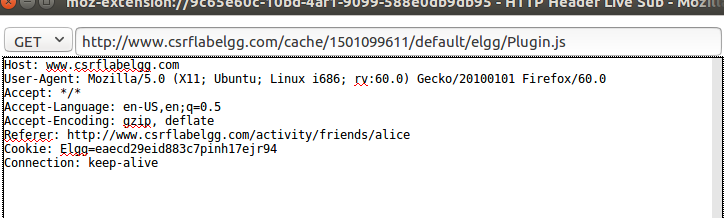
Cross-Site Request Forgery (CSRF) Attack Lab

## Task 1: Observing HTTP Request.

Get:



﻿ ﻿

Host: www.csrflabelgg.com

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

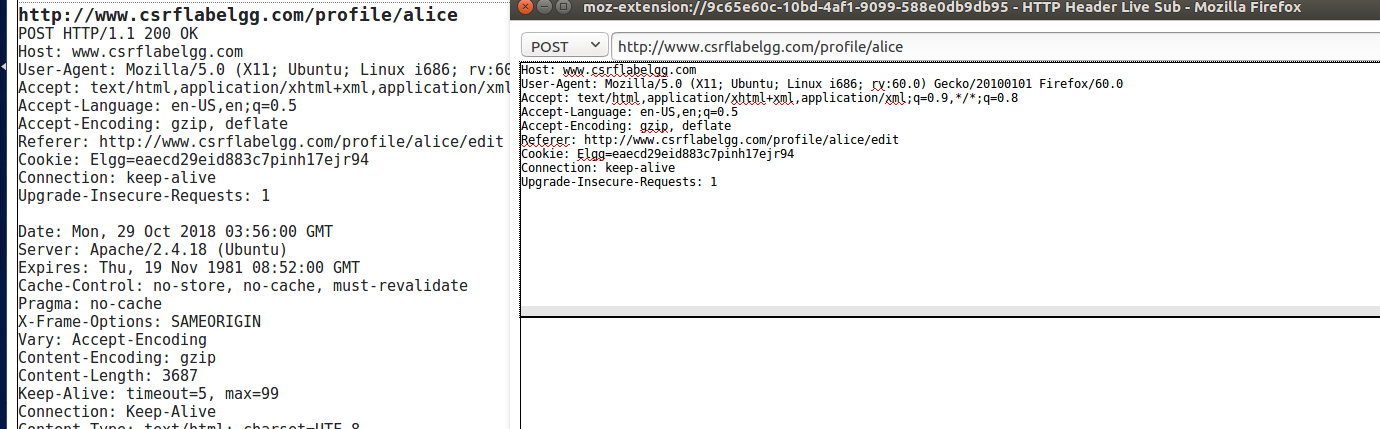
Accept-Encoding: gzip, deflate

Referer: http://www.csrflabelgg.com/activity/friends/alice

Cookie: Elgg=eaecd29eid883c7pinh17ejr94

Connection: keep-alive

Post:



﻿

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/alice/edit

Cookie: Elgg=eaecd29eid883c7pinh17ejr94

Connection: keep-alive

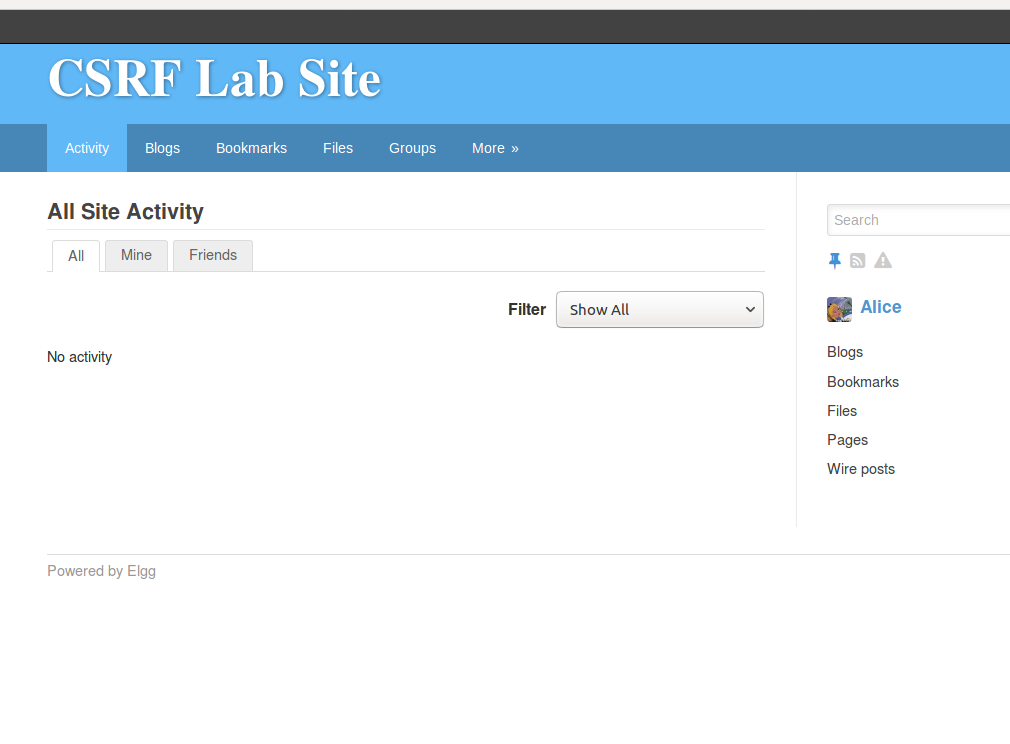
Upgrade-Insecure-Requests: 1

We can see that both Get and Post need configure parameters like Accept-Language and user-agent.

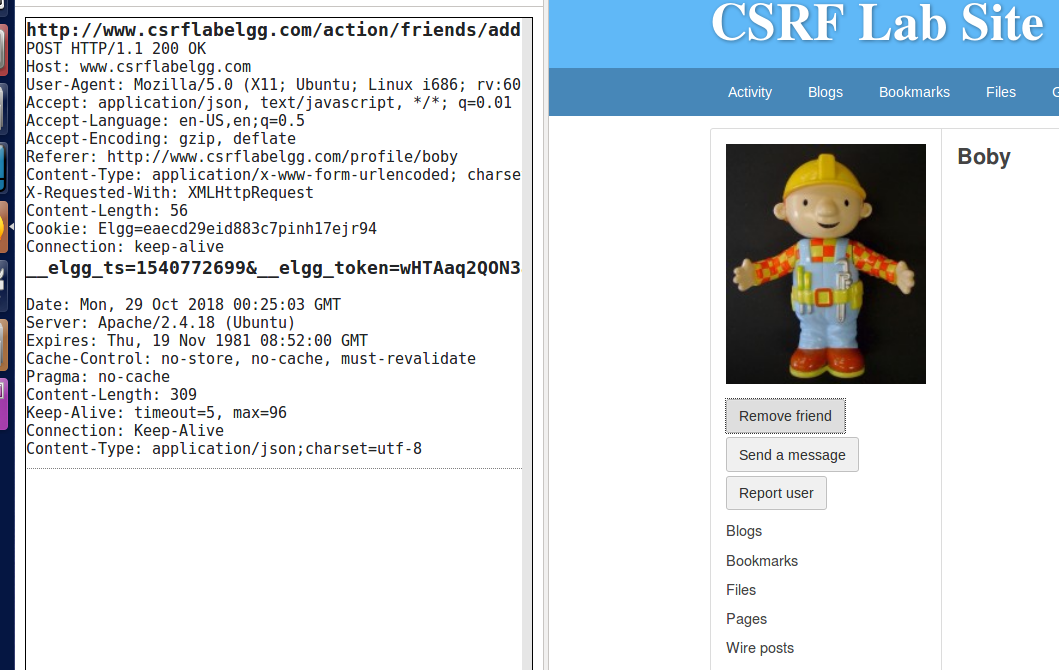
## Task 2: CSRF Attack using GET Request

Record the request of adding a friend.

Firstly, login as Alice:



Then, clear and add friend to record:

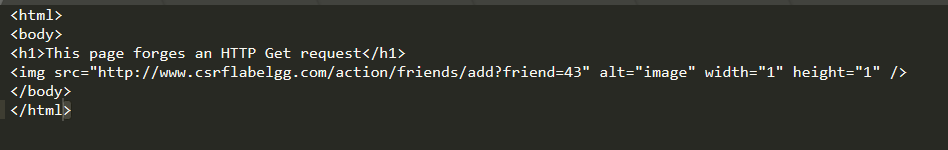




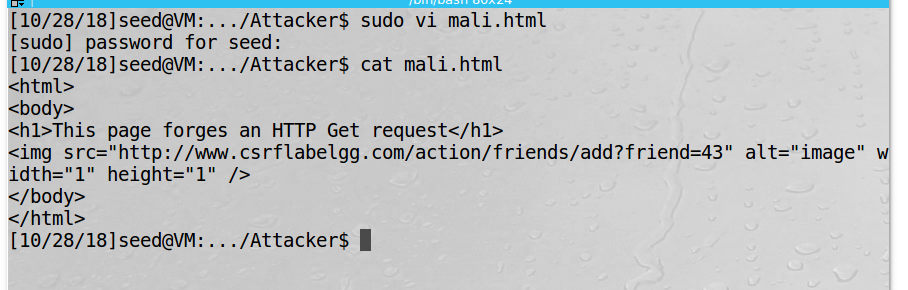
As we can see, from the HTTP Header Live, we can see all the information of the adding friend request. With headers, we can find the add friend action is <http://www.csrflabelgg.com/action/friends/add?friend=43&__elgg_ts=1540772771&__elgg_token=aeBua8N5vBvWKjMDx2urVw>

In the above action, we noticed that the number of friend is 43, which means Boby has an id of 43. It also includes elgg\_ts and elgg\_token, which we cannot design. However, in this section the countermeasure is stopped, which means we can have Boby add friend only with the first part.

