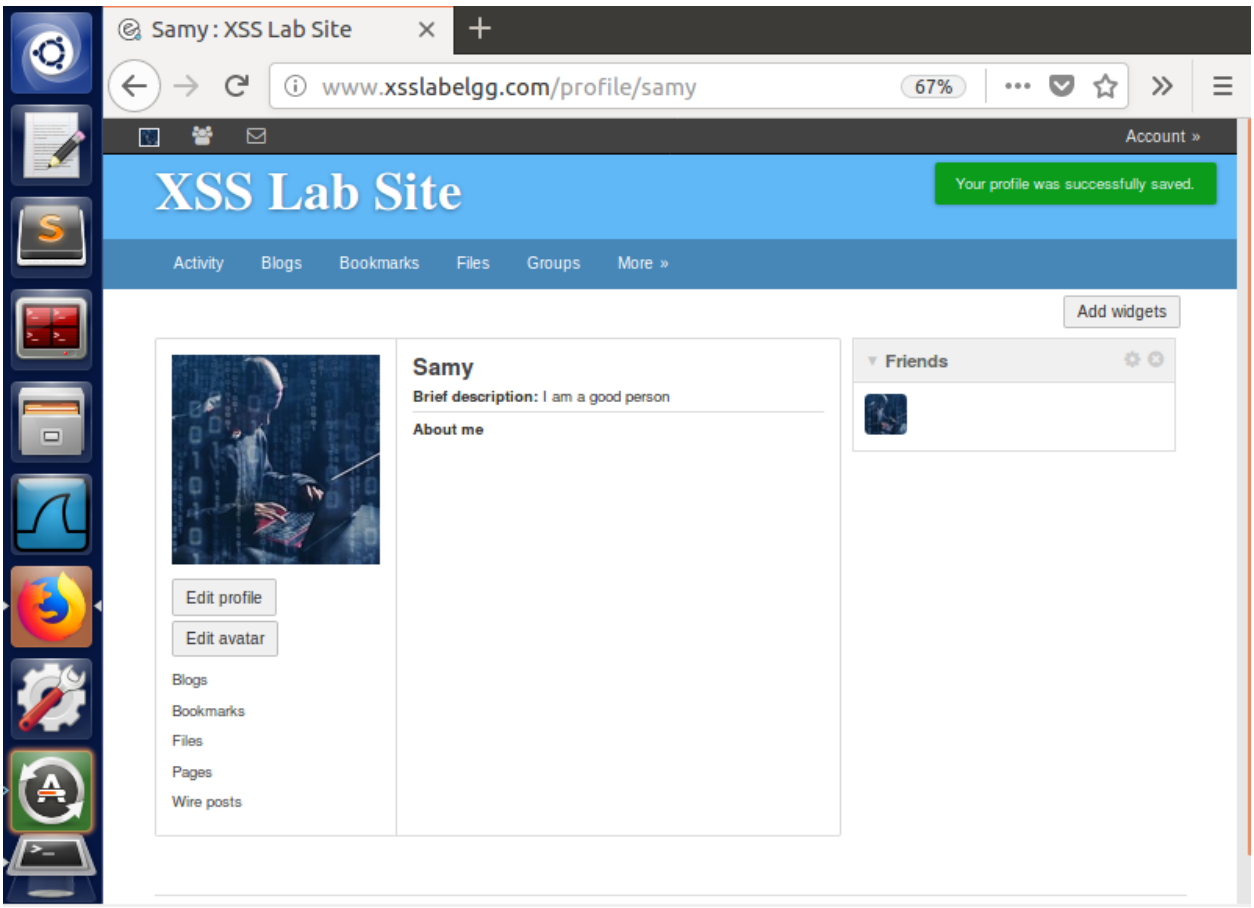


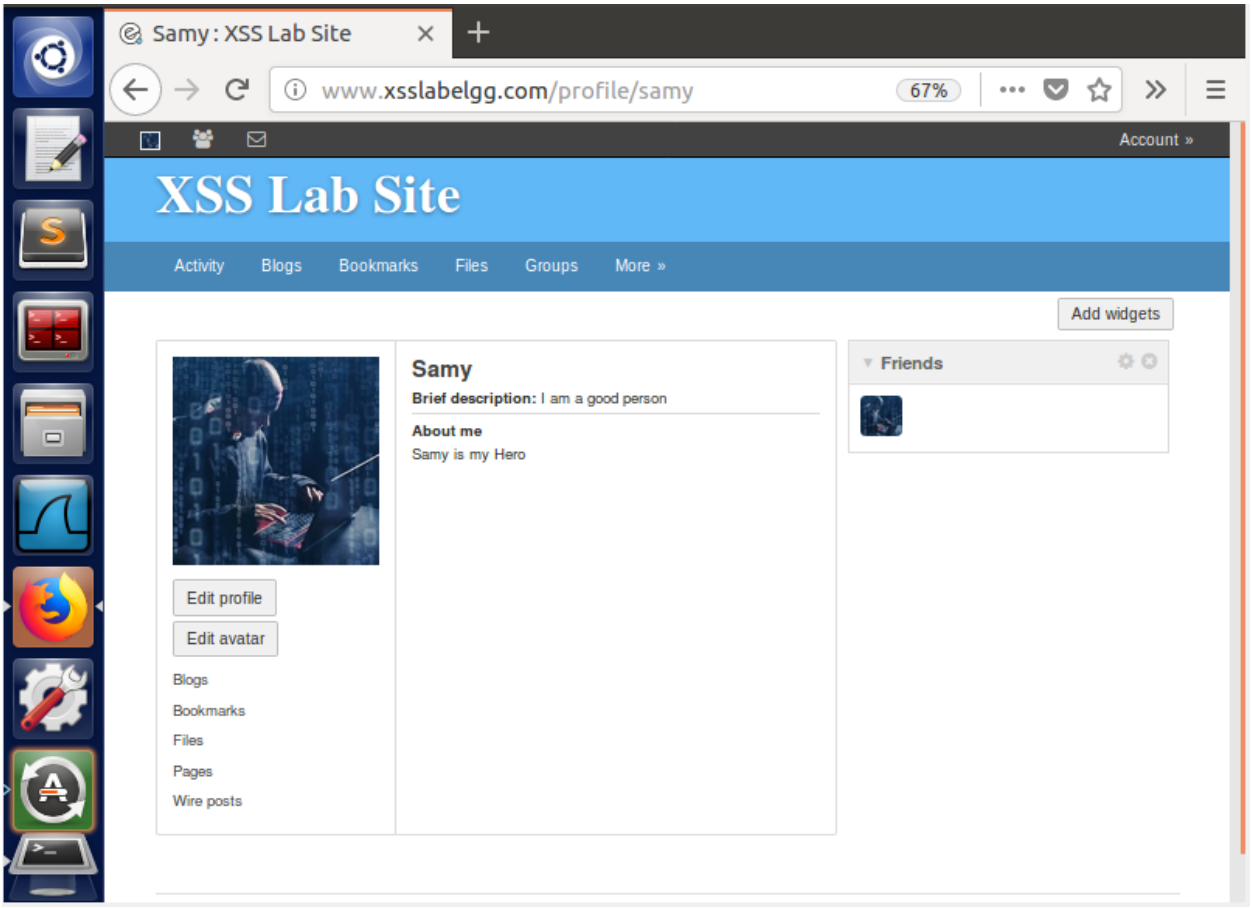


Conclusion: Samy wants to modify any user who visits his profile. So, he simply finds out the exact parameters of the POST request URL that is required to be submitted to the Elgg website. By knowing this, he writes a JavaScript code that acts as an XSS worm which can modify any user profile's 'About me' to 'Samy is my Hero'. This XSS worm is not self-propagating. This means that only if an user goes and visits the Samy's profile, they get attacked. This worm does not spread to other users. Thus, when Alice visited Samy's profile, her description changed and when Boby visited Samy's profile, his description changed.

Answer to Question 3: When the if condition line (Line 1) is commented out, the attacker's profile also gets attacked. This is explained in the below screenshots.


Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/samy/edit
67%
Account »
XSS Lab Site
Activity
Blogs
Bookmarks
Files
Groups
More »
Edit profile
Display name
Samy
About me
Visual editor
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("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send(content);
//}
Public
Brief description
I am a good person
Public
Location
Search
Samy
I am a good person
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Change your settings
Account statistics
Notifications
Group notifications

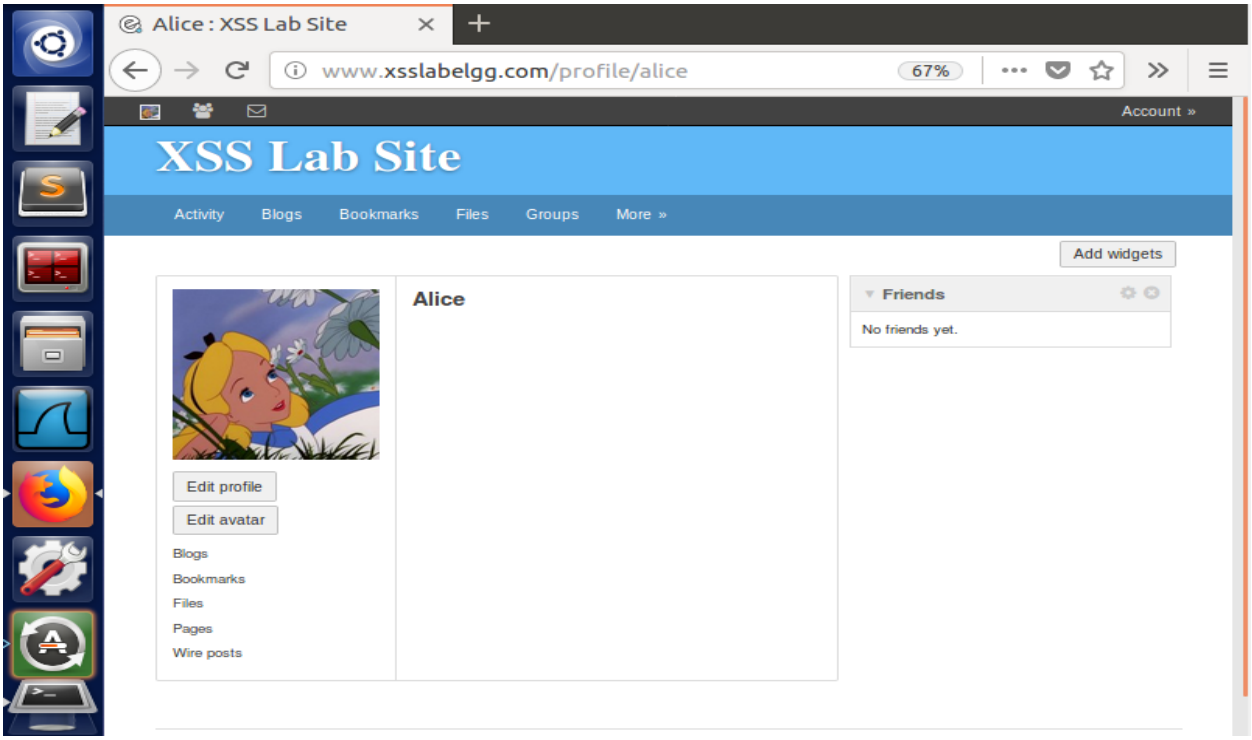

Samy : XSS Lab Site
www.xsslabelgg.com/profile/samy
67%
Account »
XSS Lab Site
Your profile was successfully saved.
Activity
Blogs
Bookmarks
Files
Groups
More »
Add widgets
Samy
Brief description: I am a good person
About me
Friends
Edit profile
Edit avatar
Blogs
Bookmarks
Files
Pages
Wire posts



Conclusion: The if condition checks whether the to-be-modified profile is not the attacker himself. If it is the attacker, it does not modify his profile. This line is commented out and he saves his profile. He refreshes the page and sees that his 'About me' description is changed to 'Samy is my Hero' which should not be the case. There is no point in Samy performing an attack on himself.

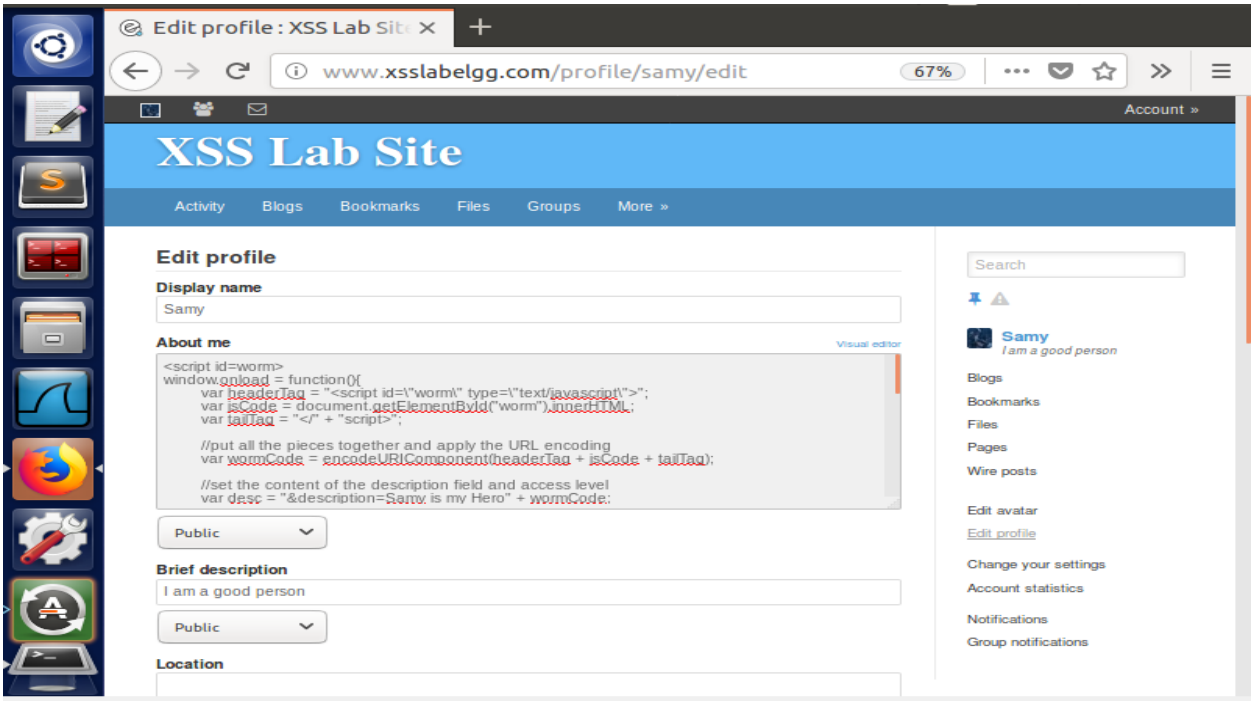
3.7 Task 6: Writing a Self-Propagating XSS Worm

The purpose of this task is to propagate the XSS worm we have been creating so far. This JavaScript code should get copied to the victim's profile when he/she visits the attacker's profile. Then the victim himself will become an attacker unknowingly and attacks any other user visiting the victim's user.

