Jorge Avila

Professor Trey

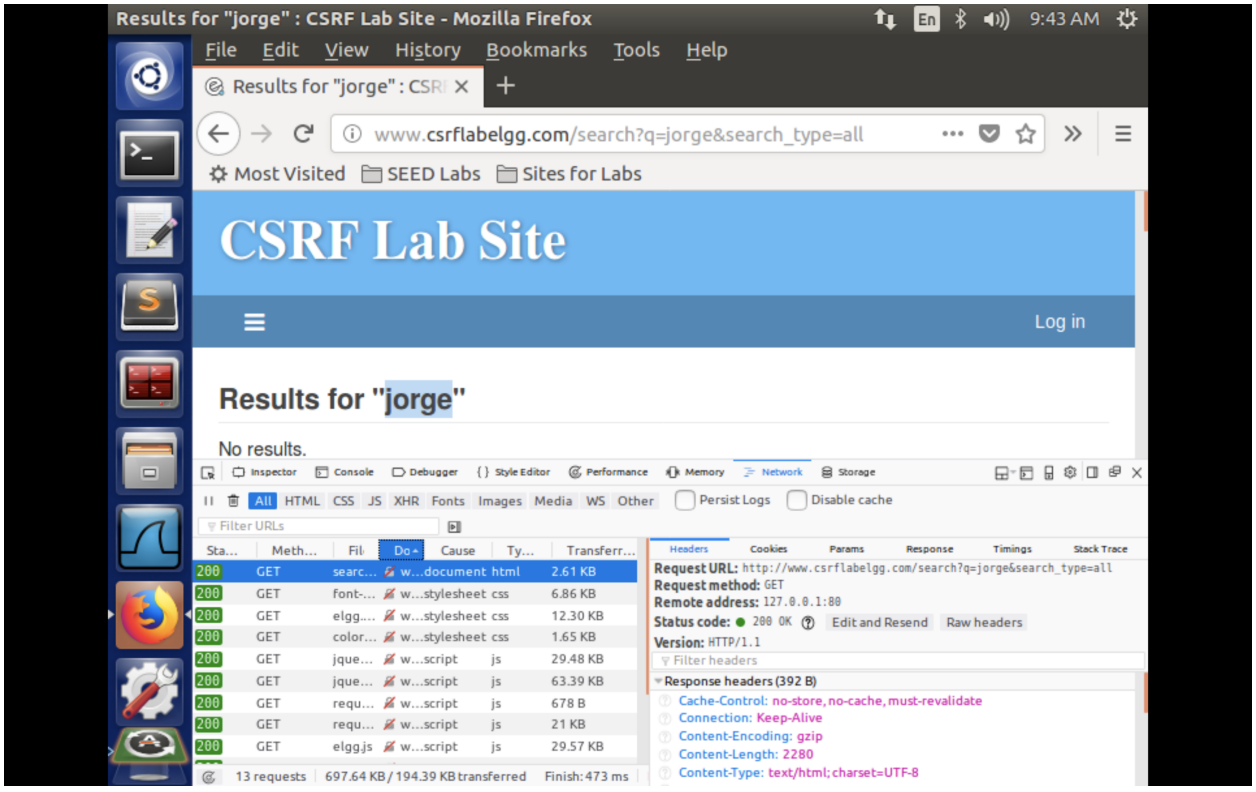
Secure Programming

02 April 2021

Cross Site Request Forgery Attack Lab

Task 1:

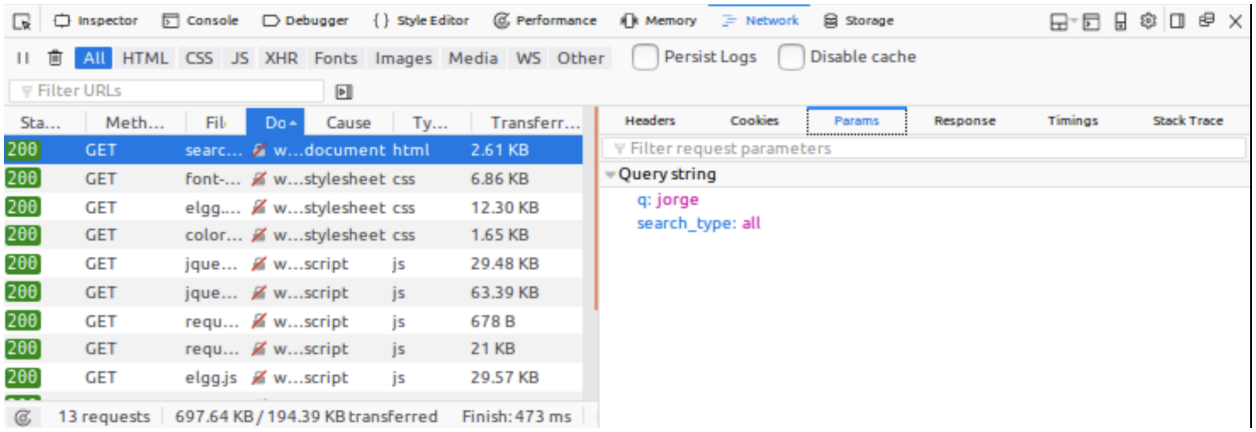
In this task we are going to observe an HTTP request and see how it works on the search engine developer tools. May use console or the actual built in add on on firefox called HTTP Header Live.



