Wenbin Li SUID: 687150735

Lab 7 Cross-Site Request Forgery (CSRF) Attack Lab

Task 1:

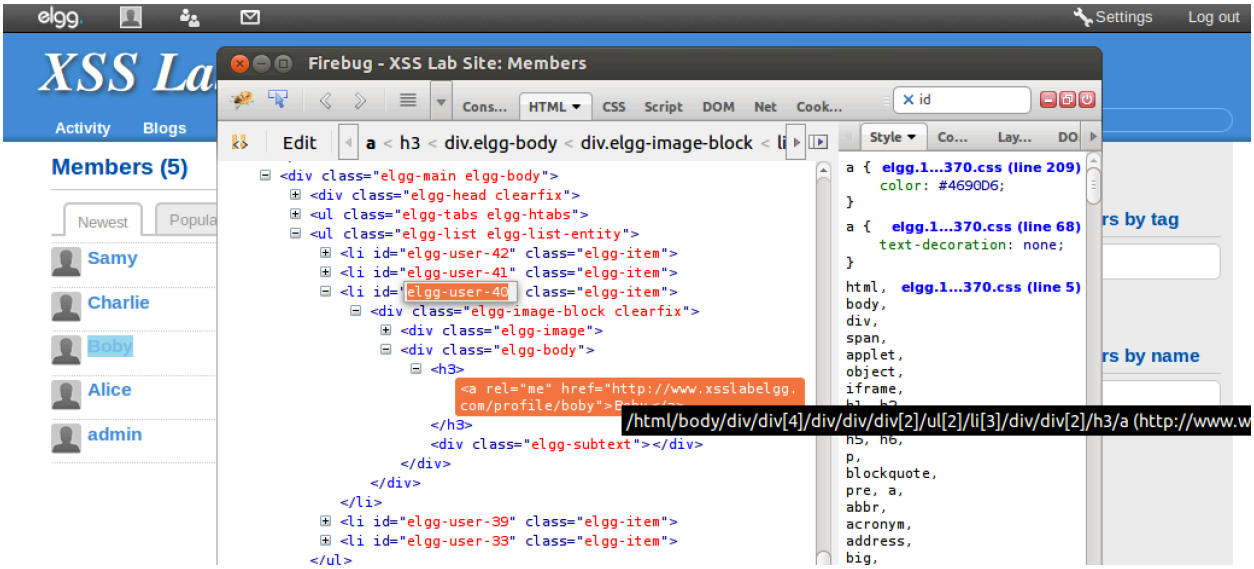


Fig 1.1 log in boby account, and inspect his user id.

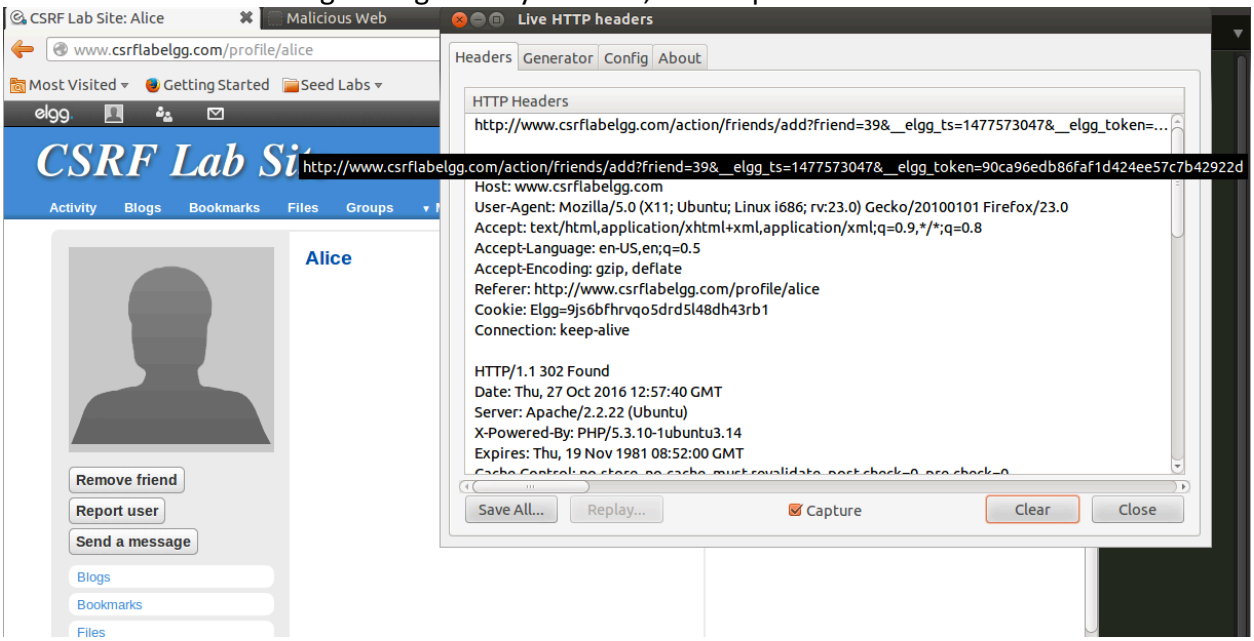


Fig 1.2 boby add alice as friend and check request command with live http head

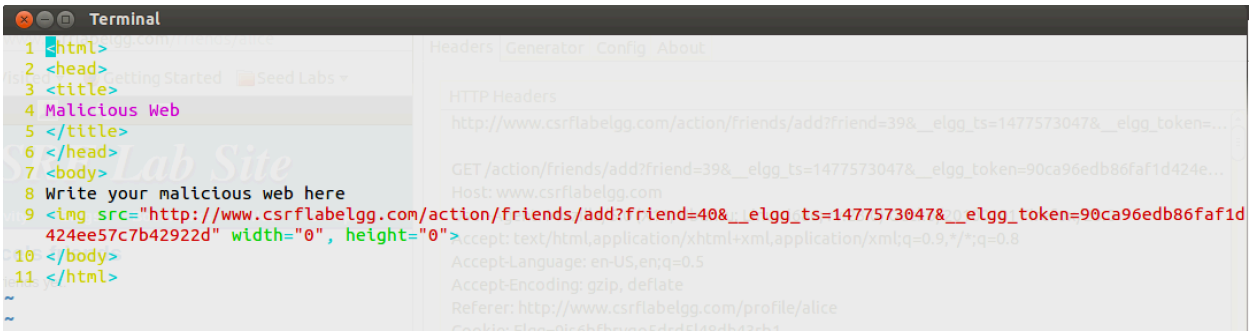


Fig 1.3 modify attack web site.

