Information Security: Cross-Site Request Forgery (CSRF) Attack

Name: Chandan N Bhat

PES1201701593

Section H

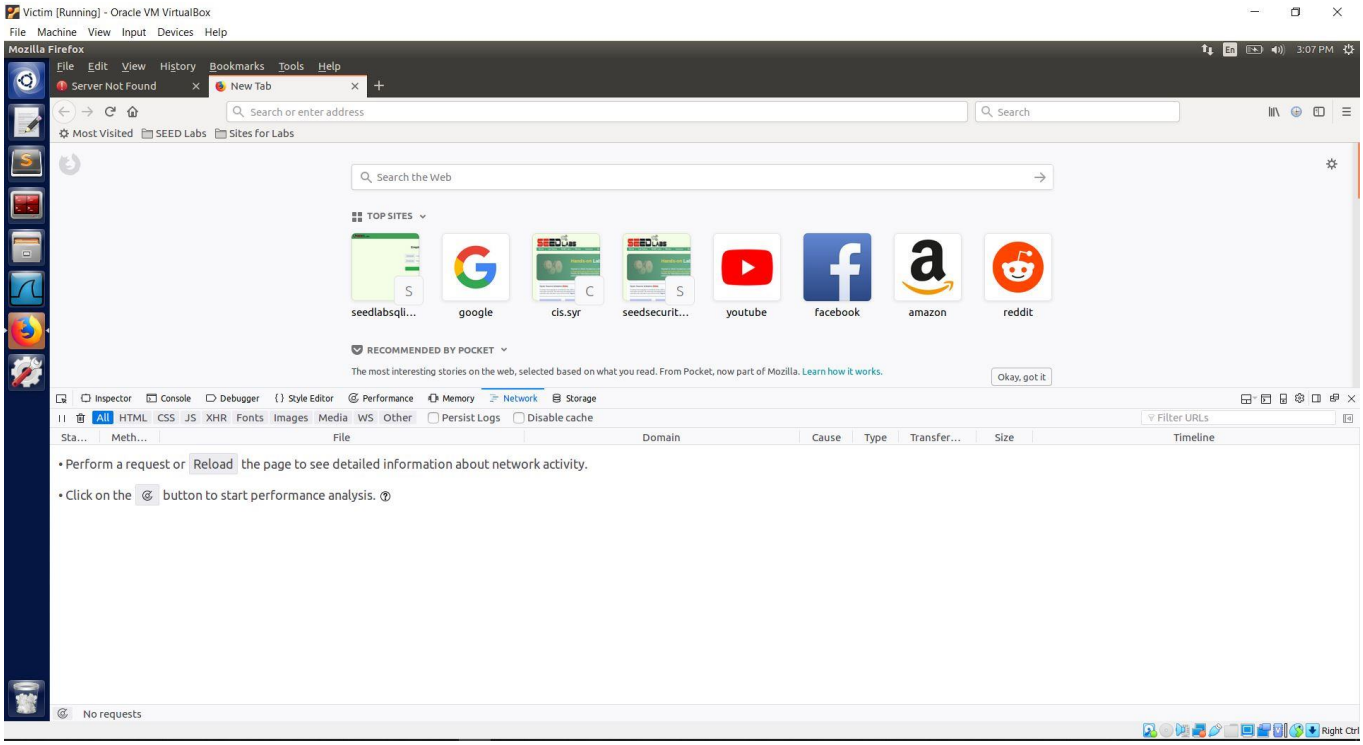
CSRF is an attack that forces the user to execute unwanted actions on the web application to which the user is currently authenticated. The attacker might trick the user of the web application into performing actions of the attacker's choice.

For this lab, we will use 2 websites that are locally set up in the VM. The first website www.csrflabelgg.com is vulnerable and the second website www.csrflabattacker.com is the malicious site of the attacker.

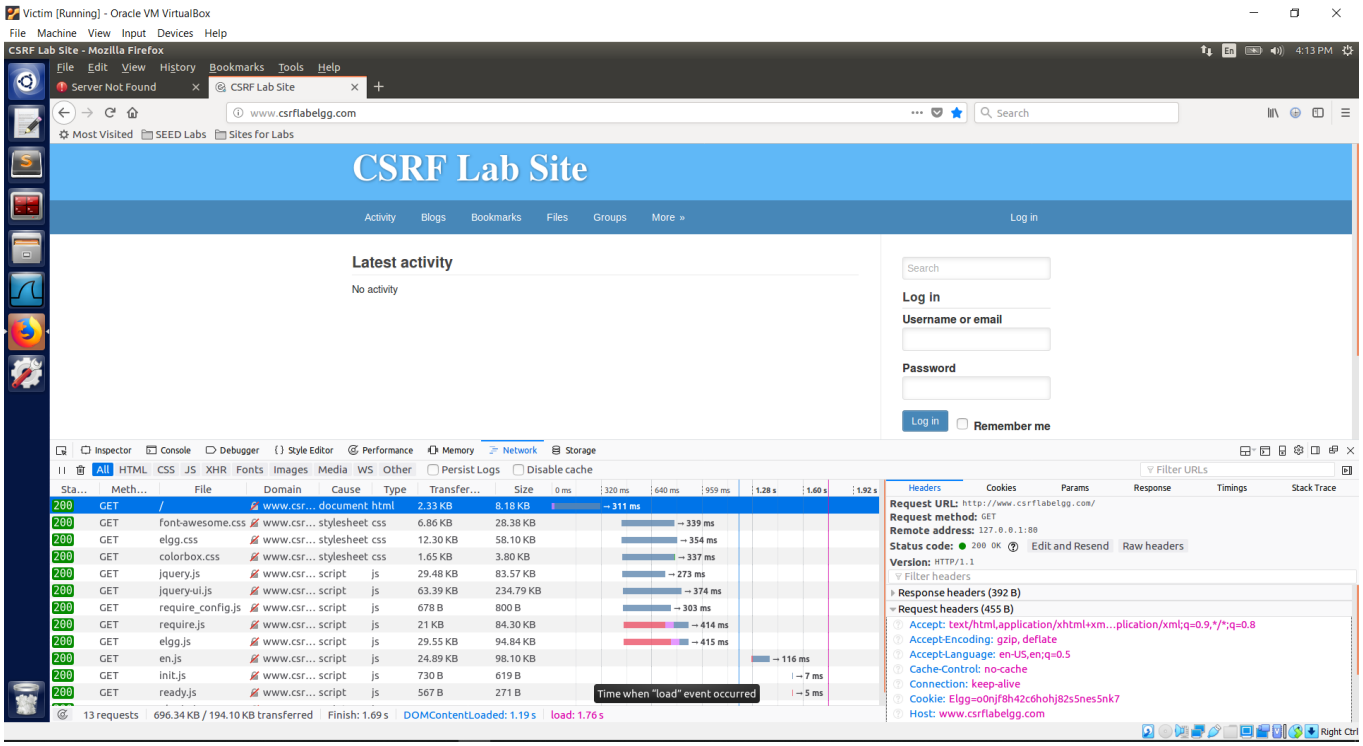
Task 1: Observing HTTP Request

In CSRF we need to forge HTTP requests. Thus, it is essential to understand a legitimate HTTP request and how it looks and its parameters. We use a Firefox add-on called 'HTTP Header Live' to achieve it. Using this tool, we will capture an HTTP GET request and POST request in Elgg.

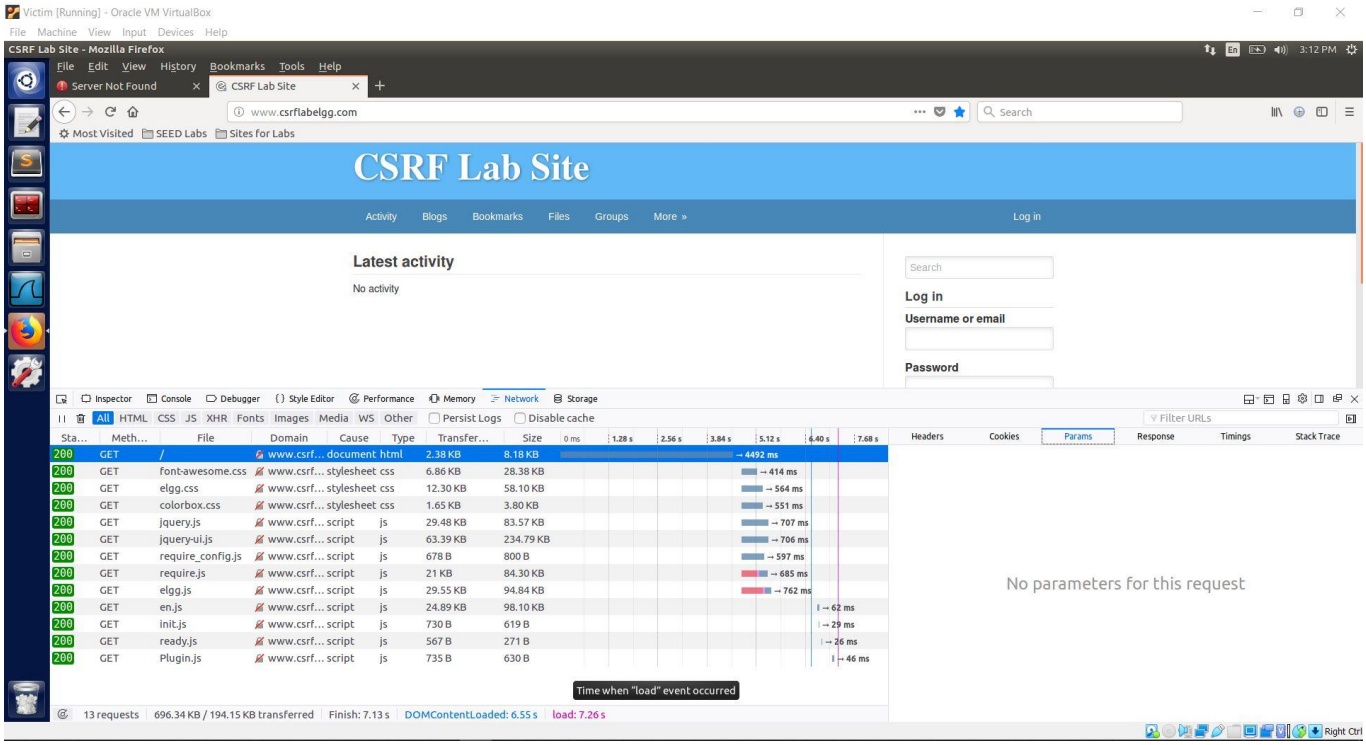
To access it press "ctrl+shift+E"



Now we make a simple GET request to www.csrflabelgg.com to view the HTTP request. The below screenshot shows the request made. The method column shows GET indicating that the request made is a GET request. To the right we have a section with columns headers, cookies, params, response etc. which has all the data associated with the request.

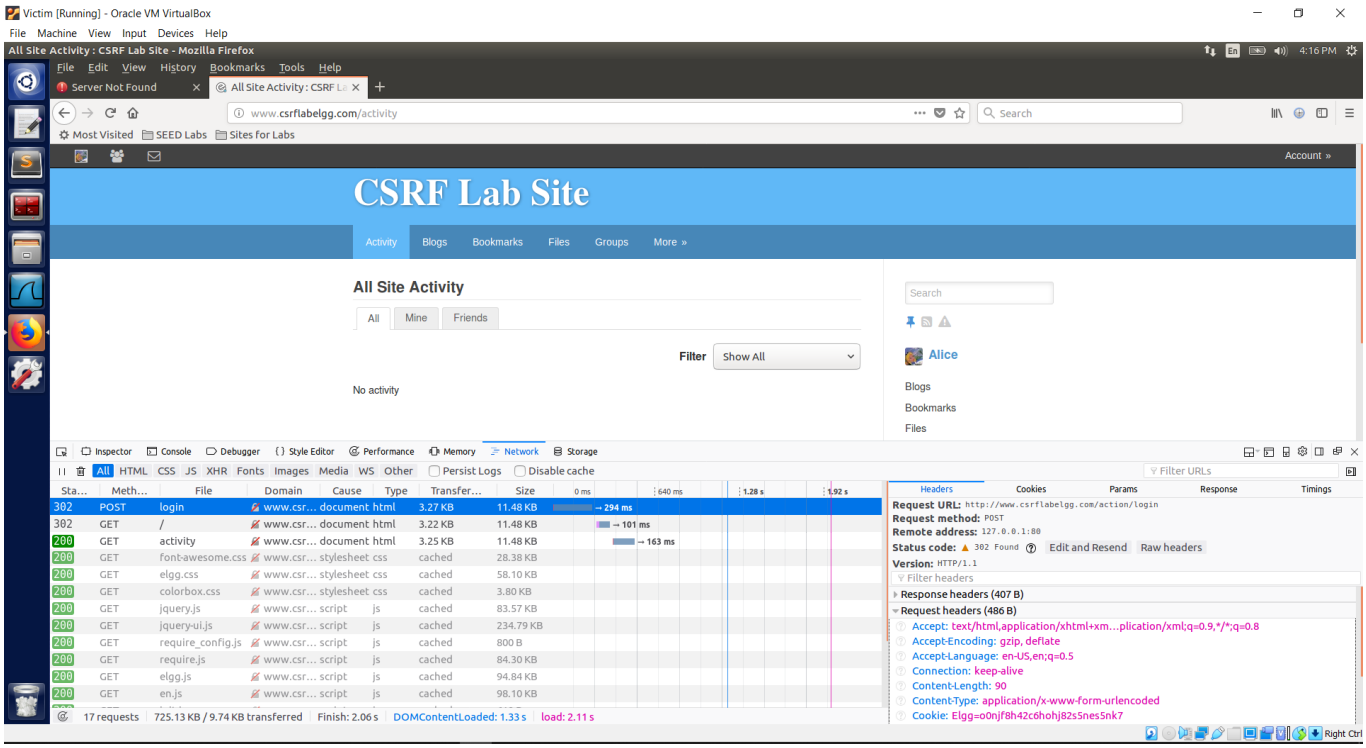


To see the parameters passed to the domain with the HTTP request, we can see the Params section as shown below. From the below screenshot we observe that there were no parameters passed with the HTTP request.