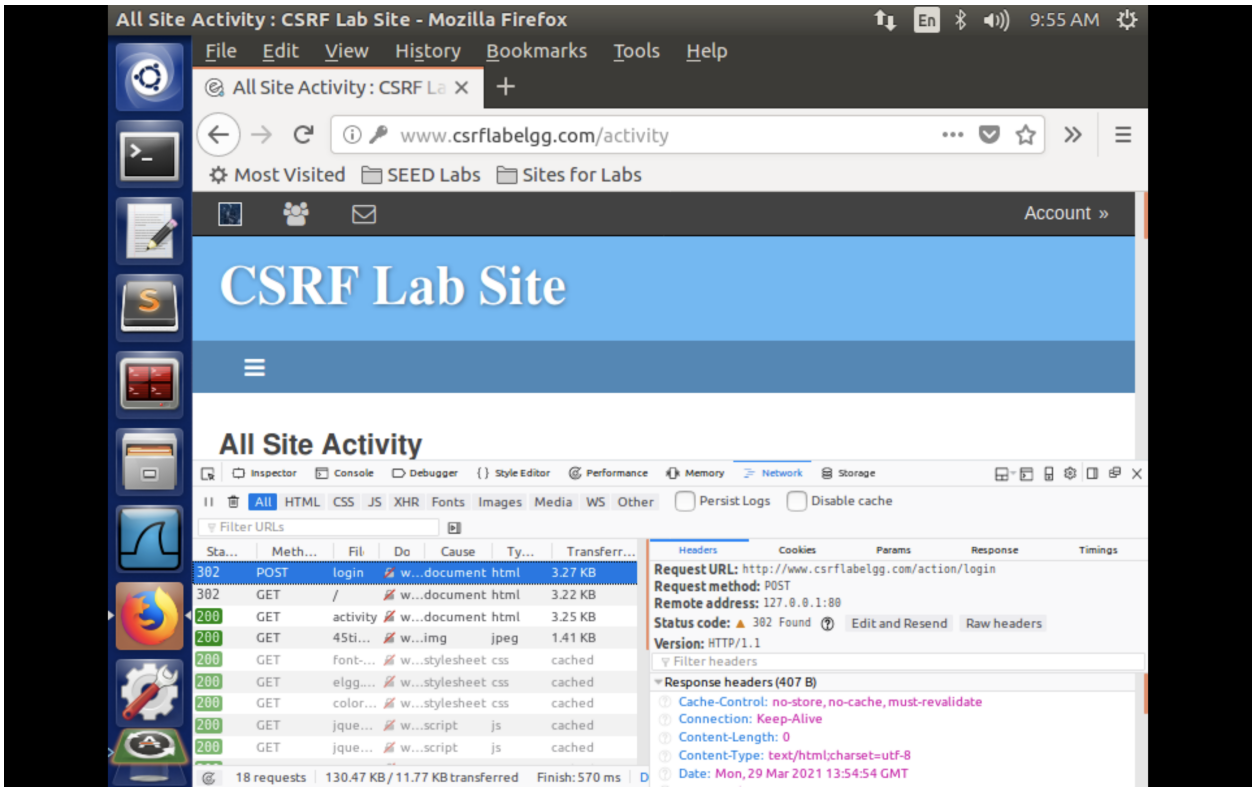
I decided to use the console and do a query on my name. As you can see it a method of GET was called on a request URL as

http://www.csrflabelgg.com/search?q=jorge&search_type=all

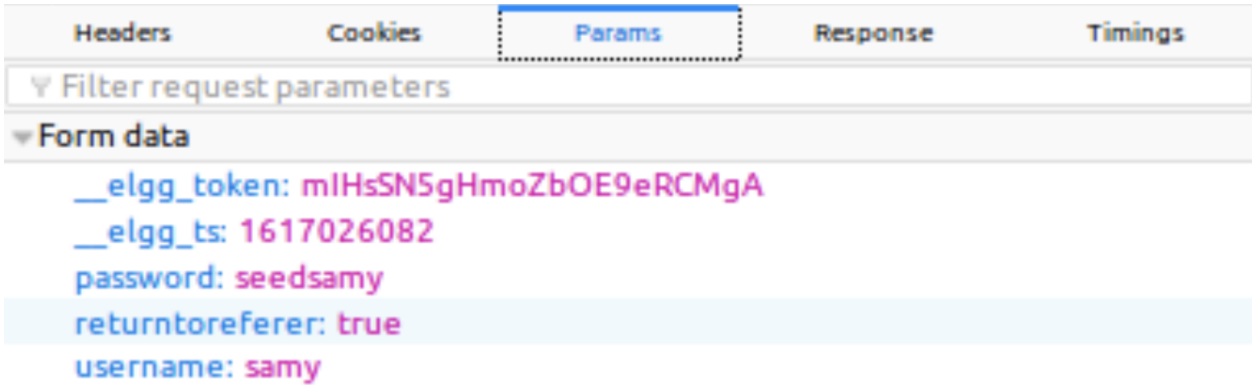
The parameters are shown below:



Next, we are going to do a POST request and to first do that we are now going to sign in as Samy



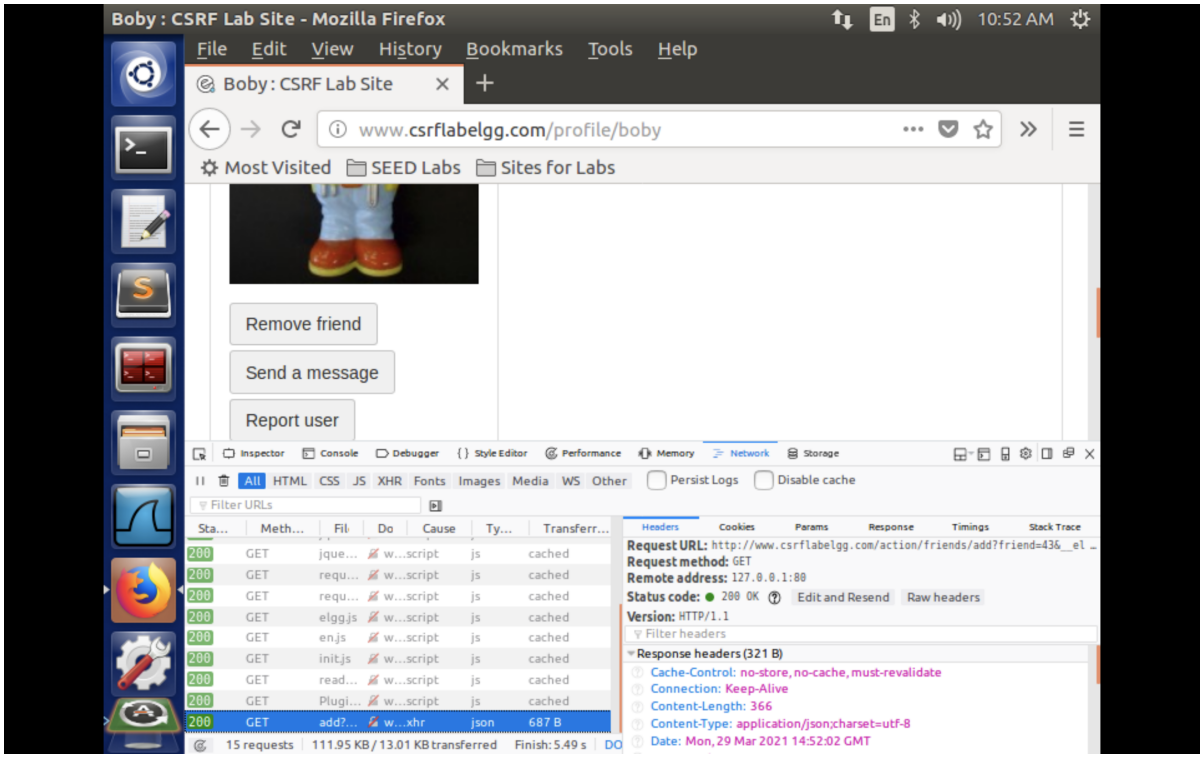
As soon as we log in, we see a POST request that we are now logged in.