To achieve this, first we need to modify a malicious web page.



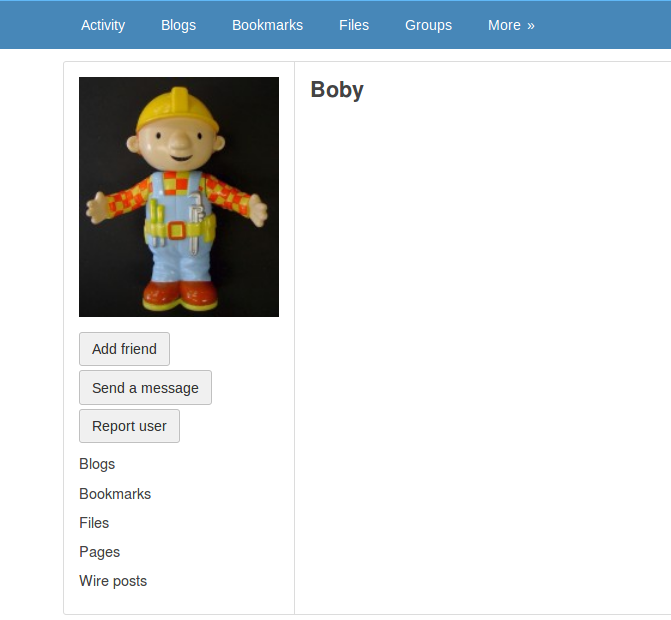
Save the page in the attacker’s server.



After doing this, we can load the web page into the server, which means we can easily get to the page only by visiting the link

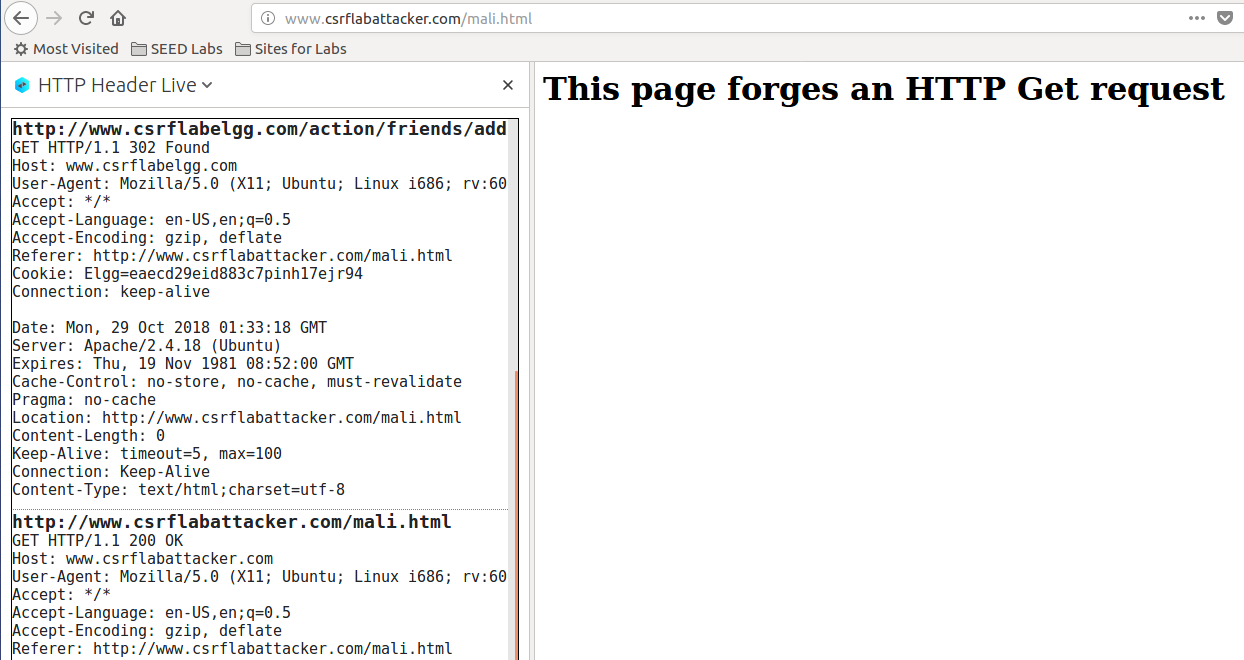
<http://www.csrflabattacker.com/mali.html>