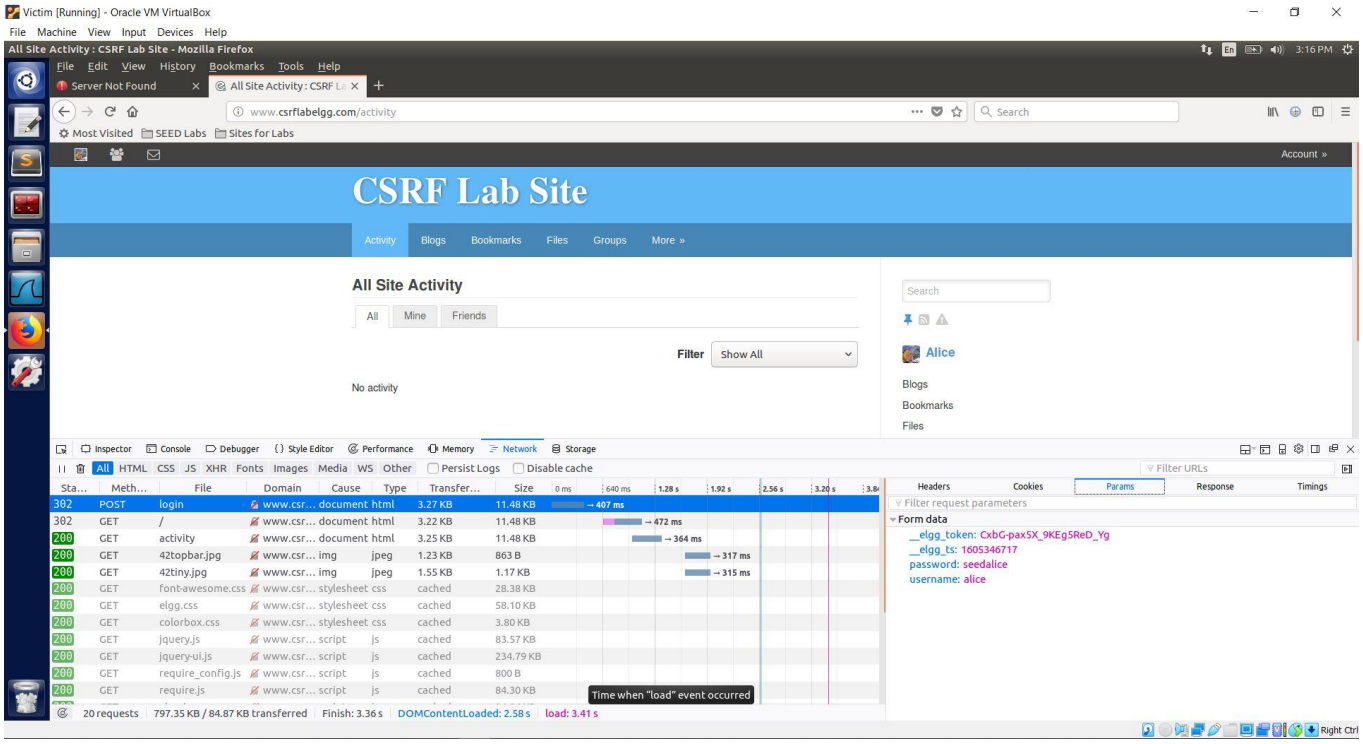


Similarly, we will send a HTTP POST request and observe the headers and data sent with the POST request. We will login to the website as a user ‘Alice’ which will send a POST request to the server. The username is ‘Alice’ and password is ‘seedalice’.

From the below screenshot we see that unlike the previous request, this request has method POST indicating POST request made to www.csrflabelgg.com/action/login.



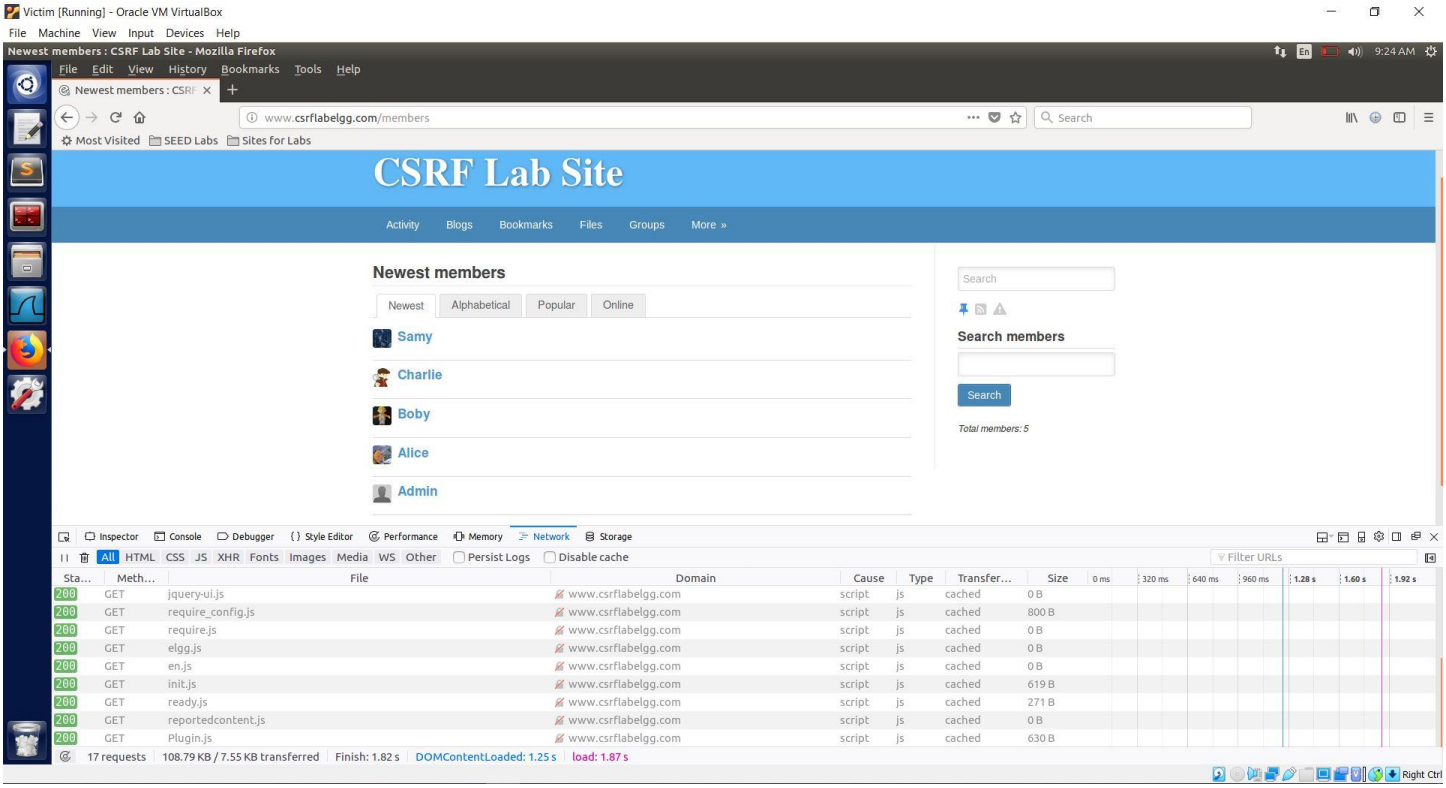
To see the parameters sent with the request we see the Params section as shown below. We observe that unlike the previous request we see that username, password and a couple more data is sent as form data along with the HTTP Post request to the server.

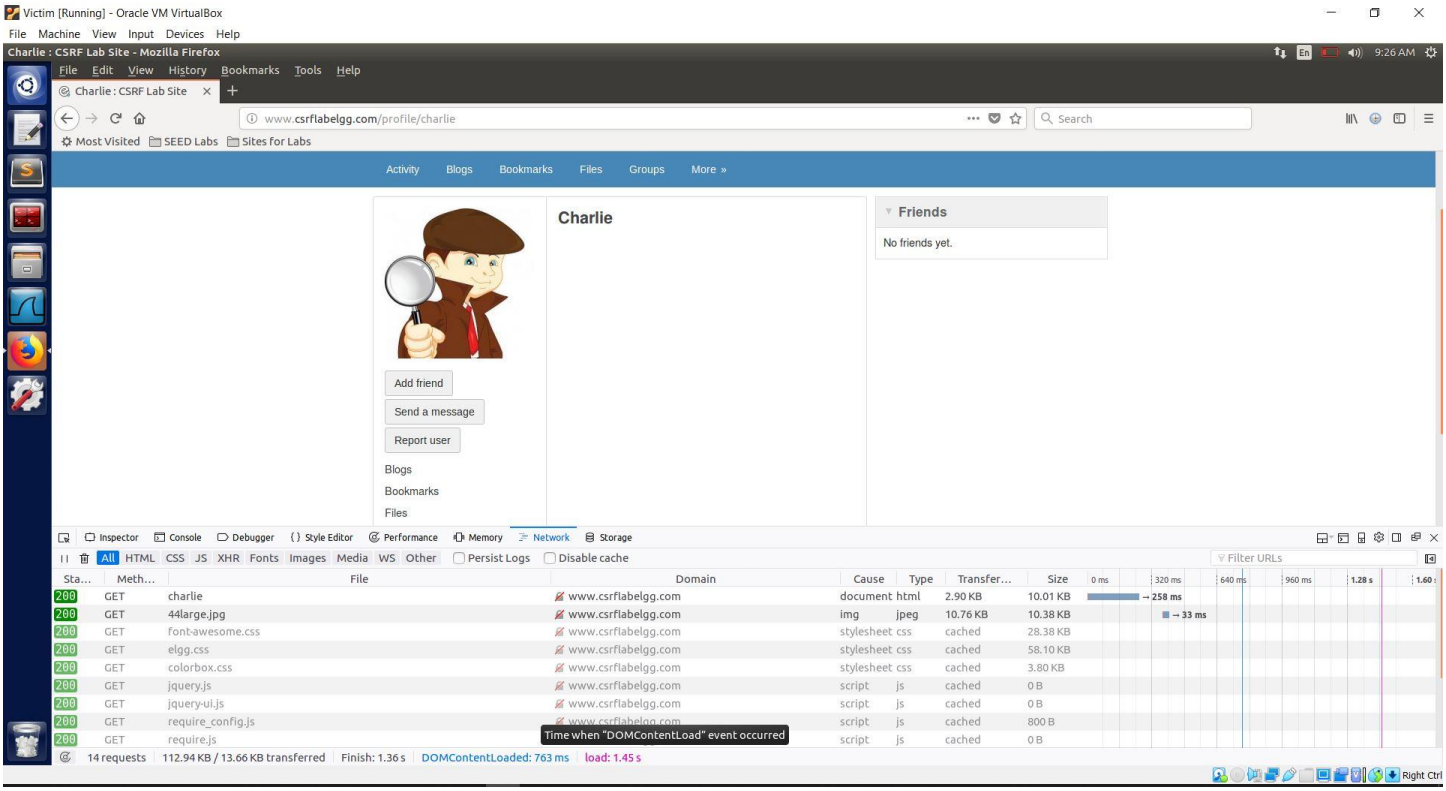


Task 2: CSRF attack using GET Request

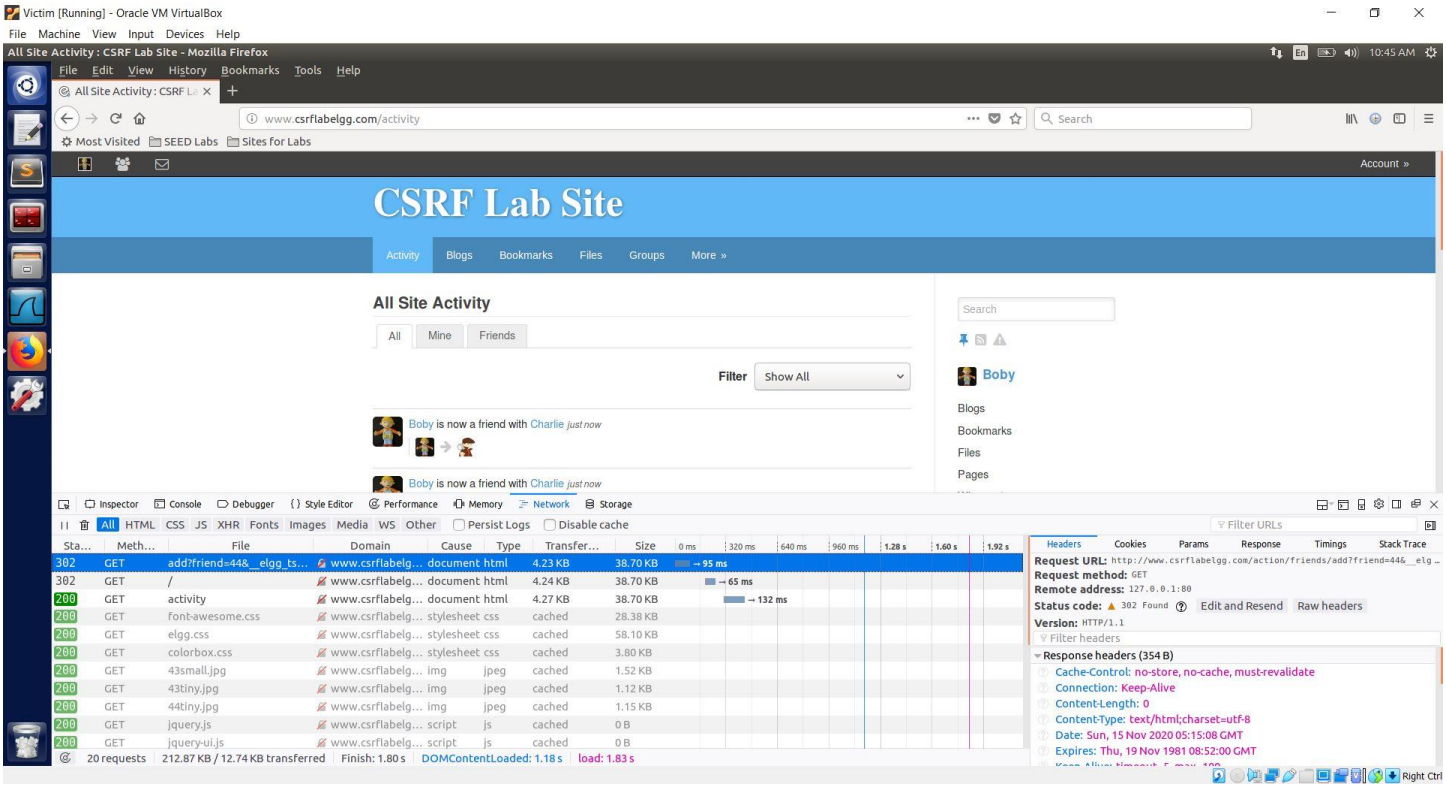
In this task we require 2 people in the Elgg network, Alice and Boby. Boby wants to be a friend of Alice, but Alice refuses to. Thus, we will try to use CSRF to achieve this goal.

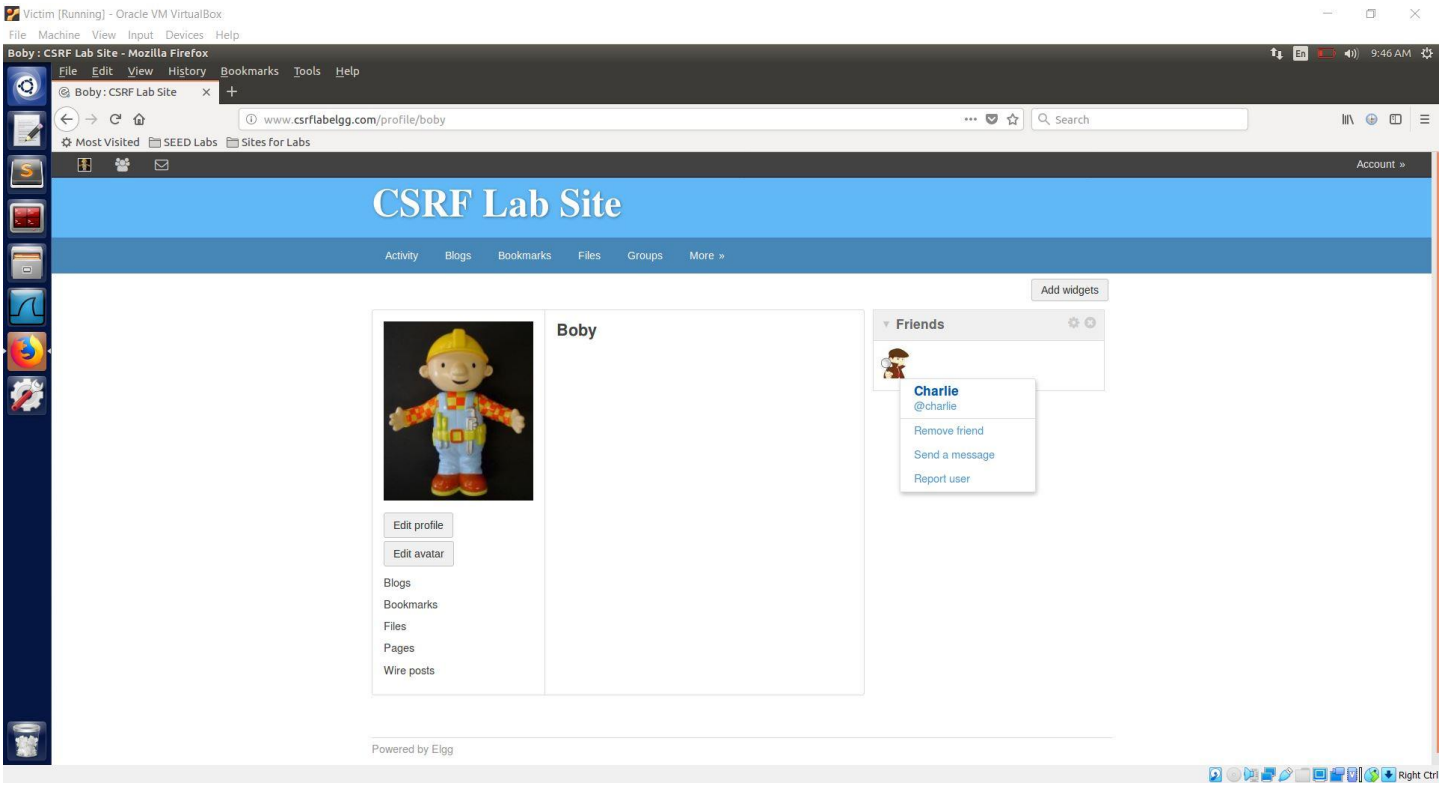
First, we will send a request from Boby to Charlie to understand how the request is sent, and the URL to which it should be sent. We go to the members page to add Charlie as Boby's friend.