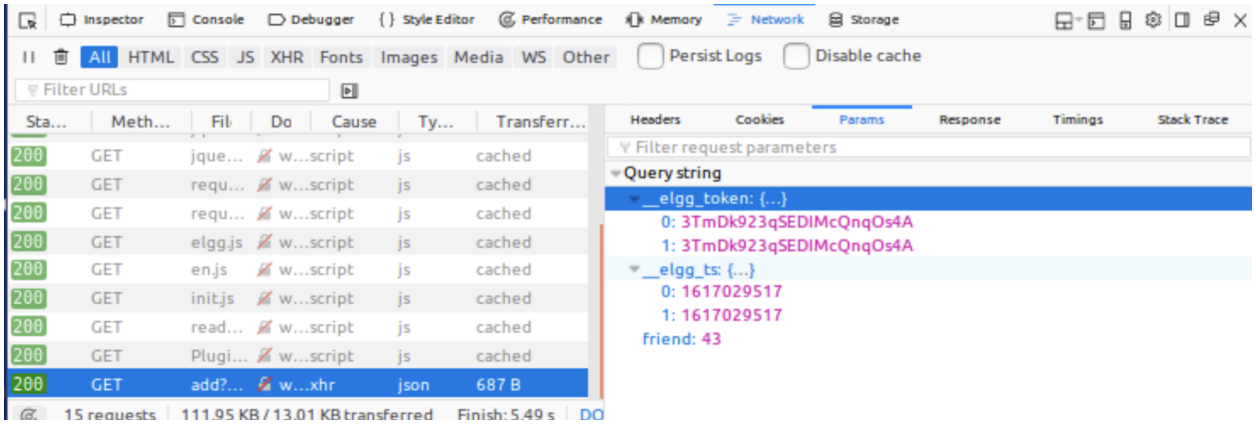
Above you can also see the parameters that present in the POST Request made by us. As well as the cookies that are sent to us, since HTTP is stateless, *cookies* makes it seem that everything is in state as talked about in lecture.

Task 2:

In this attack we need Boby (us) to add Alice forcefully to our friend list. In order to do that you create a fake account, this account is called Charlie which was already created for us then you find out how the add friend button works so that you can add the target to your list of friends.



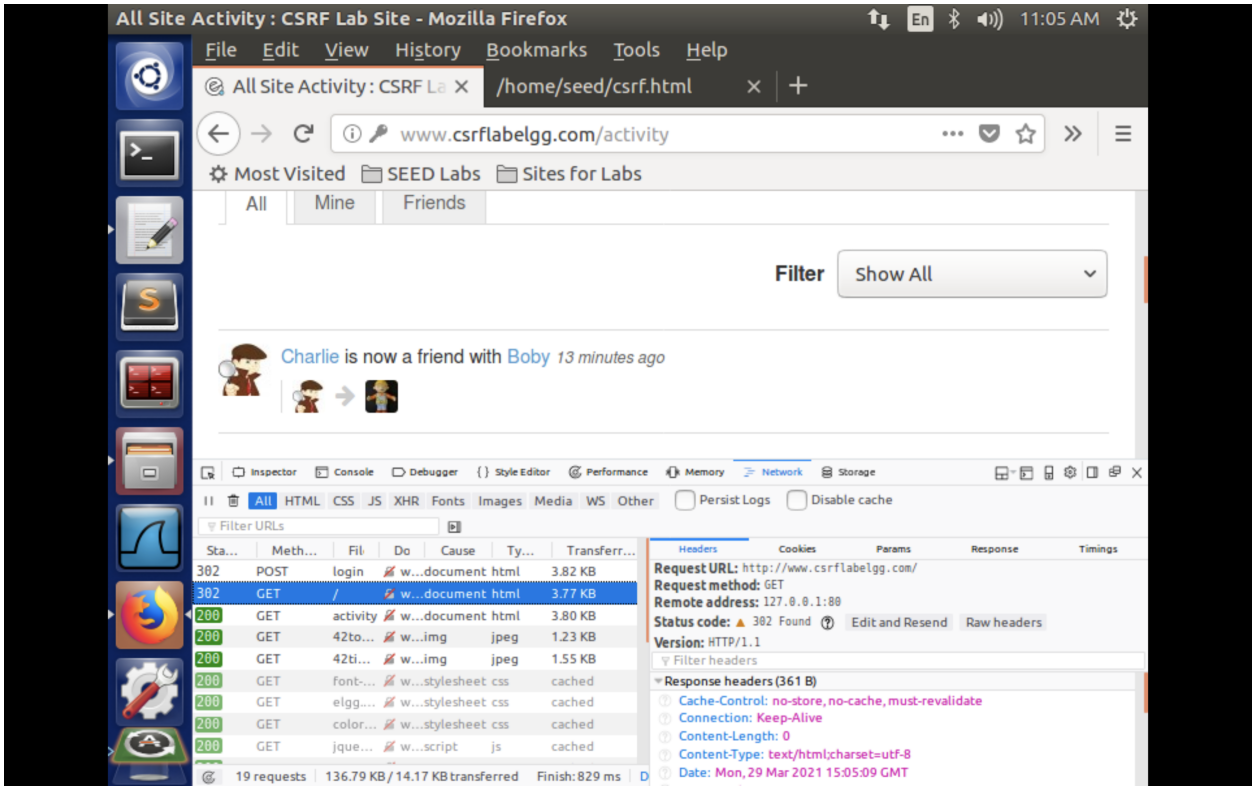
As you can see the GET request method was done with the URL indicating that add?friend=43 which 43 indicates is Boby's ID.



You can also see other information like the parameters used for this.

Now in order for this to work and we know what Bob's ID is, we are going to create an HTML file that somehow (and for the purposes of the lab) we trick Alice using social engineering to