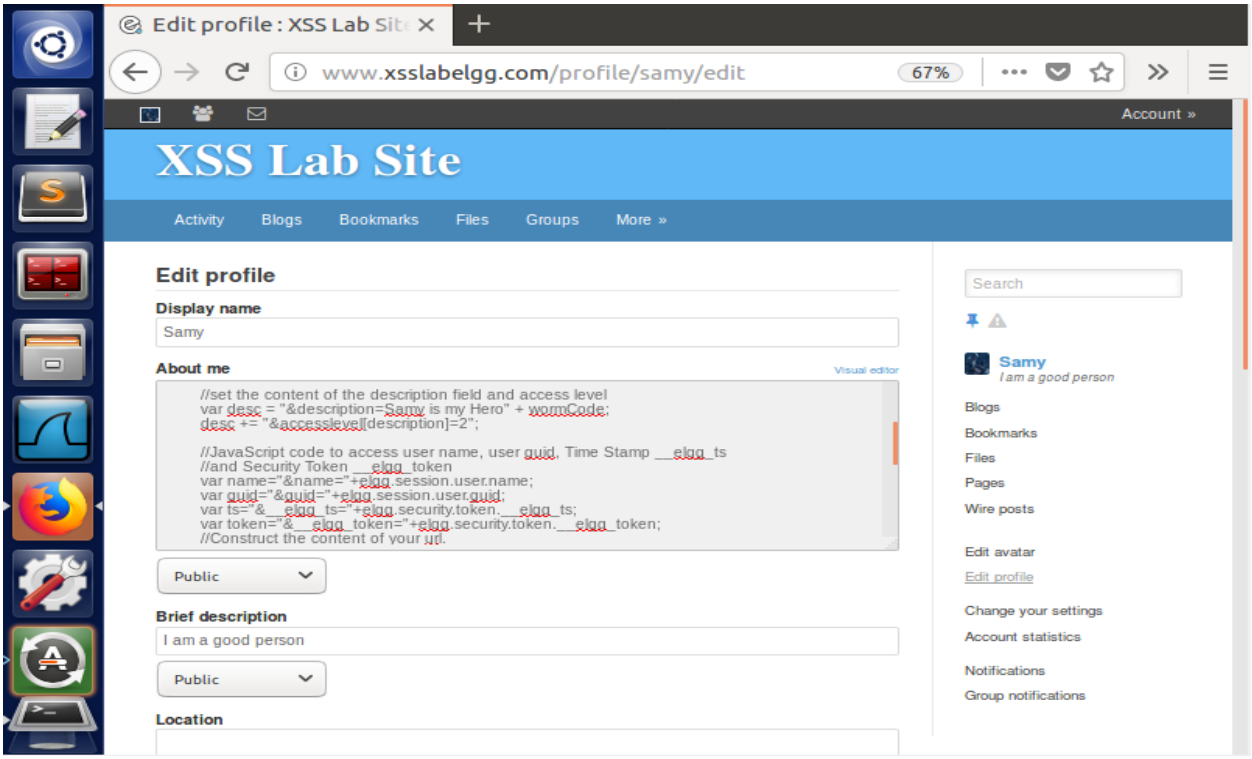


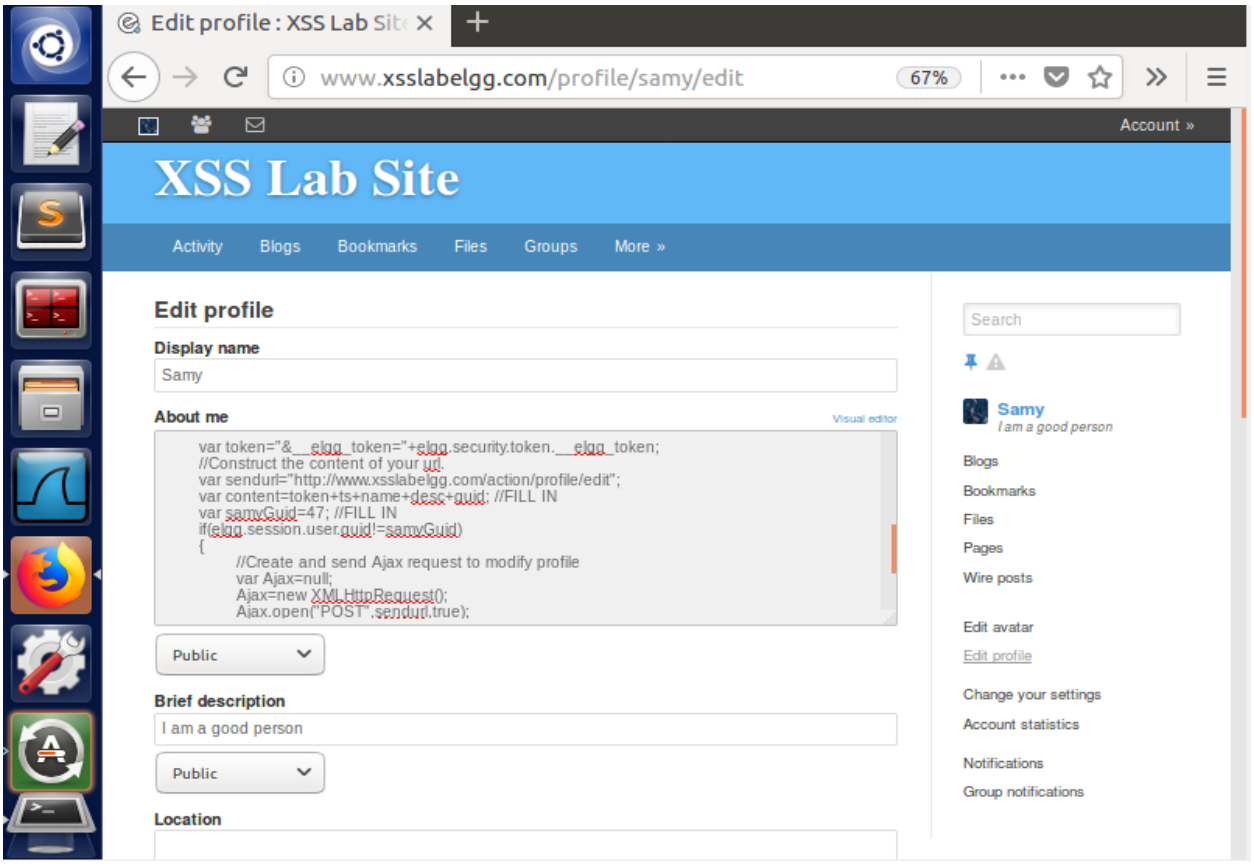
We reset Alice's and Boby's profile and delete their description before we start the task.

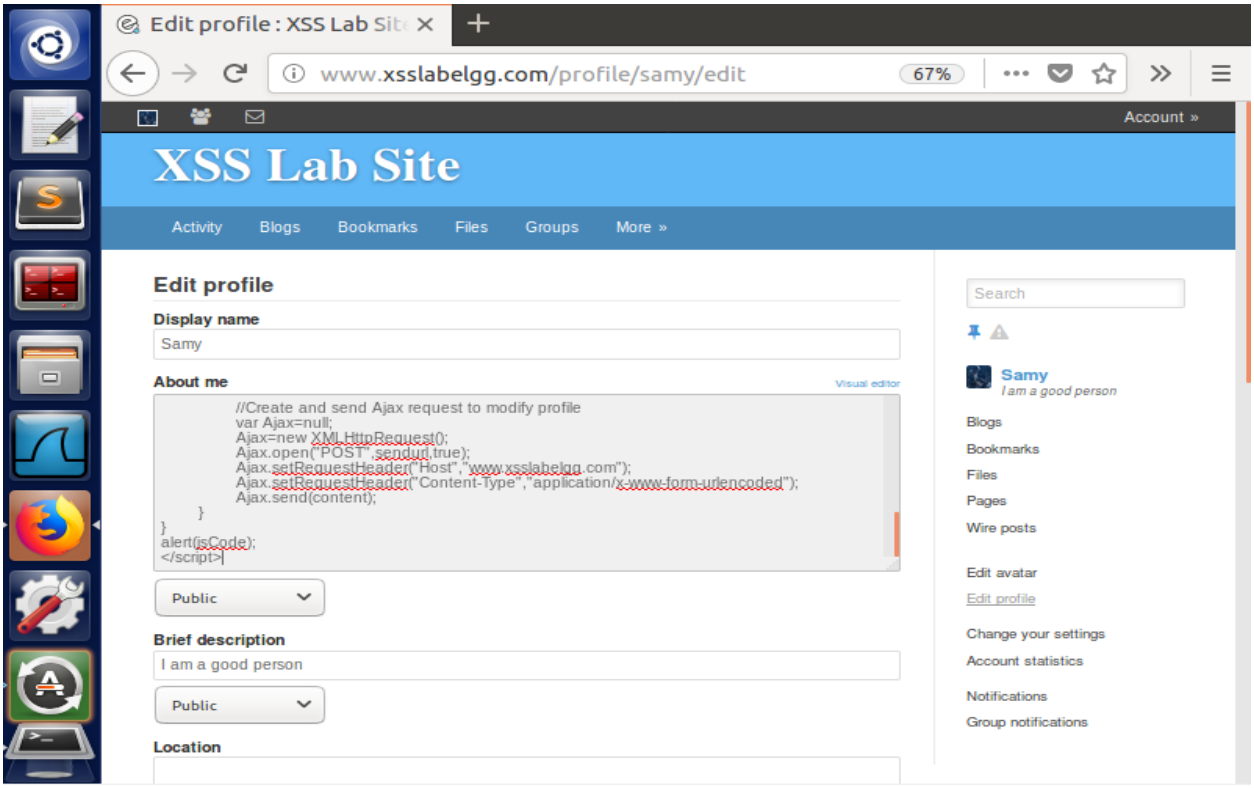
DOM Approach:

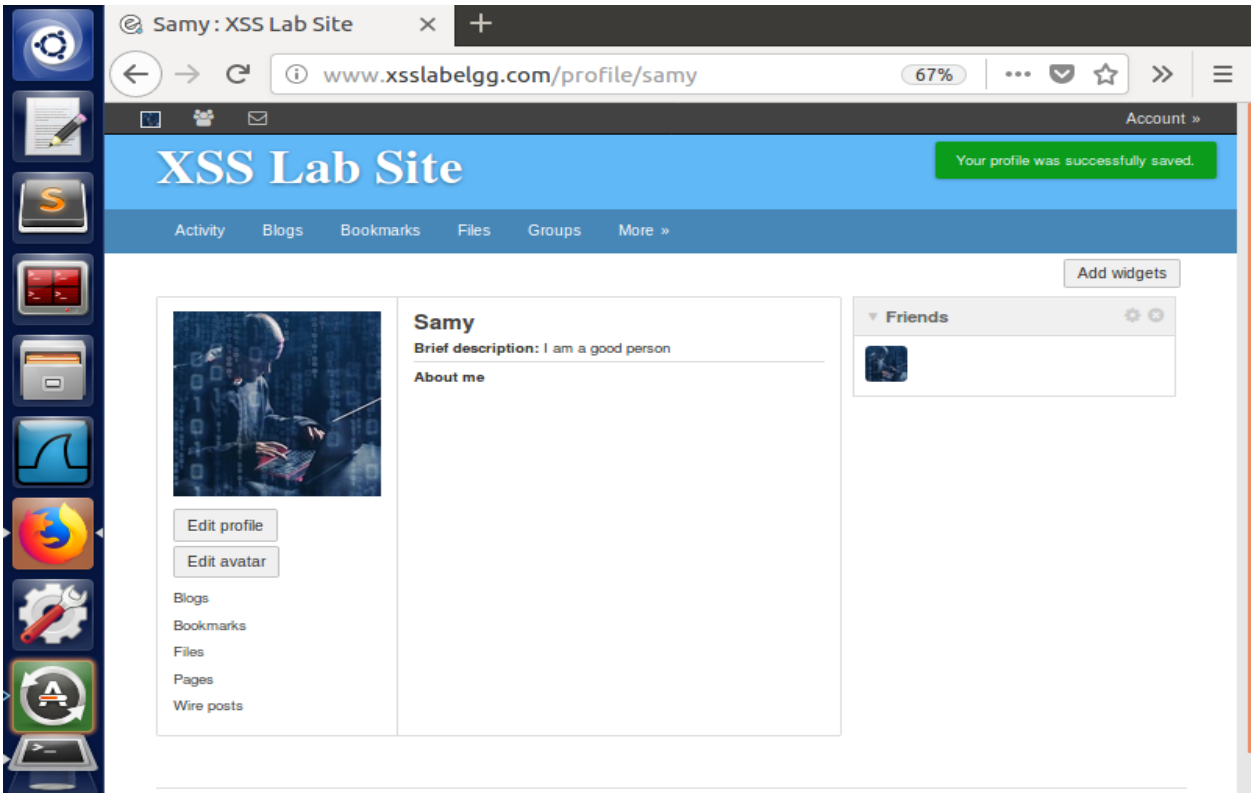


We add the wormCode to encode the URL to use the head tag, jsCode and the tail tag in the JavaScript and add the extra code needed to perform the XSS worm attack.

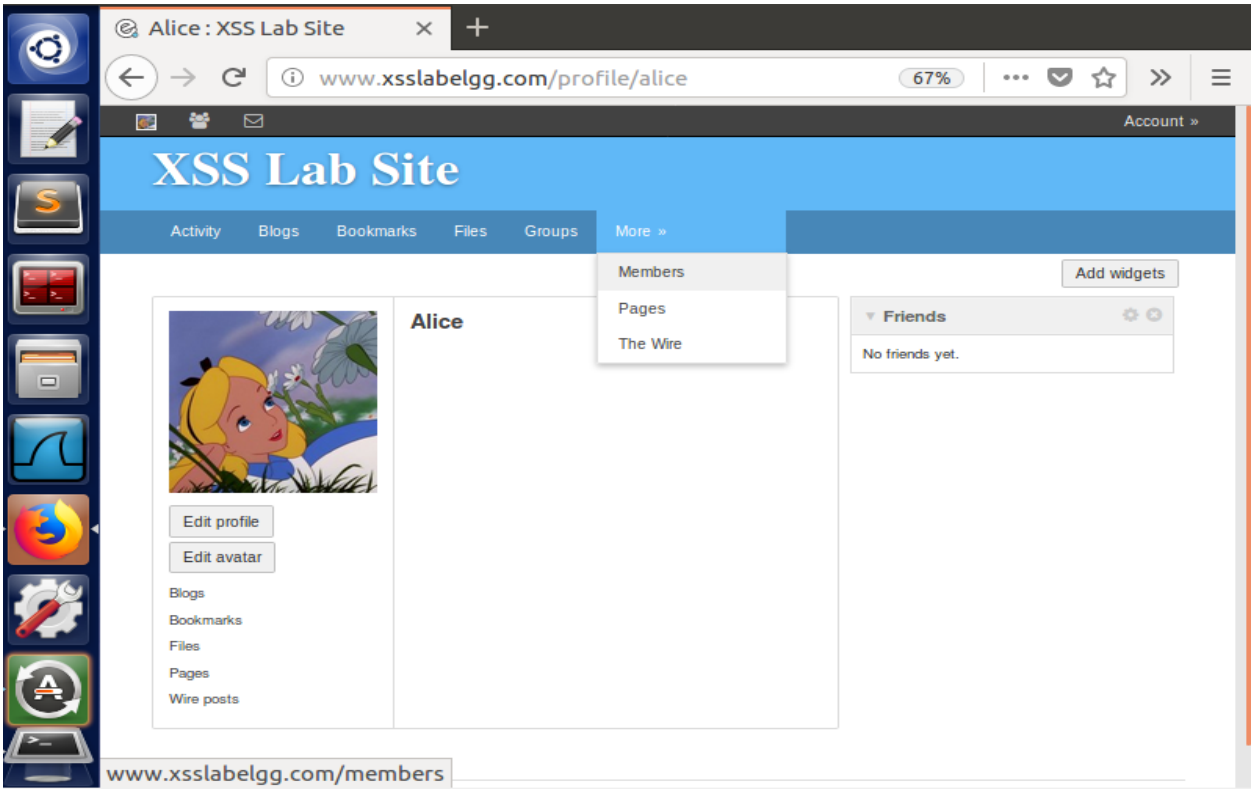


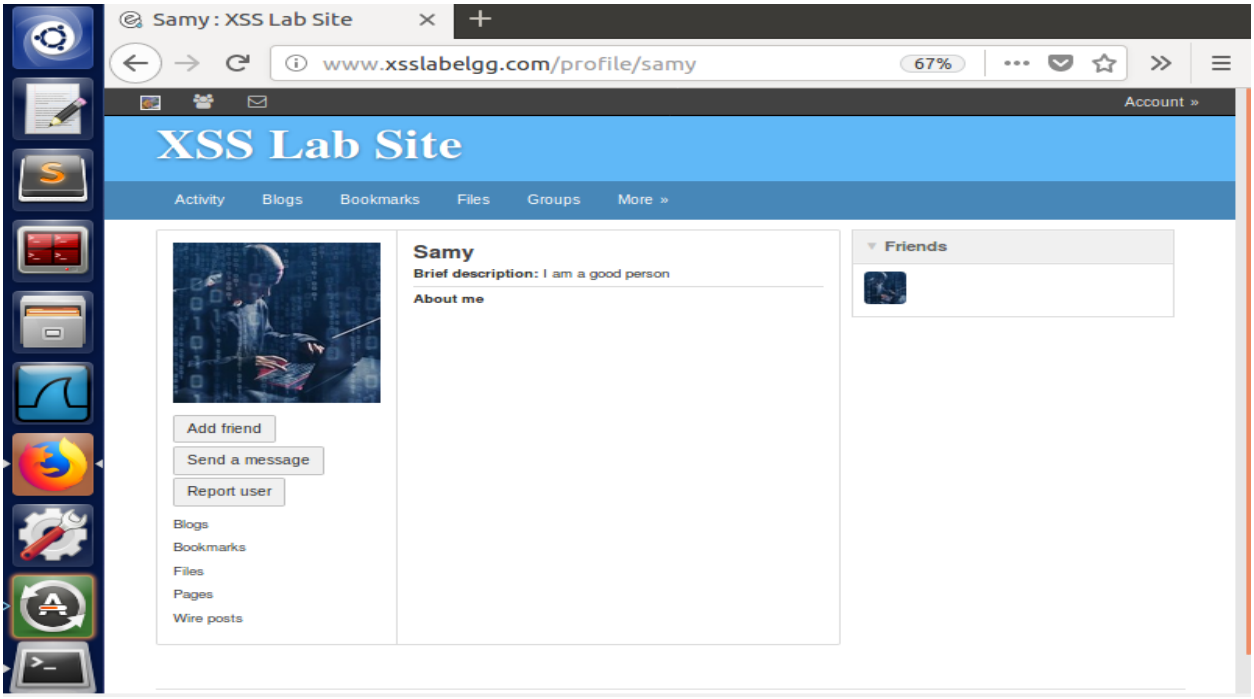

Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/samy/edit
67%
Account »
XSS Lab Site
Activity
Blogs
Bookmarks
Files
Groups
More »
Edit profile
Display name
Samy
About me
Visual editor
var token="&__elgg_token="+elgg.security.token.__elgg_token;
//Construct the content of your url.
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
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);
Public
Brief description
I am a good person
Public
Location
Search
Samy
I am a good person
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Change your settings
Account statistics
Notifications
Group notifications