Next we click on ‘Add Friend’ and observe the request sent.

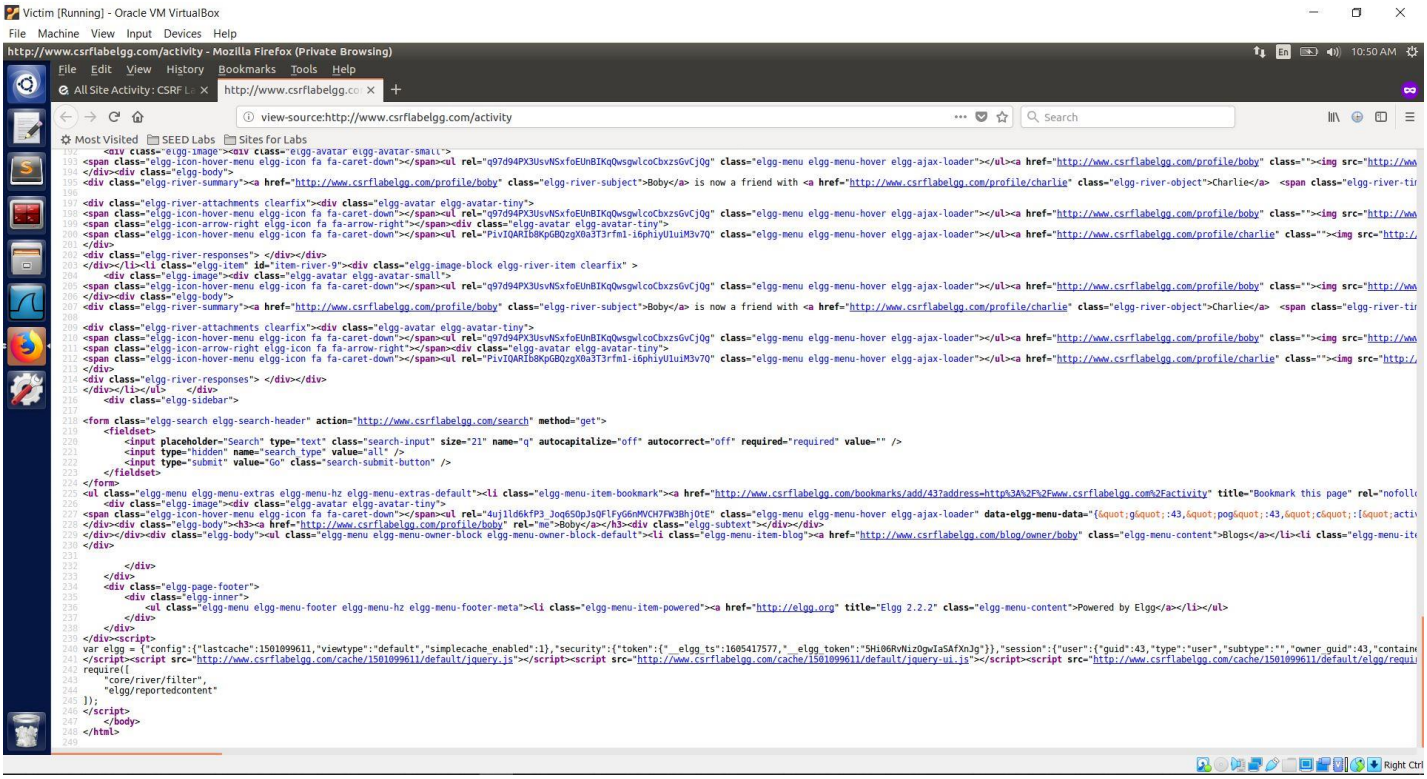




Charlie is added as Boby’s Friend as shown in the above screenshot.

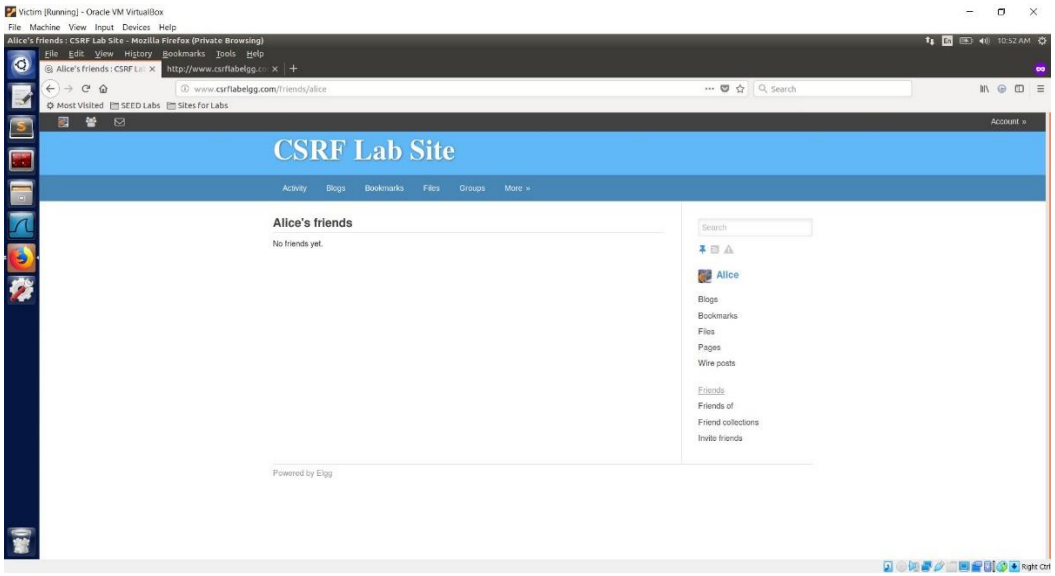
We observe that a GET request is sent to URL http://www.csrflabelgg.com/action/friends/add?friend=44. The 44 in the query string of the URL indicates the id of user Charlie. Thus, in order to add Boby as Alice’s friend we need to find the id of user Boby so that we can construct the URL request accordingly.

To find out the id of Boby we view the page source where we can find the required value in guid value in the Javascript variable which indicates the id of Boby.

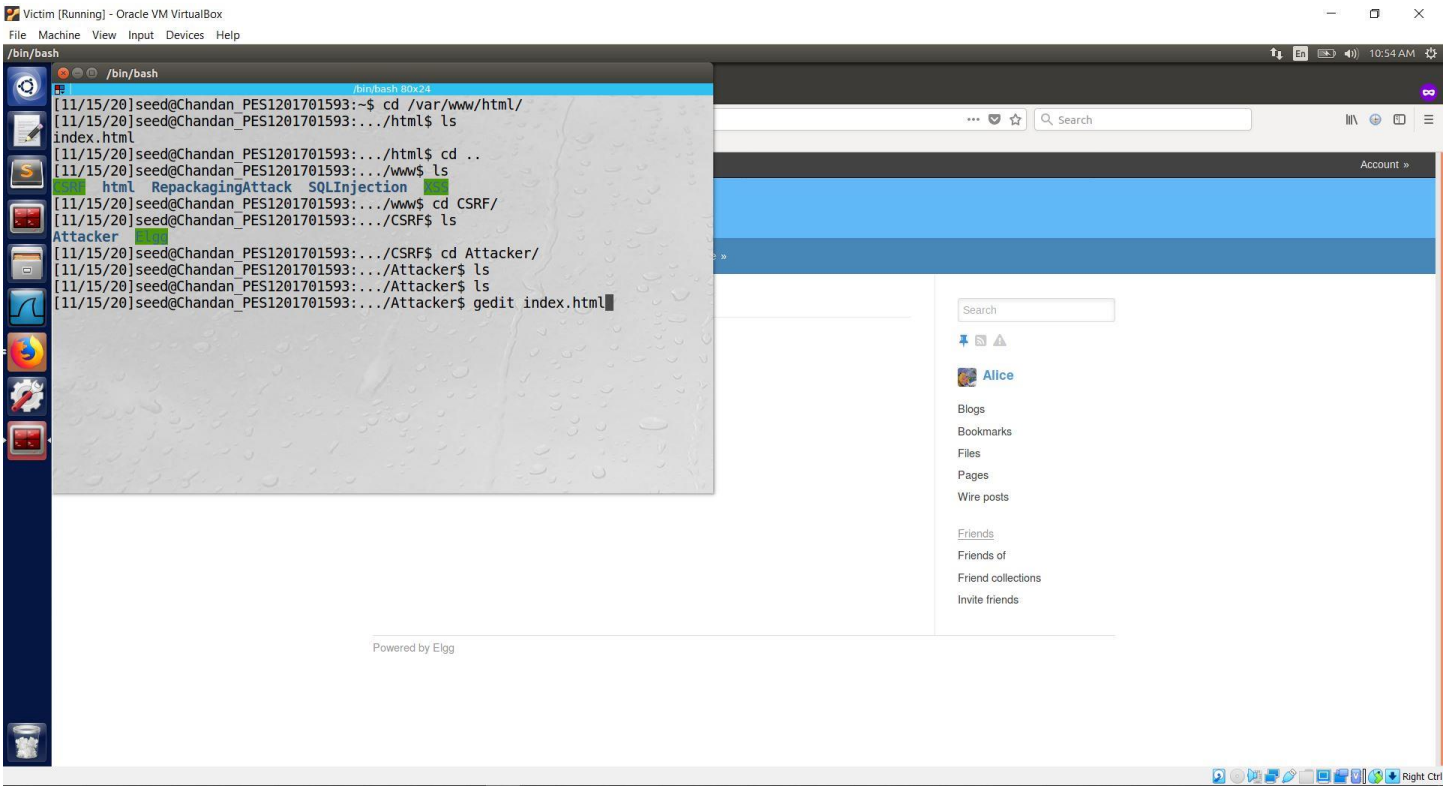


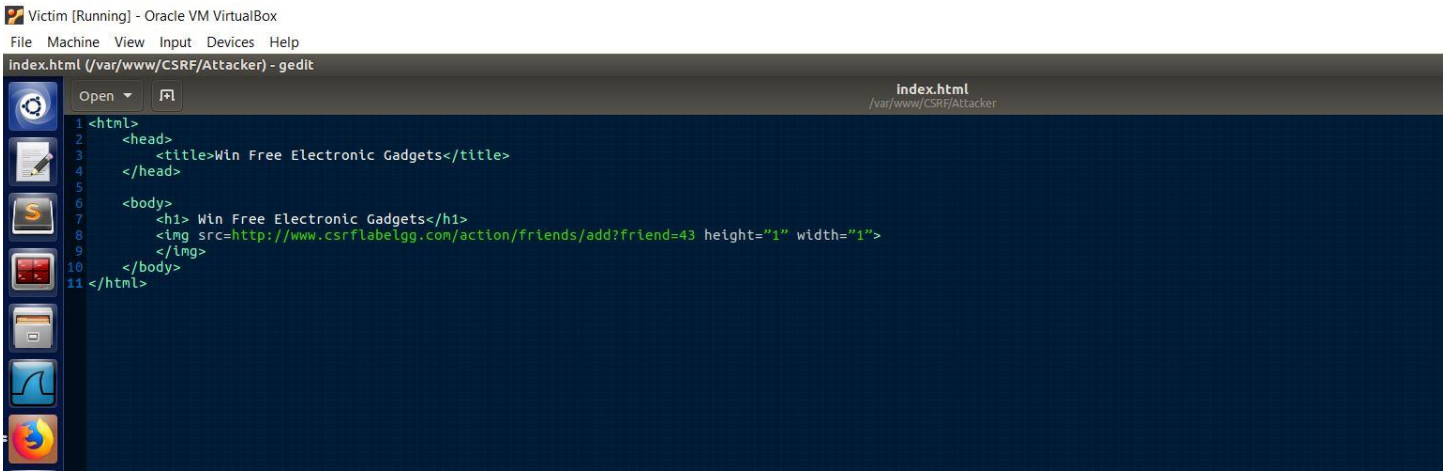
From the page source we found the guid value to be 43. To add Boby as Alice's friend a request to http://www.csrflabelgg.com/action/friends/add?friend=43 should be sent. So if Alice opens the malicious website we want it to send a GET request to http://www.csrflabelgg.com/action/friends/add?friend=43

The below screenshot shows the friends list of Alice before the CSRF attack.



Now we create a html file index.html which contains an img tag with the souce attribute of the image set to http://www.csrflabelgg.com/action/friends/add?friend=43 which makes a GET request when the page loads. The code for index.html is shown below.