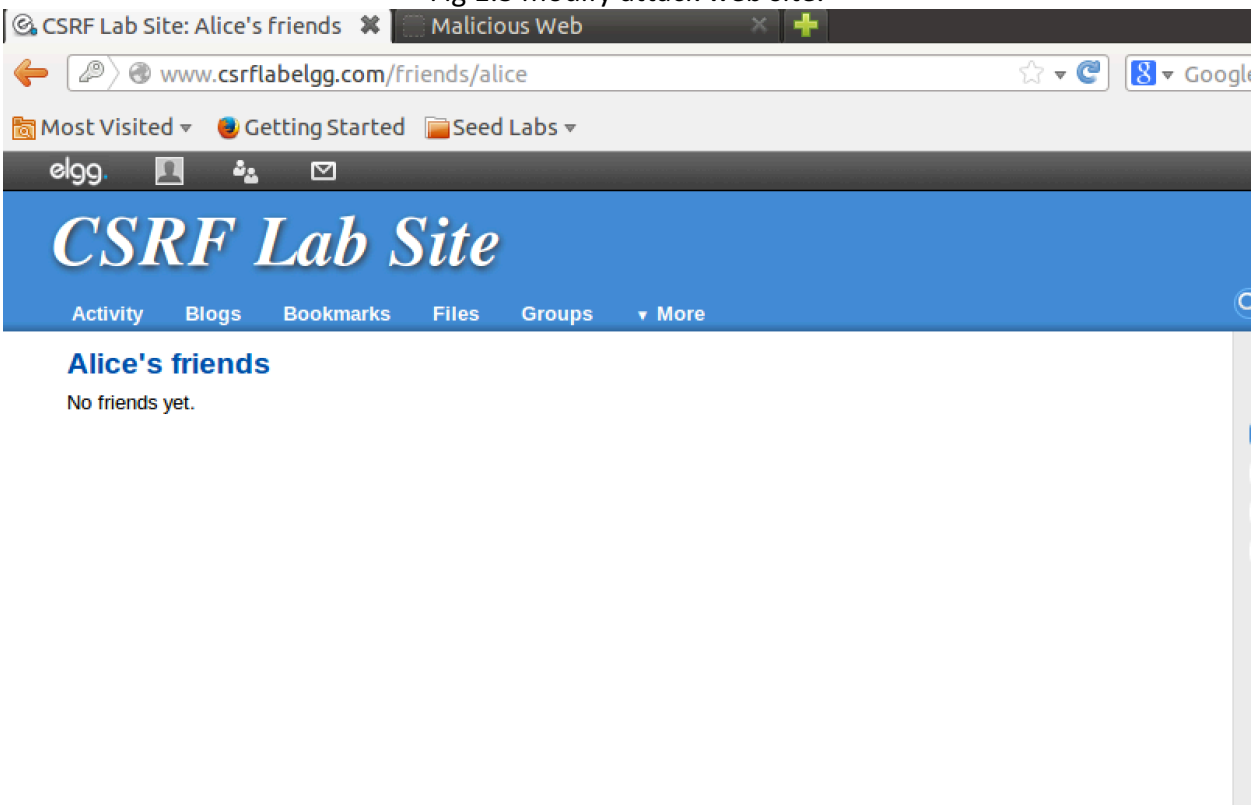


Fig 1.4 log in alice account, she had no friends

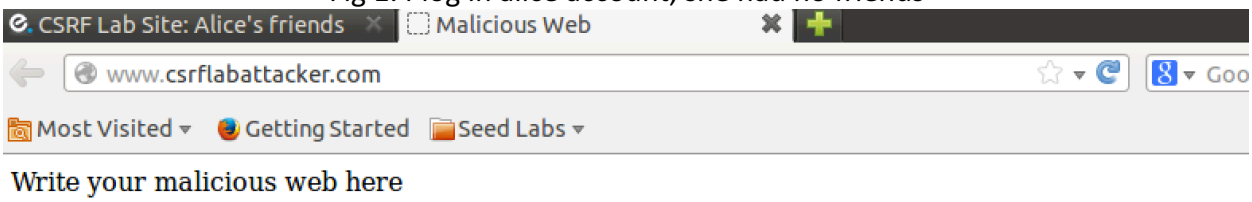


Fig 1.5 alice visited attacker website.

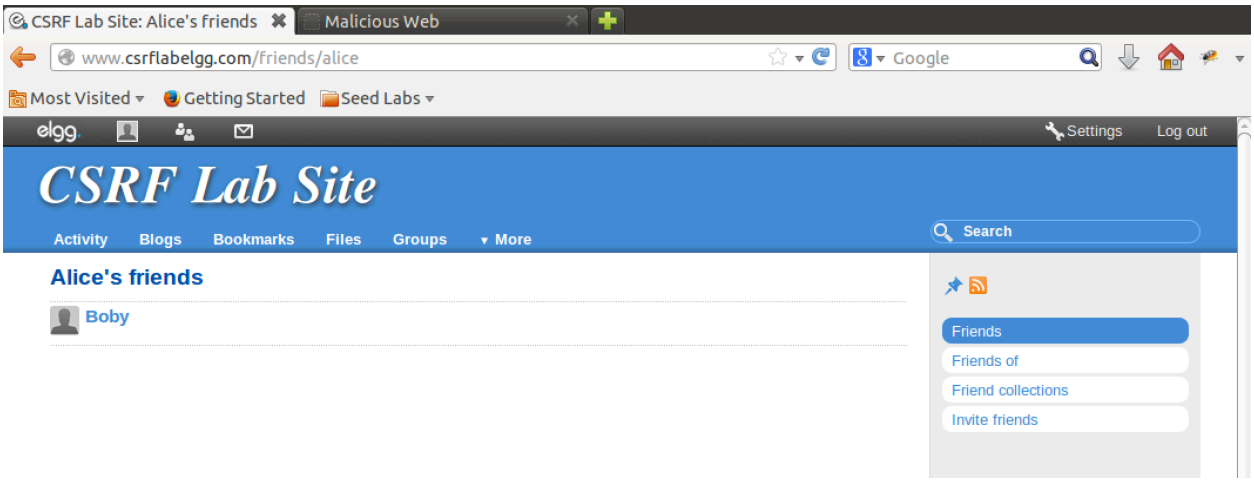


Fig 1.6 alice had added boby as a friend

Observation:

1. As fig 1.1, Boby logged in his account, and used firebug to inspect his userid in elgg application. The id is 40.
2. As fig 1.2, boby added alice as a friend, and check this http request with http live headers.
3. As fig 1.3, I modified attack web site. Actually, I wrote a http get request in <img> tag.
4. As fig 1.4, I logged in alice's account.
5. As fig 1.5, and 1.6, alice visited attacker website and then body was added in alice's friend list.

Explanation:

1. Use firebug inspect I can get boby's user id which could be used for this attack.
2. I used a real add friend request to forge the attack one, so I capture the request that boby add alice.
3. I wrote get request in img src, because this kind of request is atomically send when the html is loaded without click any buttons.
4. As fig 1.4, Alice logged into elgg. First time that alice logged in account, she needed to use password, and then elgg server sent cookies to alice's browser in headers. These cookies were saved in alice's browser. When alice sent reqeuset to server, the server could only use these cookies to verify user's identification. So when alice visited attacker website, the malicious request was sent to server with the cookies owned by alice. The server would verify these cookies successfully, including session id. So that the forgery request would be successful. So that when malicious website was loaded, the request was wrote in src was successful, boby was added as a friend for alice.

Task 2:

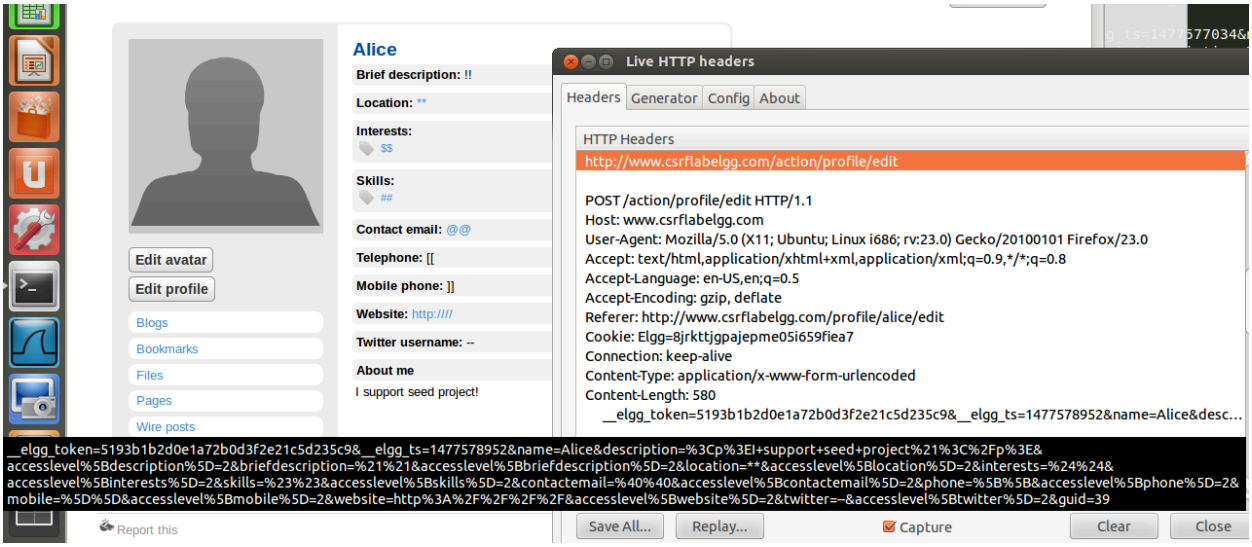


Fig 2.1 aice modified her profile to get structure of post request.