Before:



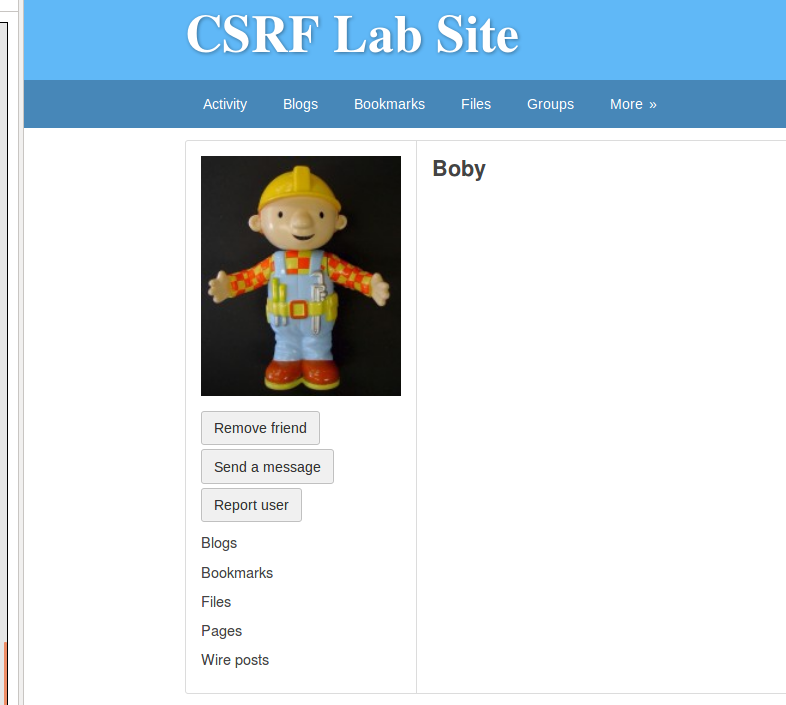
We can see that Boby is not Alice’s friend now.

After typing in the malicious website:



We can see from the left that besides the content of the page, the mali.html page sent an add friend request.

Back to the page of Alice’s friend list and refresh the page:



Obviously, the add friend button becomes remove friend button, which means that Alice has added Boby as friend.

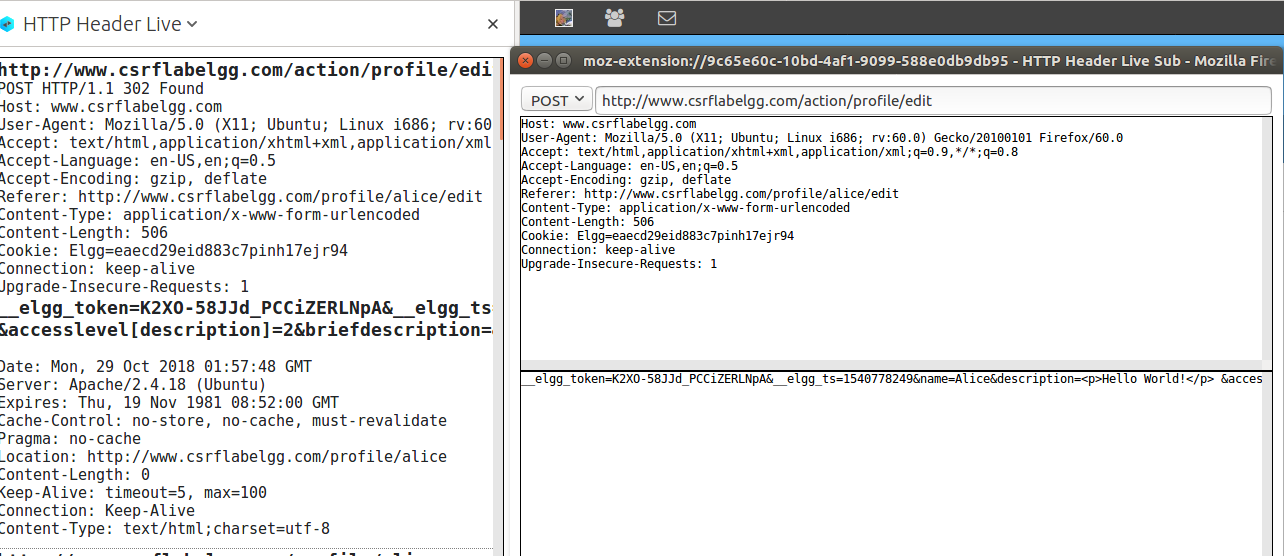
The main reason why this happens is that inside the malicious page, we use img tag to trigger an HTTP GET request automatically. After loading the page, the request has been sent to Elgg’s server and then the friend adding request make effects/

## Task 3: CSRF Attack using POST Request

Firstly, we need to construct the post request.