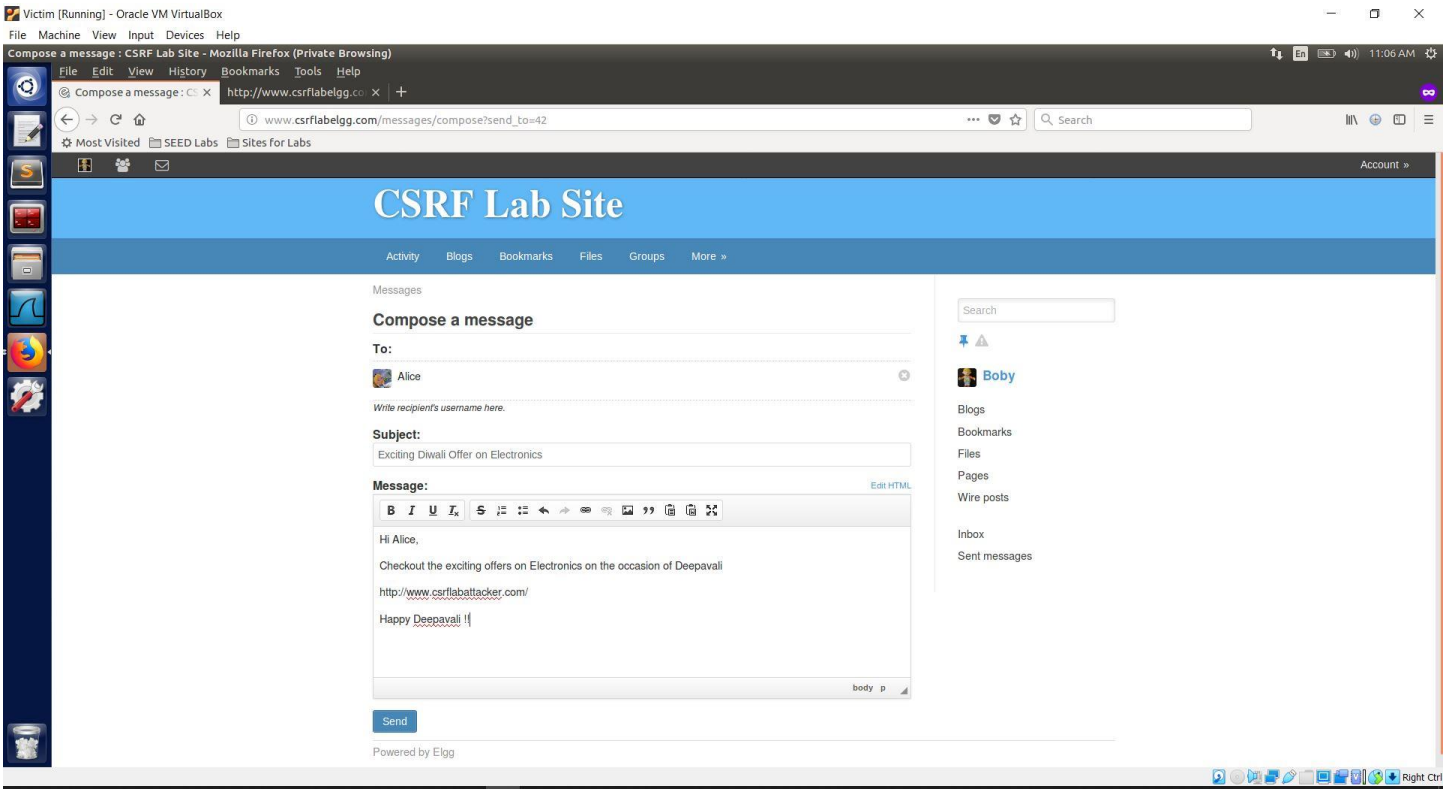




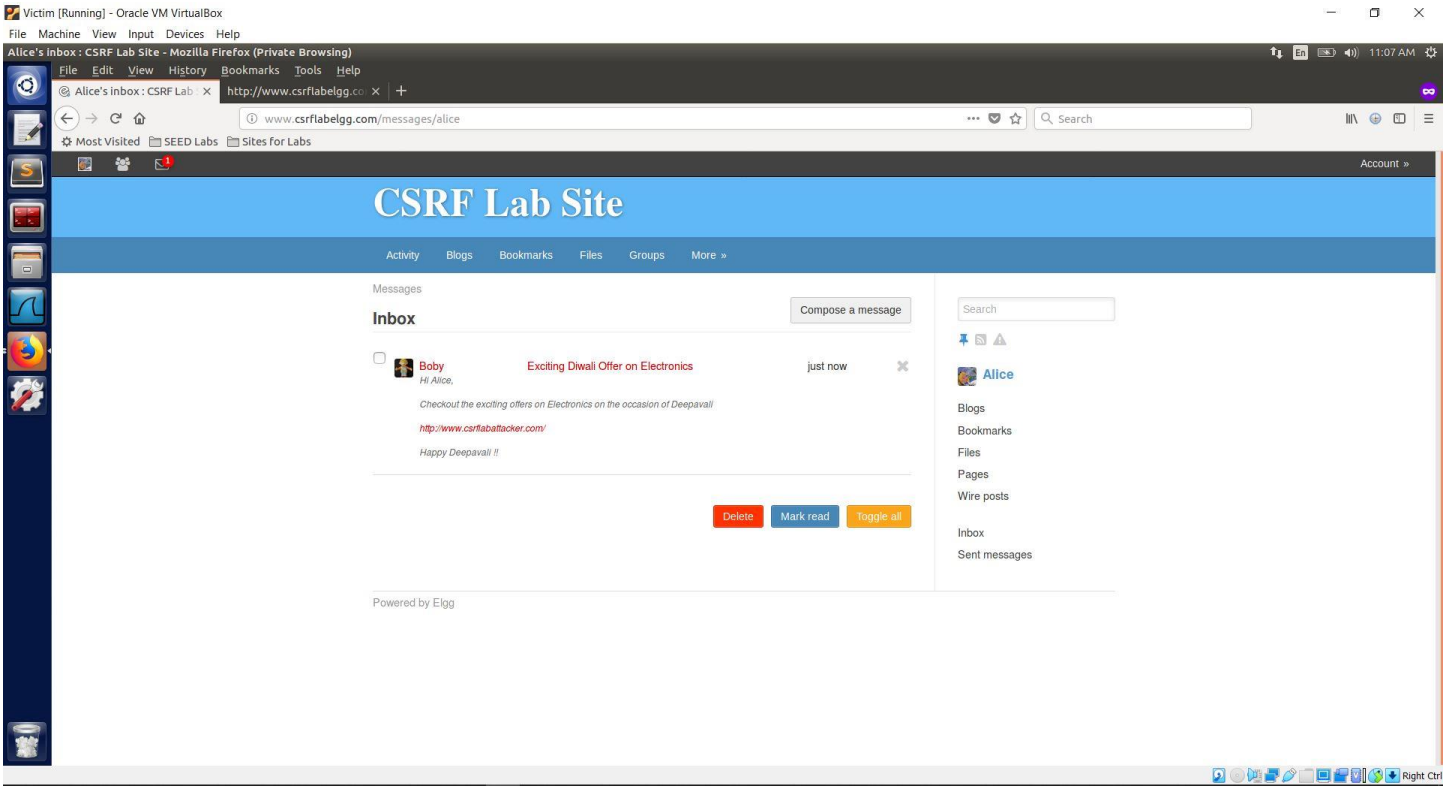
When Alice visits the attackers website it will send a GET request to the Elgg website that adds Boby as Alice's friend. Also we have ensured that the image is not visible to Alice when he visits the page.

Now our task is to send Alice a message that contains the link to the attacker's website. We could use social engineering to ensure that Alice clicks on the link and be a victim of CSRF attack.

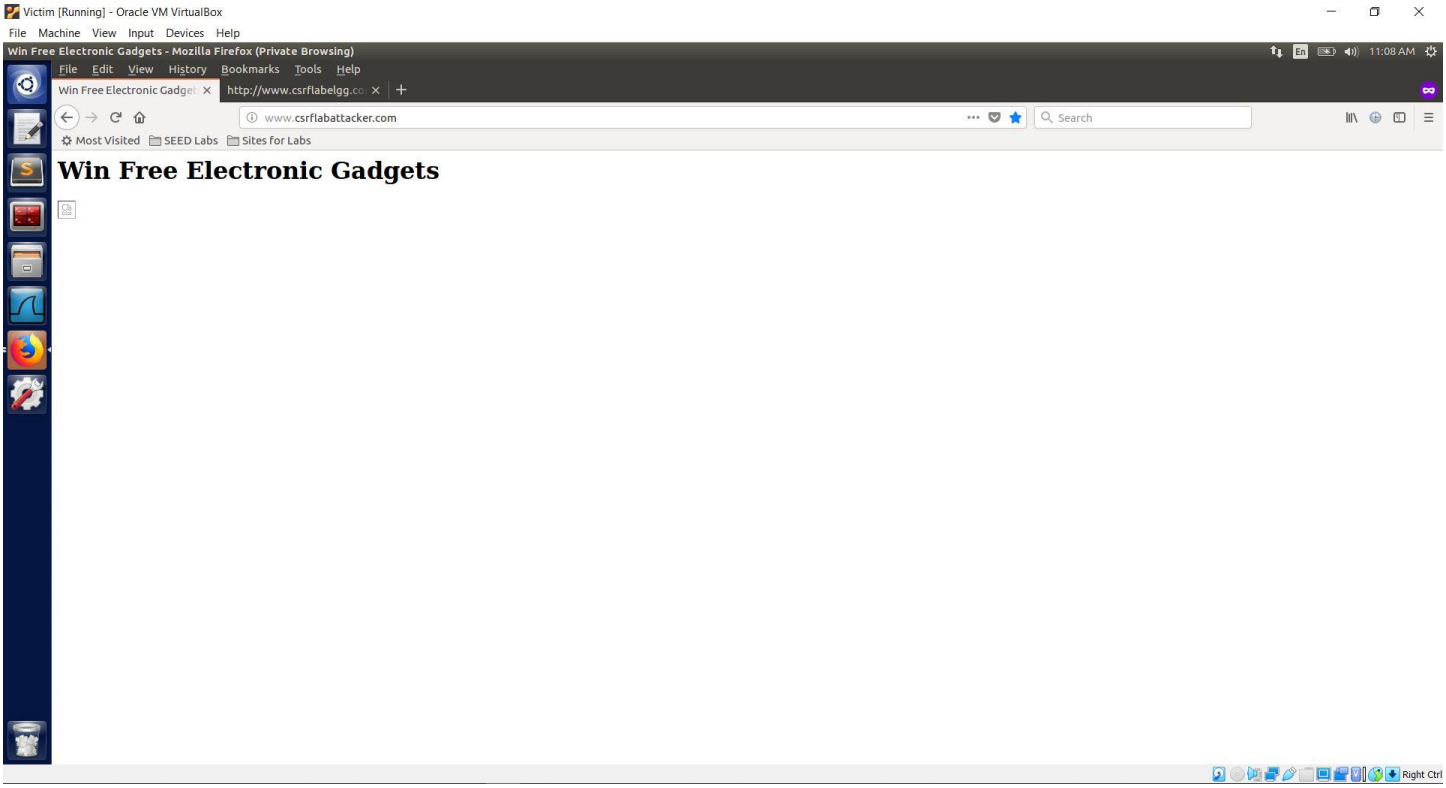
The below screenshot shows the message sent from Boby to Alice.



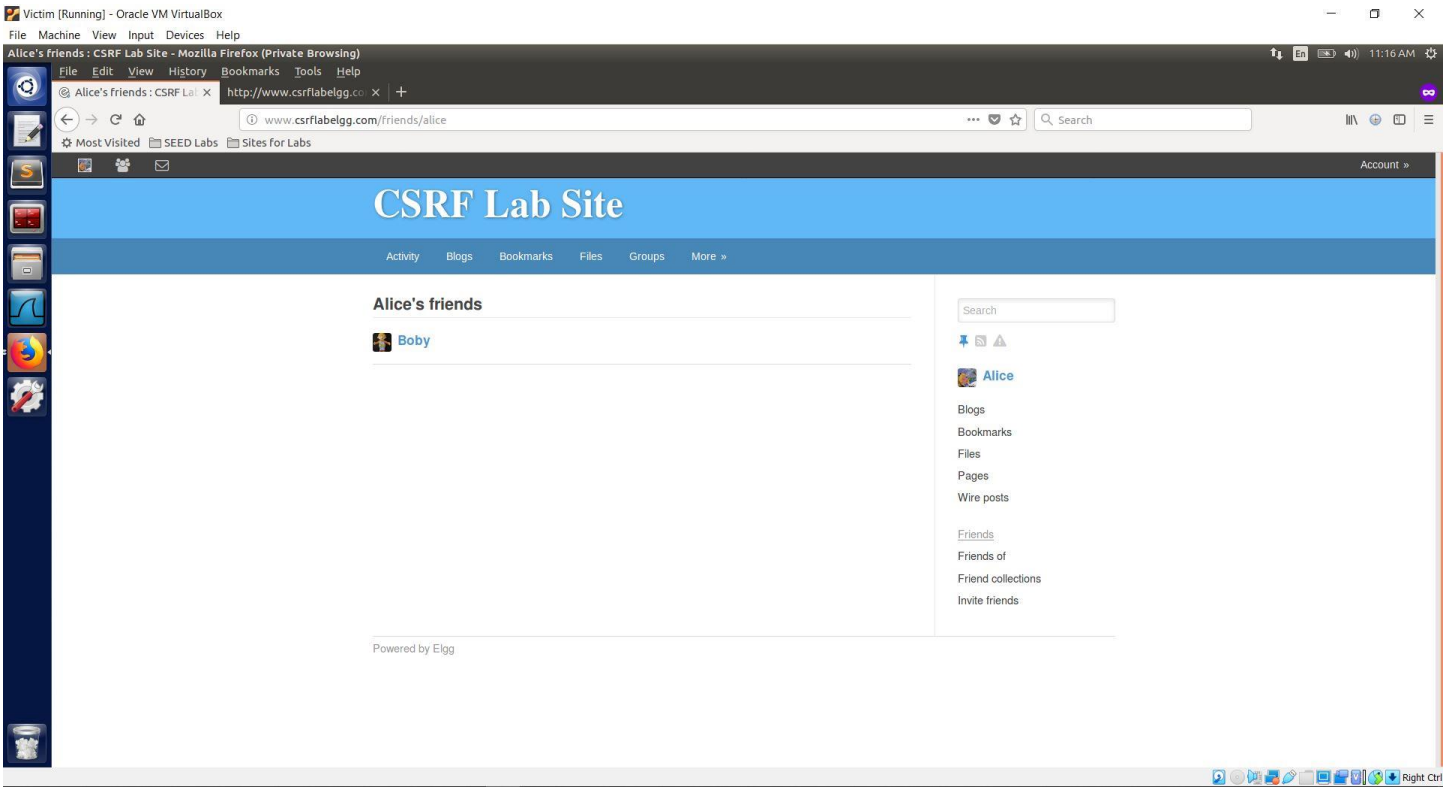
We observe from the below screenshot that Alice received the message from Boby.



When Alice clicks on the link in the message sent from Boby he is redirected to the attacker's website which makes the GET request which adds Boby as Alice's friend.



Now if we go to Alice's profile and check his friends we observe that Boby is added as Alice's friend.

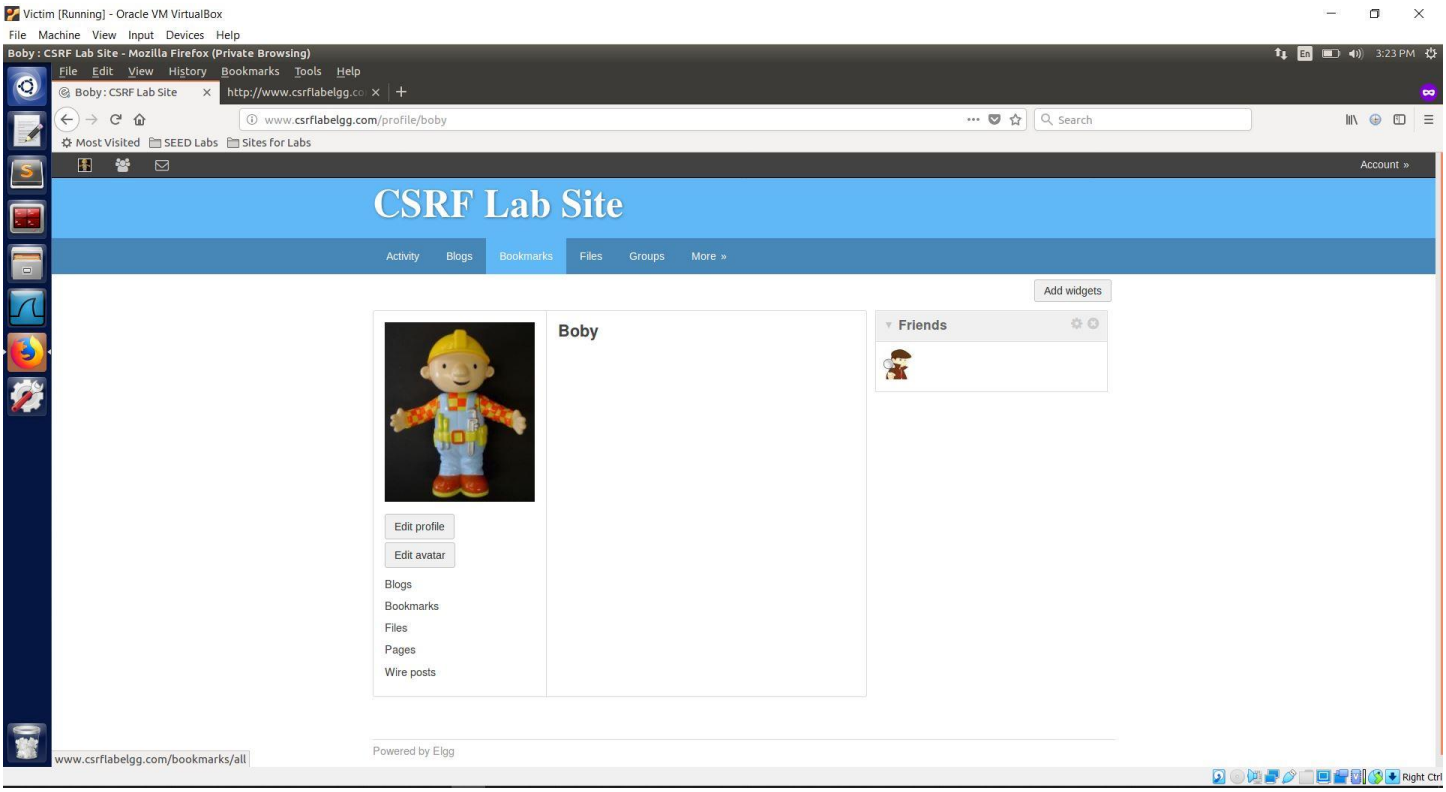


Thus we successful performed a CSRF attack using a GET request as seen above.