Edit profile : XSS Lab Site
www.xsslabelgg.com/profile/samy/edit
67%
Account »
XSS Lab Site
Activity
Blogs
Bookmarks
Files
Groups
More »
Edit profile
Display name
Samy
About me
Visual editor
//Create and send Ajax request to modify profile
var Ajax=null;
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);
}
}
alert(jsCode);
</script>
Public
Brief description
I am a good person
Public
Location
Search
Samy
I am a good person
Blogs
Bookmarks
Files
Pages
Wire posts
Edit avatar
Edit profile
Change your settings
Account statistics
Notifications
Group notifications

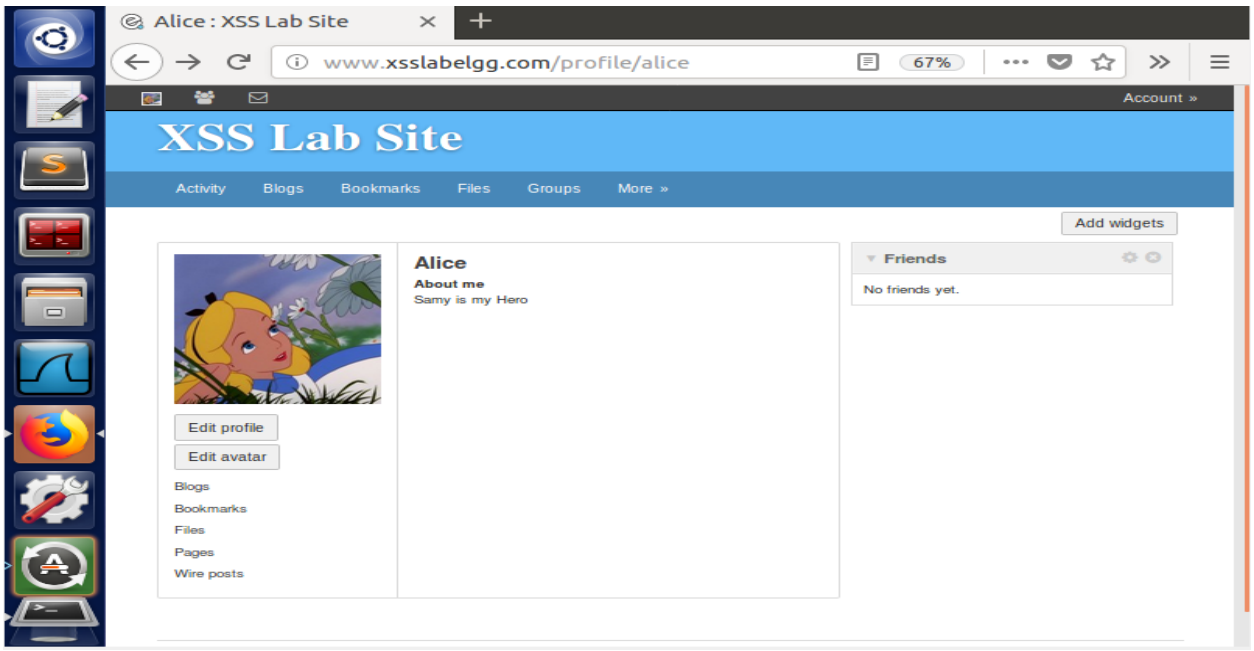


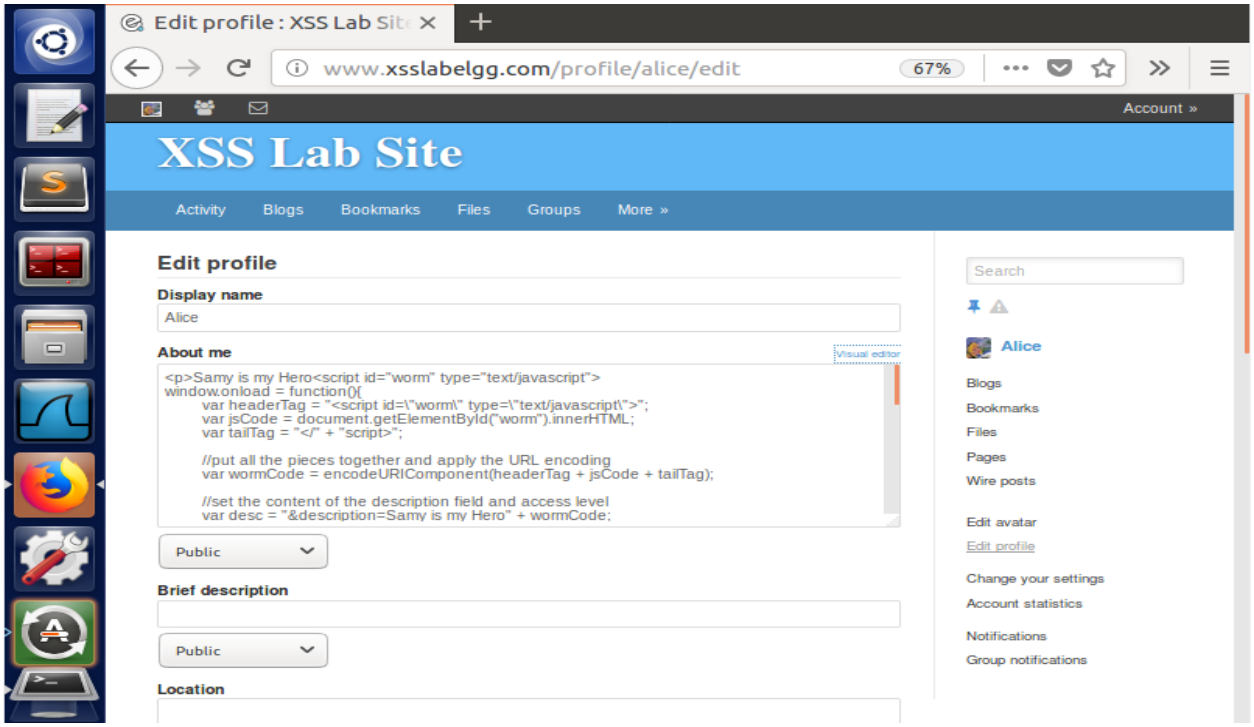
Samy saves his profile with the worm code. We then login as Alice. Alice casually goes and checks the members of the Elgg website and visits Samy's profile



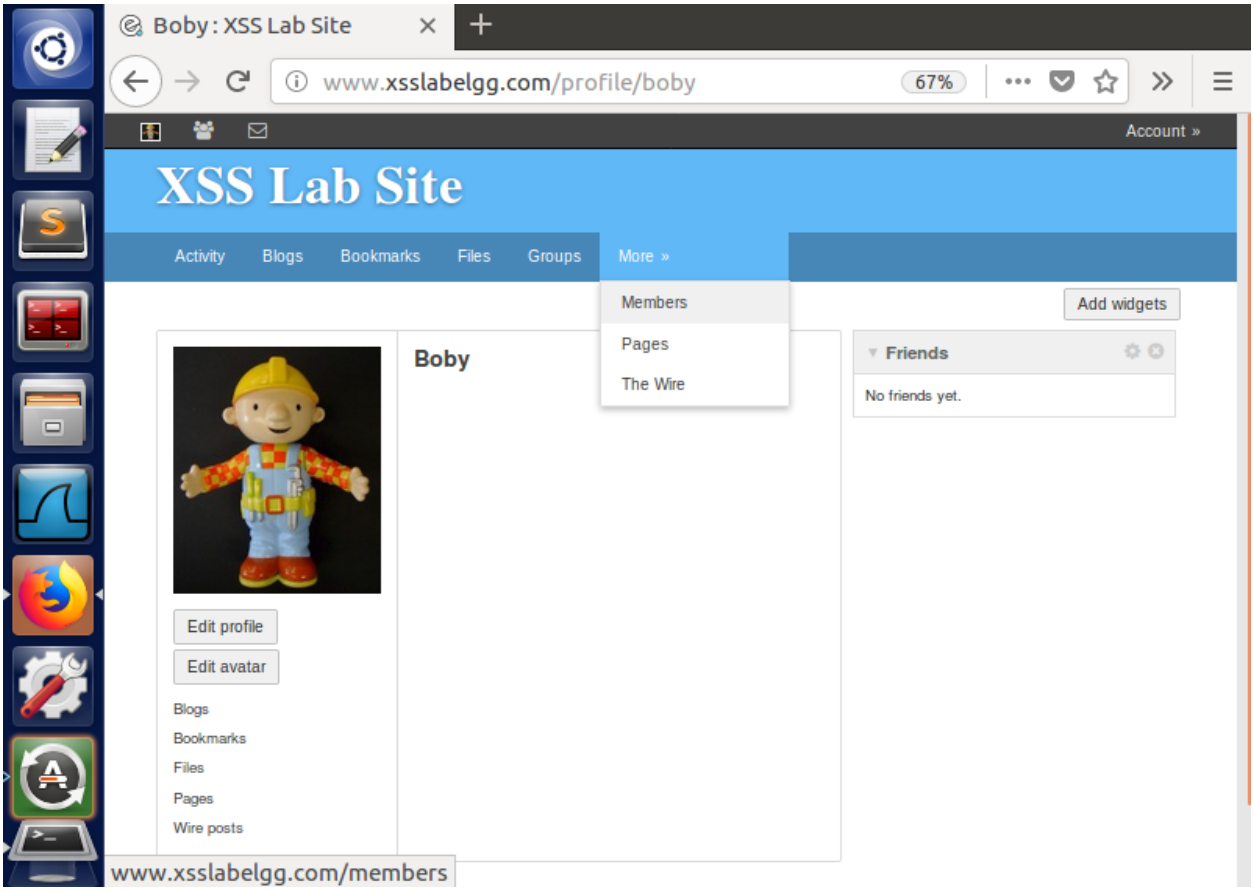


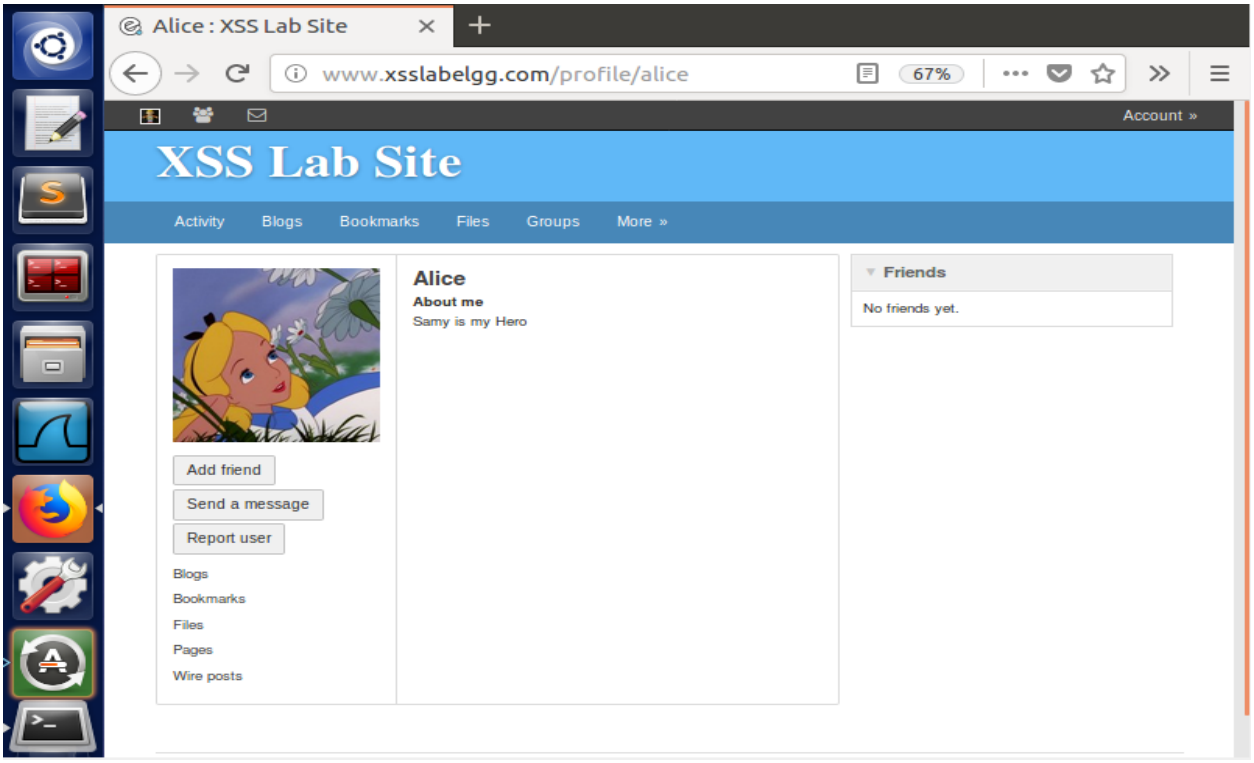
When Alice returns to her profile, she sees that her 'About me' description is changed to 'Samy is my Hero'.