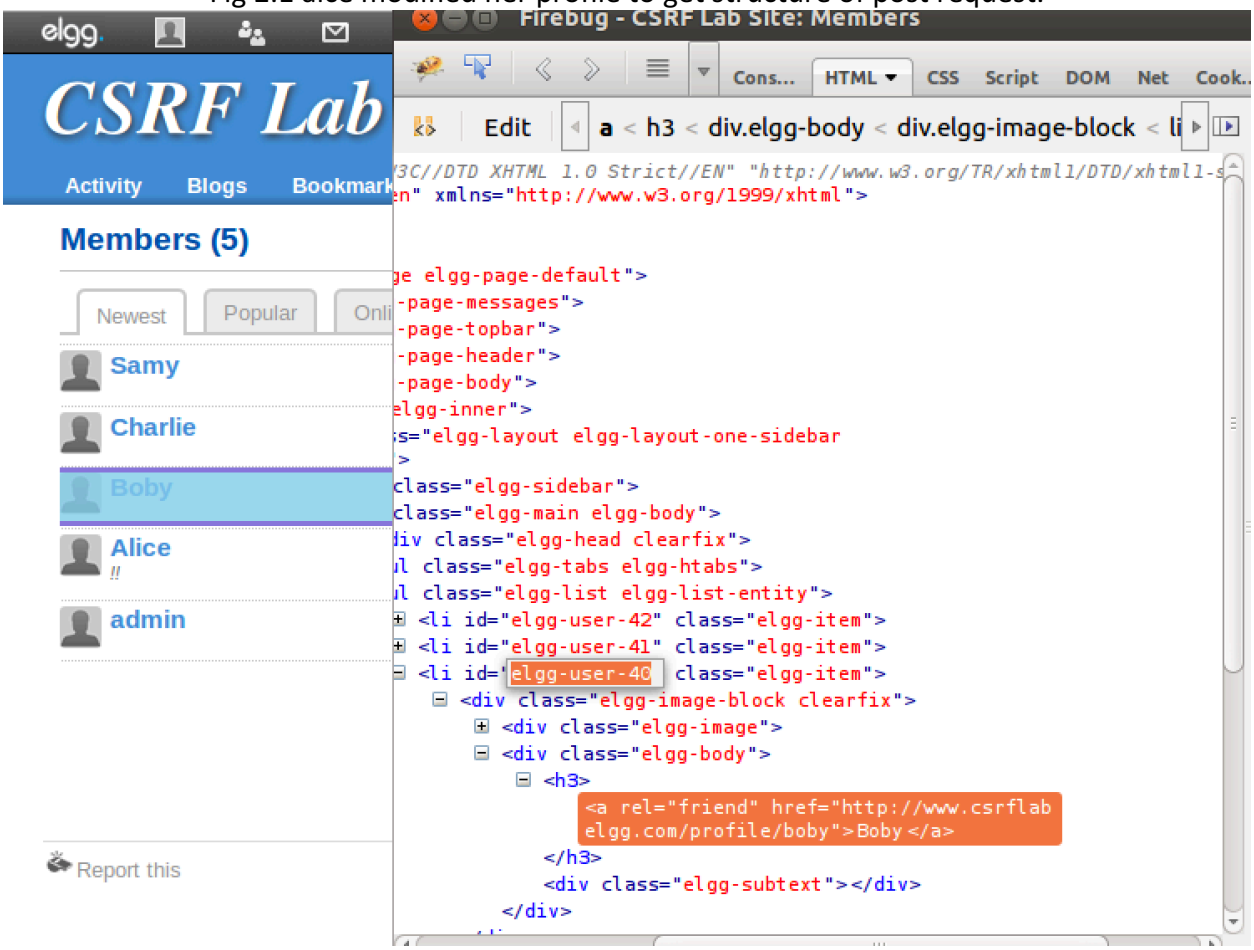


Fig 2.2 get boby's uid using firebug inspect

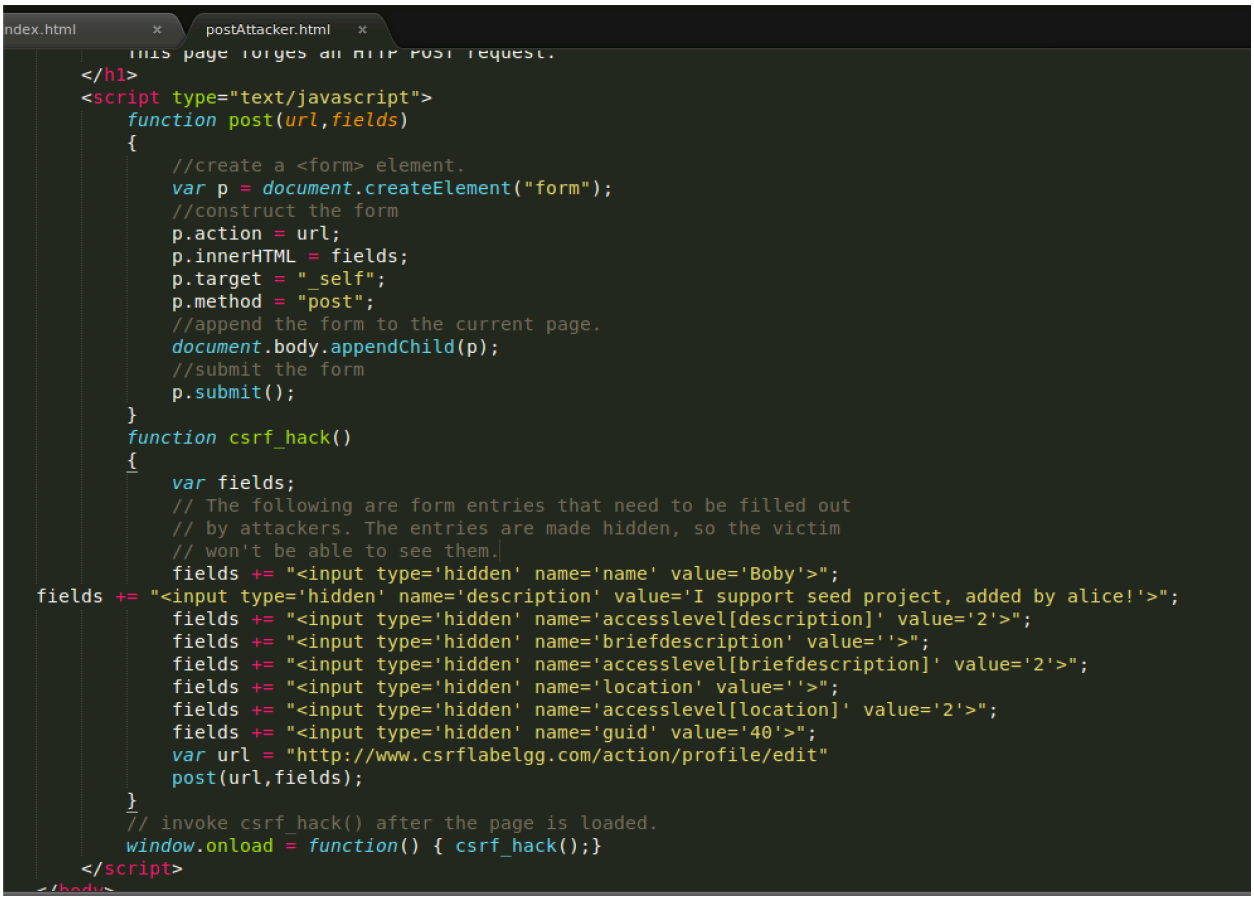


Fig 2.3 attacker code for post form

```
[10/27/2016 10:46] seed@ubuntu:/var/www/CSRF/Attacker$ sudo chmod 755 postAttacker.html
[10/27/2016 10:47] seed@ubuntu:/var/www/CSRF/Attacker$ ls -l
total 12
-rw-r--r-- 1 root root 1078 Oct 27 10:18 format.txt
-rwxr-xr-x 1 root root  263 Oct 27 09:08 index.html
-rwxr-xr-x 1 root root 1523 Oct 27 10:43 postAttacker.html
[10/27/2016 10:47] seed@ubuntu:/var/www/CSRF/Attacker$
```

Fig 2.4 make postAttacker.html to be executed.