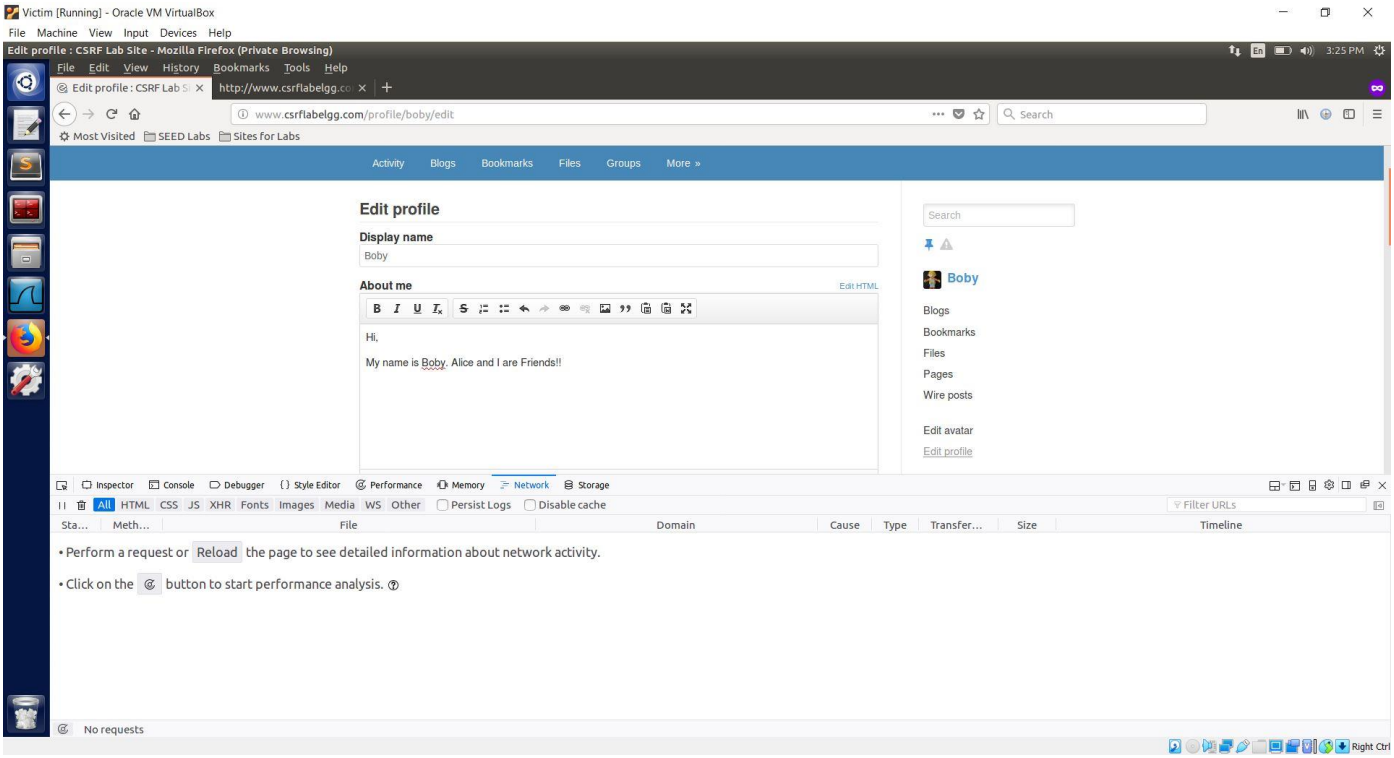
Task 3: CSRF Attack using POST Request

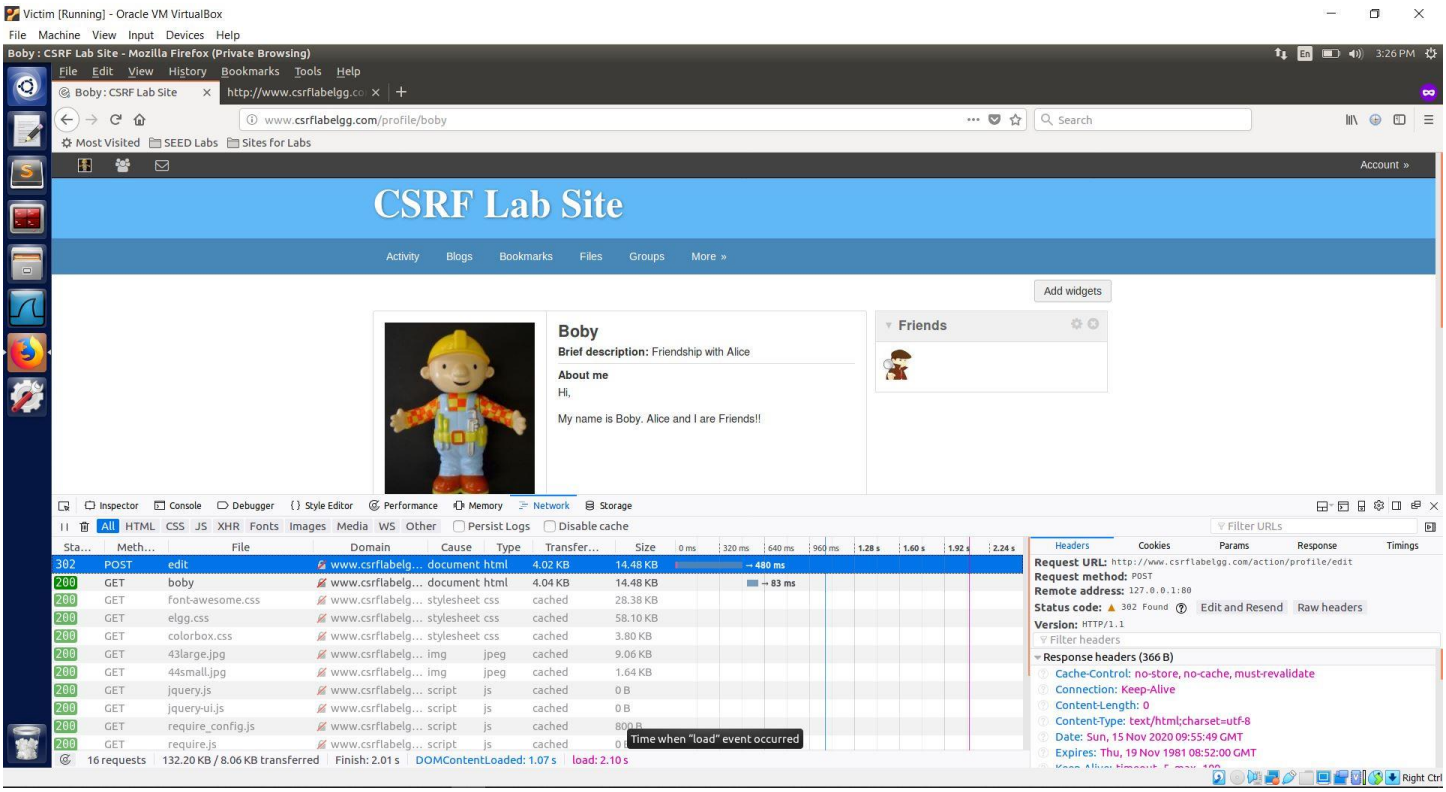
Now we will try to perform CSRF attack using a POST request. Last task we added Boby as Alice's friend. Now using CSRF attack we will add 'Boby is my Hero!' in Alice's profile.

First, we will observe a legitimate POST request sent when profile is updated. The below screenshot shows Boby's profile.

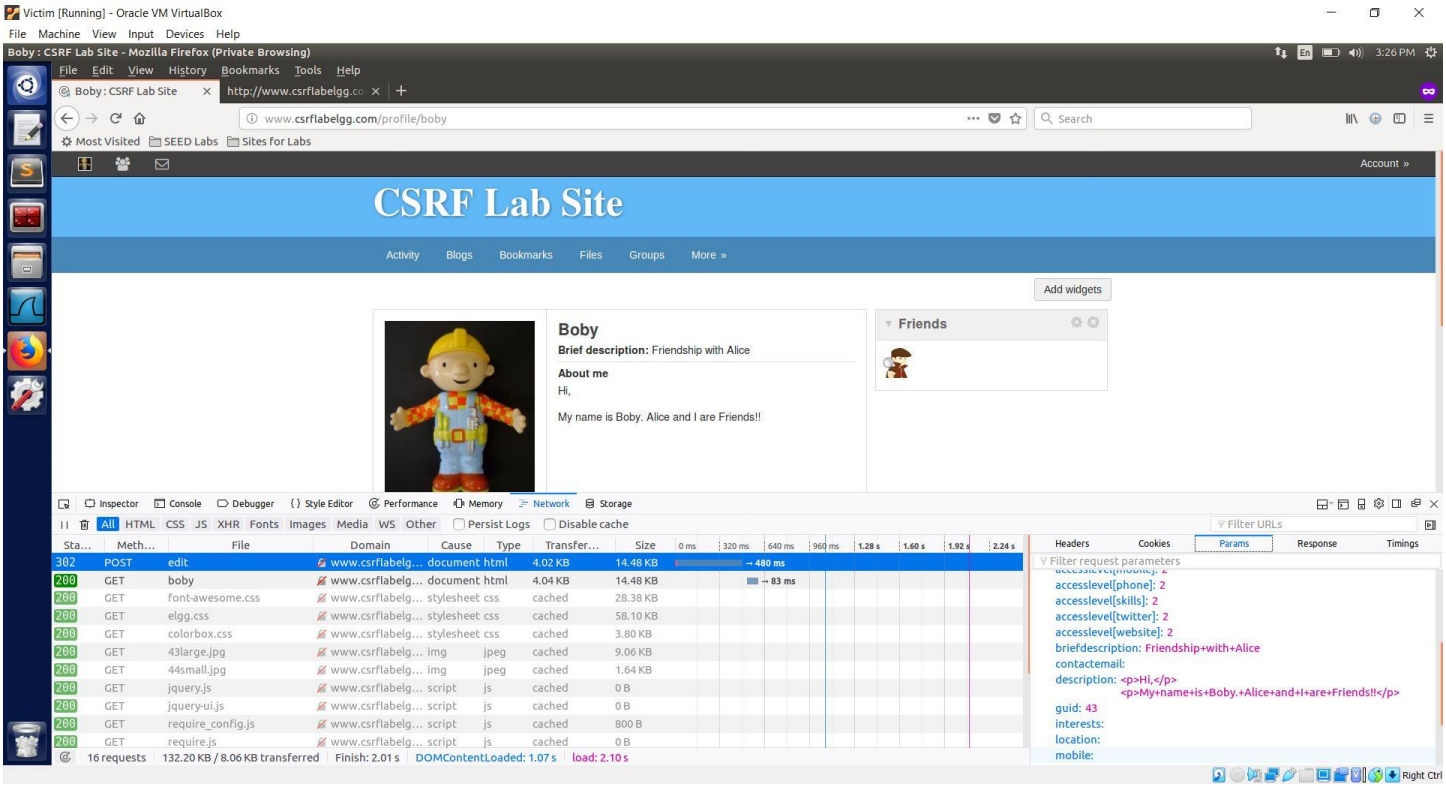


We click on 'Edit profile' and add about me to Boby's profile.



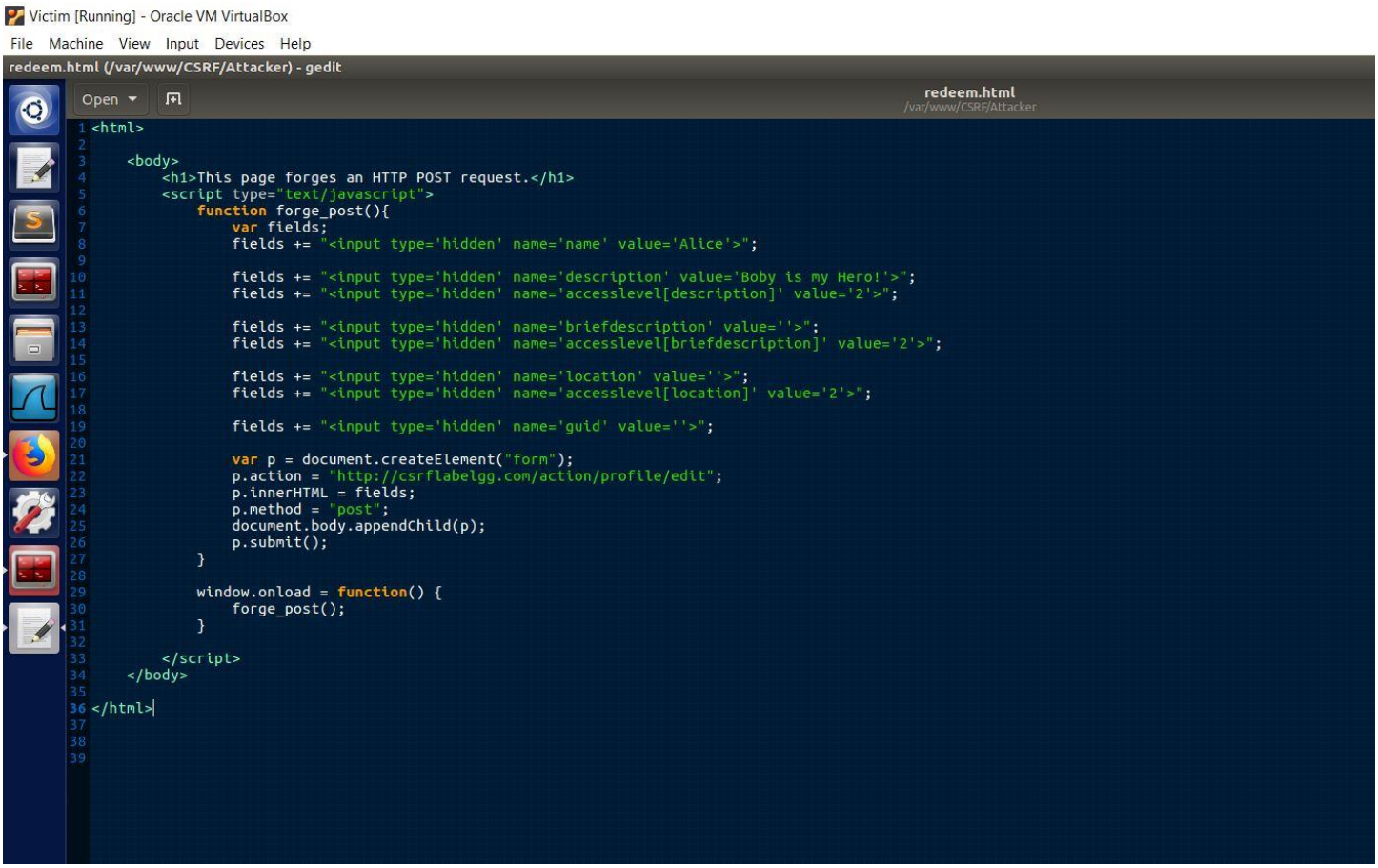


Using the developer tools, we will observe the POST request sent when profile is updated. From the above screenshot we observe that a POST request is sent to /action/profile/edit. In addition, the params section shows the form data that is sent with the request. We observe that various fields such as guid, description etc are sent as form data along with the POST request.