Also, we need to investigate what requests can change the profile.

Modify the profile and submit, capture the post request as follow:





﻿

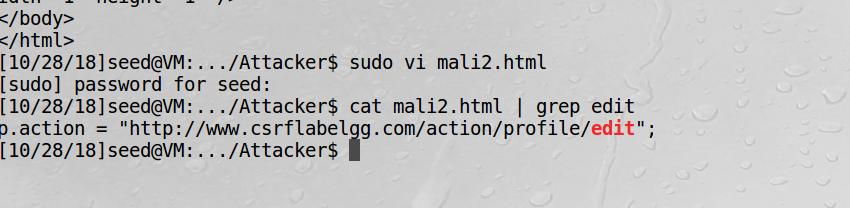
\_\_elgg\_token=mxh-MEtuLHDOH548trSXZg&\_\_elgg\_ts=1540779469&name=Alice&description=<p>Hello World!</p>

&accesslevel[description]=2&briefdescription=&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=42

Obviously, my input “hello world” is in the first post. We can easily see that the guid is 42.With the details given and investigation, we can complete codes as follow:



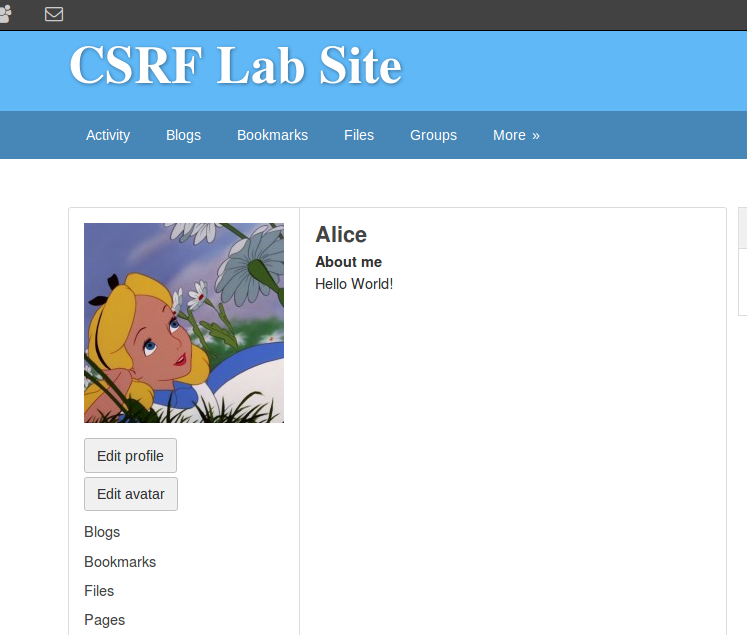
Save the html page into attacker’s server:



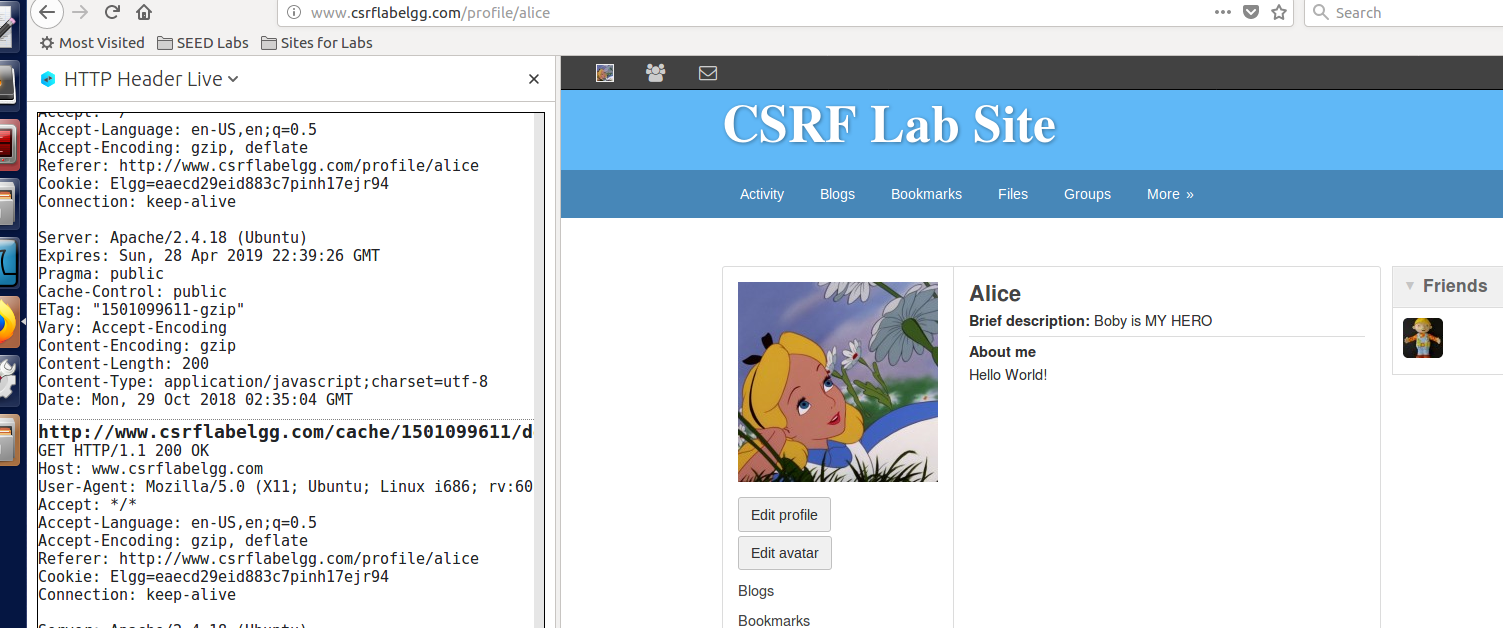
Finally, by typing in the new address, the attack will start:

<http://www.csrflabattacker.com/mali2html>

Before:



After:



We can see from the left that the post function is making effects and there is a post request sent. There is a quick flash of the malicious page content and then jump to Alice’s main page. And we see that the brief description is changed to “Boby is my hero” successfully.

The same, while loading the page a malicious post request was generated and sent to the Elgg’s server. If we write the request right, then the attack succeeds.

**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.**

To get Alice’s user id, Boby only needs to add friends to Alice or just send a message to Alice and after capture the requests he would get the user id. Only with the user id the attack will not succeed. What Boby needs as well is Alice’s cookie to that website. As a result, the attack cannot be down on Boby’s computer. Only when Alice logs in to the website and connections and cookies has been stablished and generated the attack can succeed on Alice’s computer.

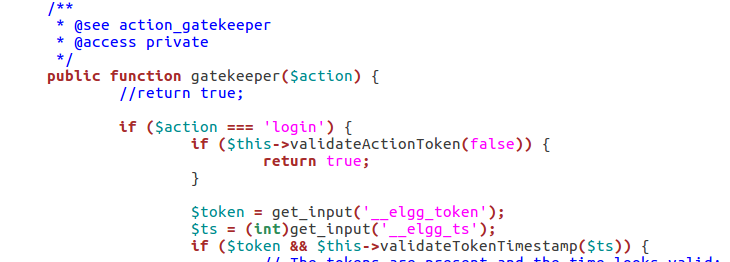
**Question 2: If Boby would like to launch the attack to anybody who visits his malicious web page. In this case, he does not know who is visiting the web page beforehand. Can he still launch the CSRF attack to modify the victim’s Elgg profile? Please explain.**

No he cannot. This is because he is not the Elgg’s server and cannot figure out who is visiting his malicious web page only from the ip addresses. As a result, he cannot get the user id for the visitors. Thus, Boby cannot design codes to generate malicious post request automatically.

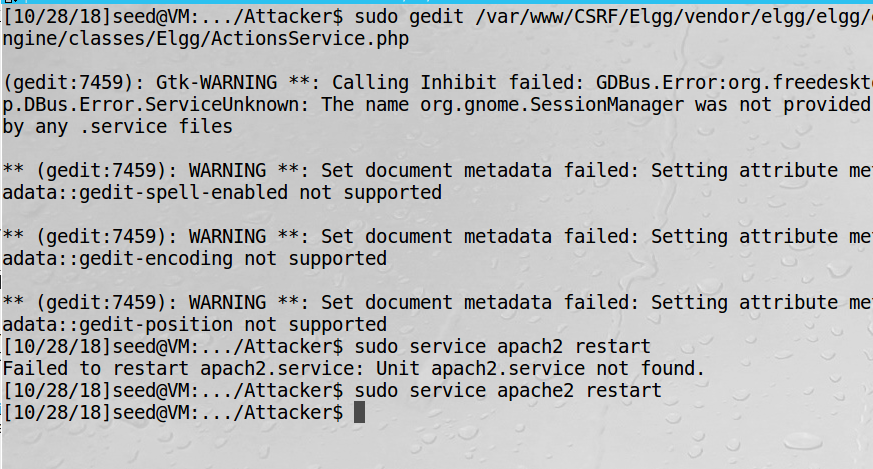
## Task 4: Implementing a countermeasure for Elgg

Turn on countermeasure.