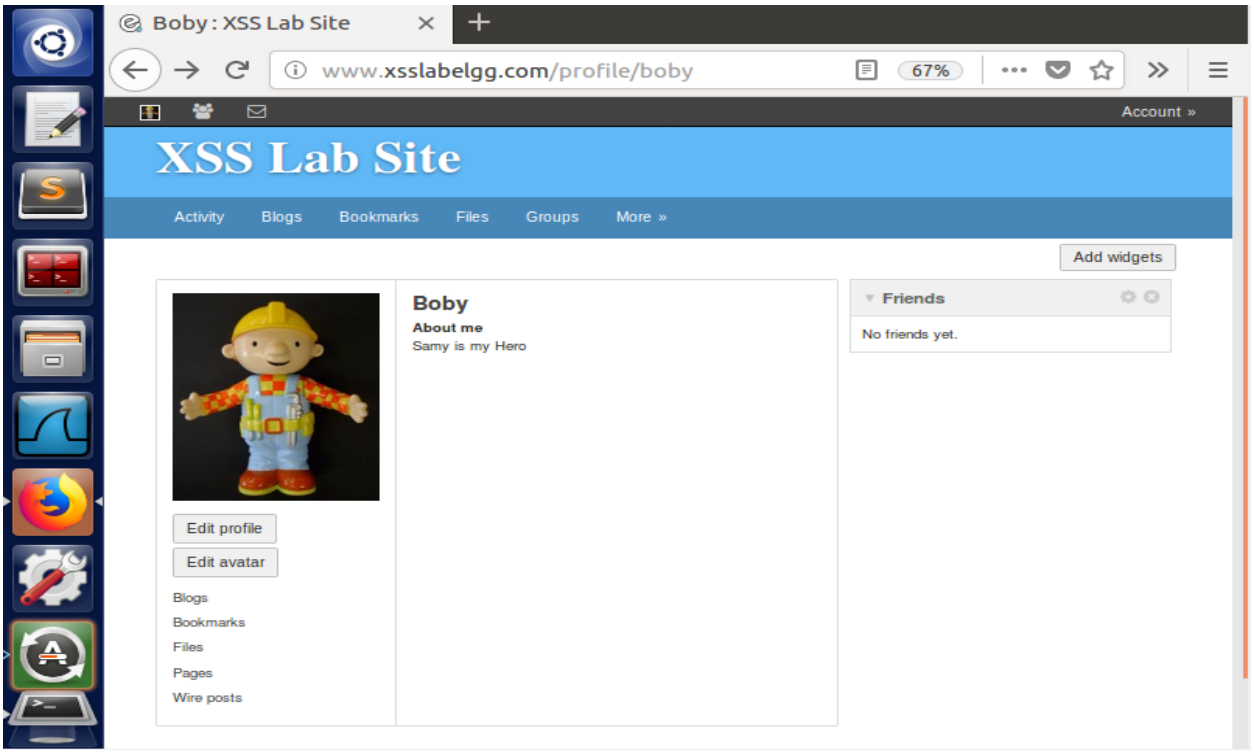


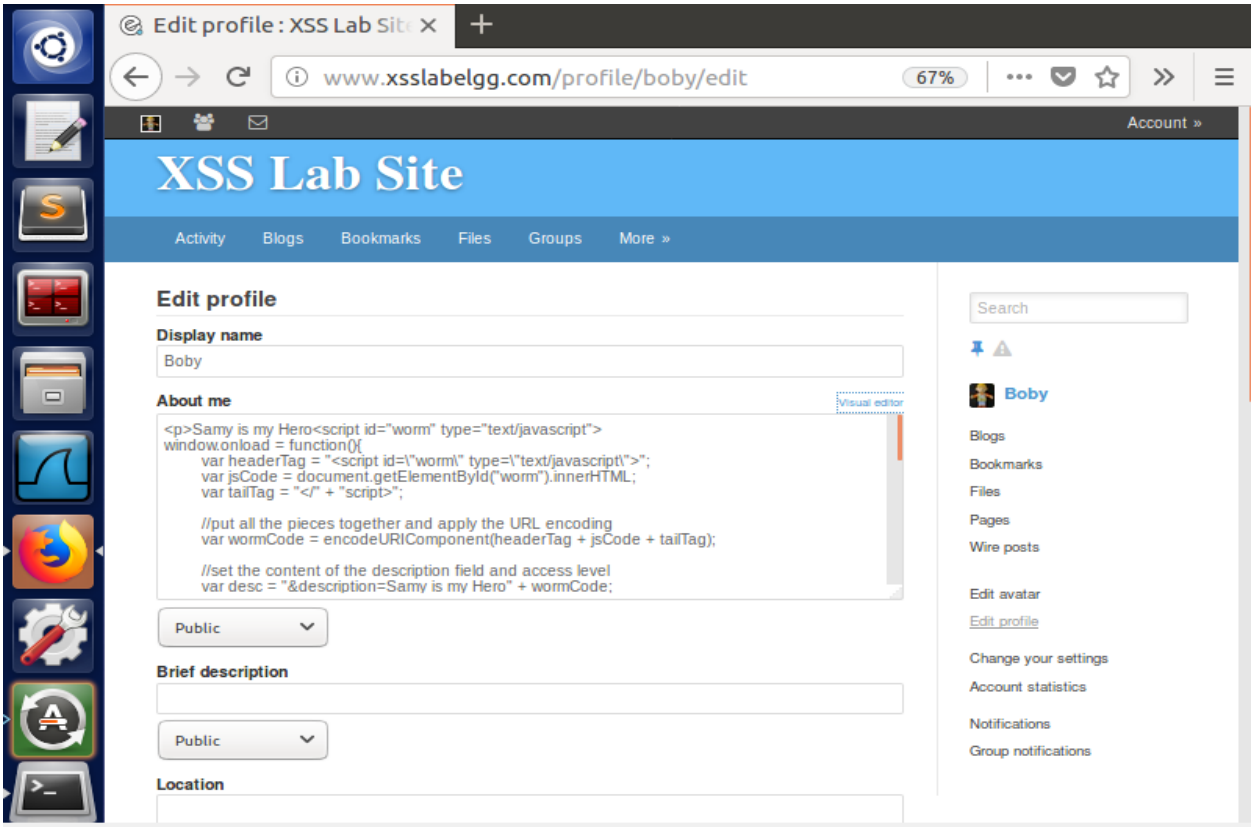
When Alice goes and checks what is wrong with her profile, she sees the worm code in her 'About me' section 'Edit HTML'. We then login as Boby and check Alice's profile.





When Boby returns to his profile he sees that his 'About me' description is changed to 'Samy is my Hero'.



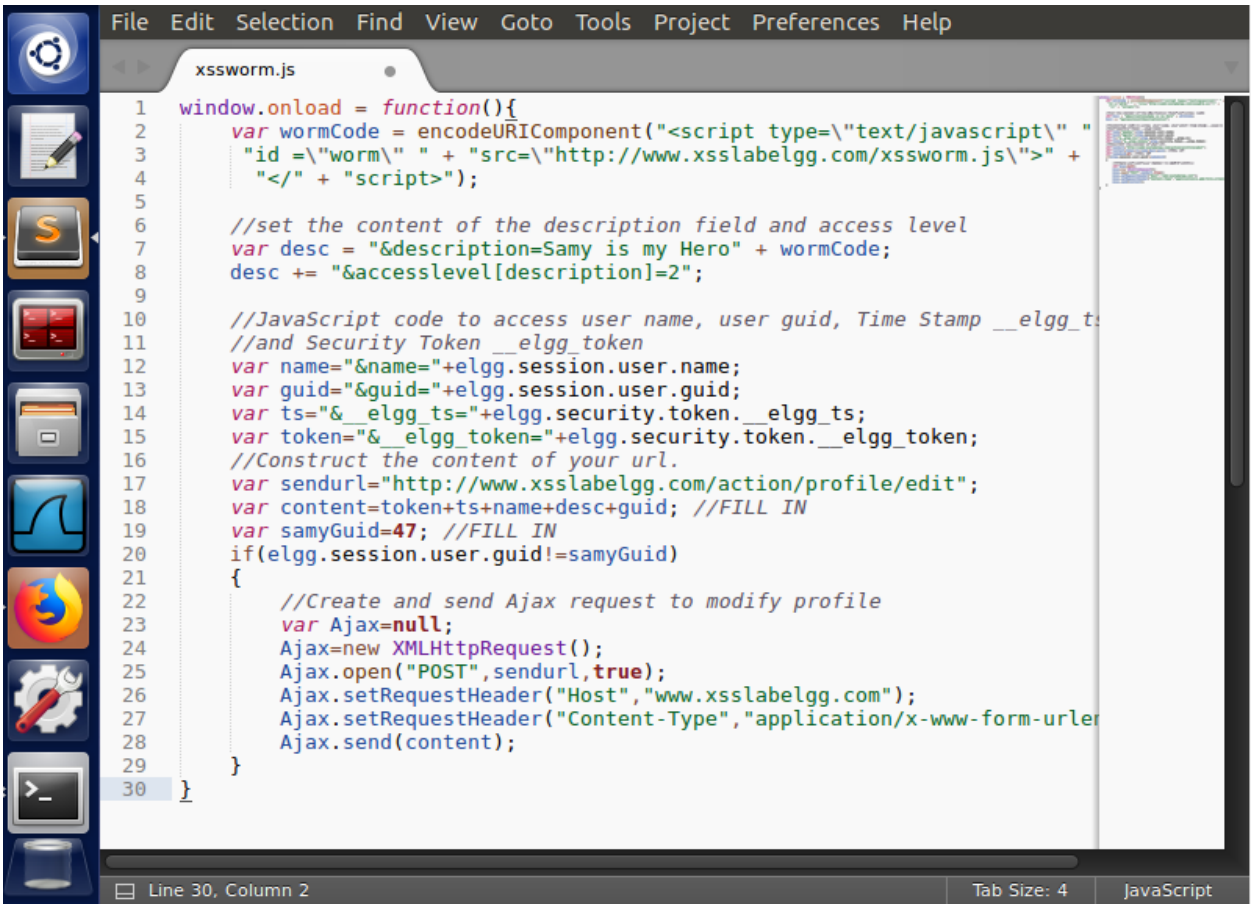


When Boby goes and checks his own 'About me' section 'Edit HTML', he sees that the worm code is now copied to his profile. Now if any other user comes and checks Boby's profile, that user will also get infected and this XSS worm code is copied to that user's profile.

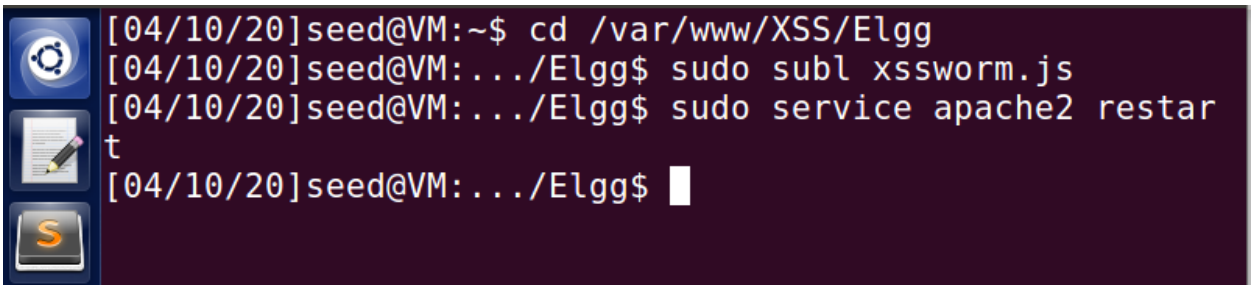
Conclusion: Samy wants to modify everyone's profile in a short period of time. So, he creates an XSS worm and puts it in his profile that self-propagates itself. This means that whenever some victim comes and visits Samy's profile, the worm code gets copied to that victim's profile. Now, whenever someone else comes and visits this victim's profile that user also gets affected and so on. This is called self-propagating. When Alice visits Samy's page, she gets the XSS worm in her profile. When Boby visits Alice's profile, he gets the worm code in his profile and the network goes on and on. Here, we used the DOM paramets and performed the attack. This is a DOM approach for this XSS self-propagating worm attack.

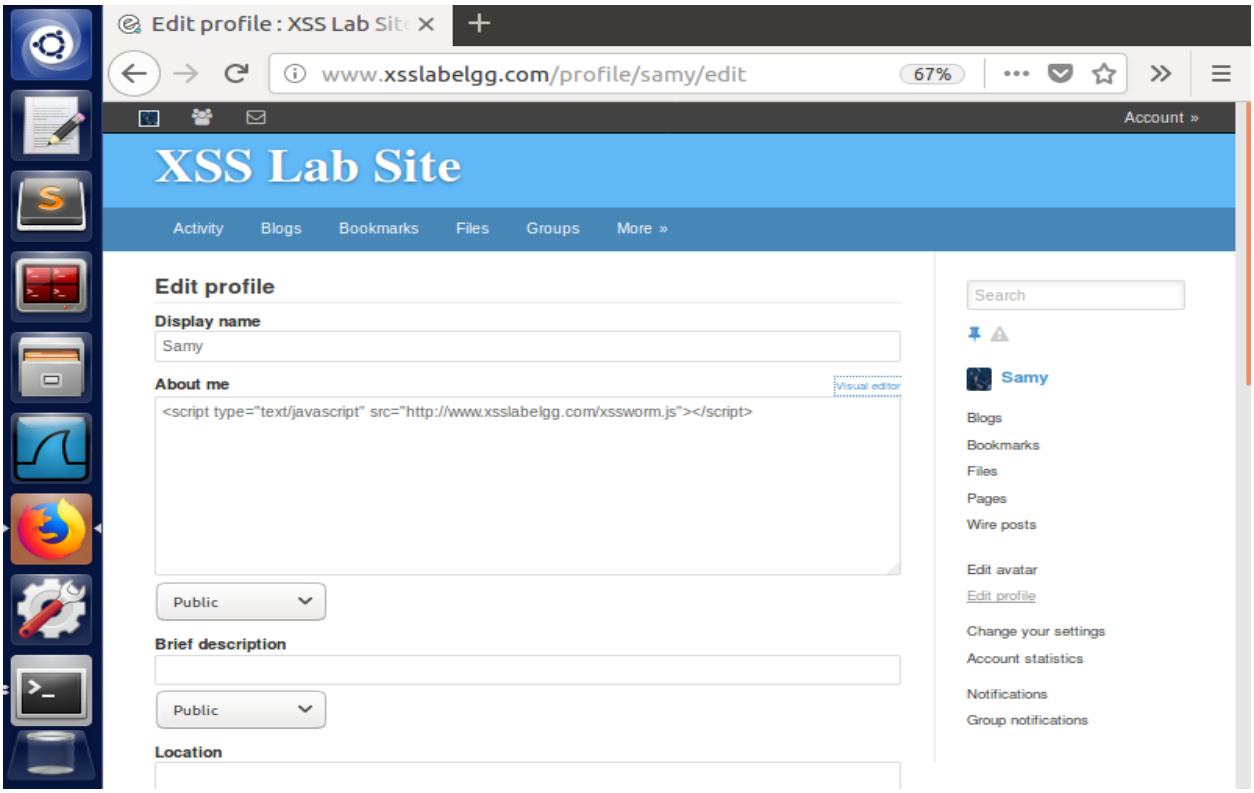
LINK Approach:

The purpose of this task is to make the XSS worm to self-propagate but by using the LINK approach. We copy the XSS worm code as given in Modules that Professor taught and maintain the rest of the code as such like the DOM approach. See screenshot below. We place this JavaScript code in the path /var/www/XSS/Elgg

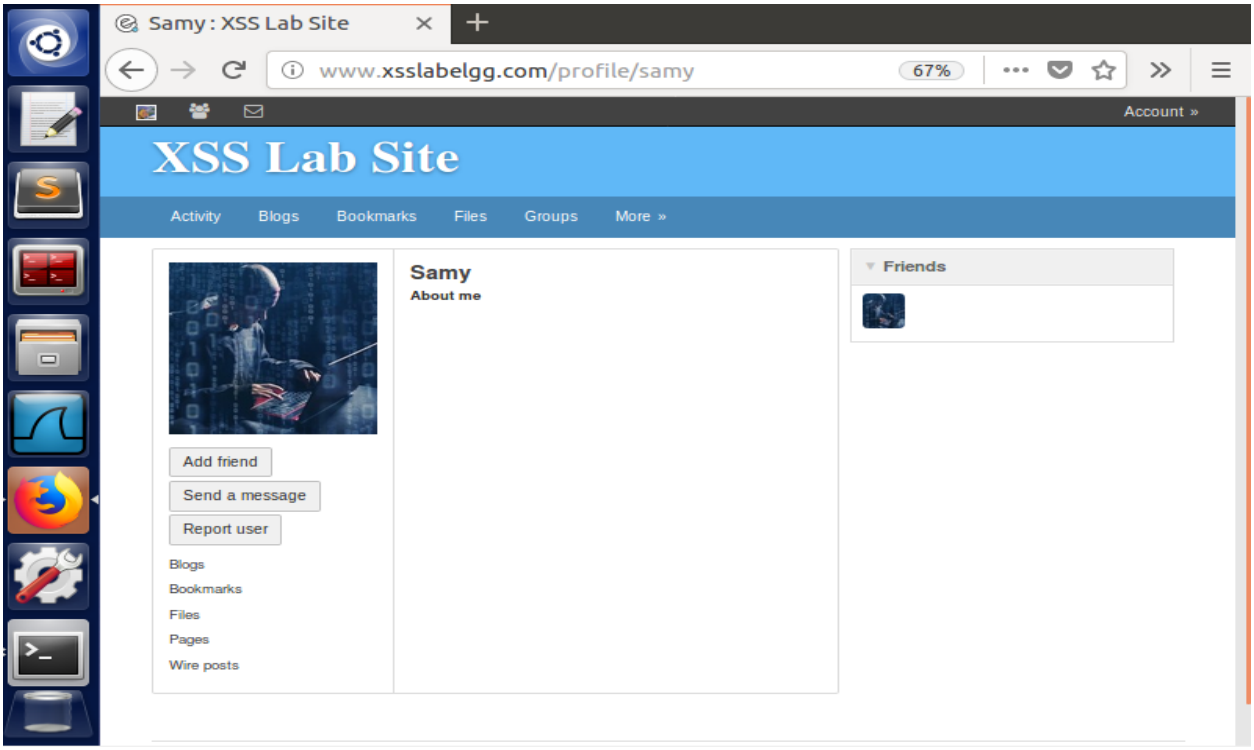


After placing this worm code in the specified path, a complete Apache 2 service restart must be done.