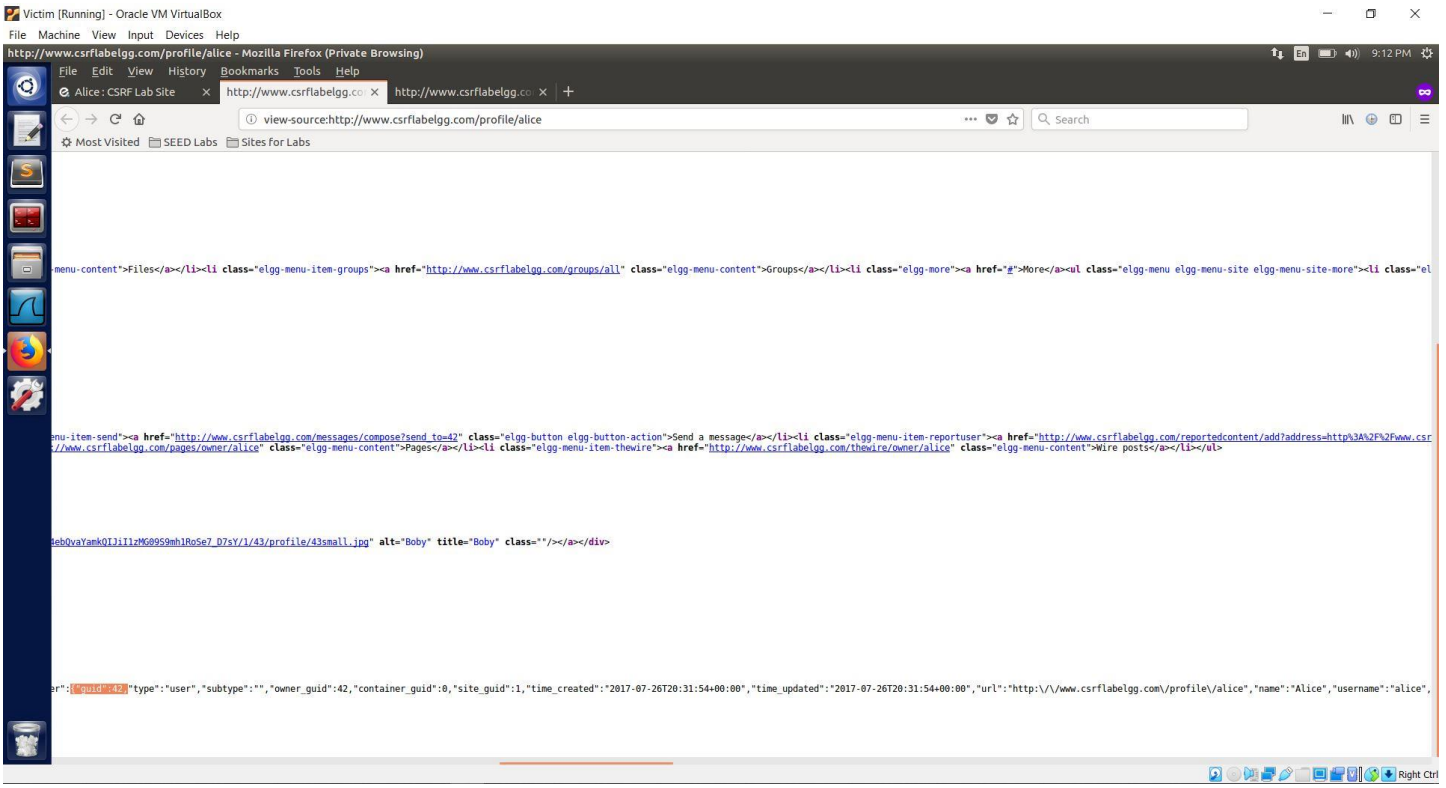


Now we will create the redeem.html of the attacker's website to send a POST request to http://www.csrflabelgg.com/action/profile/edit with the appropriate form fields. This form is again

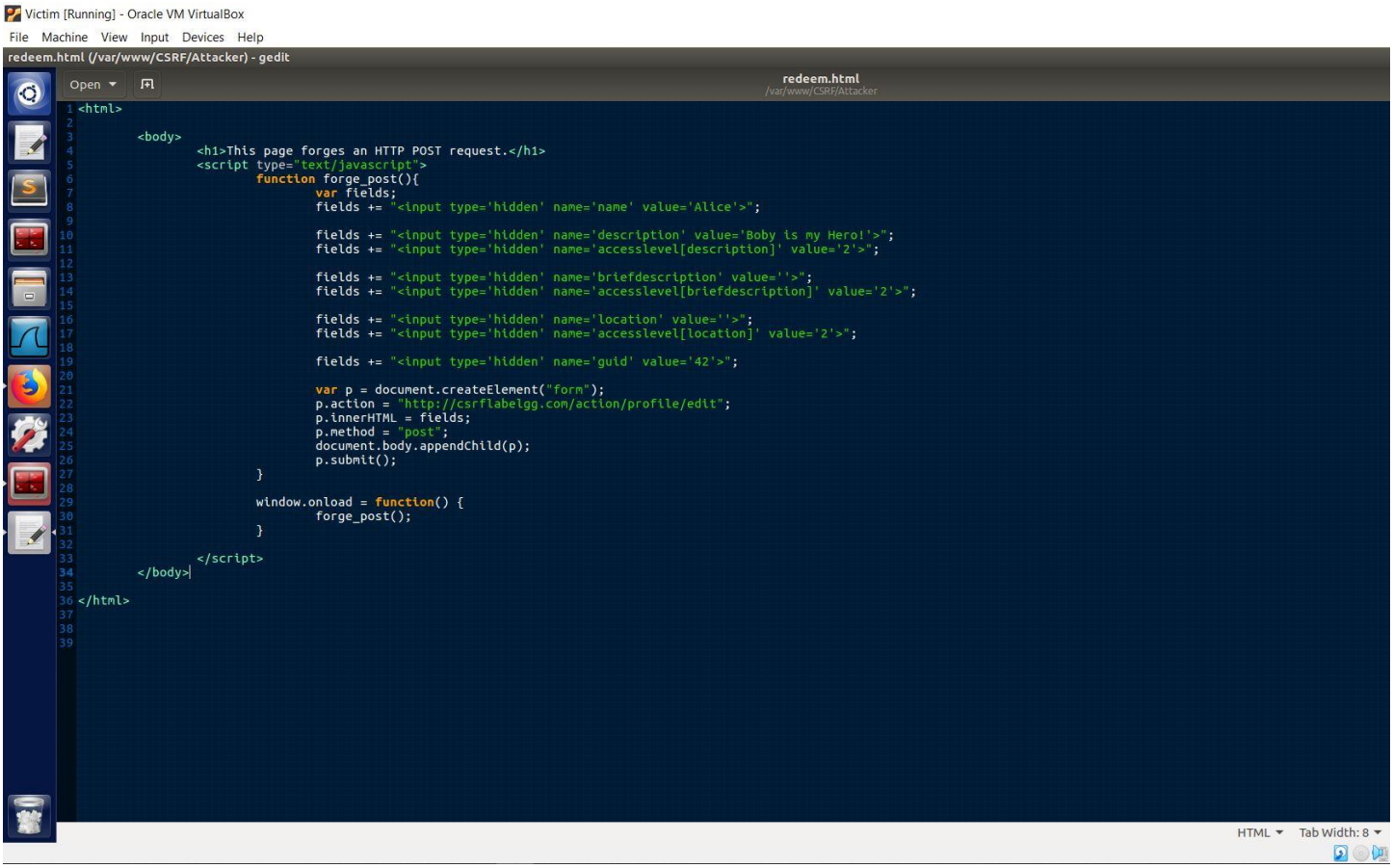
kept hidden from the user and the form is automatically submitted when the page loads. The updated redeem.html is shown below.



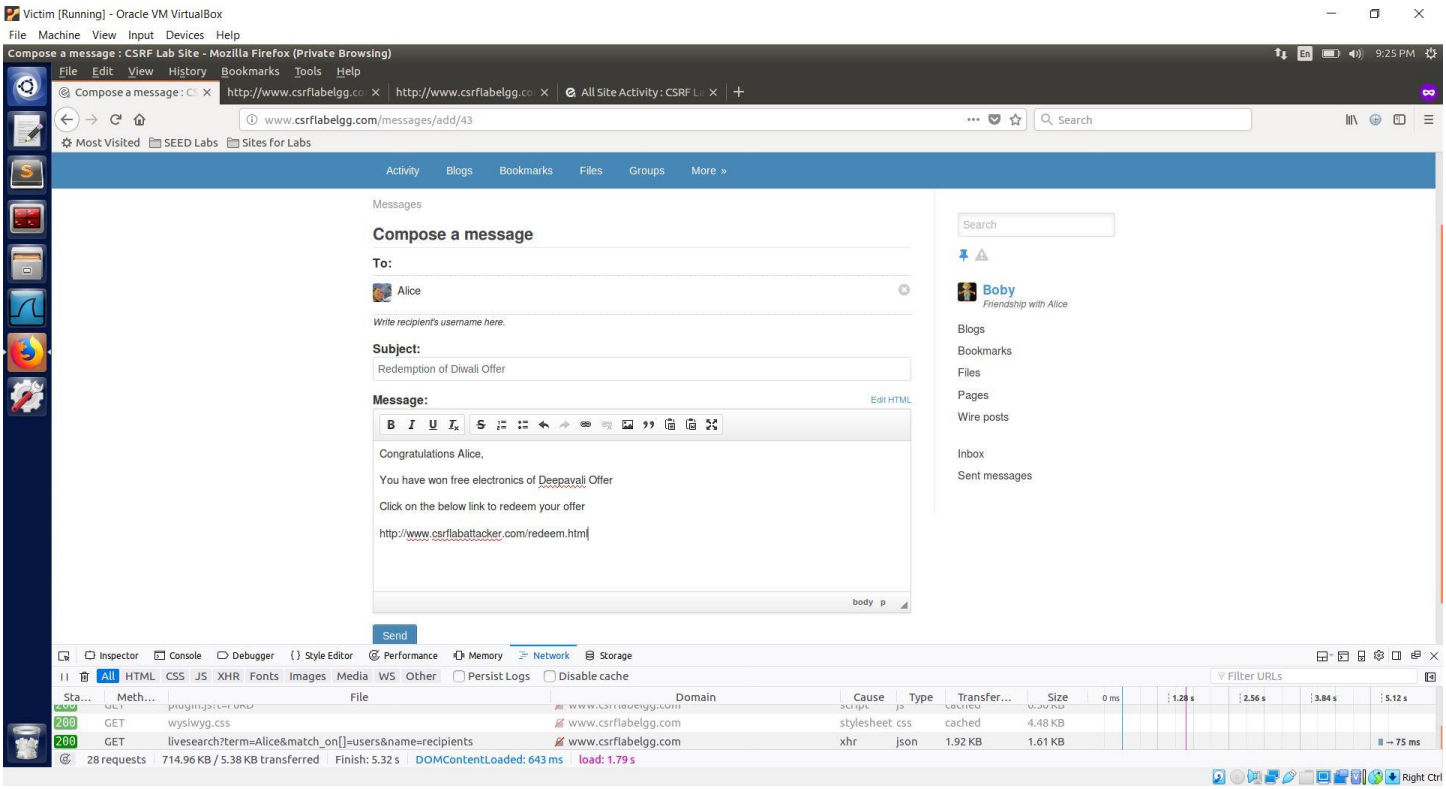
We also require the guid of Alice as the form field. We can find this by going to members page and then viewing the page source that contains the guid value of Alice saved in a Javascript variable.

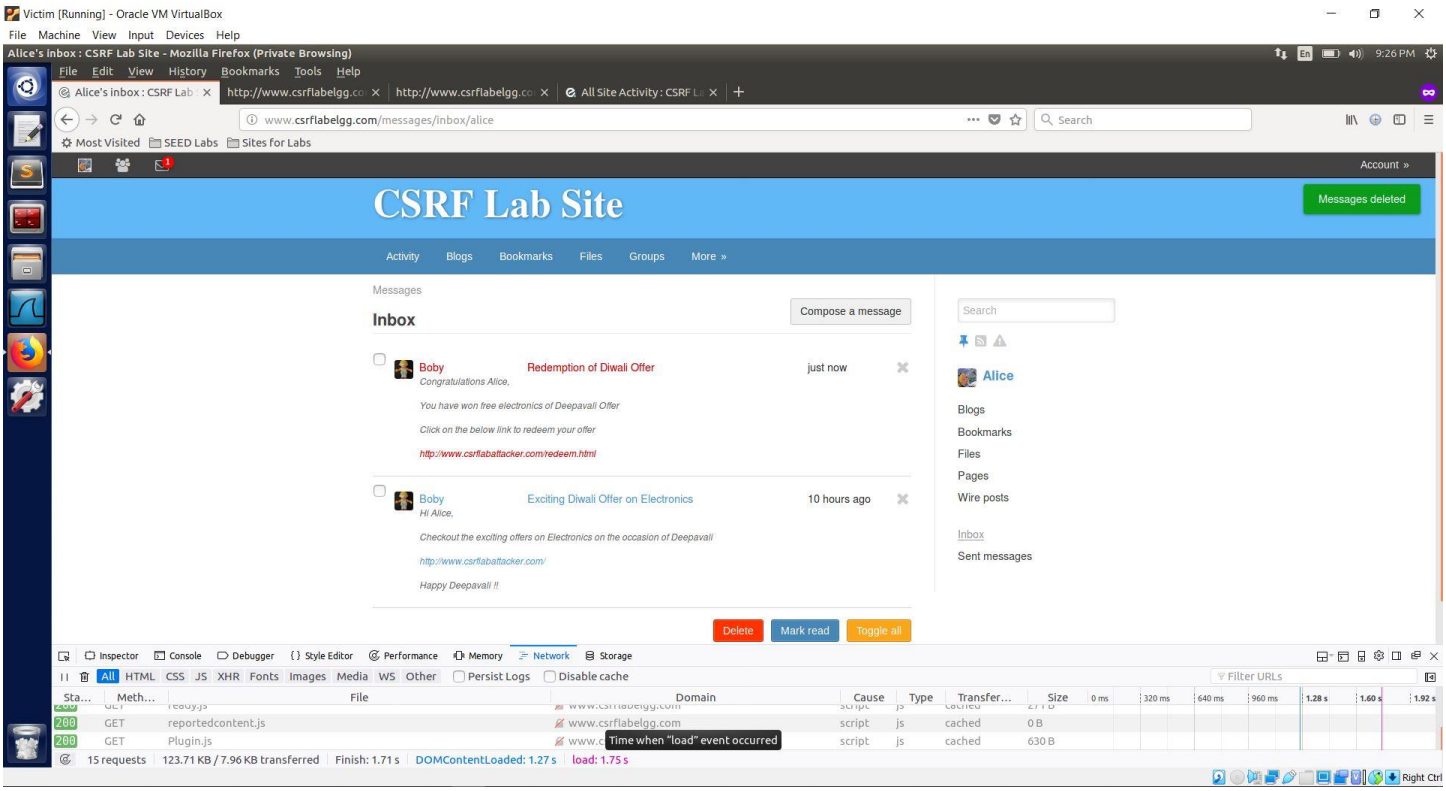


We update the redeem.html with the guid value of Alice which is 42.

