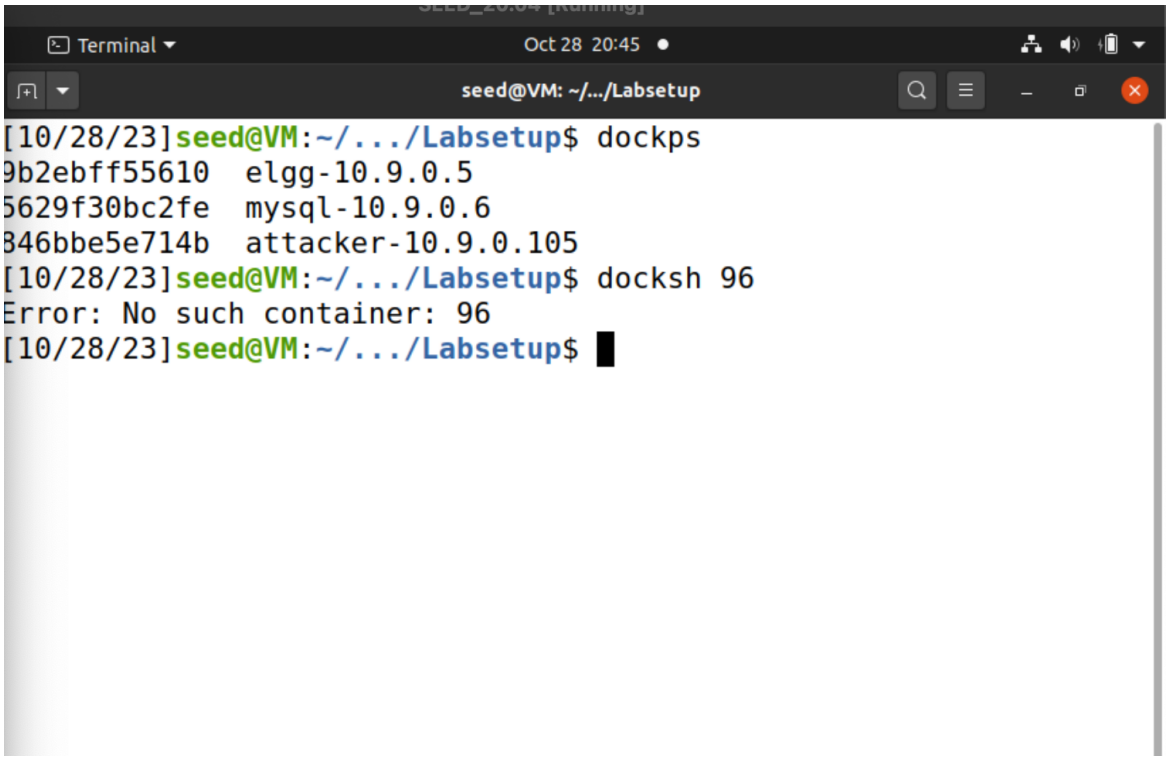
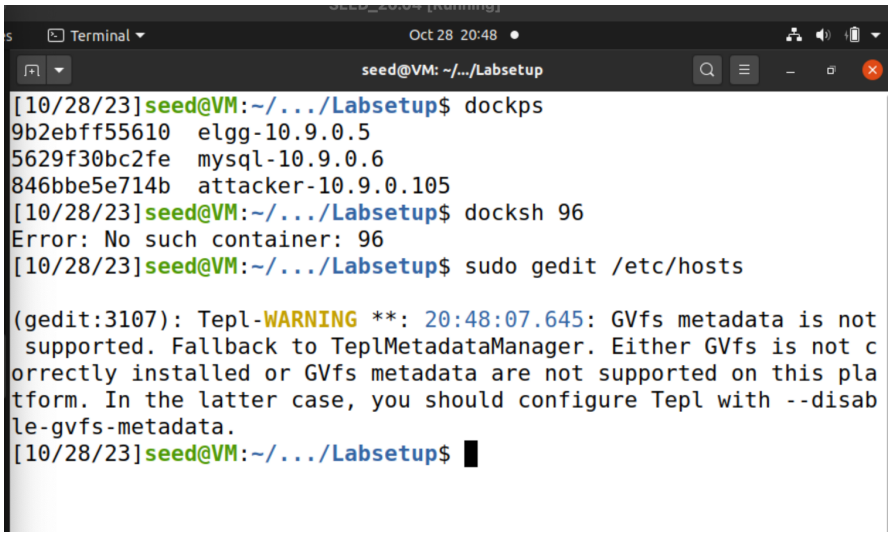
ICSI 424 lab7
Seoyeon Choi