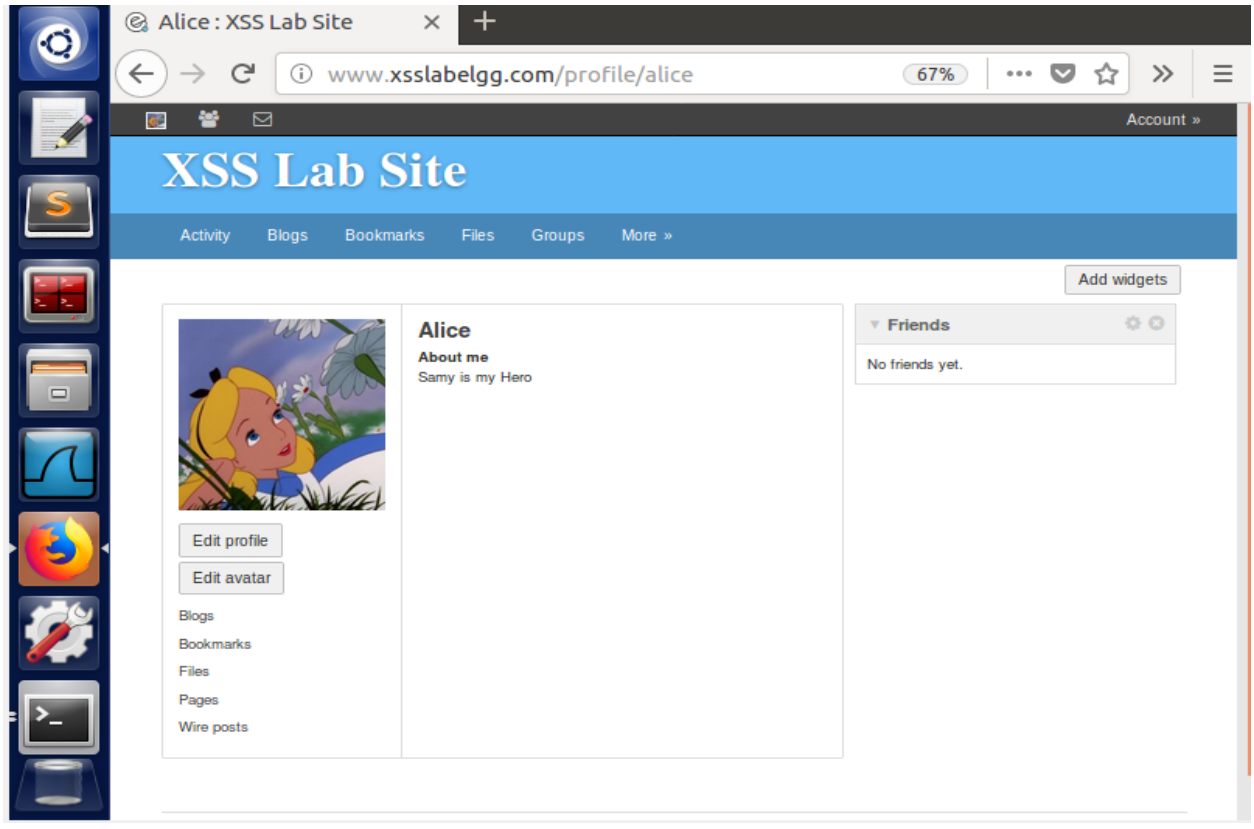


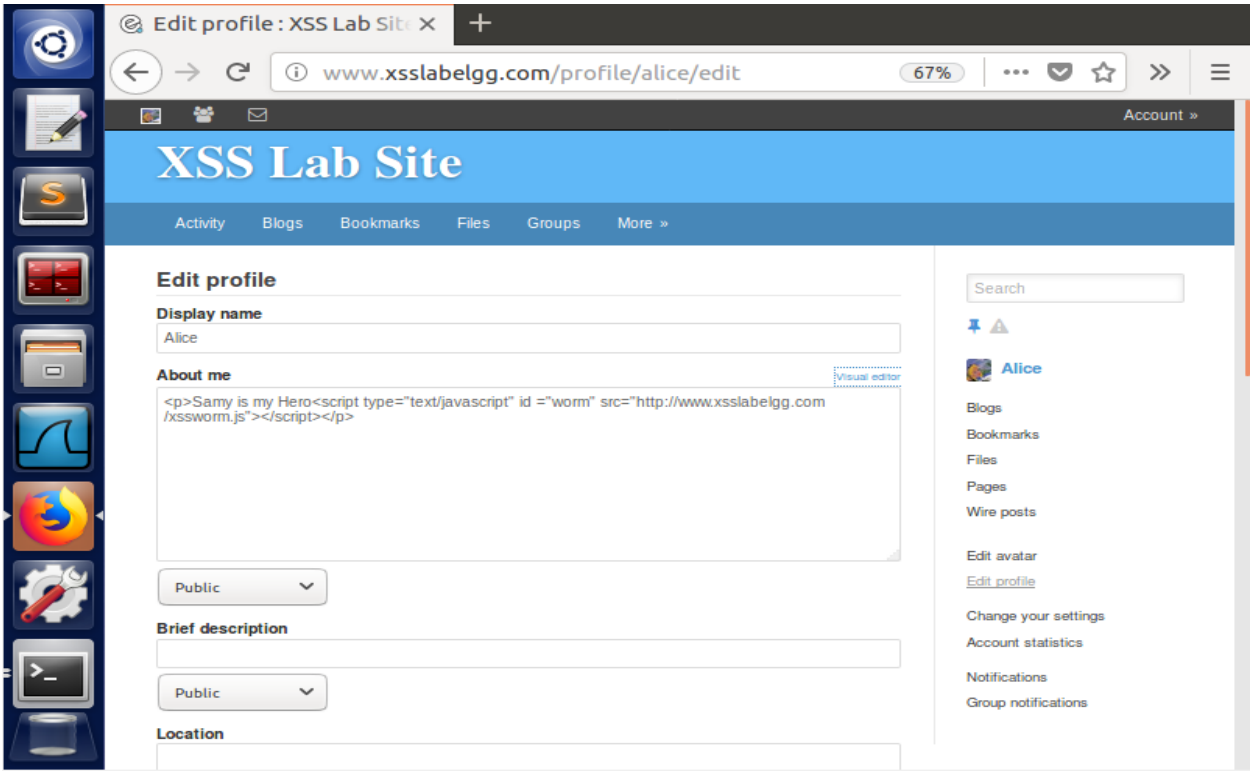


We then login as Samy. Samy edits his profile and places the script with the JavaScript code path, http://www.xsslabelgg.com/xssworm.js in the img tag src attribute in the Edit HTML field About me section. We then login as Alice. Alice goes and casually checks Samy's profile.

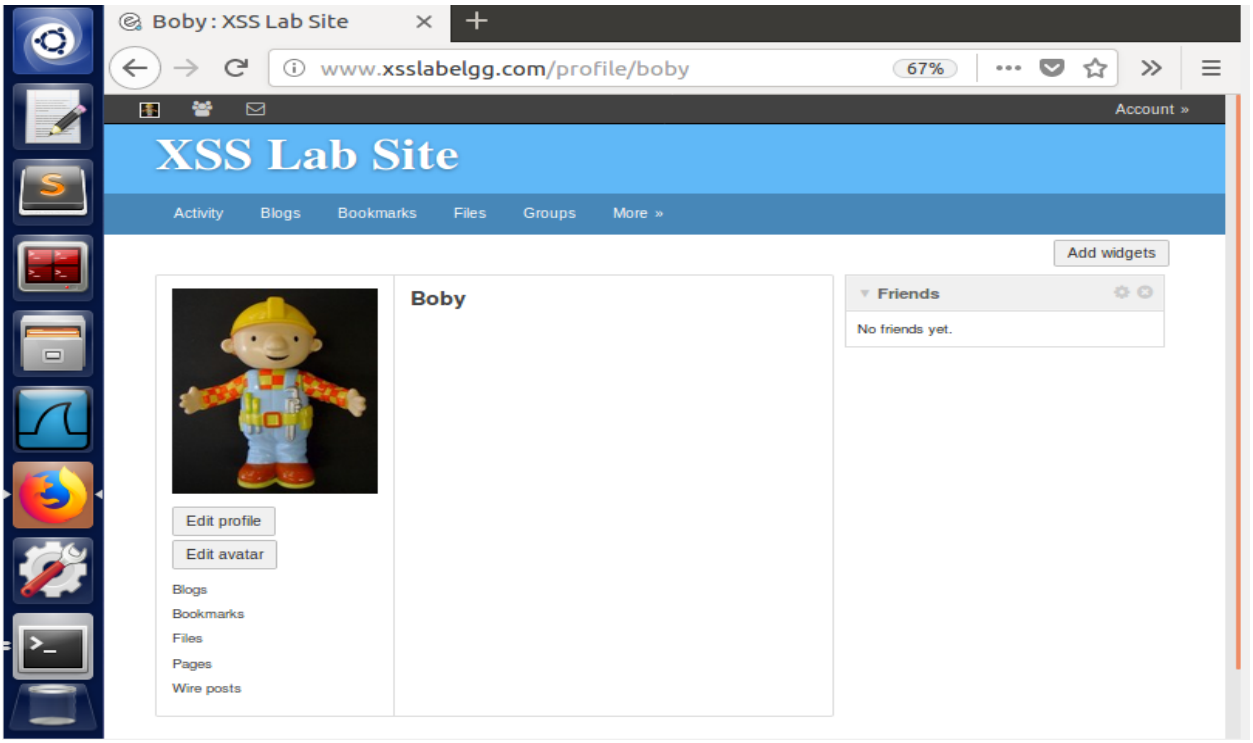


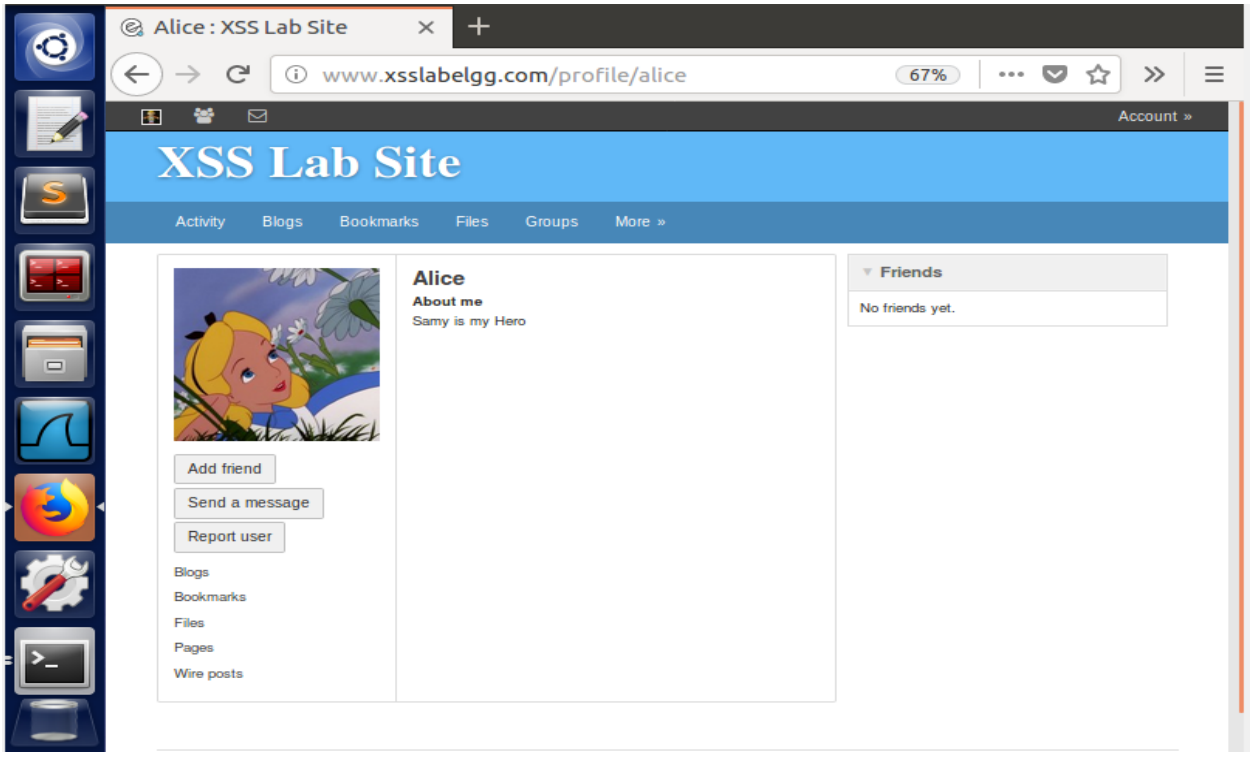
When Alice returns to her profile, her About me description has changed.



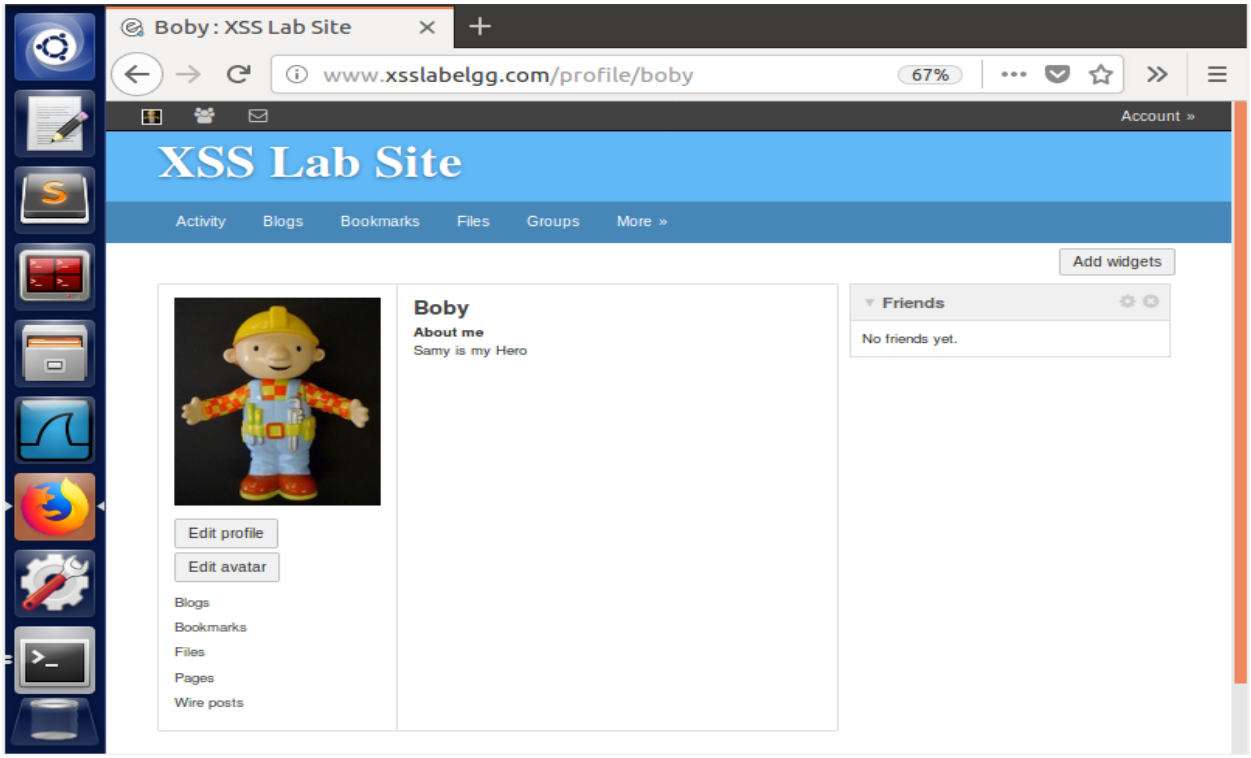


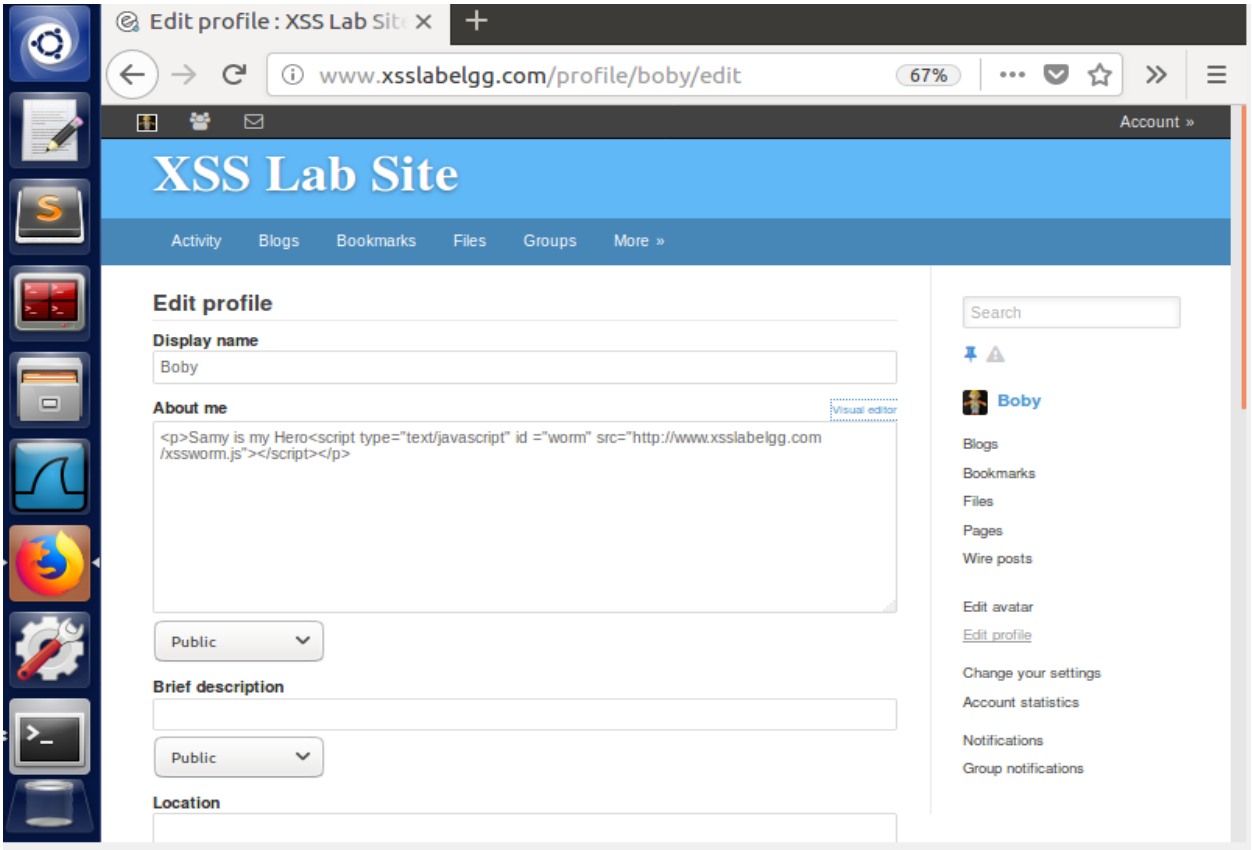
When she investigates what is wrong with her profile, she sees that About me section 'Edit HTML' has been infected with the worm and the exact JavaScript code is found in Alice's profile as we saw in Samy's profile. We then login as Boby.