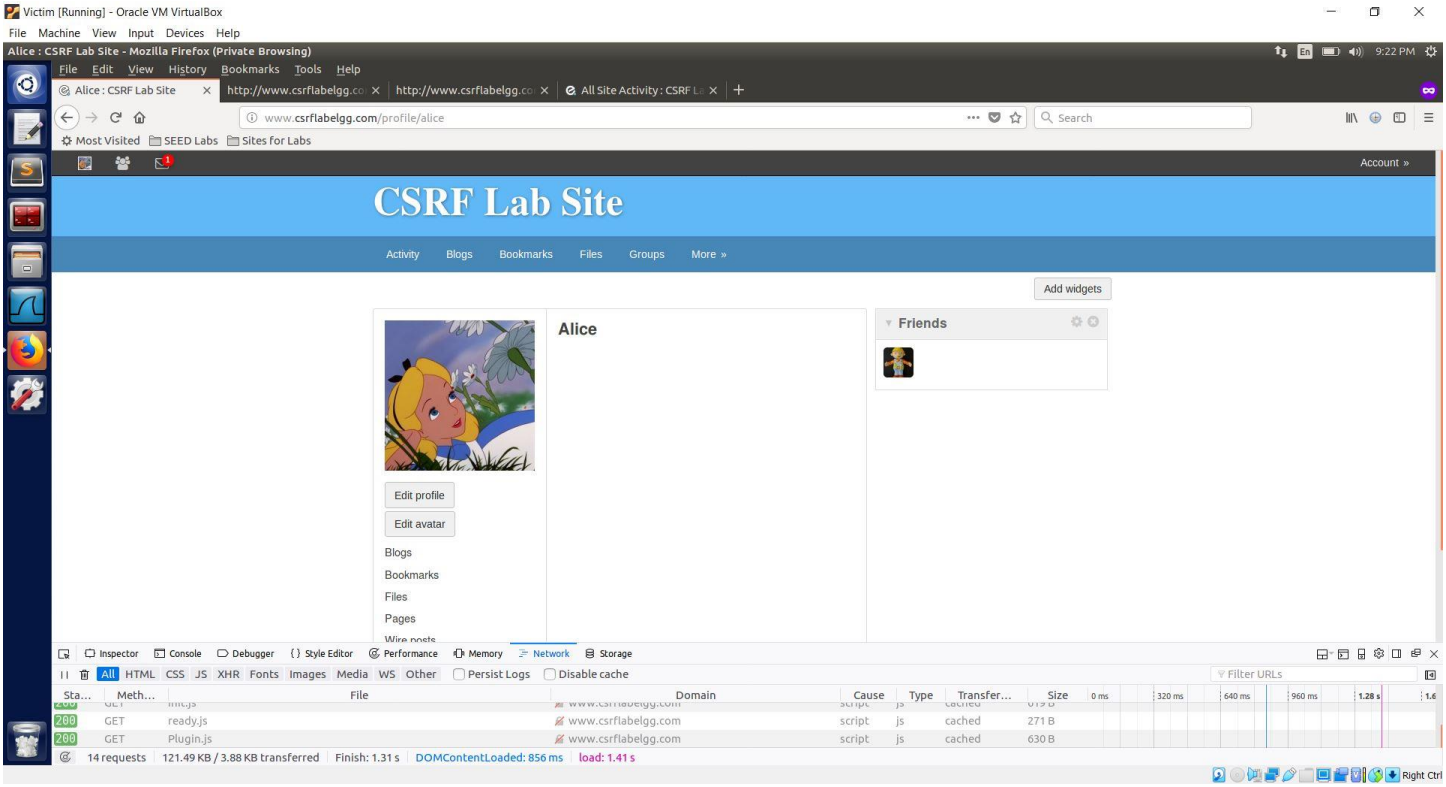


Again we compose a message and send it to Alice.



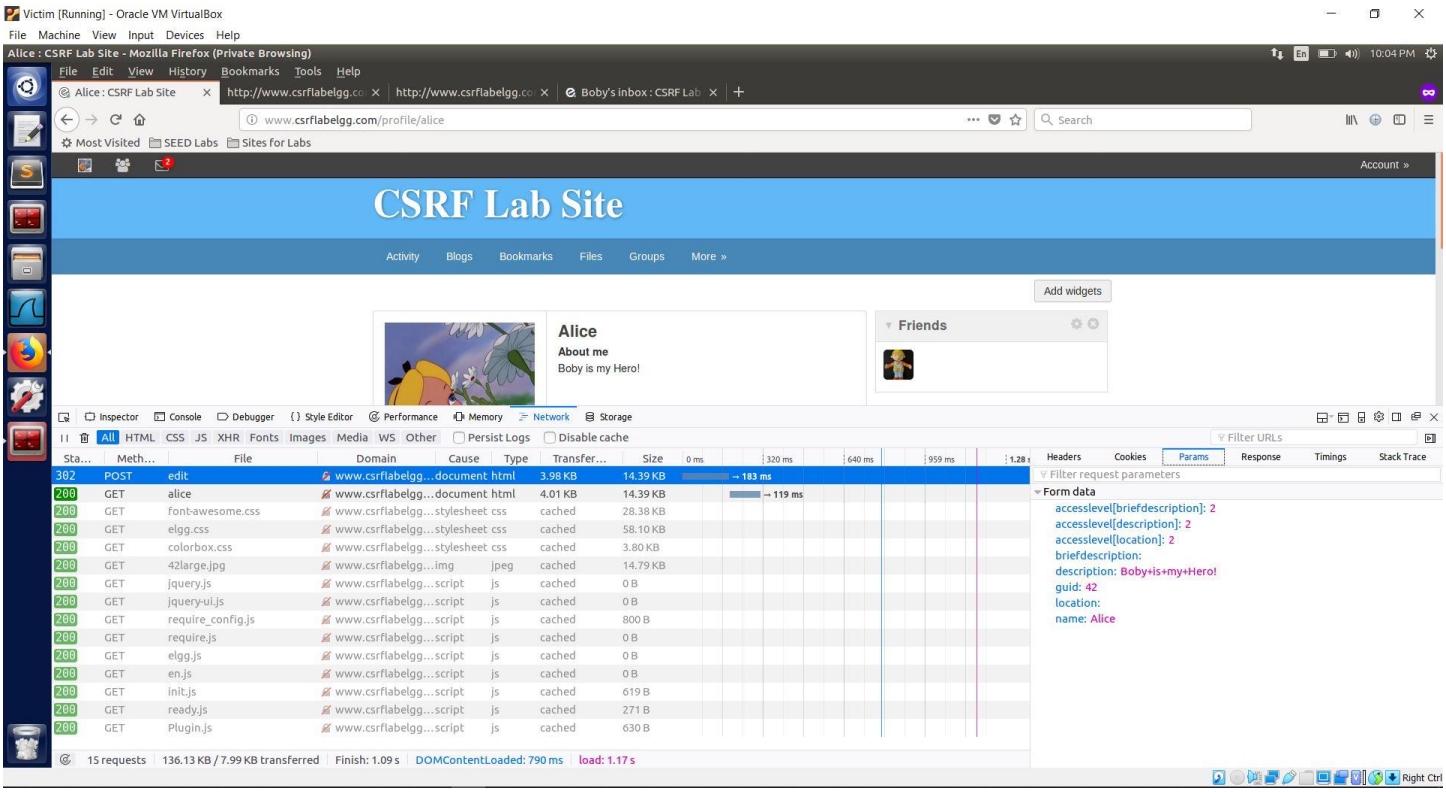


The below screenshot shows Alice's profile before the CSRF attack. We observe that there is not about me section in Alice profile.



When Alice clicks on the link sent in the message he is redirected to the attackers website which further sends a POST request which adds 'Boby is my Hero!' in Alice's profile.

The below screenshot shows Alice's profile after the CSRF attack. We observe that 'Boby is my Hero!' is added as Alice's about me. Thus the CSRF attack using POST request was successful.



Question 1: The forged HTTP request needs Alice's user id (guid) to work properly. If Boby targets Alice specifically, before the attack, he can find ways to get Alice's user id. Boby does not know Alice's Elgg password, so he cannot log into Alice's account to get the information. Please describe how Boby can solve this problem.

Answer 1: We can retrieve the guid of Alice from Boby's account by visiting Alice's profile and then clicking on view page source, the guid of Alice is stored in elgg variable in the javascript tag in the html along with the guid of Boby

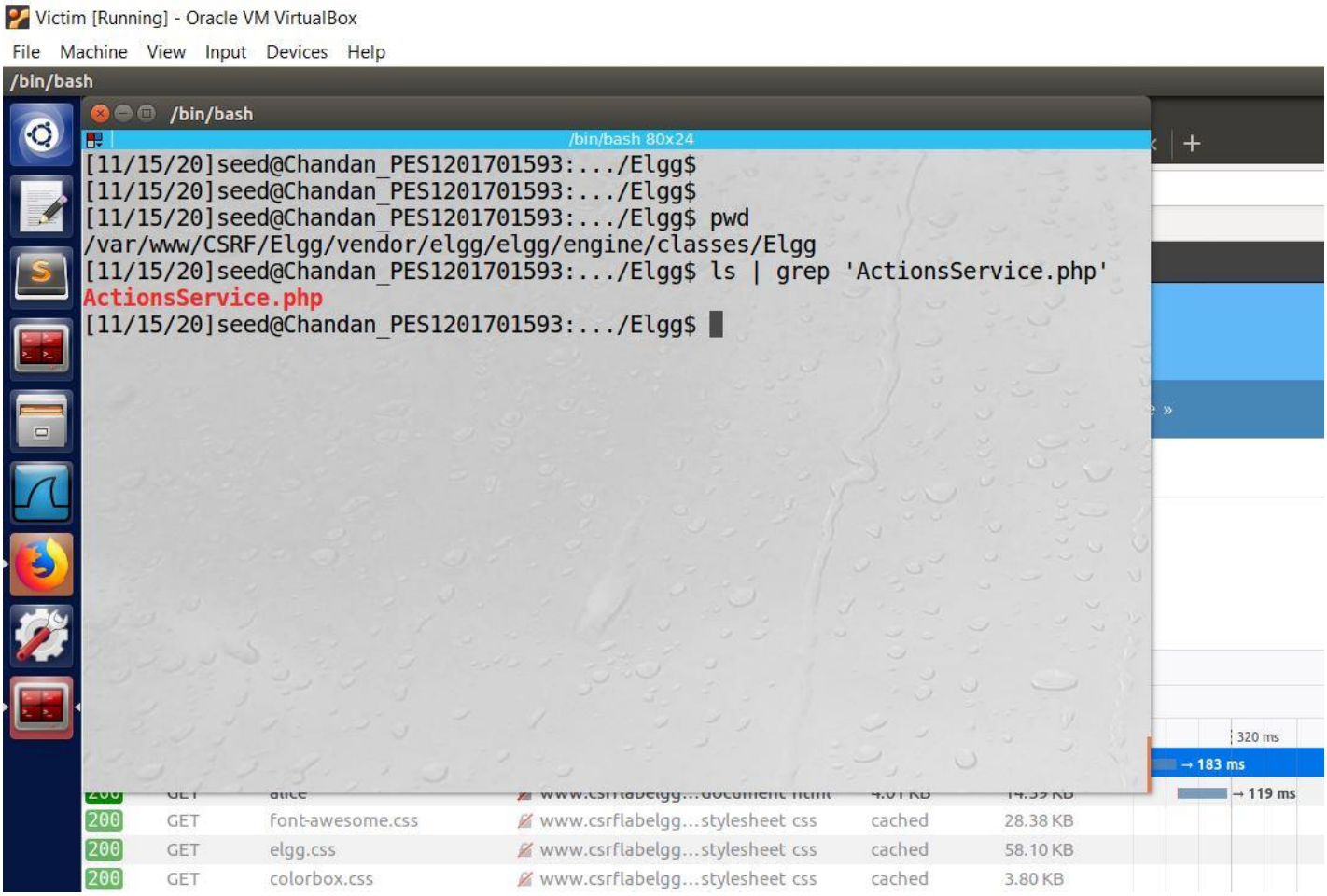
Question 2: If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim's Elgg profile? Please explain.

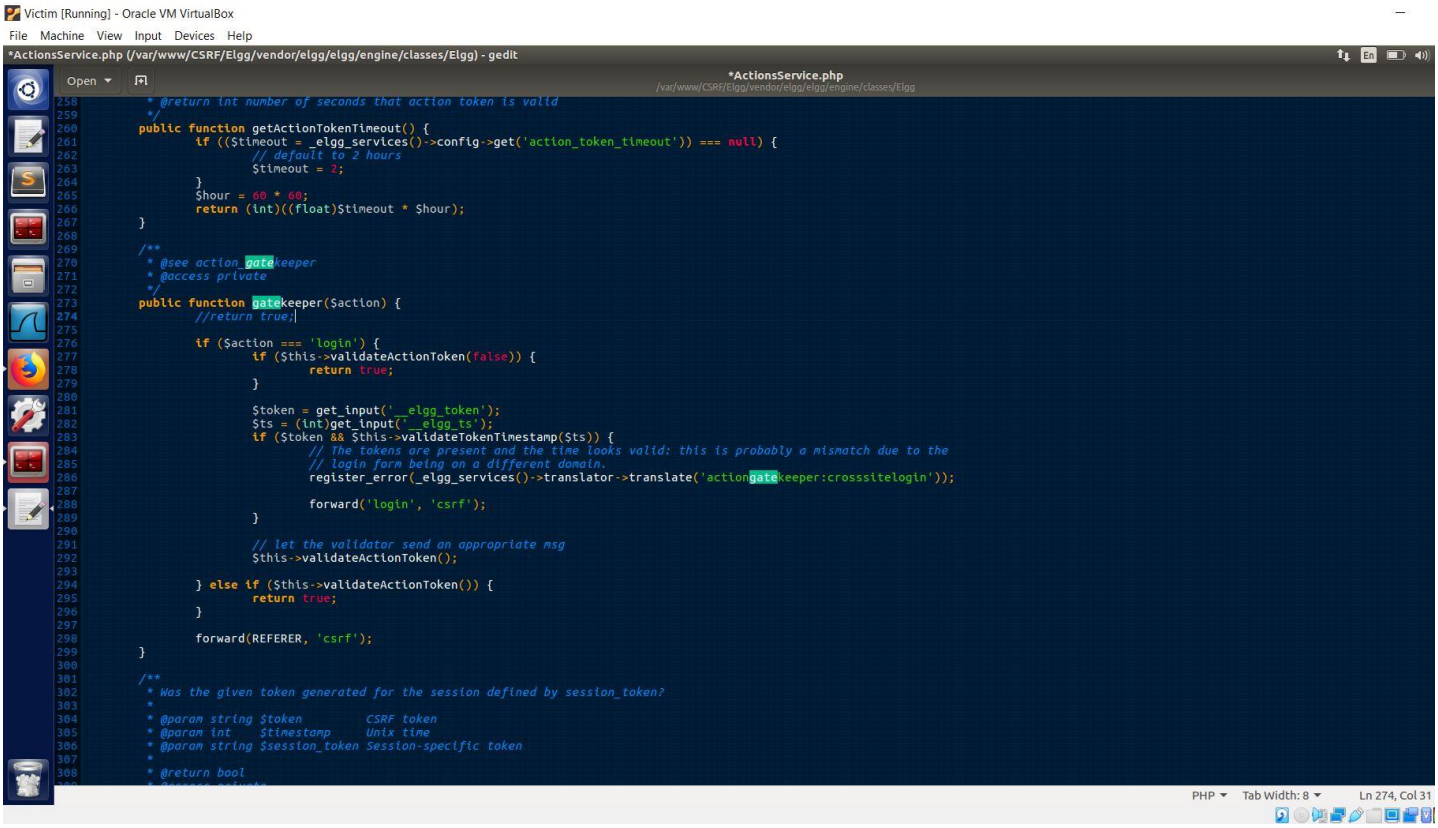
Answer 2: In this case it is not possible to perform a CSRF attack because we will not be able to access the guid of the user, since only the cookie is passed and we will not be able to view the source page from Javascript disabling us from getting the username or guid of the user.