1. **Setup for cross site request forgery attack lab and 2 set-ups**

I typed “dcbuild” , ‘“dcup” , and “docks”



2. Open /etc/host and store
3. 10.9.0.5
4. 10.9.0.5
5. 10.9.0.105
6.
7. www.seed-server.com # Elgg For SEED Labs
8. www.example32.com
9. www.attacker32.com # attackers website
10. Int /etc/hosts

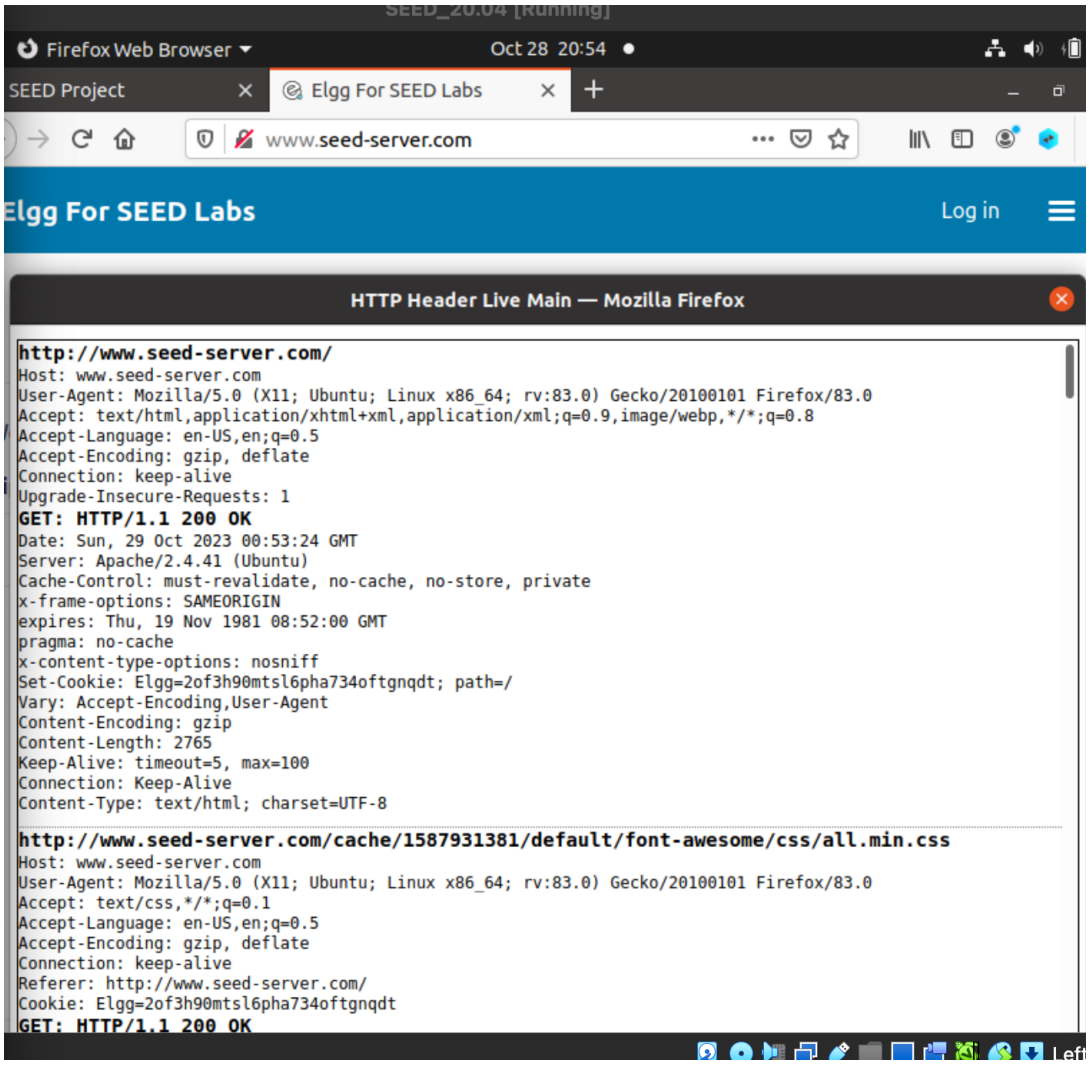


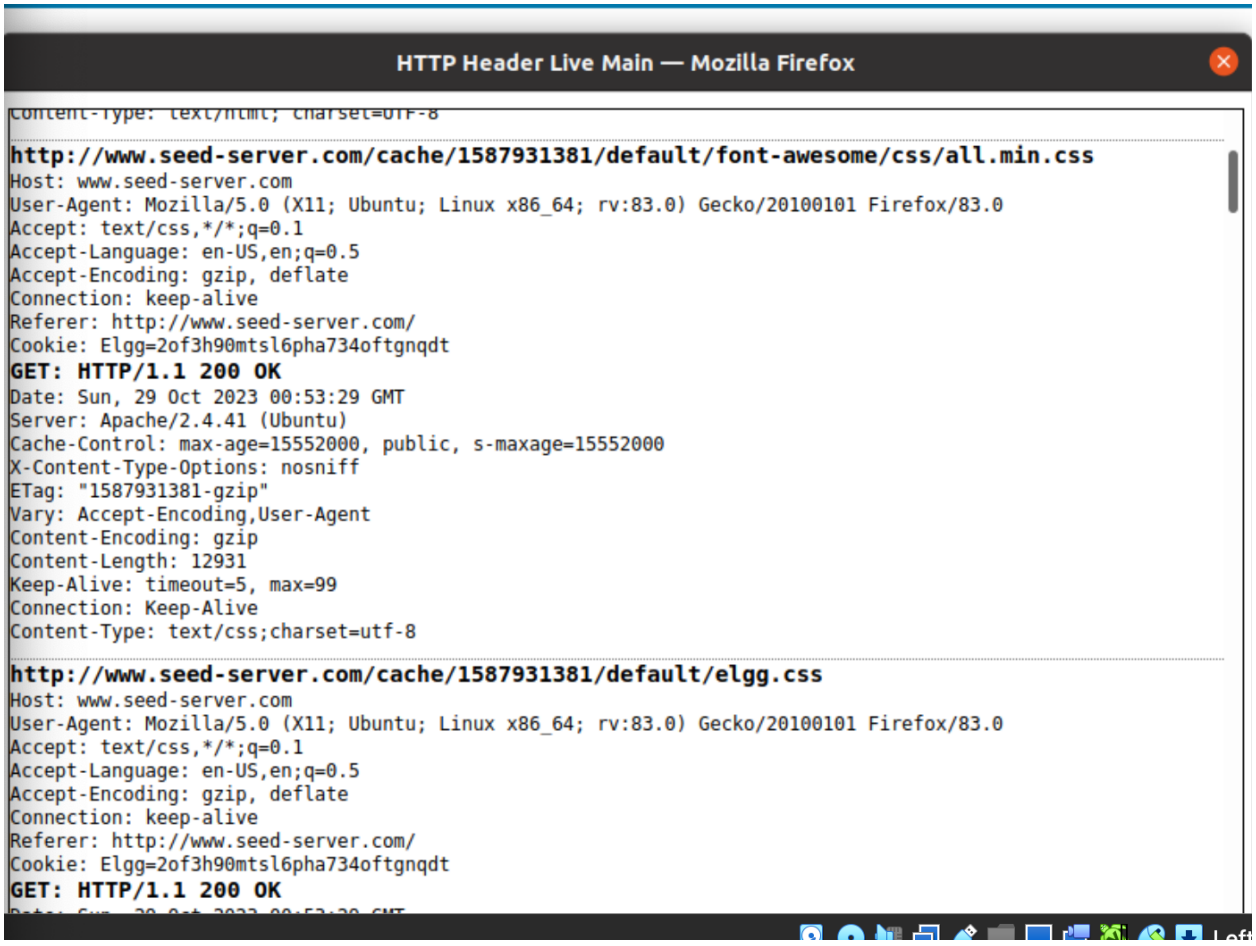
// used sudo gedit to open /etc/hosts

3 Lab Tasks: Attacks