Boby goes and checks Alice's profile. When he returns to his profile, he sees that his About me description has changed.





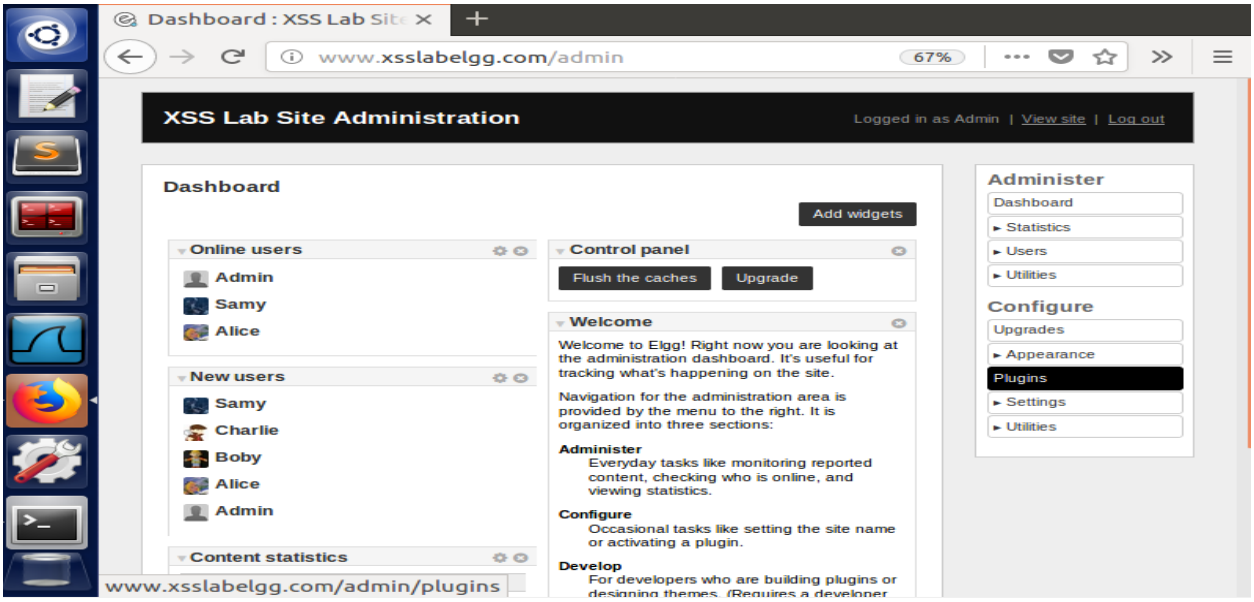
When he investigates by editing his profile, he finds that his profile has been infected by XSS worm and he finds a JavaScript code that we saw in Samy's and Alice's profile.

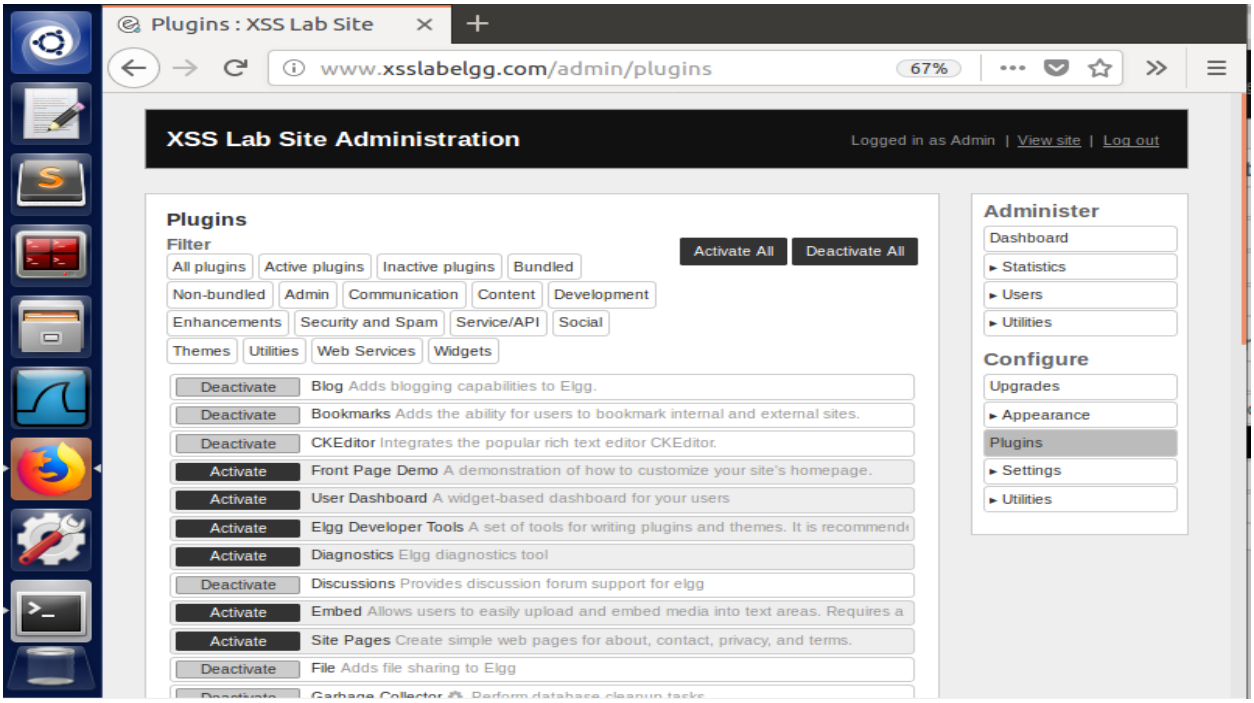
Conclusion: By using the LINK approach, we just paste the link to the JavaScript worm code in the attacker's profile and this worm travels from one profile to another based on the visits made on the infected profiles. Here, Samy is the attacker. He attacks Alice's profile and infects it and passes the XSS worm to Alice's profile. When Boby visits Alice's profile, Boby is also affected and carries the XSS worm to the next person who visits Boby's profile and so on. Unlike DOM approach, here, we place the worm code in the path /var/www/XSS/Elgg path and just give the link to this worm code in the attacker's profile.

3.8 Task 7: Countermeasures

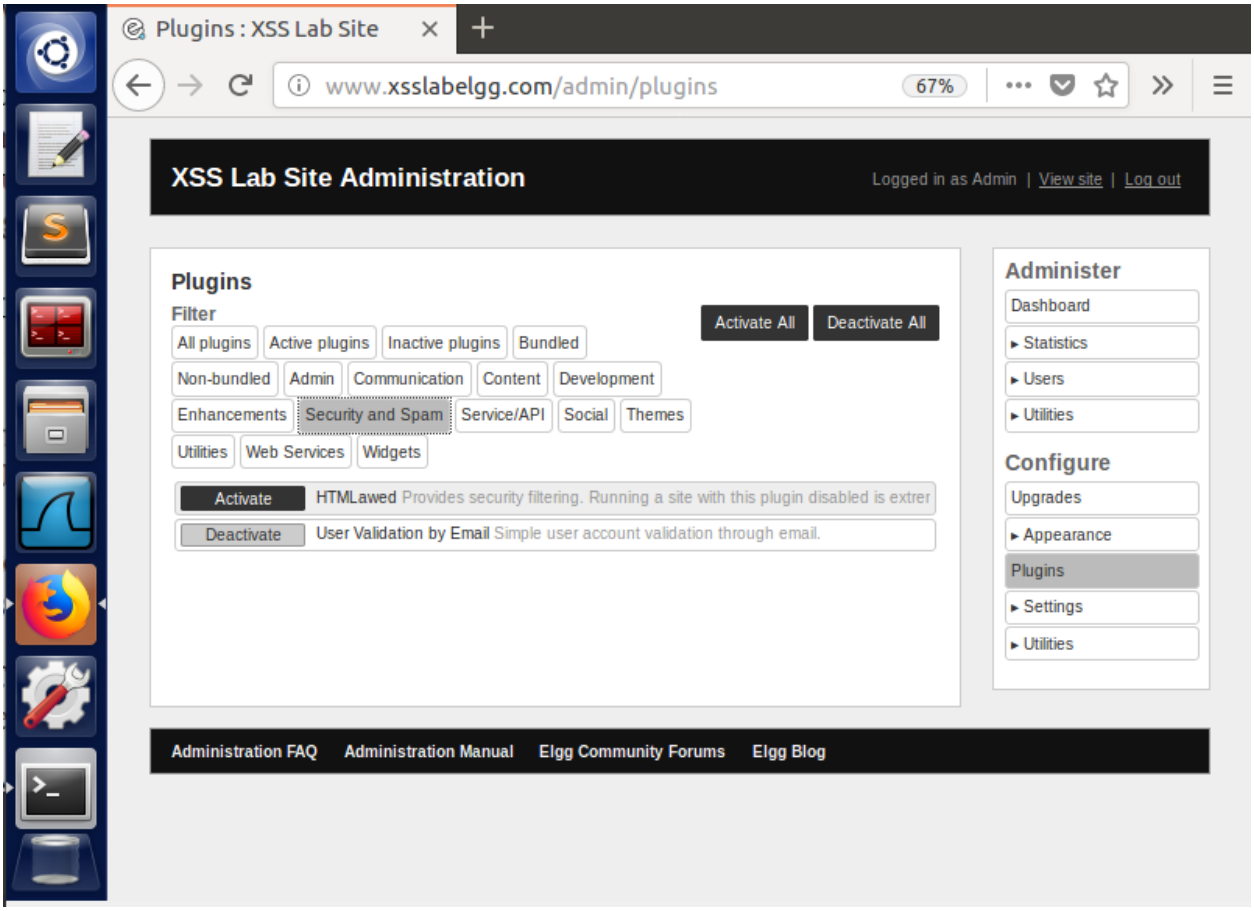
The purpose of this task is to turn on the countermeasure and perform the XSS attack using Smay's profile again. We check whether this attack works even after turning on the countermeasures.