Task 4: Implementing a Countermeasure for Elgg

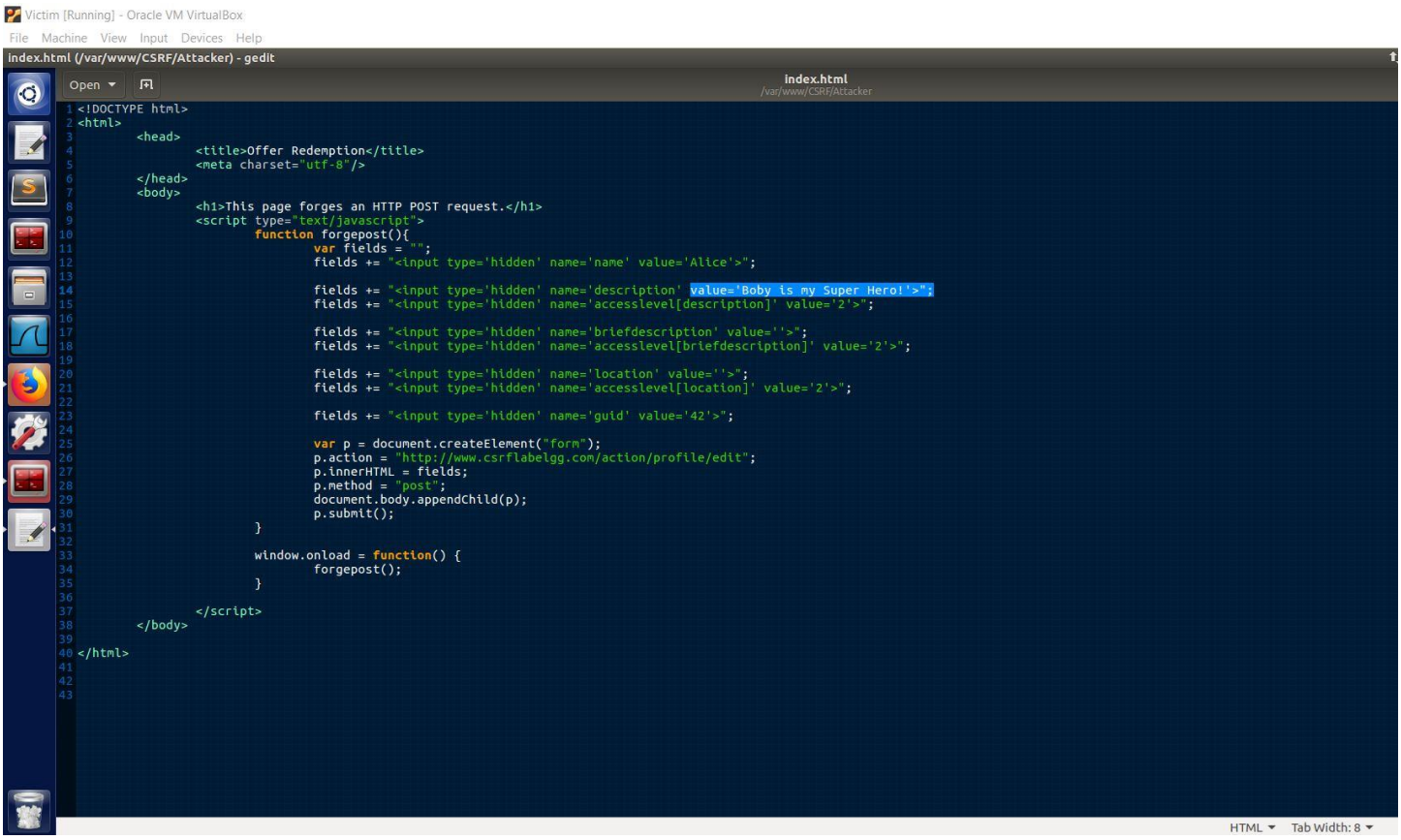
In this task we try to defend against CSRF attacks. There are two approaches which can be used to counter them. They are: Secret-token approach - Web applications can embed a secret token in their pages, and all requests coming from these pages will carry this token. Because cross-site requests cannot obtain this token, their forged requests will be easily identified by the server. Referrer header approach - Web applications can also verify the origin page of the request using the referrer header. However, due to privacy concerns, this header information may have already been filtered out at the client side. We now turn on the secret-token countermeasure by visiting '/var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg'. We need to find the function named

gatekeeper in ‘ActionService.php’ and then we comment out ‘return True’ statement. This is shown in the screenshot below.

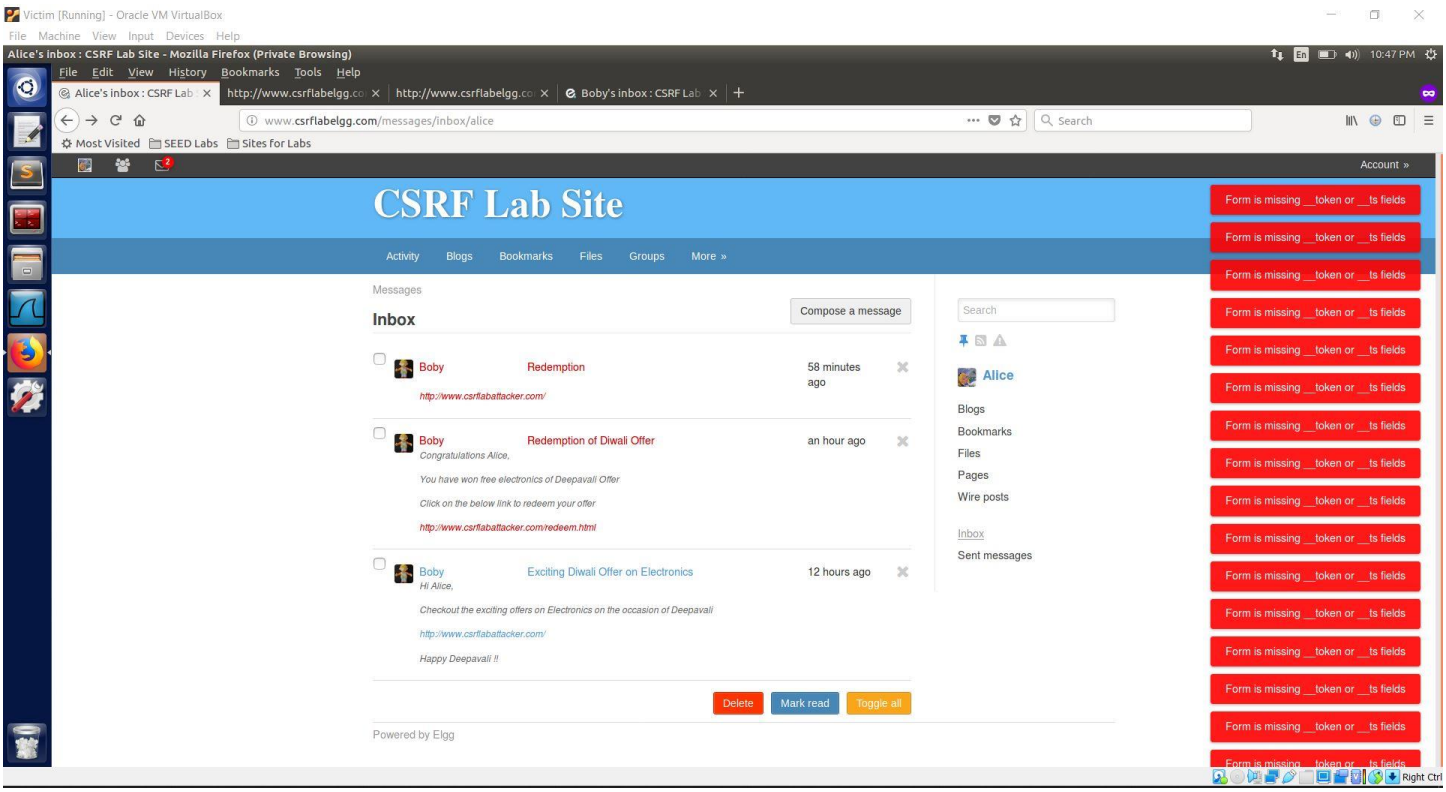


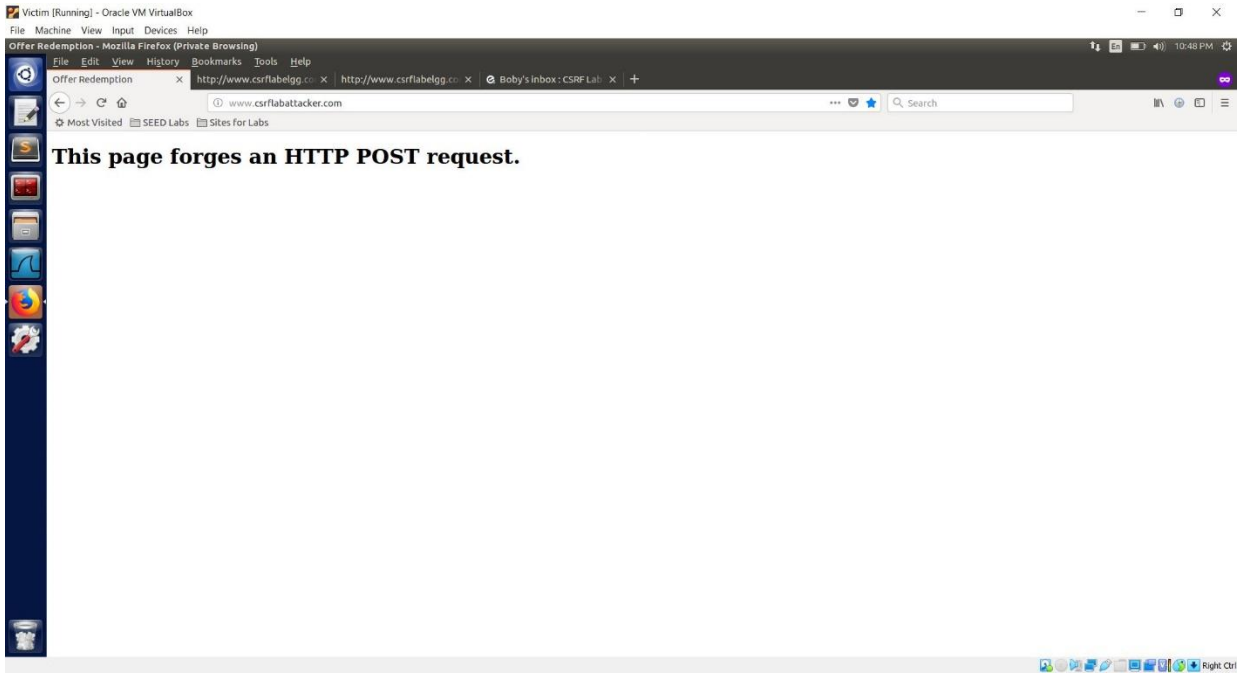


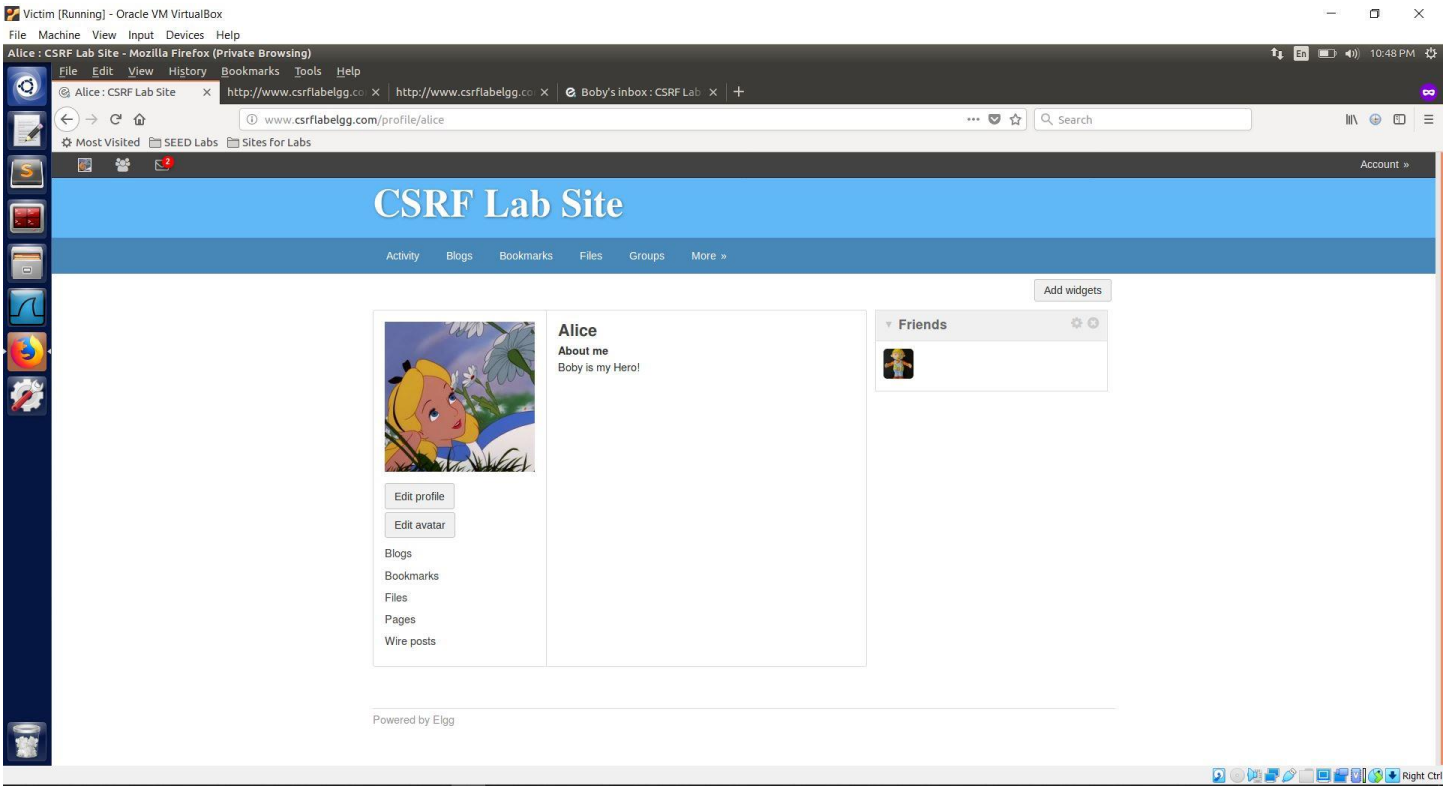
We now try clicking on the same malicious file from Alice account and check if we able to change the profile. We first edit our index.html to change the profile from 'Boby is my hero' to 'Boby is my super Hero!. This change is shown in the screenshot below.



Now we click on the link and check if we can change the profile.







From the above screenshot we observe that the CSRF attack wasn't successful. Thus, the profile of Alice was not changed/updated. We are unable to submit the form because the form is not assigned a secret token which is being checked in the backend before any update is made to the profile. Every time the backend sends the form to the browser it attaches a newly generated secret token which needs to be submitted by the webpage to validate its authenticity. Since the attacker's site is not generated by the elgg backend it does not contain the token and hence it will not be able to send any token and the form is invalidated resulting is countering the CSRF attack.

THANK YOU