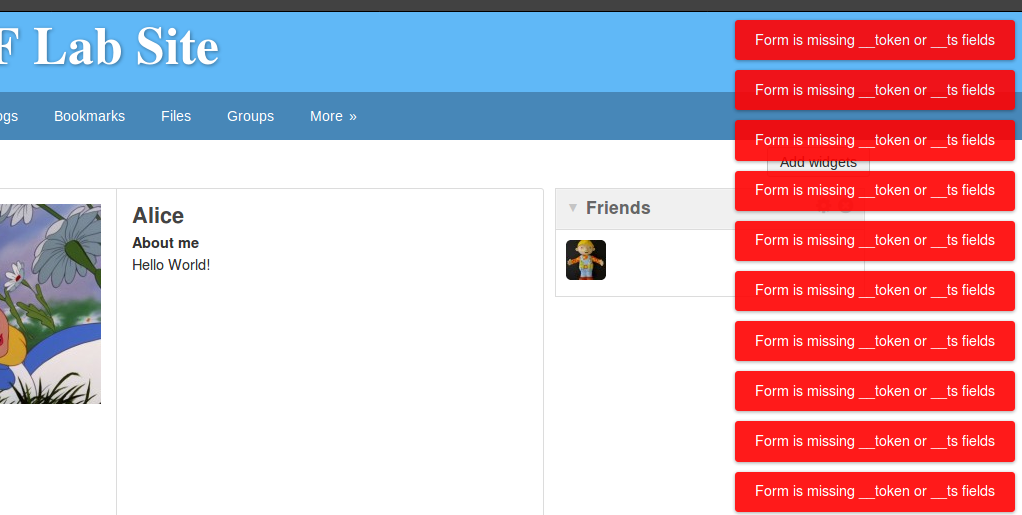
sudo gedit /var/www/CSRF/Elgg/vendor/elgg/elgg/engine/classes/Elgg/ActionsService.php

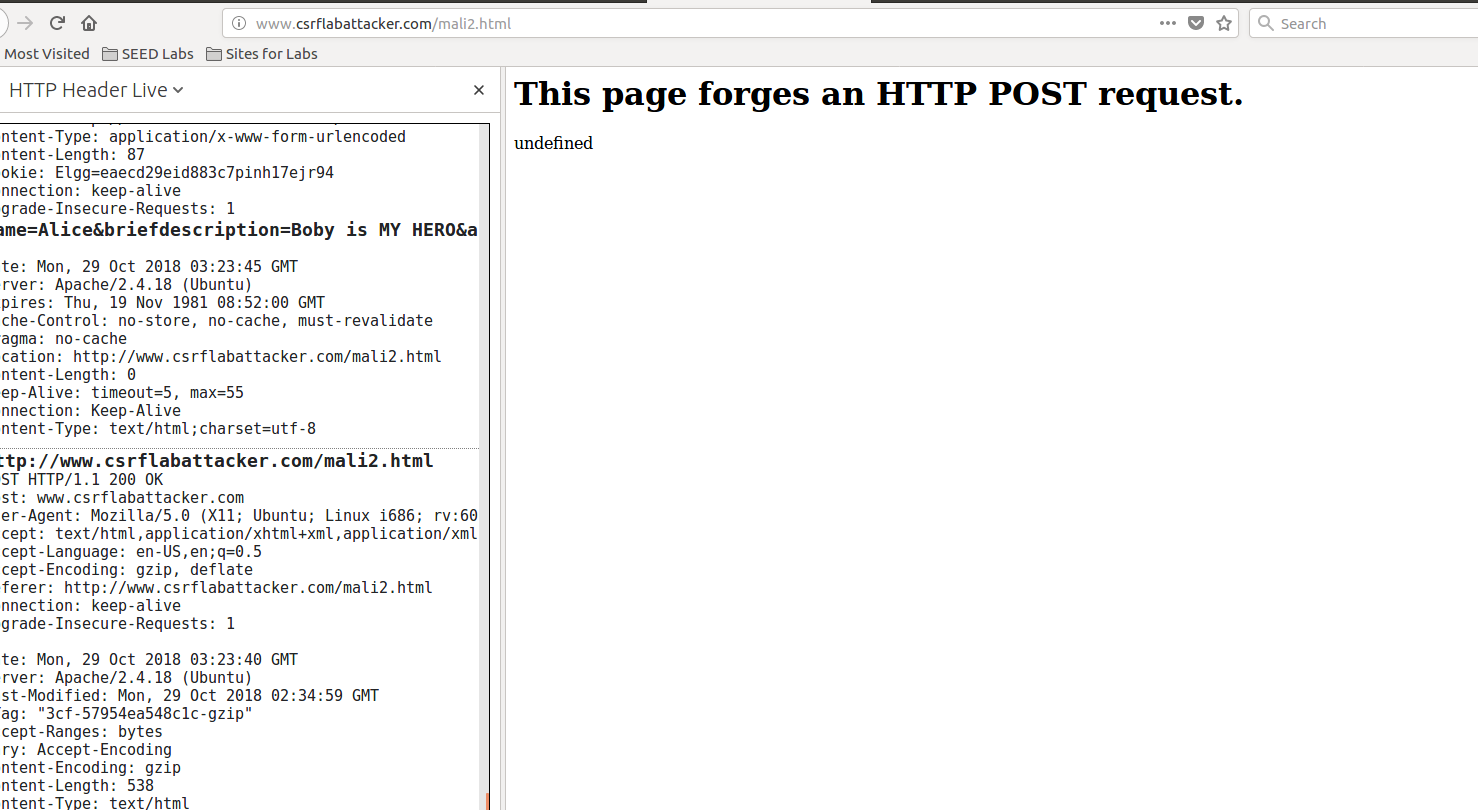


Then, restart the server.