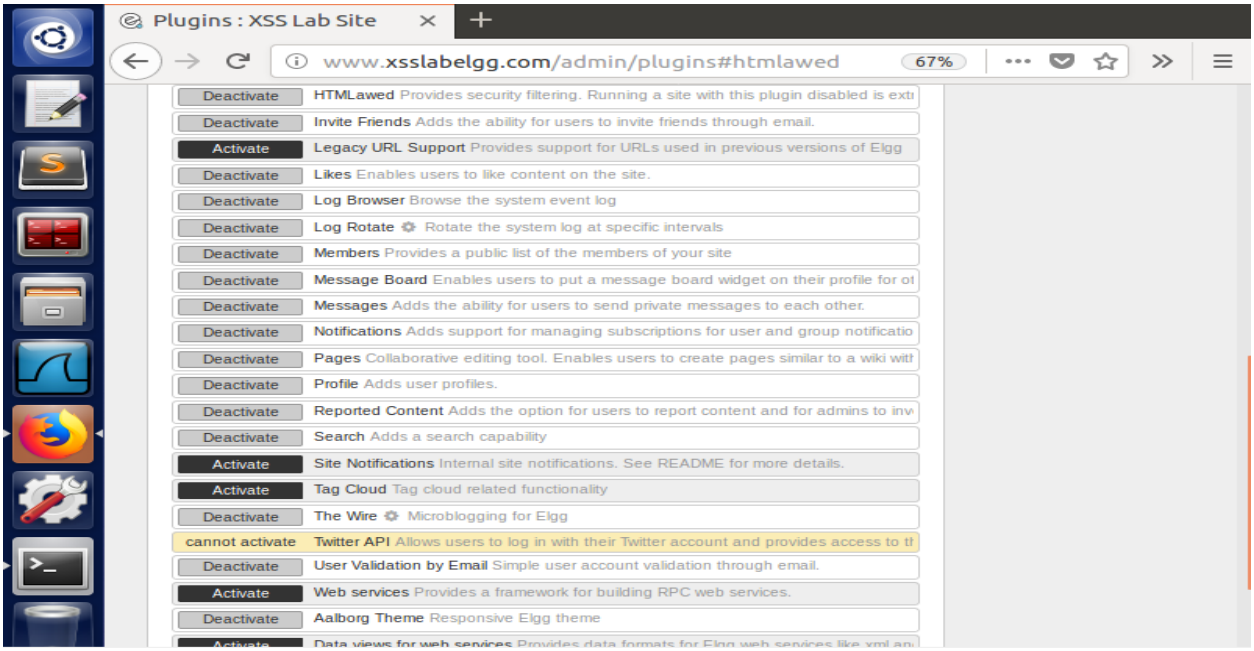
We login as Admin and go to Plugins



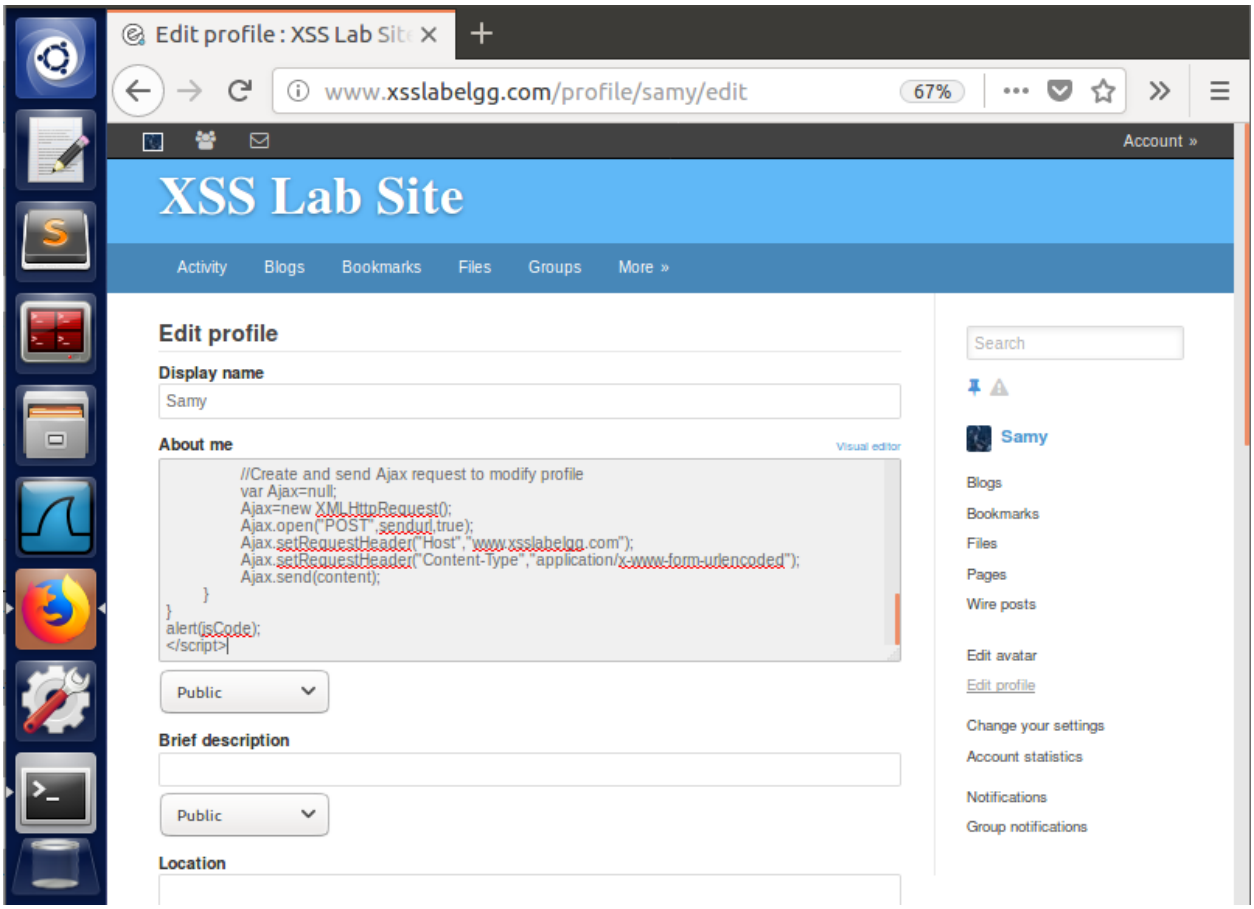


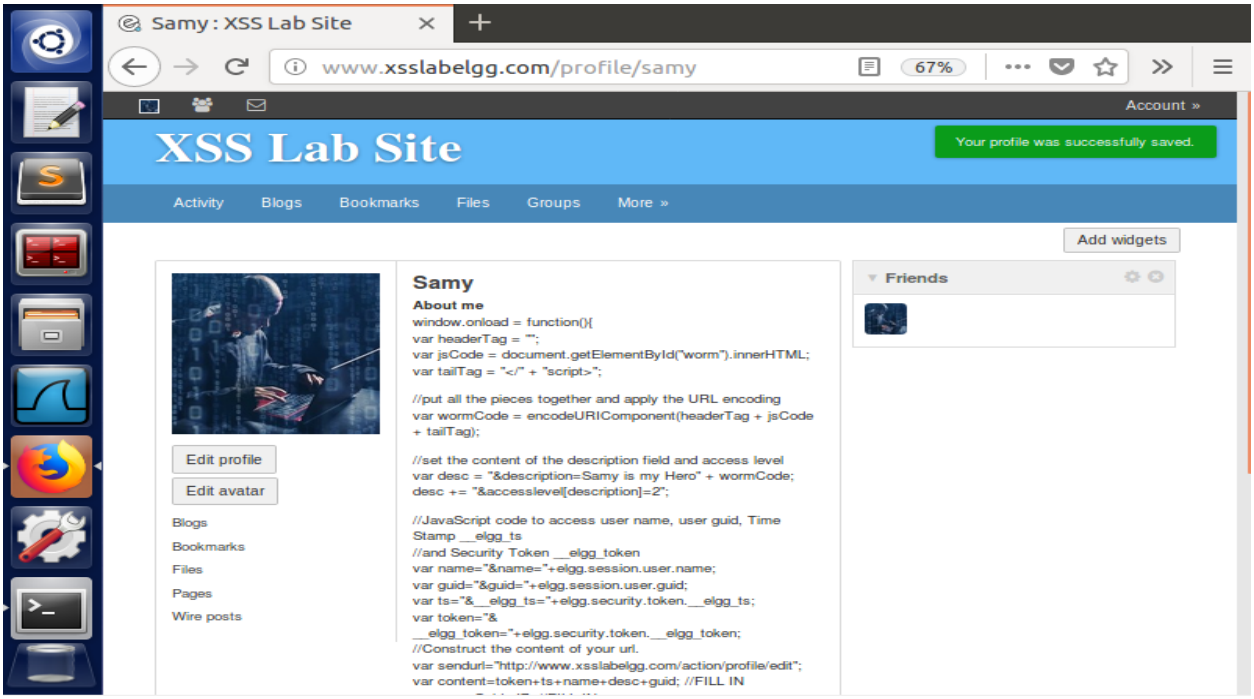
We select the Security and Spam button.



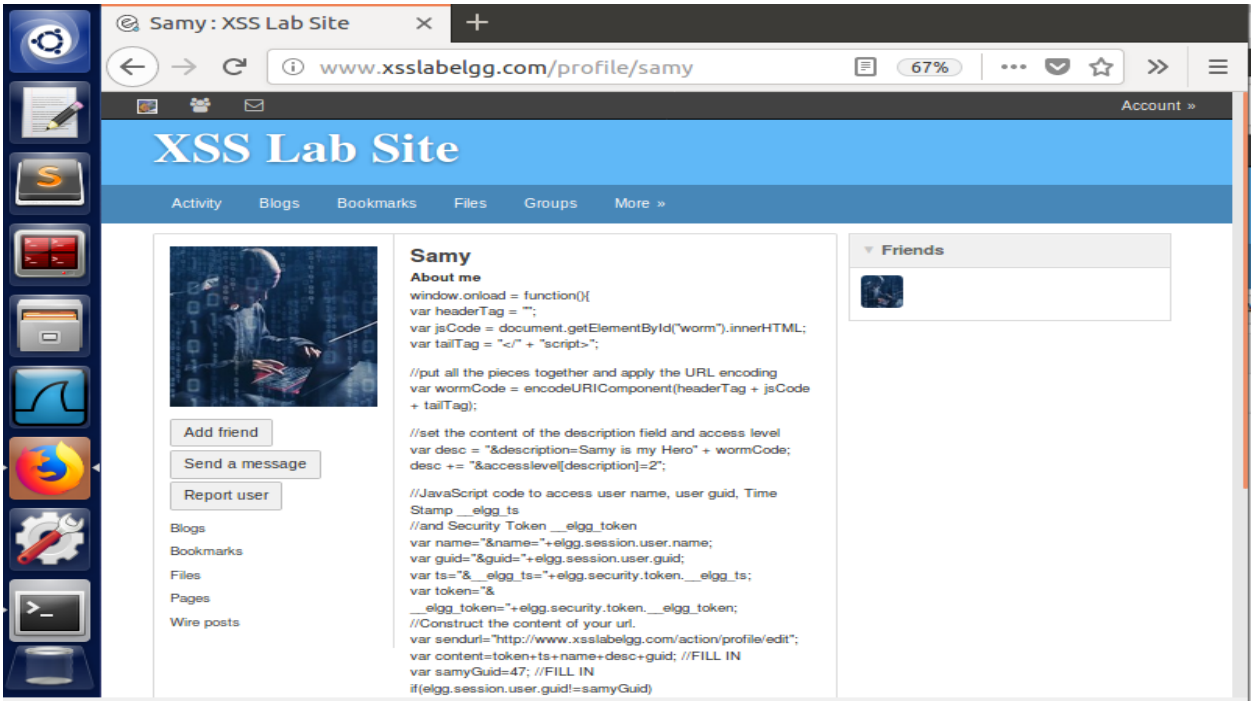


We Activate the Security and Spam option in Plugins. We login as Samy again. Samy writes the XSS worm code in the 'About me' Edit HTML section.

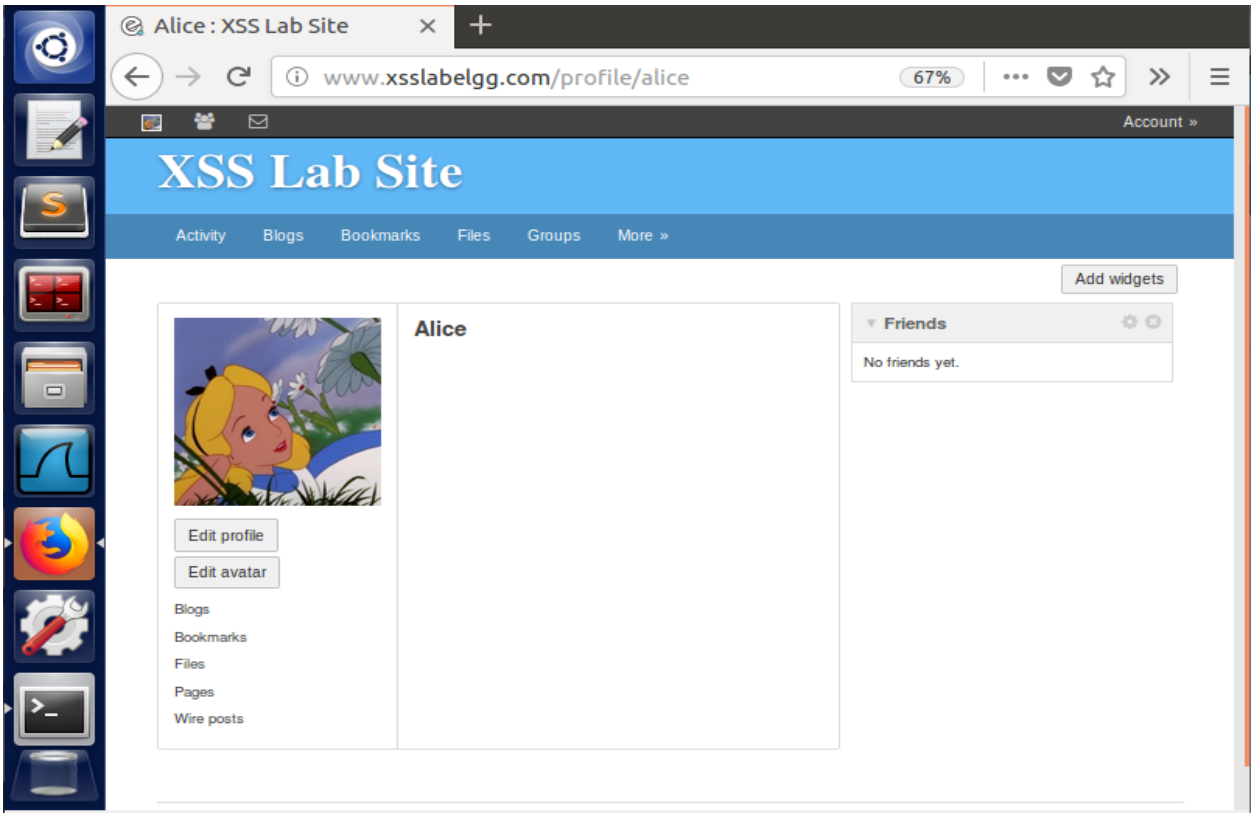


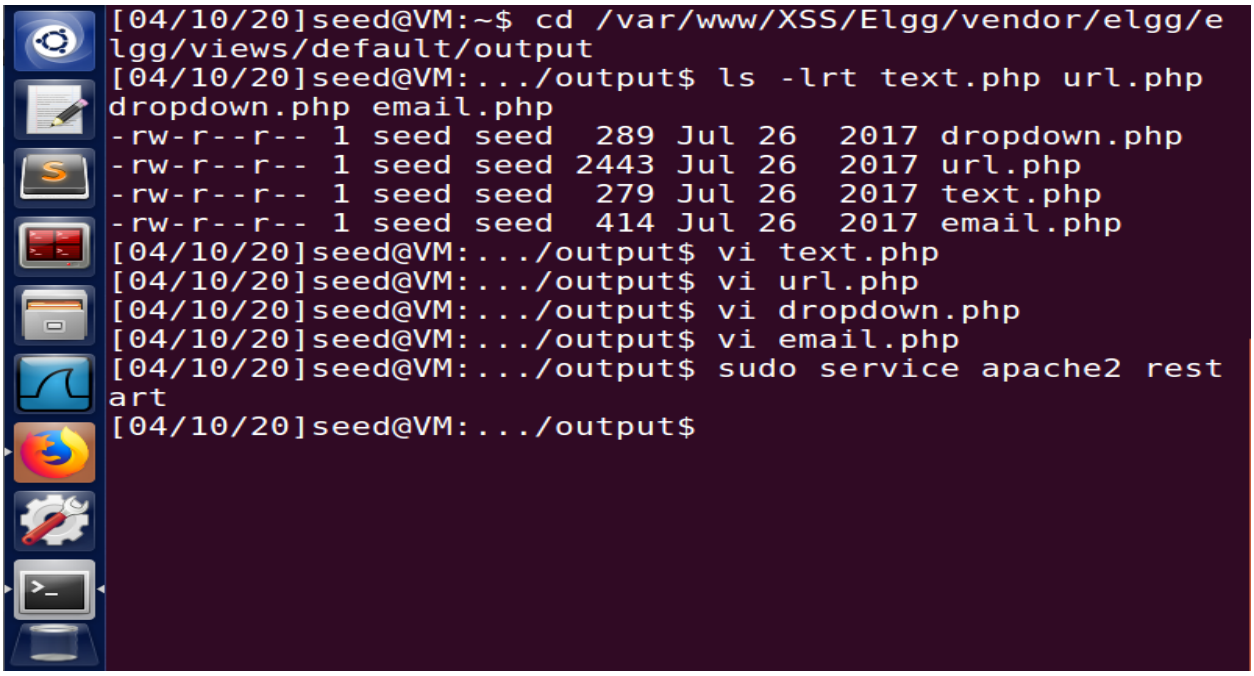


When Samy writes the code and saves his profile, he immediately sees that his code is displayed in form of text in 'About me' section. When he goes and edits the profile again, he sees that Edit HTML field in About me section is modified to replace special character such as '<' with &lt, '>' with &gt, normal single quotes with &quot, etc. This is the countermeasure that we have enabled.

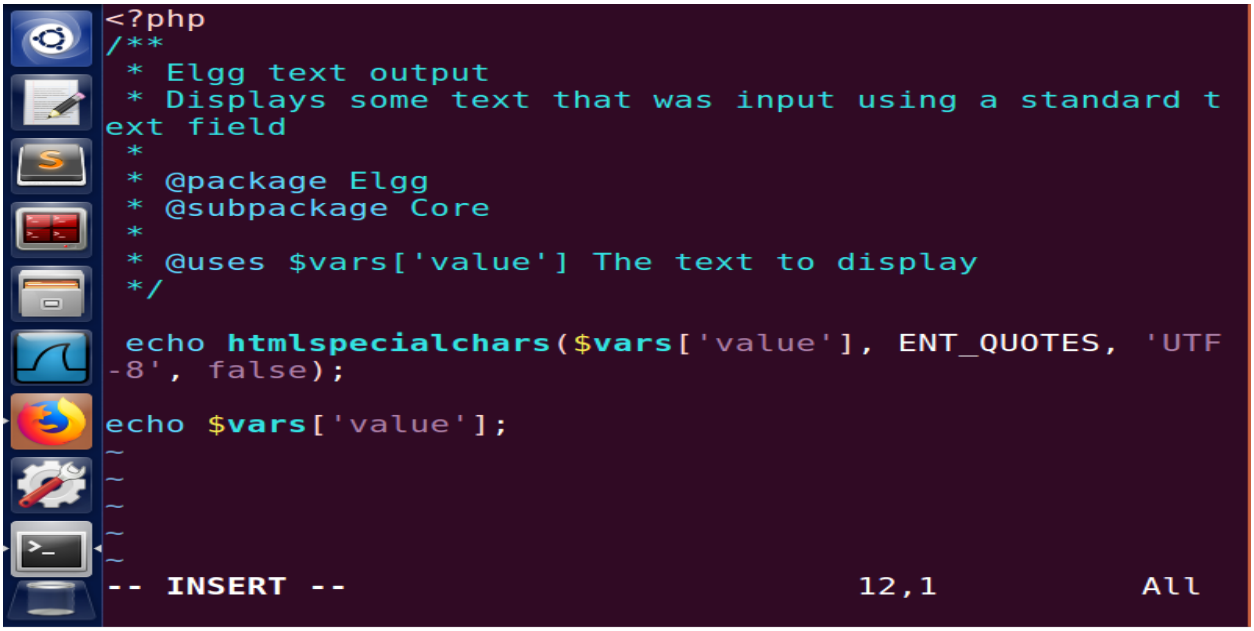


We then login as Alice. Alice goes and checks Samy's profile and returns to her profile to see no changes in her profile.