3.1 Task 1: Observing HTTP Requests.

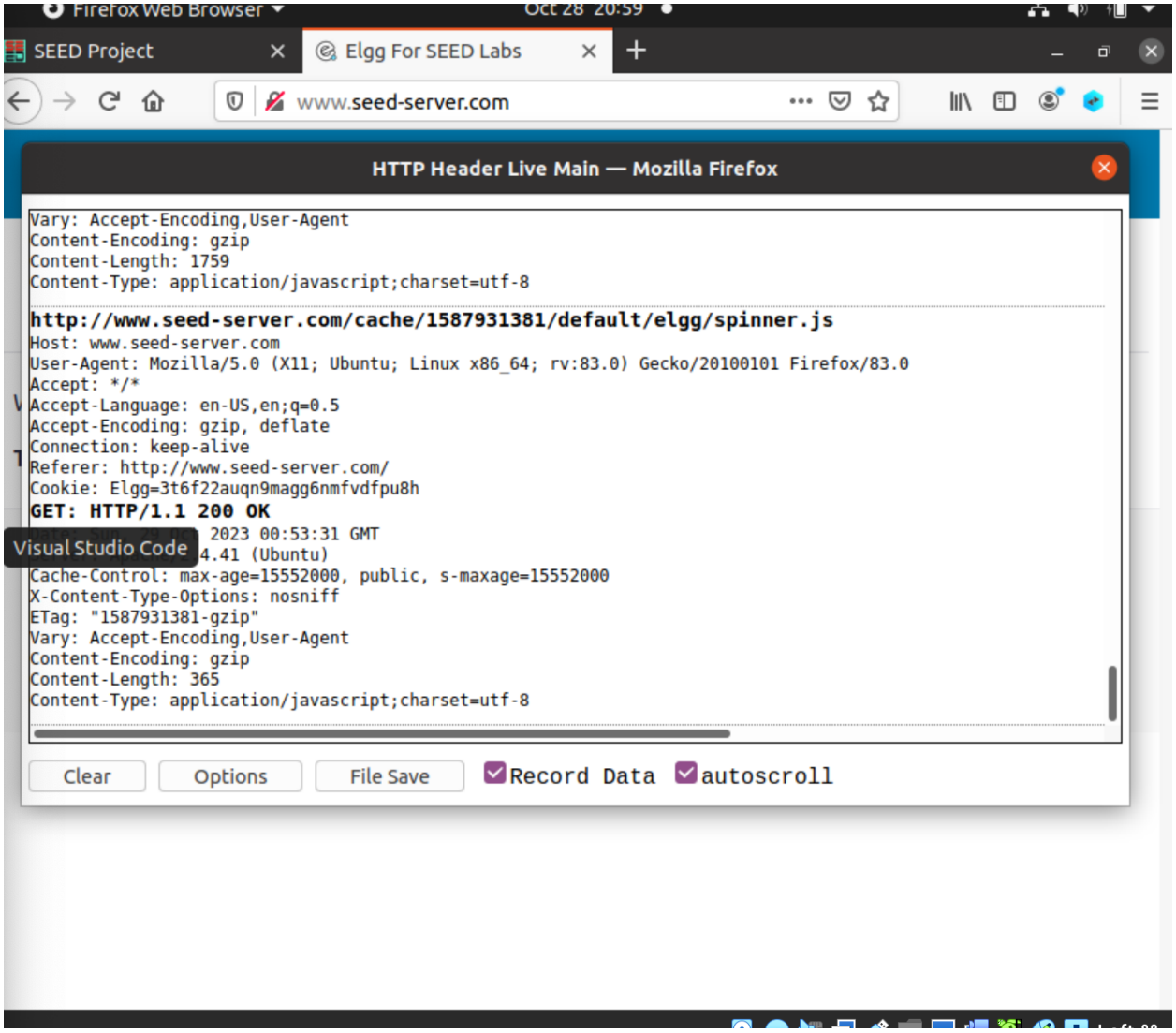
Please use this tool to capture an HTTP GET request and an HTTP POST request in Elgg. In your **report, please identify the parameters used in these requests**, if any.