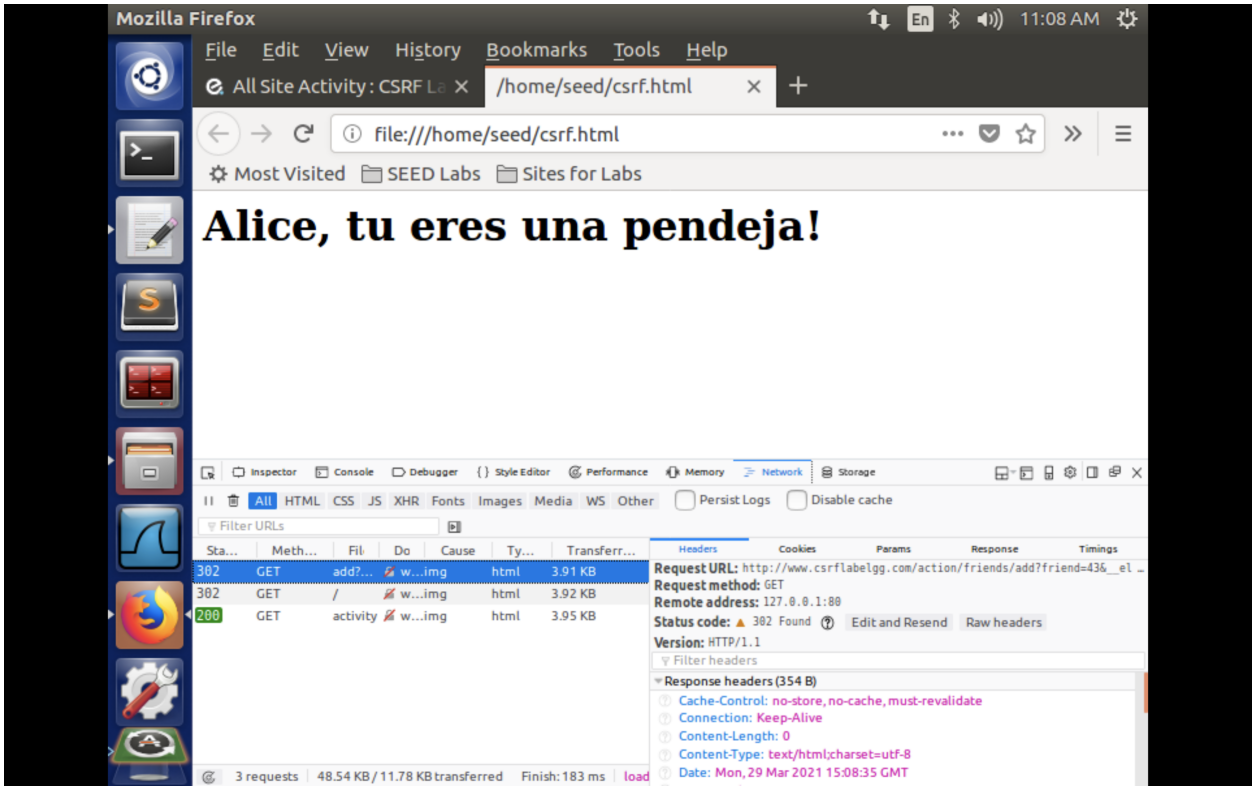
open this link while she is signed in and it will make a GET request to add Bob as a friend without her even knowing it.



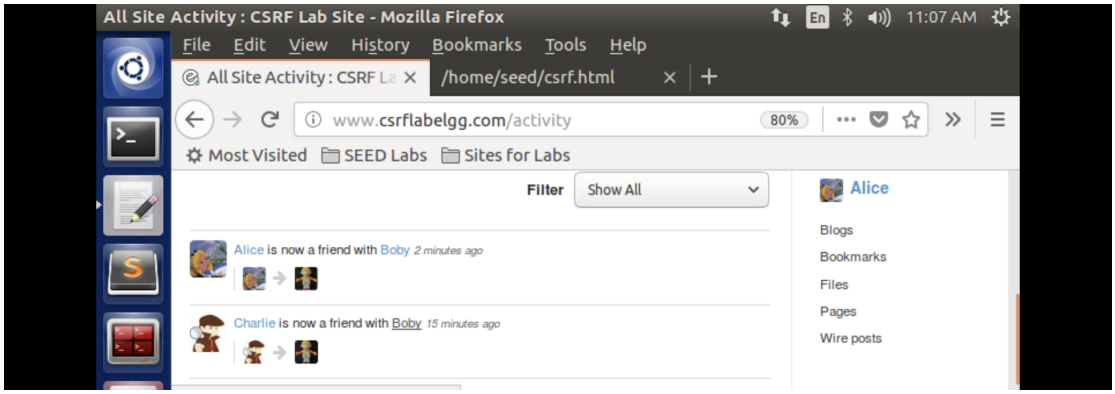
At first you do not see that Alice has any friends, a prerequisite for this to work is that she has to have a live session. Once Alice opens that HTML file with the GET request URL, she now has a friend of Bob

```
<html>
  <body>
    <h1> Alice, tu eres una pendeja! </h1>
     <img src="http://www.csrflabelgg.com/action/friends/add?
friend=43&__elgg_ts=1617029517&__elgg_token=3TmDk923qSEDIMcQnqOs4A&__elgg_ts=1617(
width="1" height="1">
  </body>
</html>
```

And since we are not allowed to use javascript, I used the <img> tag as shown above.



Alice, then clicks on the html that I created as shown above, then the bottom occurs:

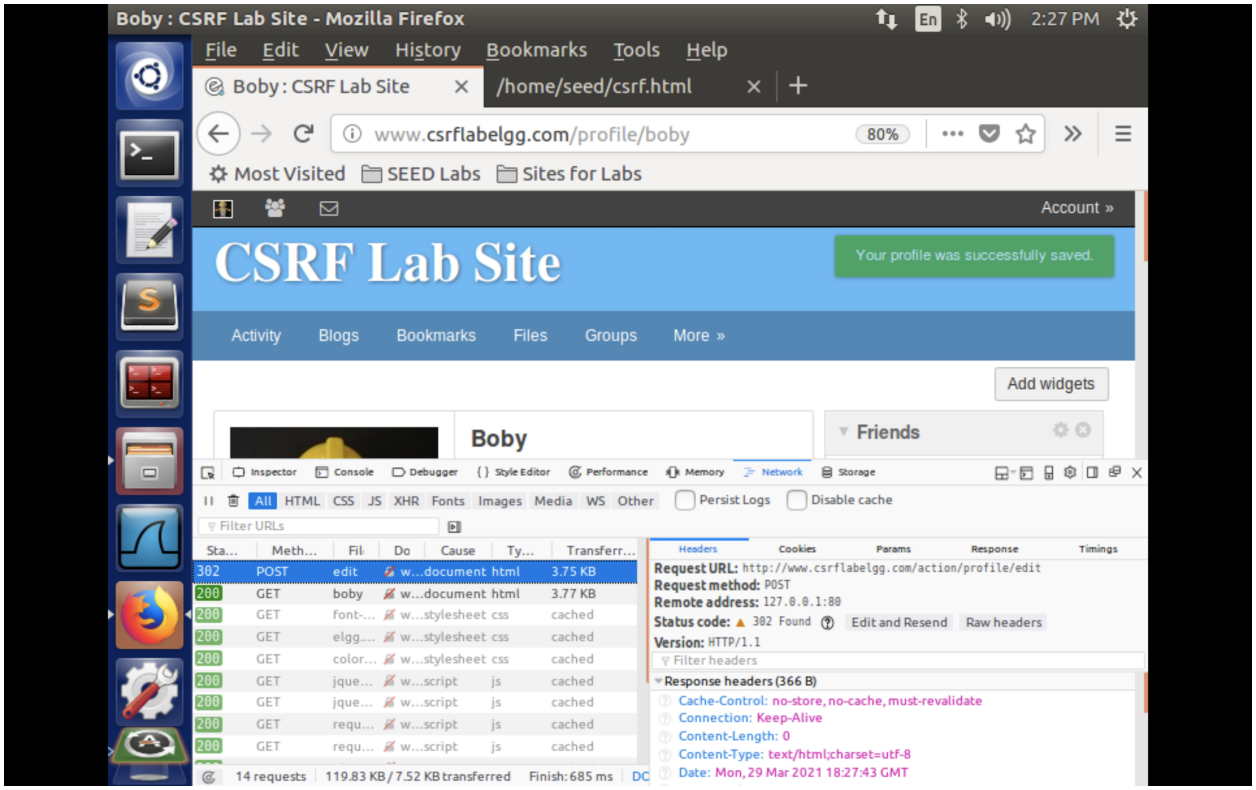


Then as you can see Alice is now a friend of Boby.

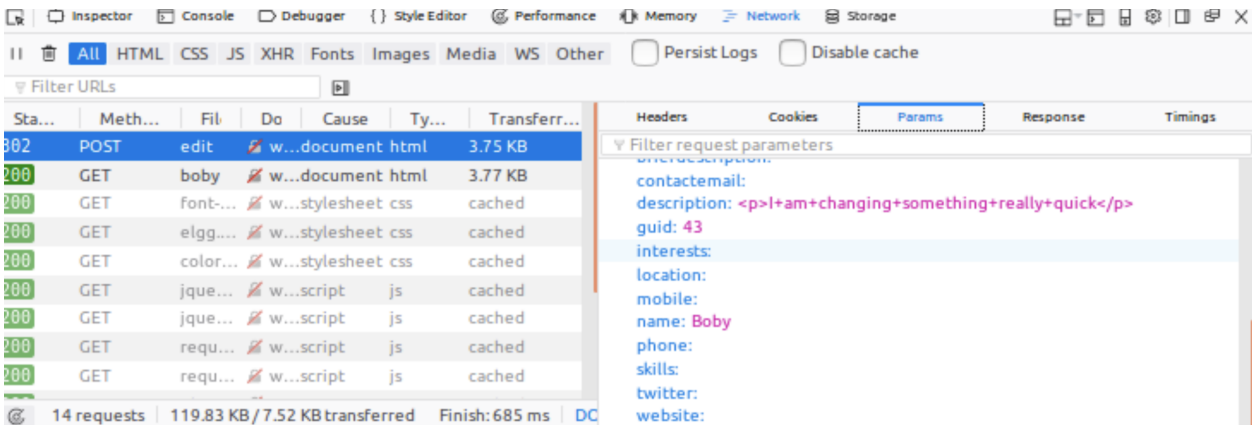
Task 3:

Next, in this task we are going to change some of Alice's information and say in the description box that "Boby is my hero". We know that Alice will never do this, but Boby will go ahead and do this attack using the POST Request.

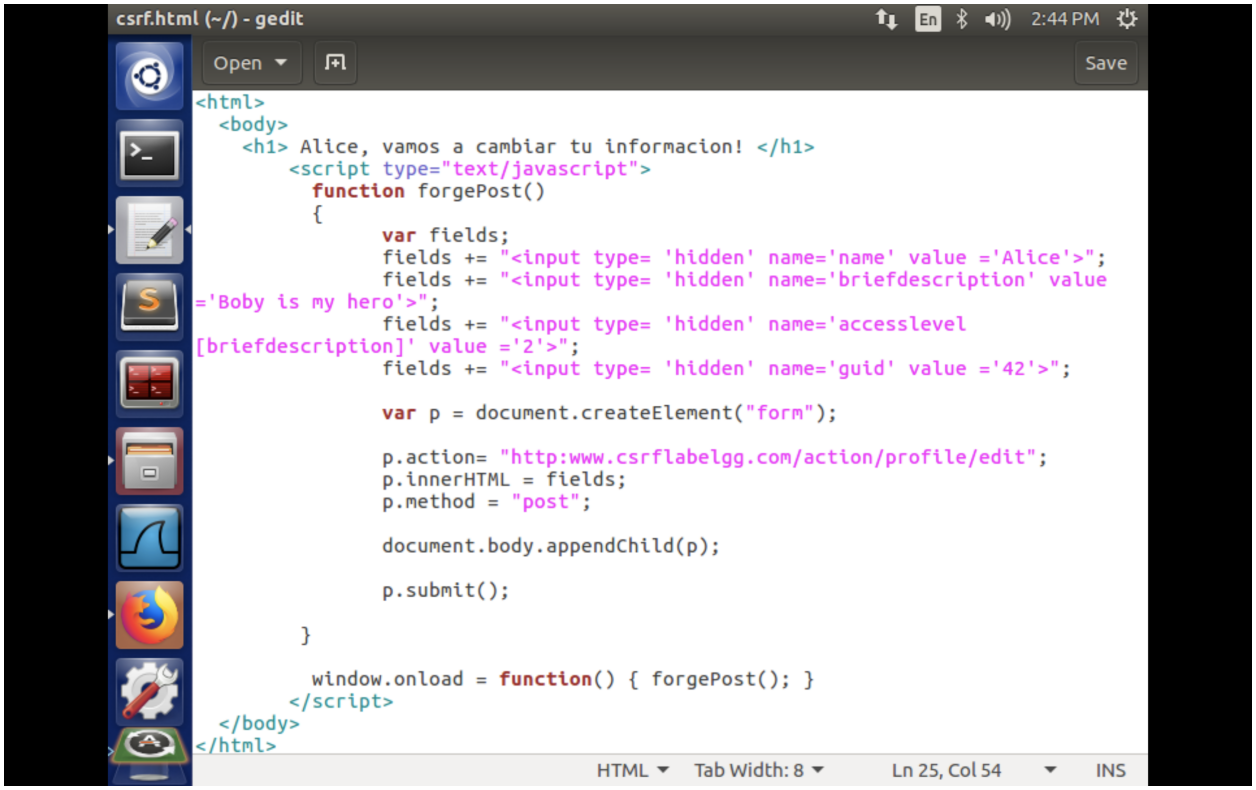
In order to view how this works, lets go ahead and change the brief description under Boby's account and see how it is done, as shown below:



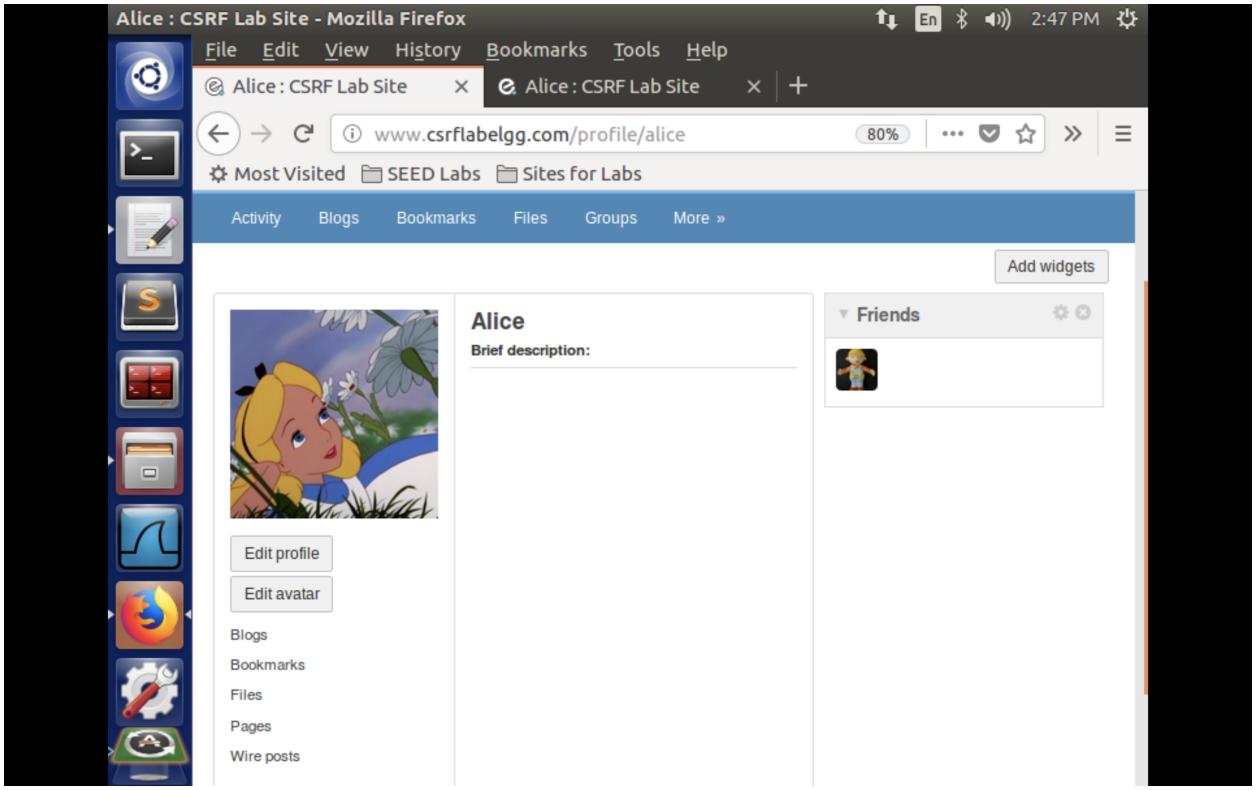
You can see the POST request is right there as /action/profile/edit with parameters as shown below:



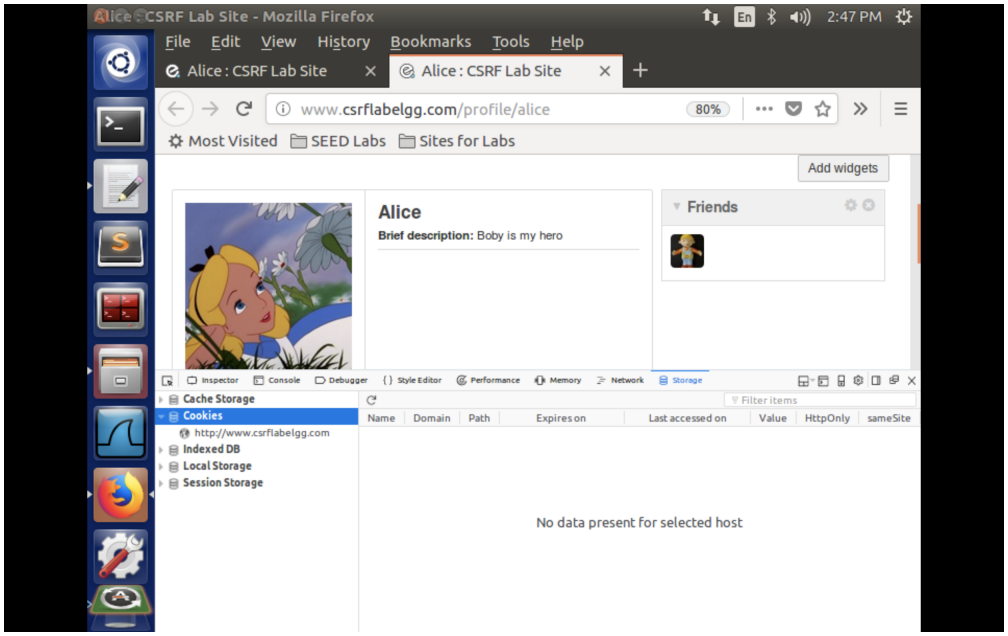
We know that these are what is needed for a POST to occur. Next we know that the guid of Alice is 42. (you can check it on the inspection tool) Next we are going to construct an html with a script tag that will ensue on a window.reload and call the function forge(), of our malicious code. The code is shown as below:



Once this is done, we go ahead and go into the attack, we are logged in as Alice, and somehow this malicious input is opened by her. First we have no description as you can see below:



Then when running the attack, we see that it changed it as below:

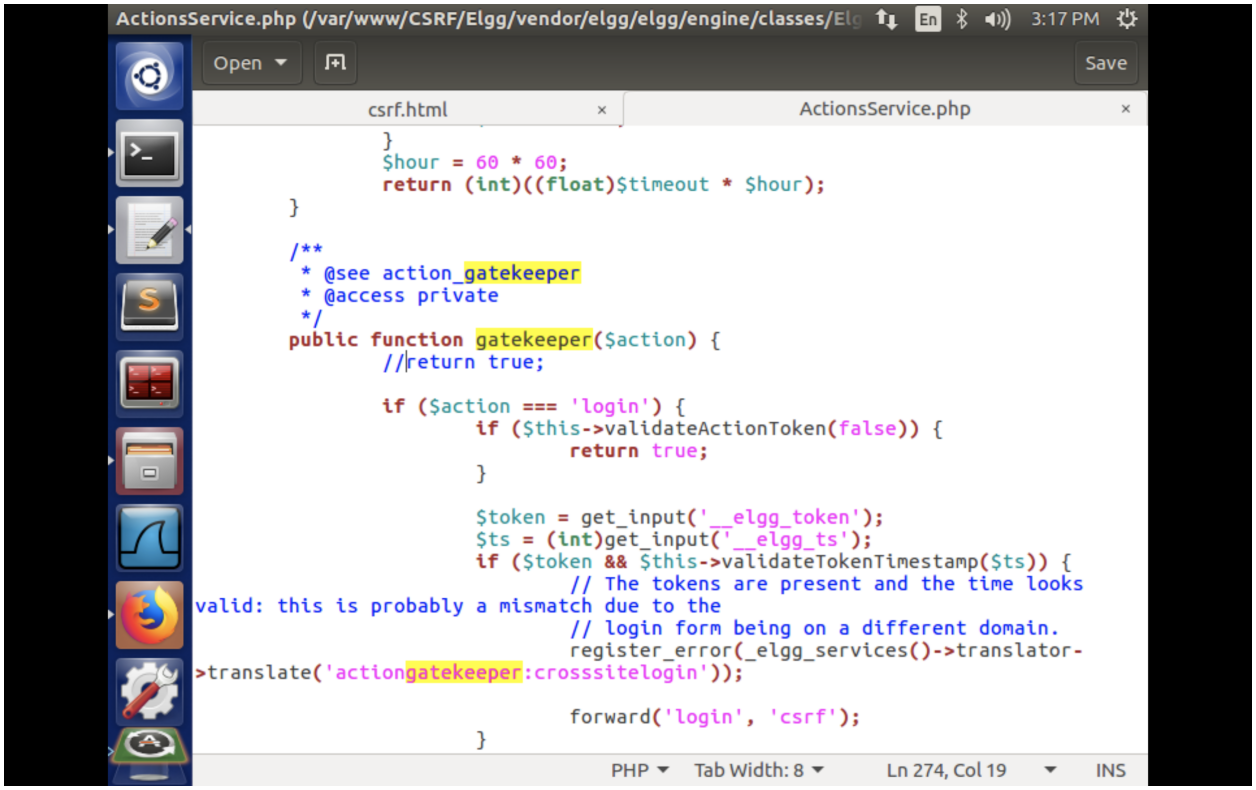


Questions:

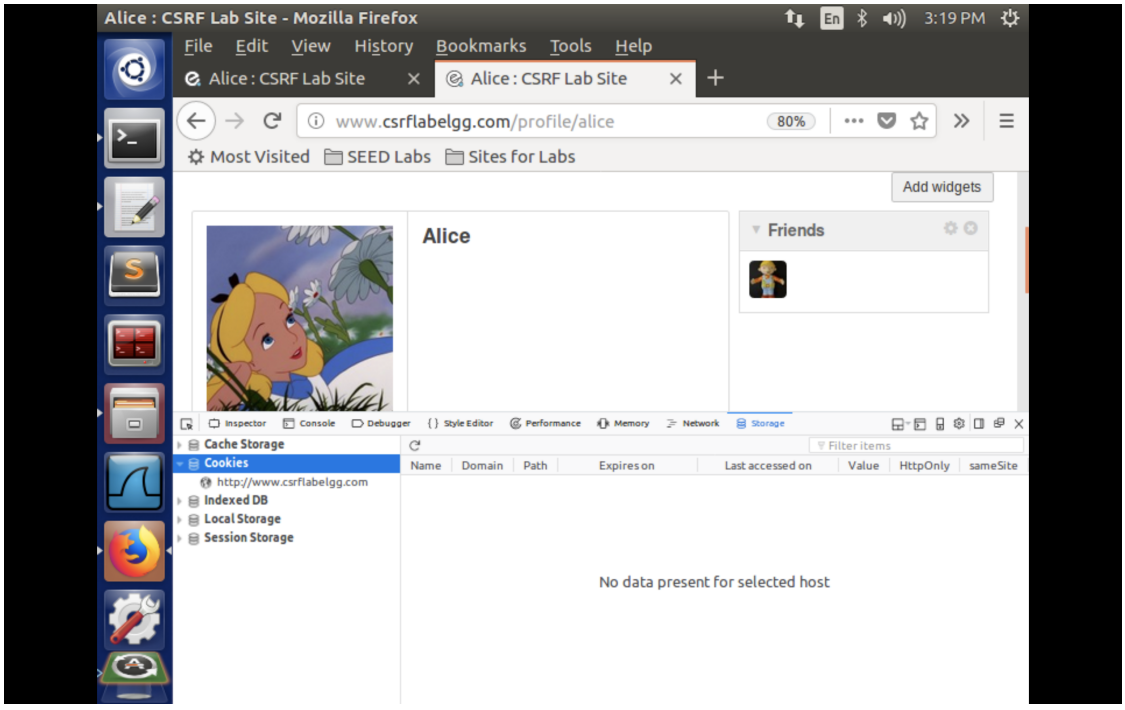
1) We are able to find the guid within the source code of the website if the people who develop it are aware that they stick that type of code there. If for some reason Boby did not know Alice's credentials then he would have had to try other mechanisms such as finding out her username, email, and/or guid, by looking over the requests that HTTP sends in order for Bobby to figure out her GUID. other than that he may just guess.
2) Boby would not be able to do this attack because he would have to know what their guid first and this is POST request. Bobby's malicious site is different from the targeted site. Moreover the guid is sent to the targeted website not the one we created.

Task 4:

Finally, in this task we are going to turn back the countermeasure that was initially turned back off. This is done like so, as shown below:



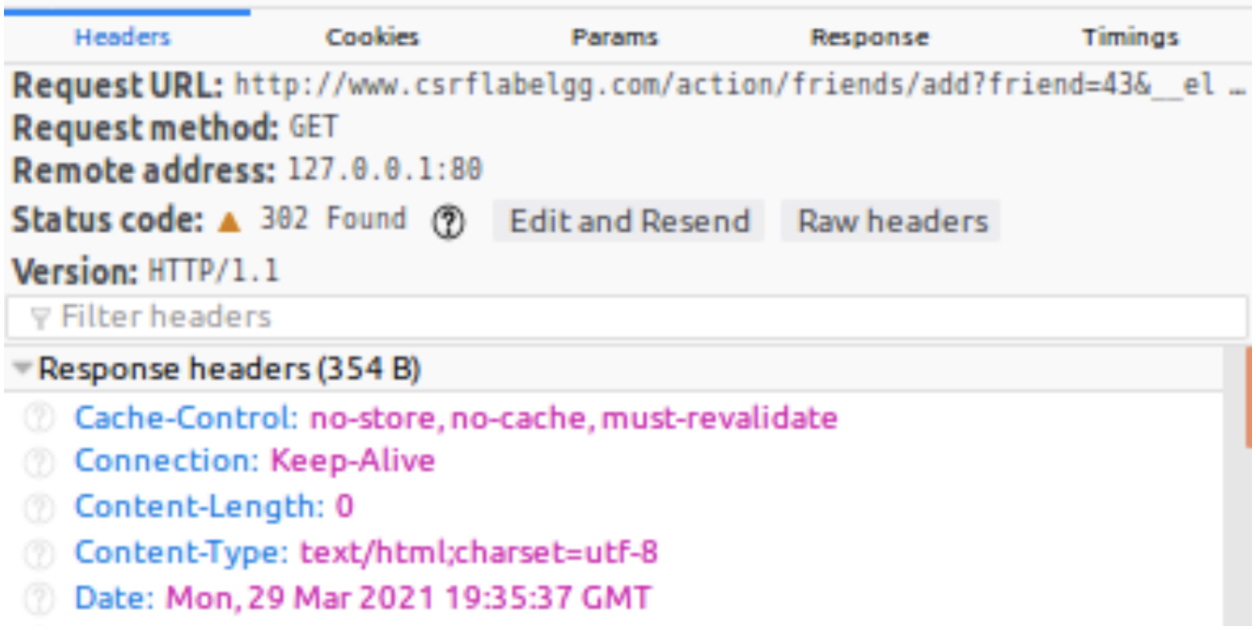
We first commented out the return statement as you can see above.