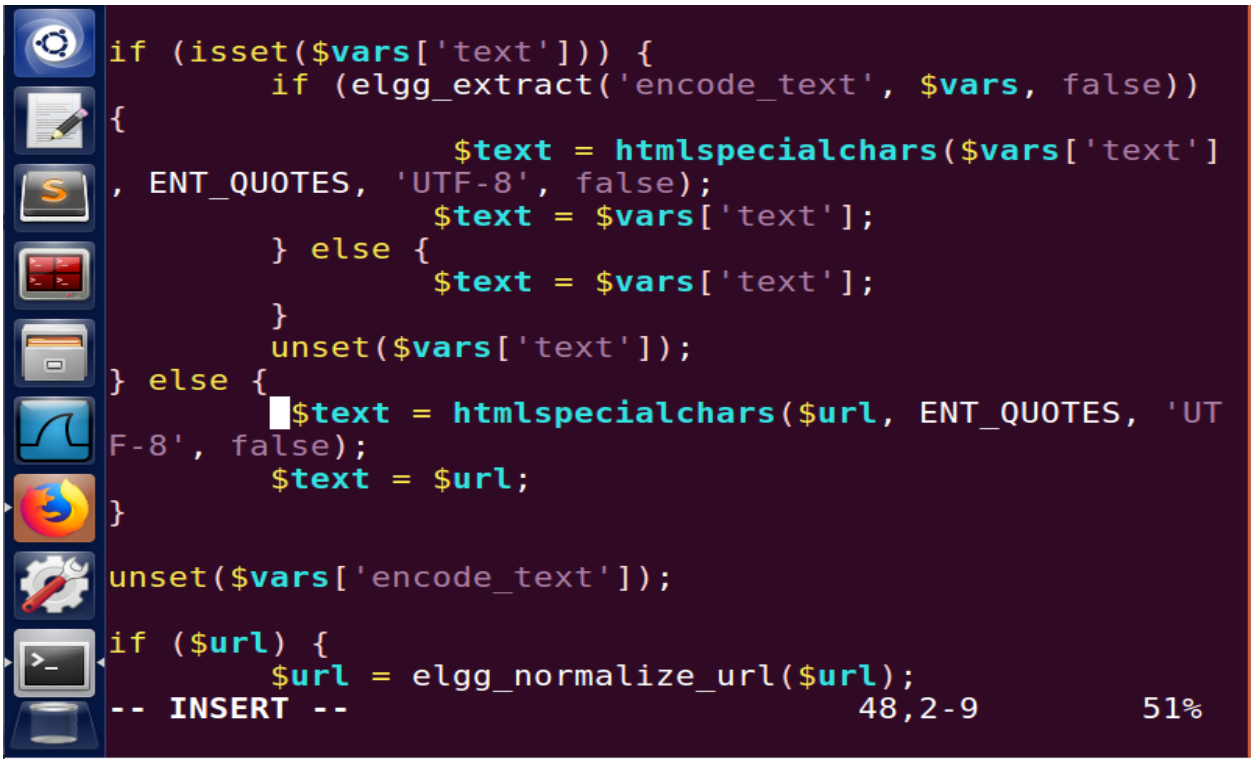




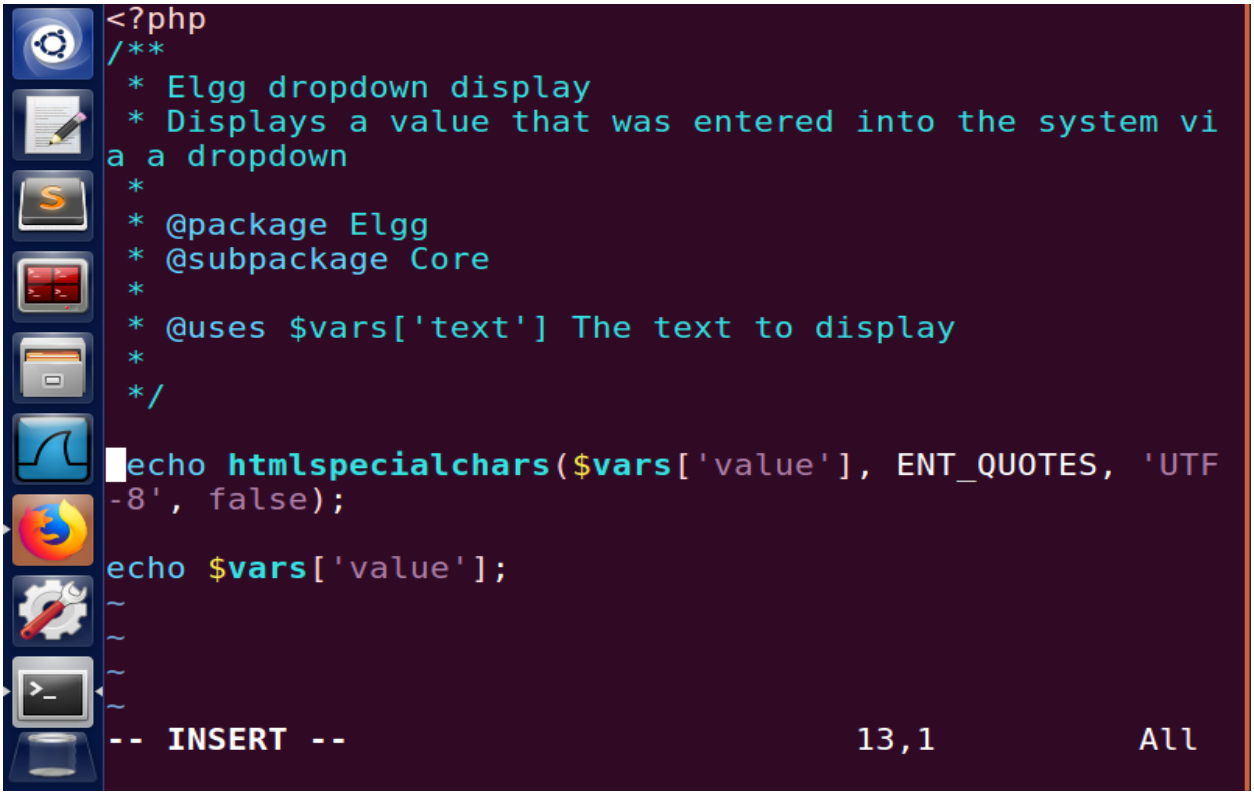
We then go to /var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output folder to check if we have the four PHP files (text.php, url.php, dropdown.php and email.php). We edit these files to encode special characters using htmlspecialchars() function.



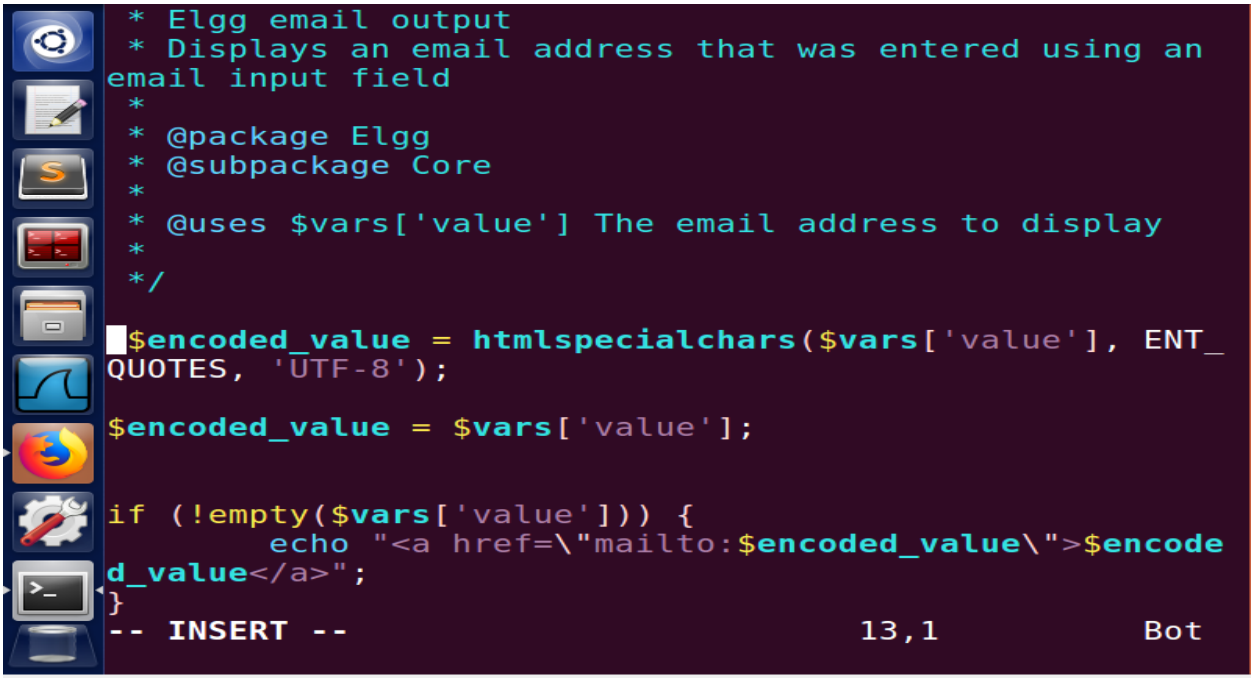
```
<?php
/**
 * Elgg text output
 * Displays some text that was input using a standard t
ext field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */

 echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF
-8', false);

echo $vars['value'];
~
~
~
~
~
-- INSERT --                              12,1           All
```



```
if (isset($vars['text'])) {
        if (elgg_extract('encode_text', $vars, false))
{
                $text = htmlspecialchars($vars['text']
, ENT_QUOTES, 'UTF-8', false);
                $text = $vars['text'];
        } else {
                $text = $vars['text'];
        }
        unset($vars['text']);
} else {
        $text = htmlspecialchars($url, ENT_QUOTES, 'UT
F-8', false);
        $text = $url;
}

unset($vars['encode_text']);

if ($url) {
        $url = elgg_normalize_url($url);
-- INSERT --                              48,2-9         51%
```

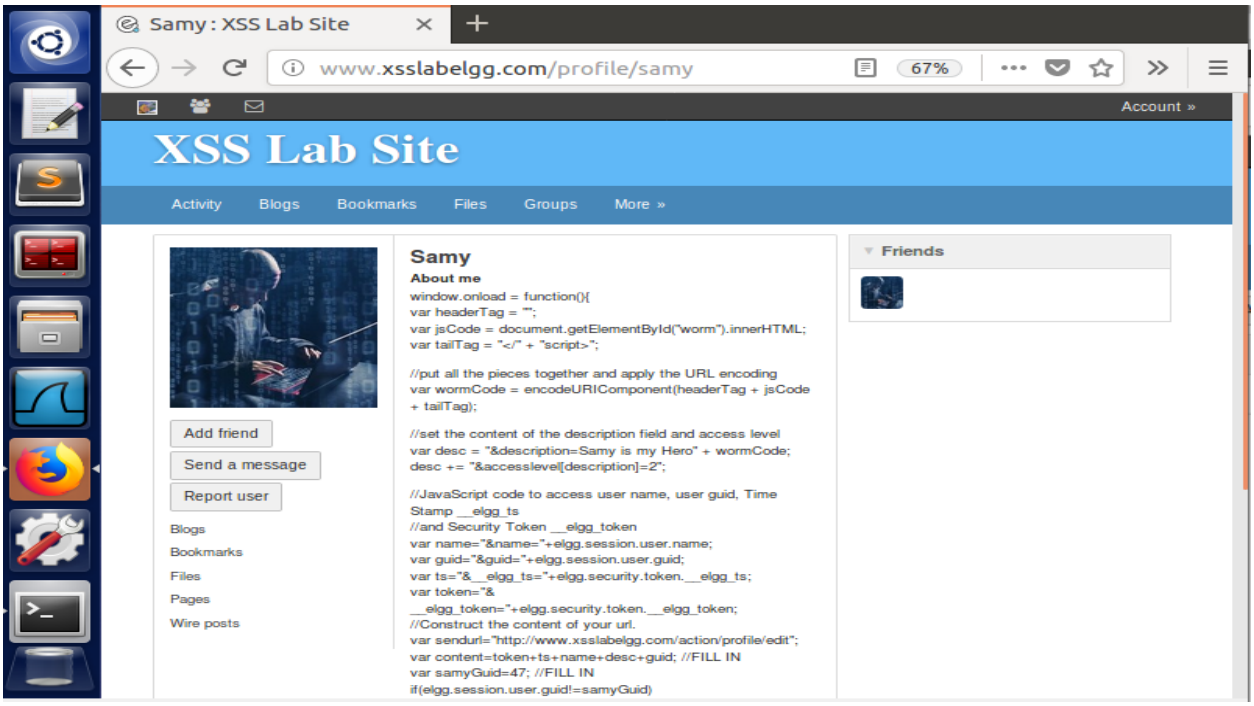


```
<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system vi
a a dropdown
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['text'] The text to display
 *
 */

echo htmlspecialchars($vars['value'], ENT_QUOTES, 'UTF
-8', false);

echo $vars['value'];
~
~
~
~
-- INSERT --                          13,1          All
```

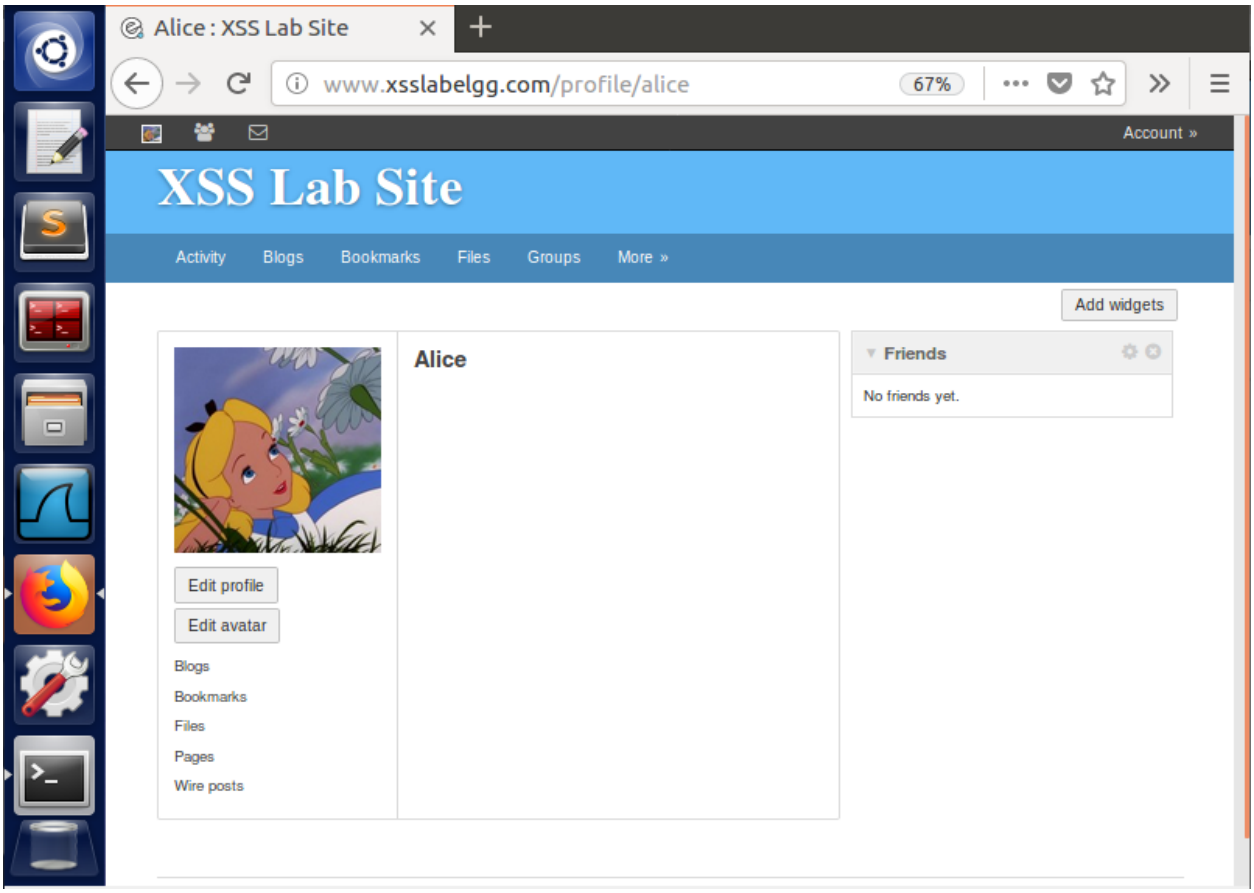


```
 * Elgg email output
 * Displays an email address that was entered using an
email input field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The email address to display
 *
 */

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

$encoded_value = $vars['value'];


if (!empty($vars['value'])) {
        echo "<a href=\"mailto:$encoded_value\">$encode
d_value</a>";
}
-- INSERT --                          13,1          Bot
```

After making the changes we login as Alice. Alice goes and visits Samy's profile.



Alice returns to her own page to see no changes to her profile.



Conclusion: When we uncomment all the htmlspecialchars() function, the XSS worm attack does not work. This is because the htmlspecialchars() function encodes special characters like '<' '>' and the start of all script tags have these symbols. When this encoding happens, the script tags do not get executed and the JavaScript code just turns into normal text being written in the profile. Thus, the countermeasure turned on, works properly and does not allow the XSS worm attack to occur.

References:

https://github.com/aasthayadav/CompSecAttackLabs/blob/master/9.%20XSS%20Attack/Lab%209%20XSS%20Attack.pdf