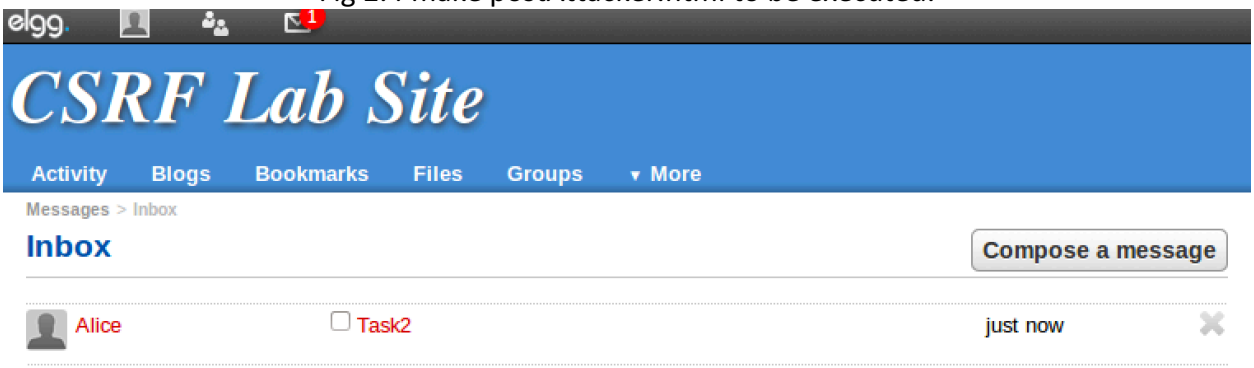


Fig 2.5 boby received a message from alice

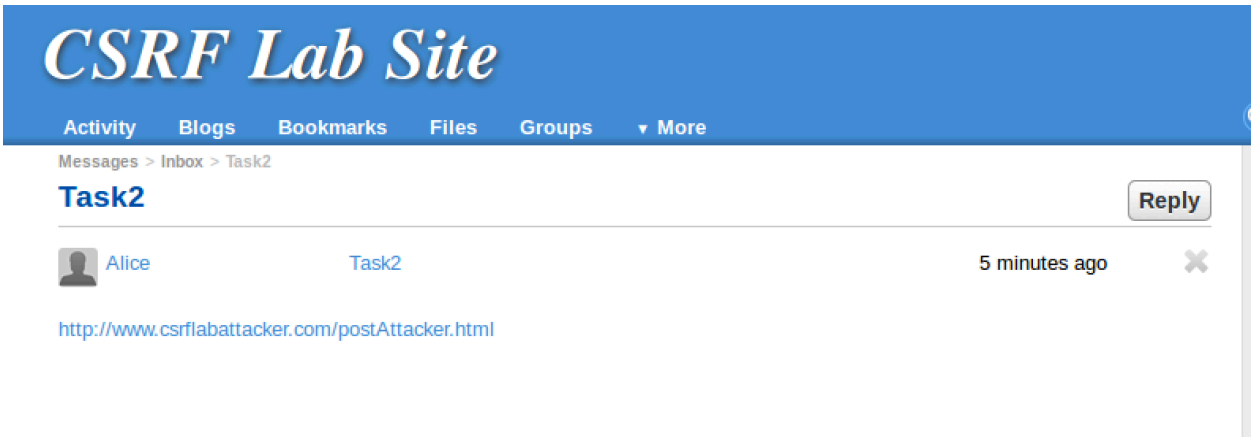


Fig 2.6 link contained a link

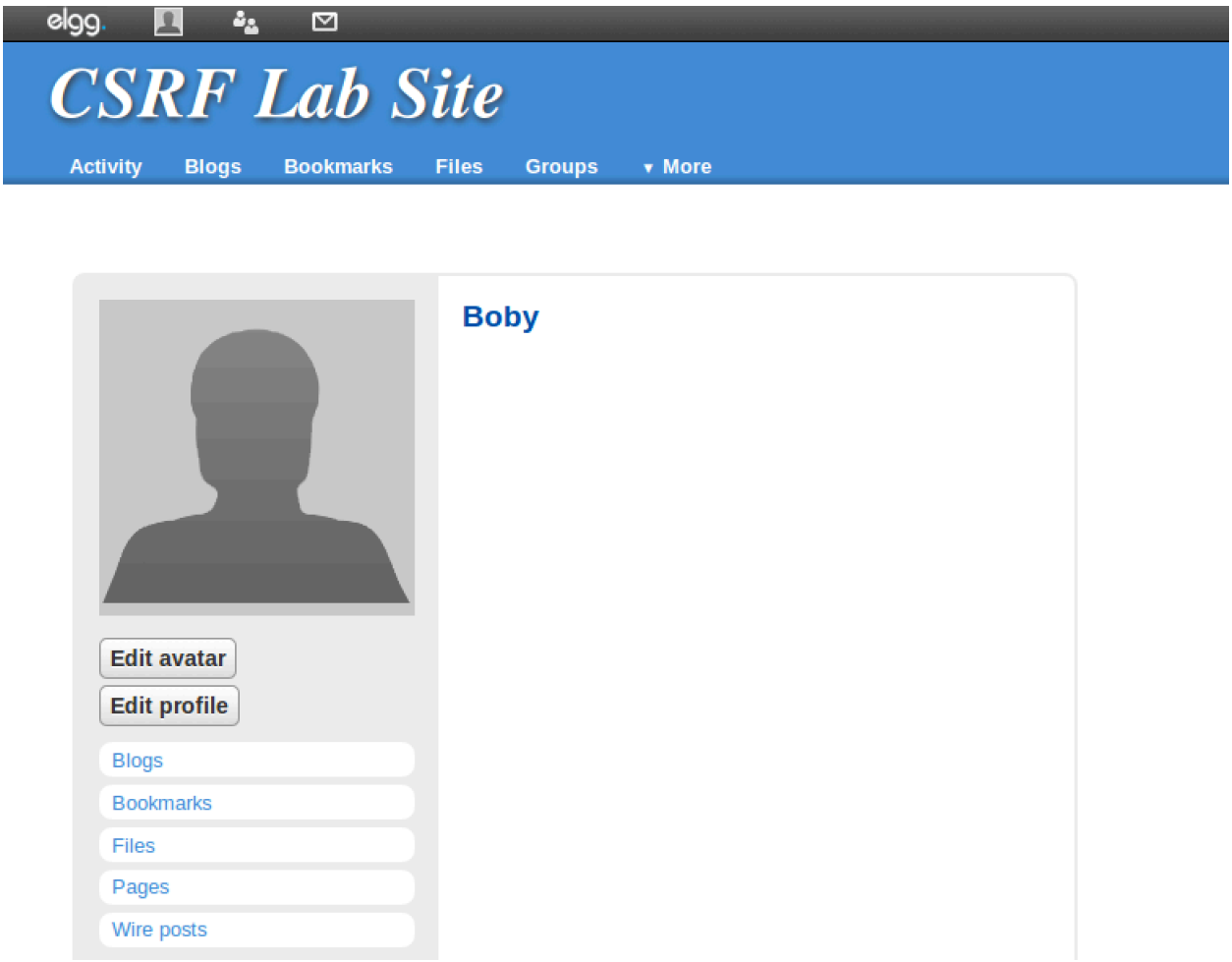


Fig 2.7 boby has no description in his profile

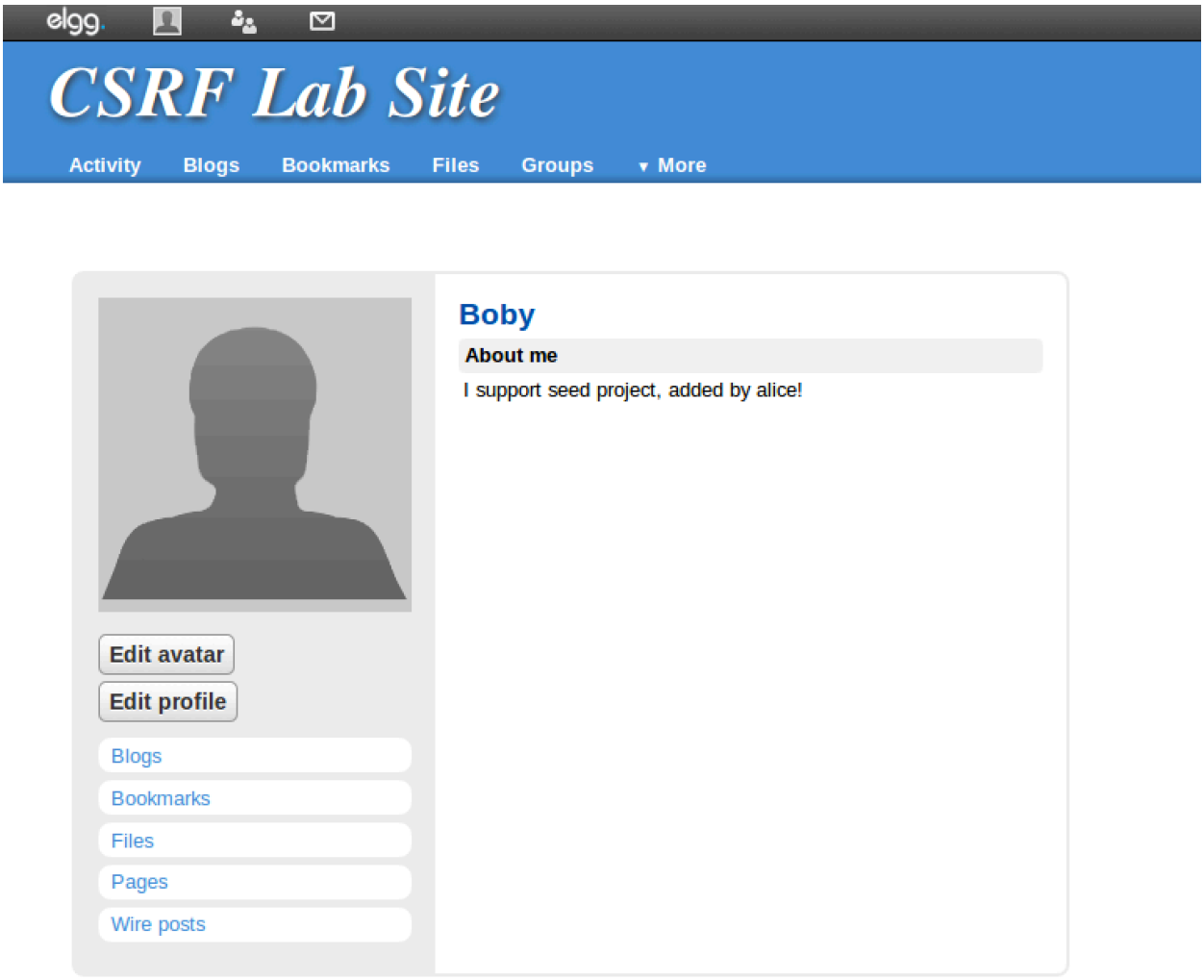


Fig 2.8 after boby clicked the link, boby had a description in his profile.

Observation:

1. Fig 2.1 alice made a real post request, and she could get structure of this request. This is used to forge an attack code.
2. Fig 2.2 use firebug to insepct boby's uid.
3. Fig 2.3 the code to send post request.
4. Fig 2.4 make postAttacker.html to be executable.
5. Fig 2.5, and fig 2.6, Alice got a message. This message includes a link for attacking sent by Boby.
6. Fig 2.7, Boby has no description in his profile.
7. After clicked the link, Boby's profile was changed.

Explanation:

1. From fig 2.1, I know that guid needed to be changed to boby's guid. And some descriptions should be written for this attack with the same structure.
2. Guid is identified for boby to target attak user id.

3. The code shows that some input fields should be organized with name and value pair. When window is loaded, the forgery post request would be sent.
4. Html file should be executable in apache server.
5. Alice sent a link to boby. This link is used to get a cross-site forgery post request.
6. As fig 2.7 and 2.8, after the post request is sent, the boby's profile was changed. Because when boby logged in to elgg application, the cookie was set by server and stored at browser, including session id. The session id and post request were sent to server, then server would recognize this request was sent by boby. So that the profile of boby was changed.

Questions:

1. Like Fig 2.2, Alice logged in to the elgg application and then use firebug to inspect the html elements. Especially in members list of this application.
2. The guid is necessary for this attack. The attack code including post and get request must be specified guid number. If attacker does not know the number, he could guess a range of the guid number and then write a loop for guid and then start the attack loop, then I think attacker's attack must be successful in one of them.

Task 3:

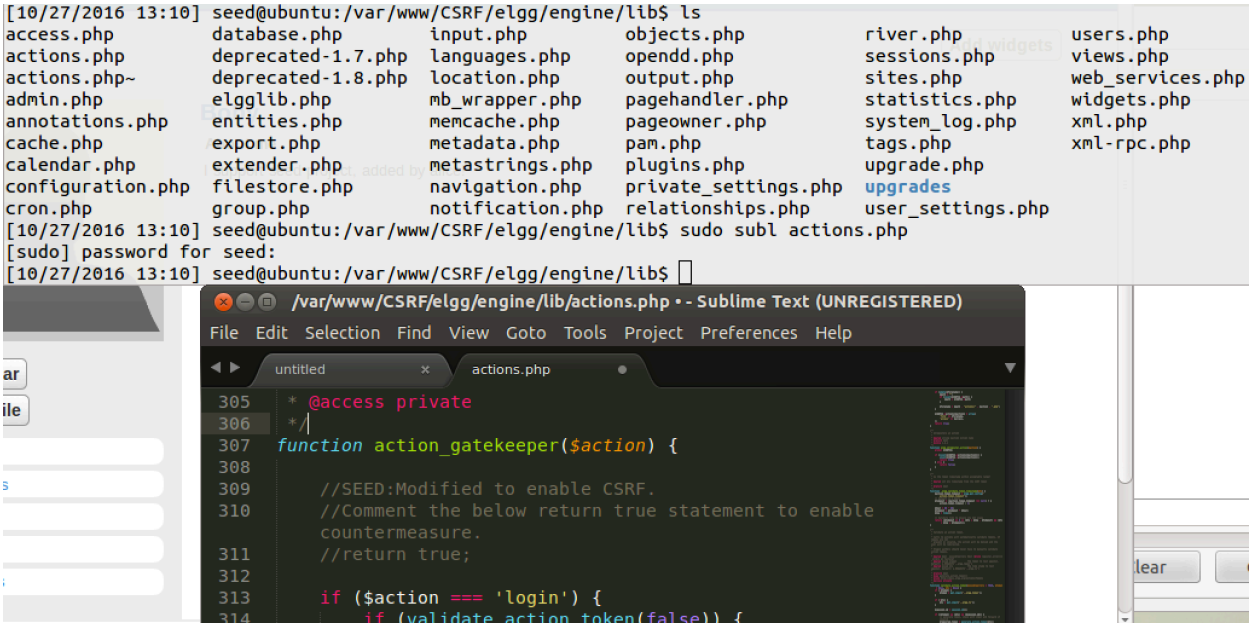


Fig 3.1 comment out the return true in action_gaterkeeper function

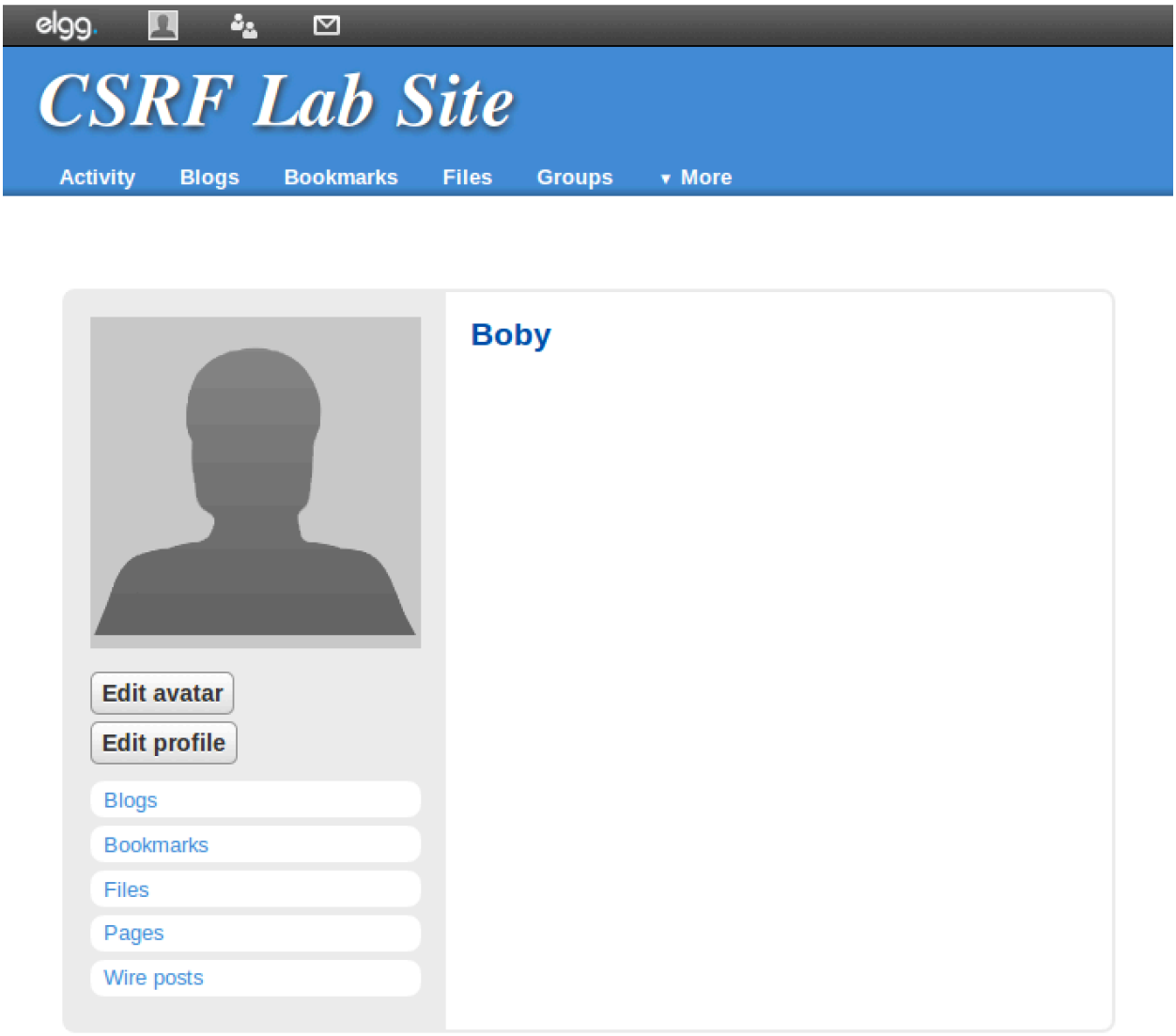
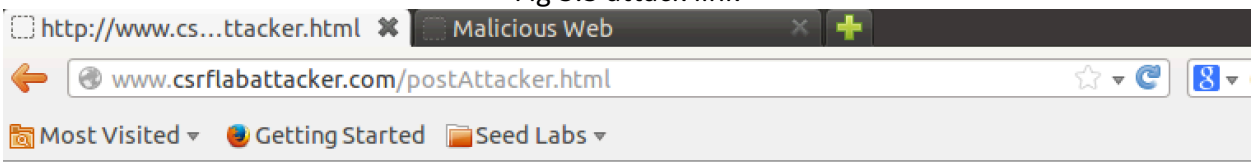


Fig 3.2 boby had no decription

Fig 3.3 attack link



This page forges an HTTP POST request.