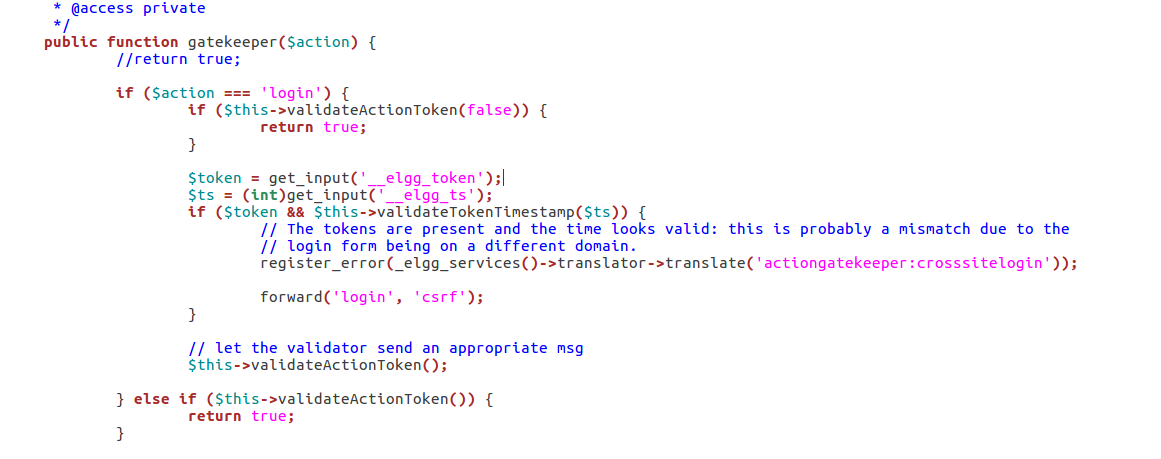


Reset Alice’s profile and redo task3 again:

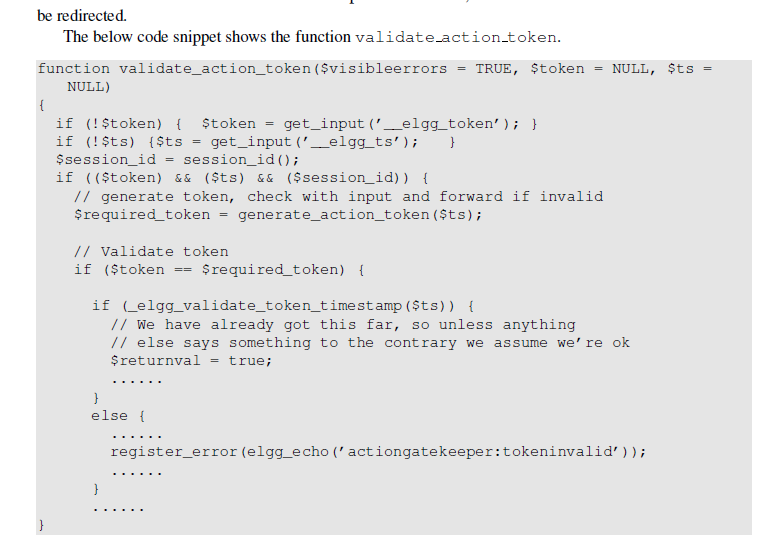




We can see that after lunching the malicious page, the attack is not successful. Back to the main page of Alice, we can see that plenty of error message appears saying form is missing \_token or \_ts field.



From the function we just modified, we can find the error is from validate\_action\_token. Before modifying the php file, the function gatekeeper will only execute the firstline and return true. But without that line, the following validations call a function called validateActionToken. In this function, token and ts are get from below:



The get\_input function will read the contents of the request and copy the values of ts and token. If we send the request as well without those two fields, errors will occur. Besides, if we just fill the two fields with random numbers, the attack fails as well because inside the validation the server will check if token equals required token and if ts is valid. Thus, without the two fields correctly, the attack fails.

To prevent attackers get the two fields, Elgg set the two parameters “hidden”. They are generated by the views/default/input/secruritytoken.php module. They are added dynamically to each page. So, we cannot read the two parameters directly from the website and we cannot guess what the two parameters are.