Then as you can see in the above screenshot, I removed everything from what we had on the brief description. We then at the screenshots below cause it to happen when we run the same attacks.

GET Request:

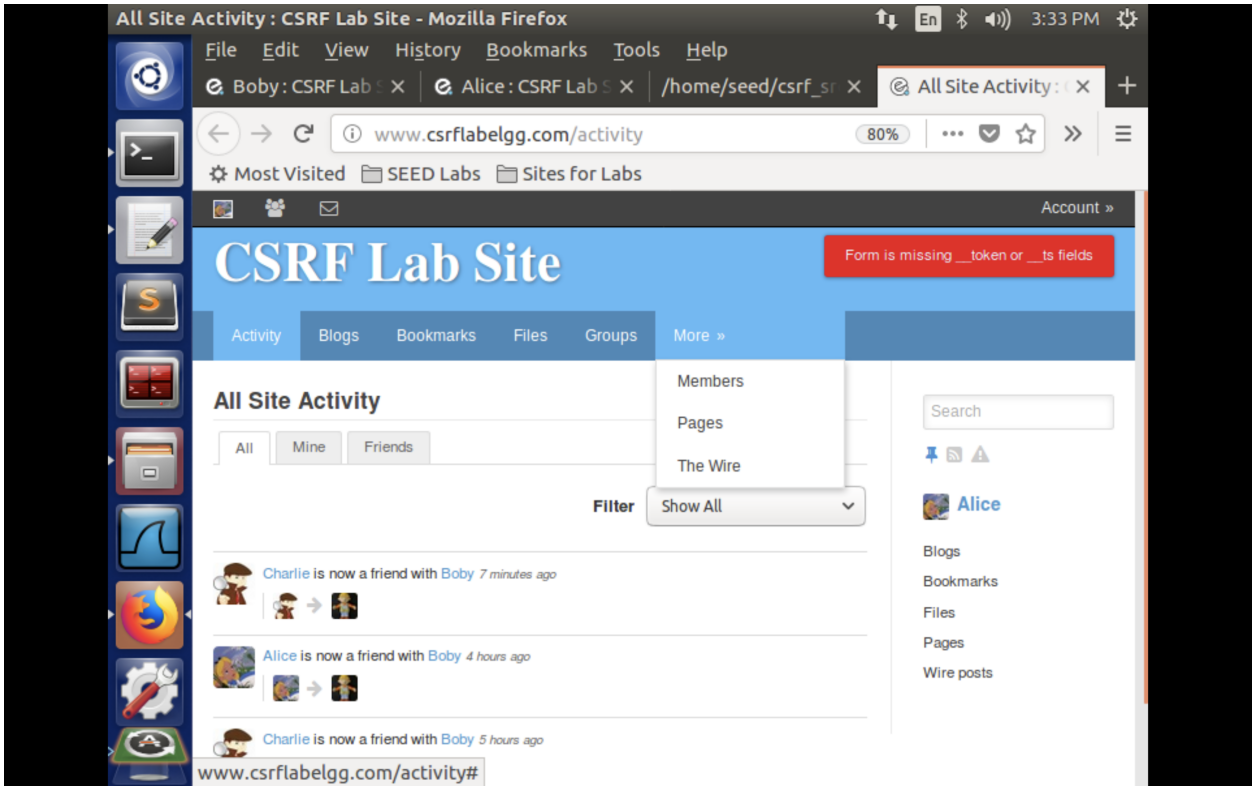
You can see we get some error as shown below;



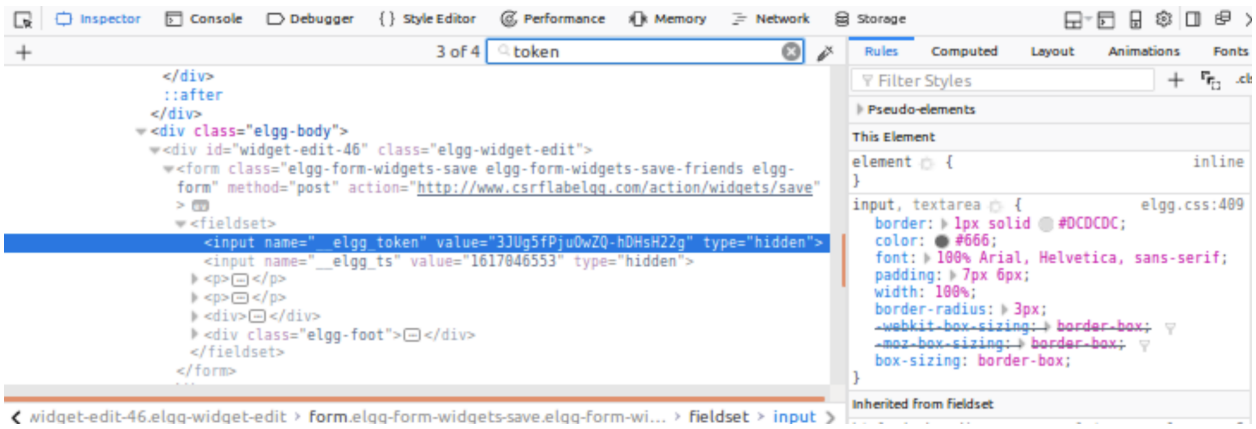
We are getting a 302 code and adding Boby as a friend is not working.

POST Request:

Here you get a missing token error:



In conclusion, timestamp (ts) and secret-token (st) are the countermeasure that we turned back on.



And as you can see this is sent from the attackers computer to the elgg server and not the elgg server to the elgg server (same site), thus it does not do what we want. That is why they cannot

attach tokens to others websites. No one can guess these values and are only visible to the user that is logged in.