this is a fine web

undefined

Fig 3.4 get into attack link

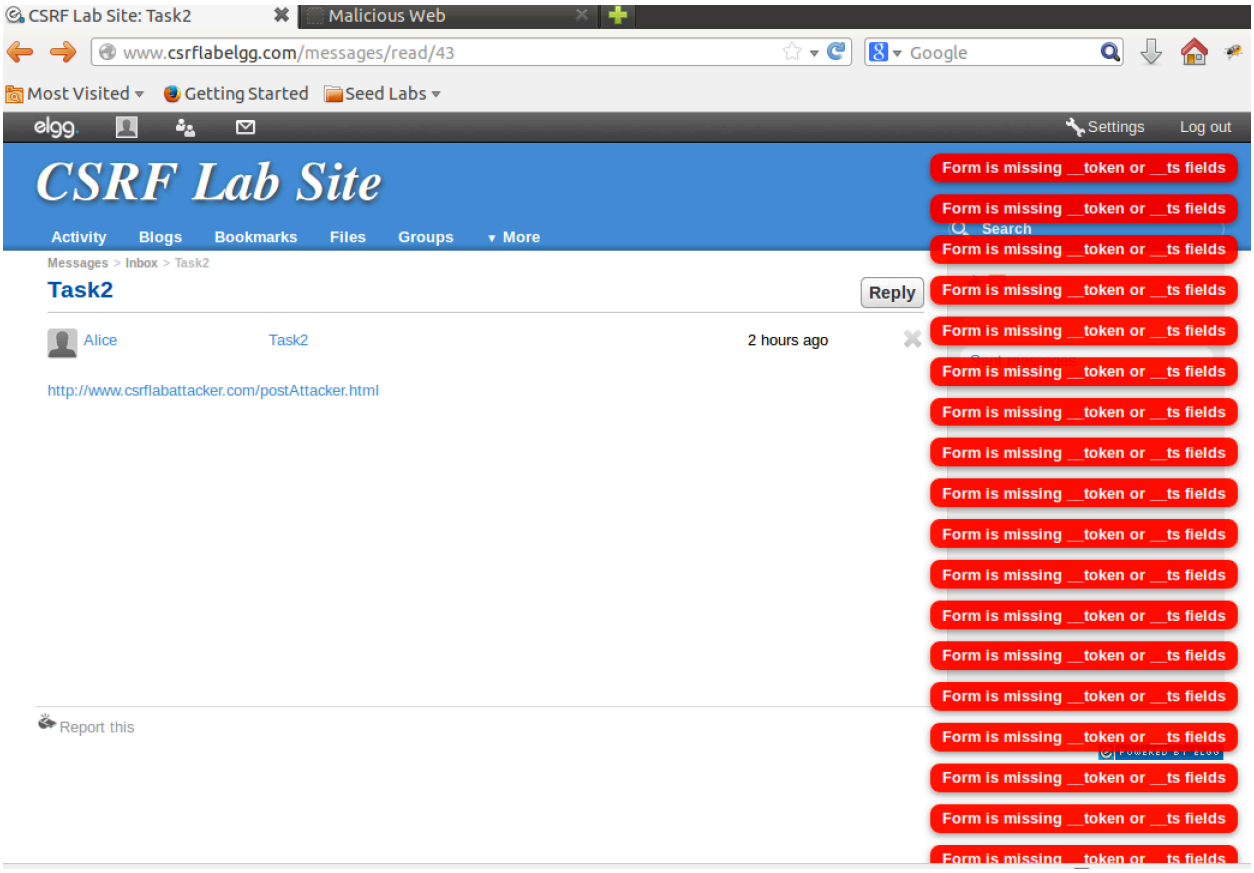


Fig 3.5 error messages printed out

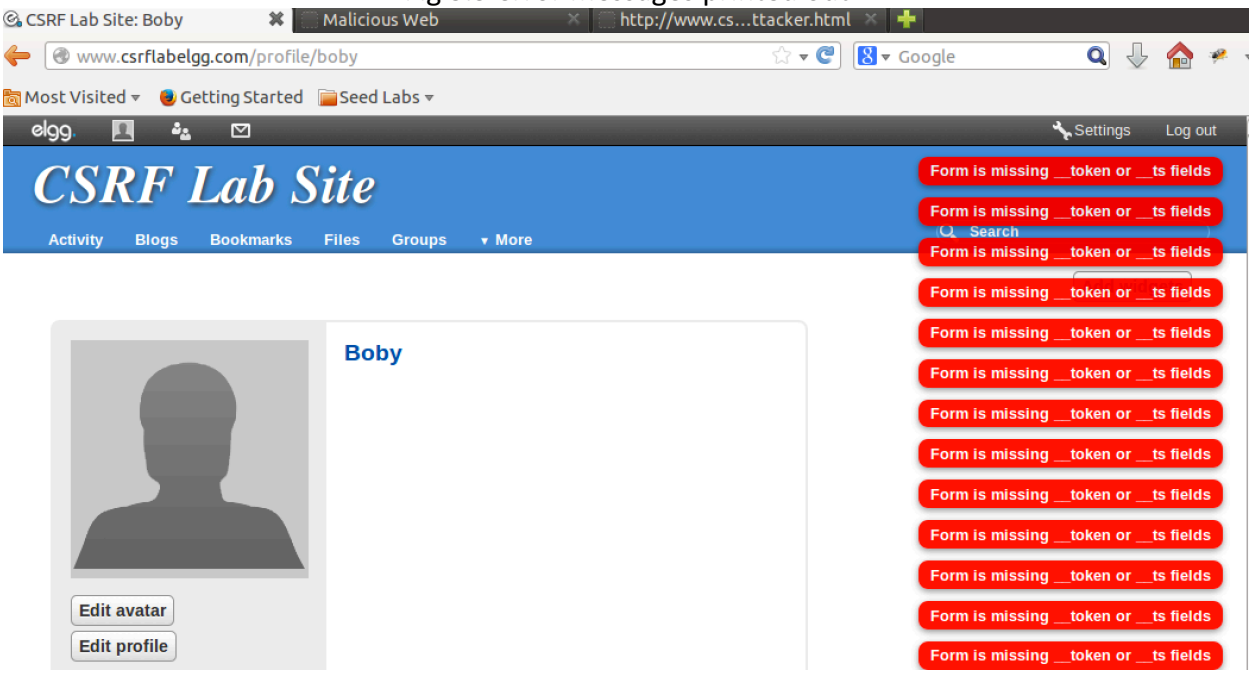


Fig 3.6 boby's profile was not changed

Fig 3.7 capture body's post request

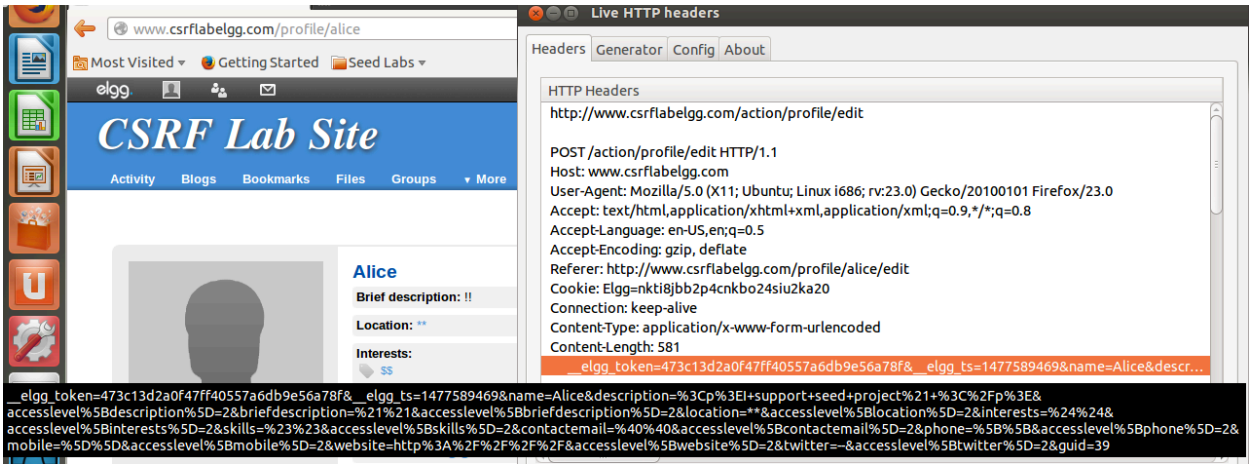


Fig 3.8 capture alice post request tokens

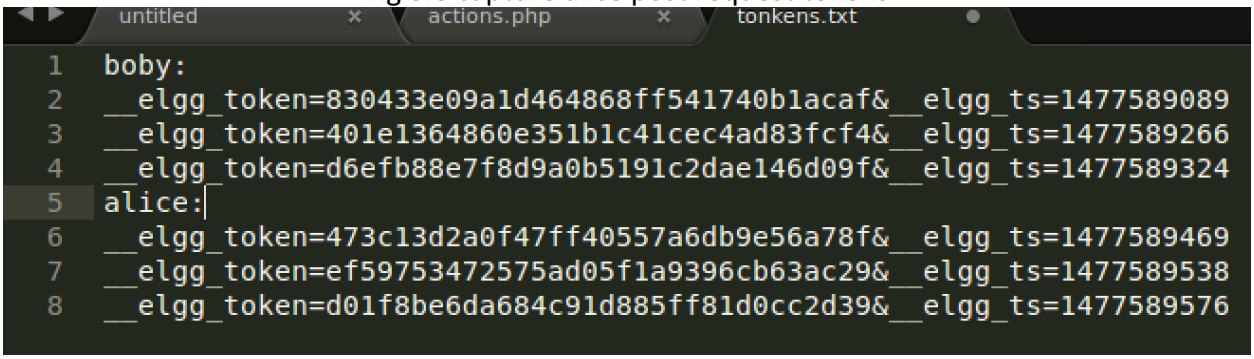


Fig 3.9 summary boby and alice tokens which were captured by live http headers

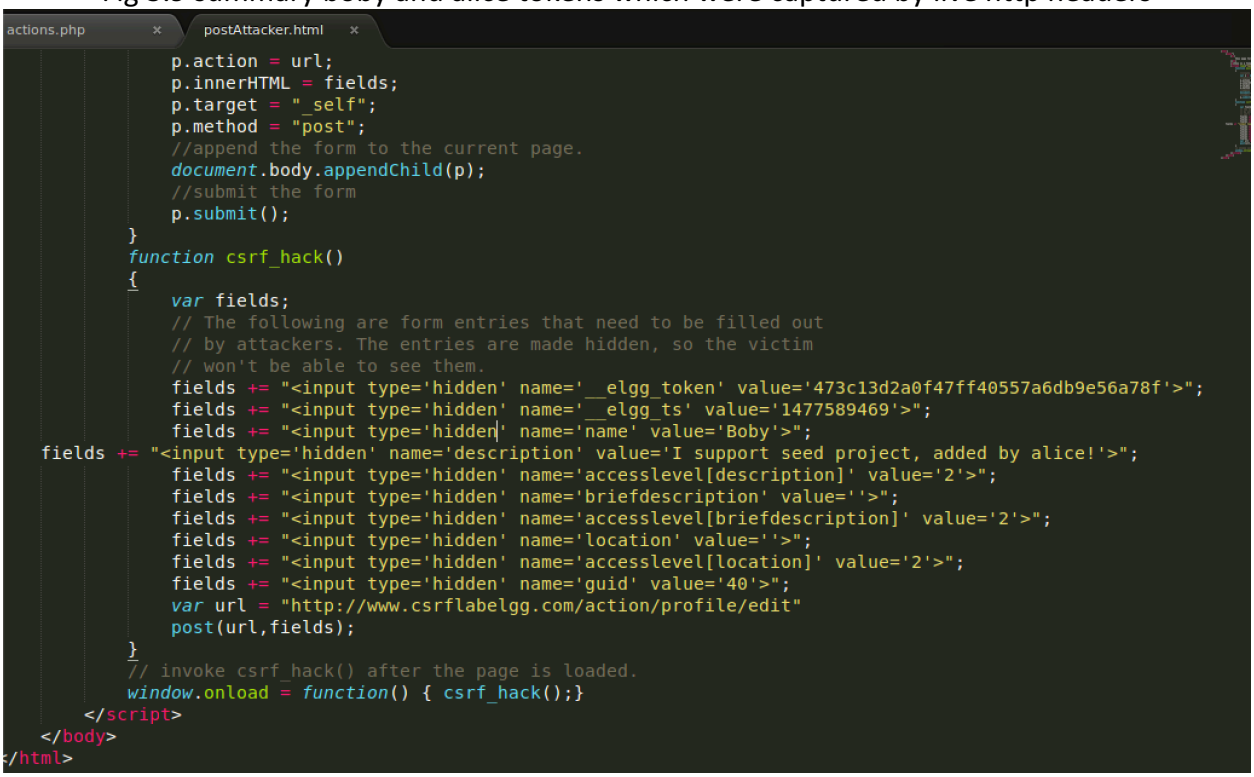
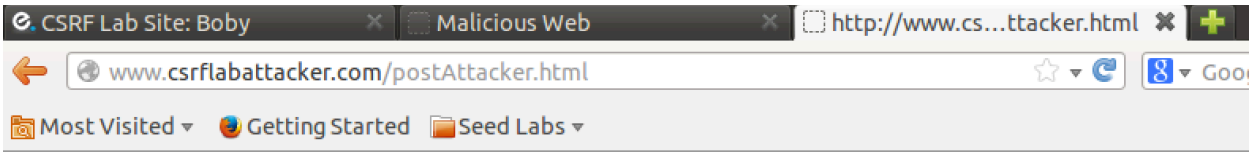


Fig 3.10 add __elgg_token and __elgg_ts tokens for attack link code.



This page forges an HTTP POST request.

this is a fine web

undefined

Fig 3.11 boby visited attacker link filled alice's encoded tokens

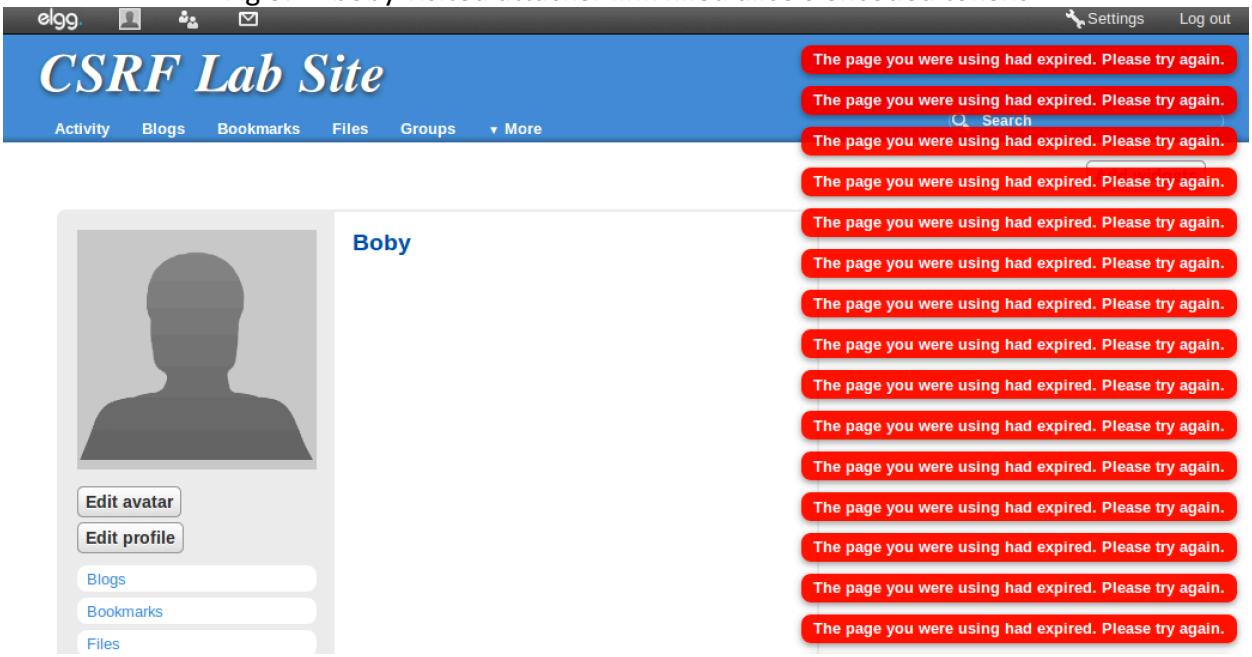


Fig 3.12 error messages 'The page you were using had expired. Please try again'

Observation:

1. Fig 3.1 comment out the return true in action_gaterkeeper function to turn on the countermeasure against CSRF.
2. Fig 3.2 boby had no description in his profile
3. Fig 3.3, with the attack link, boby entered the malicious web site.
4. Fig 3.5 and 3.6 shows that boby's profile was not changed with this above request. And the error message is 'Format is missing _token or _ts fields'.
5. Fig 3.7 I captured three different post request. And 3.8 was alice's post request tokens. Fig 3.9 was summary of Boby and Alice __elgg_token and __*elgg*_ts value.
6. I wrote the __elgg_token and __elgg_ts with Alice's specific value in attack code.
7. Fig 3.11 and 3.12 shows that when bobby opened attack link, his profile was not changed and error message were 'The page you were using had expire'.

Explanation:

1. Fig 3.1, I commented out the return true, so that the function would not always get a true result and other check need to be executed.
2. Fig 3.2 was used to cooperation.
3. The link used in Fig 3.3 that was as same as version in the task 2.
4. Fig 3.5 and 3,6 showed that the __elgg_token and __elgg_ts was necessary in post request after the 'return true' was commented out.
5. Fig 3.7, 3.8, and 3.9 were used to summary __elgg_token and __*elgg*_ts value.
6. Fig 3.10 alice add the value __elgg token and ts value in attack code.
7. The Fig 3.12 shows that the __elgg token and ts had expired. Alice could not use the values. Attacker could not successfully generate the right token, as it doesn't have all the details to generate this token and get it validated by the website being attacked. The attacker is unable to place the correct tokens in his request and so this attack was failed. And the secret tokens are generated as a result of php code, source of which is unavailable for the attacker to grab, this acts as added security to the website. And the tokens were easy to be expired.