SEED_20.04 [Running]
Firefox Web Browser
Oct 28 20:54
SEED Project
Elgg For SEED Labs
www.seed-server.com
Elgg For SEED Labs
Log in
HTTP Header Live Main — Mozilla Firefox
http://www.seed-server.com/
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
GET: HTTP/1.1 200 OK
Date: Sun, 29 Oct 2023 00:53:24 GMT
Server: Apache/2.4.41 (Ubuntu)
Cache-Control: must-revalidate, no-cache, no-store, private
x-frame-options: SAMEORIGIN
expires: Thu, 19 Nov 1981 08:52:00 GMT
pragma: no-cache
x-content-type-options: nosniff
Set-Cookie: Elgg=2of3h90mtsl6pha734oftgnqdt; path=/
Vary: Accept-Encoding,User-Agent
Content-Encoding: gzip
Content-Length: 2765
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
http://www.seed-server.com/cache/1587931381/default/font-awesome/css/all.min.css
Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/css,*/*;q=0.1
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://www.seed-server.com/
Cookie: Elgg=2of3h90mtsl6pha734oftgnqdt
GET: HTTP/1.1 200 OK

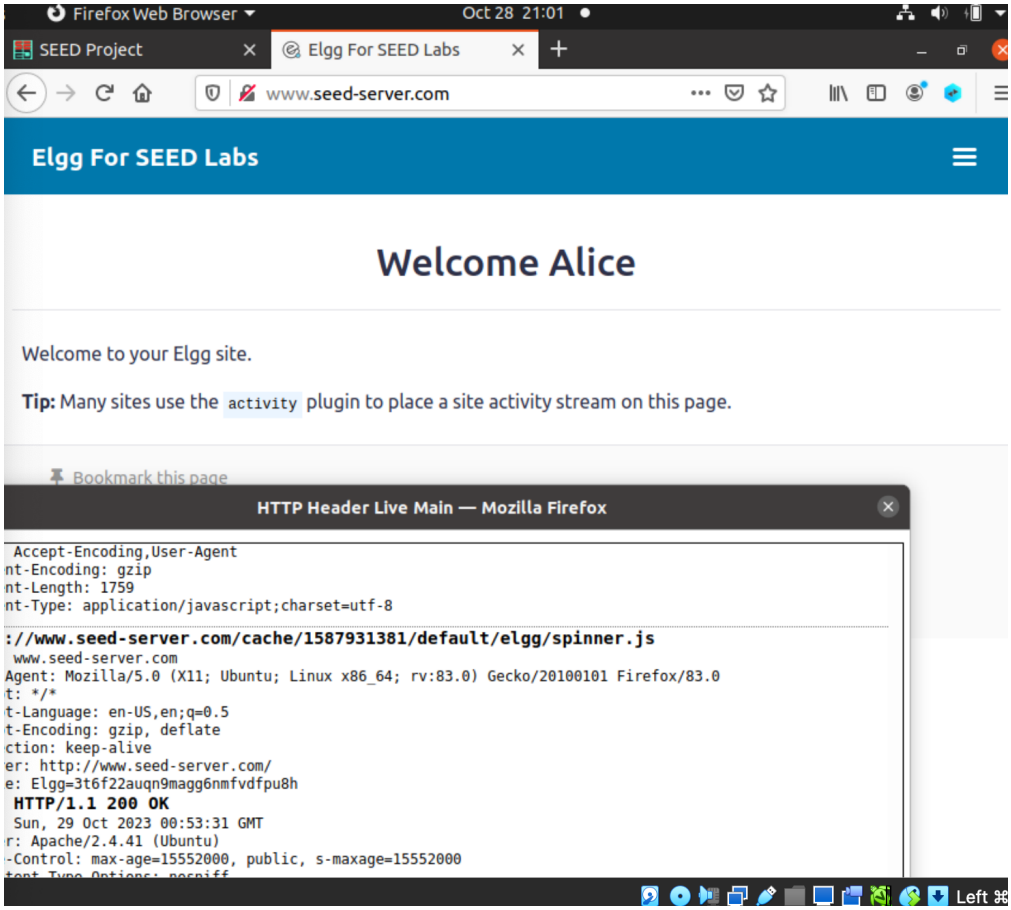


I opened www.seed-server.com website link on my machine's firefox I used HTTP header live to see how get, post were used to open above website.

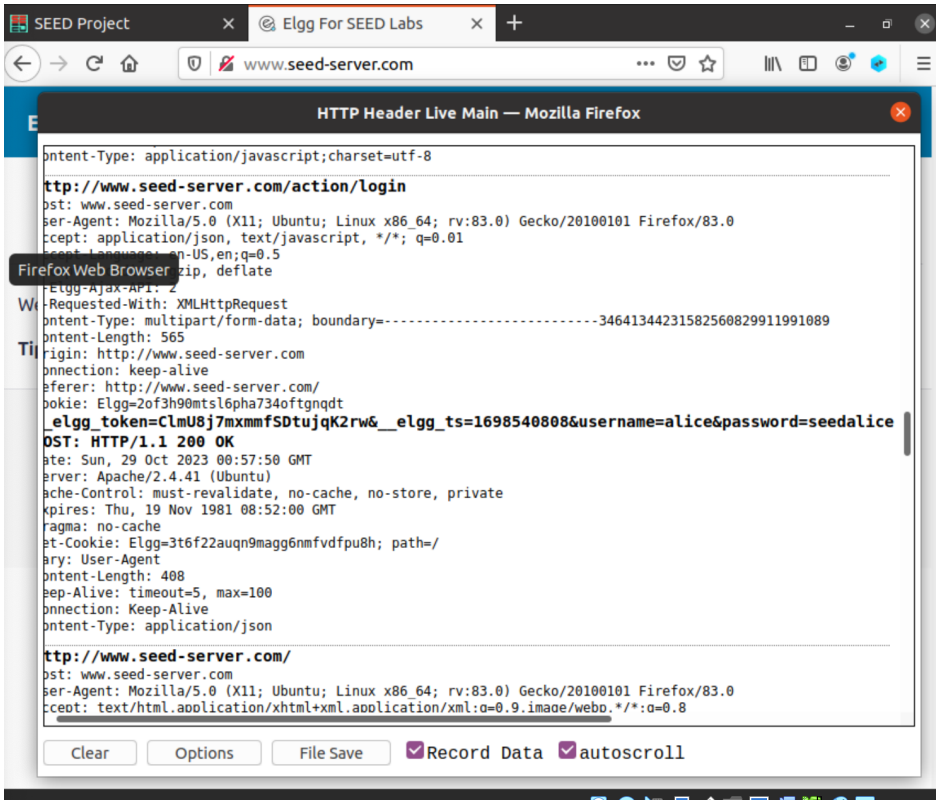
I used the given name alice and password in instruction, to see how ww.seed-server.com changes when a user login. “ alice | seedalice “



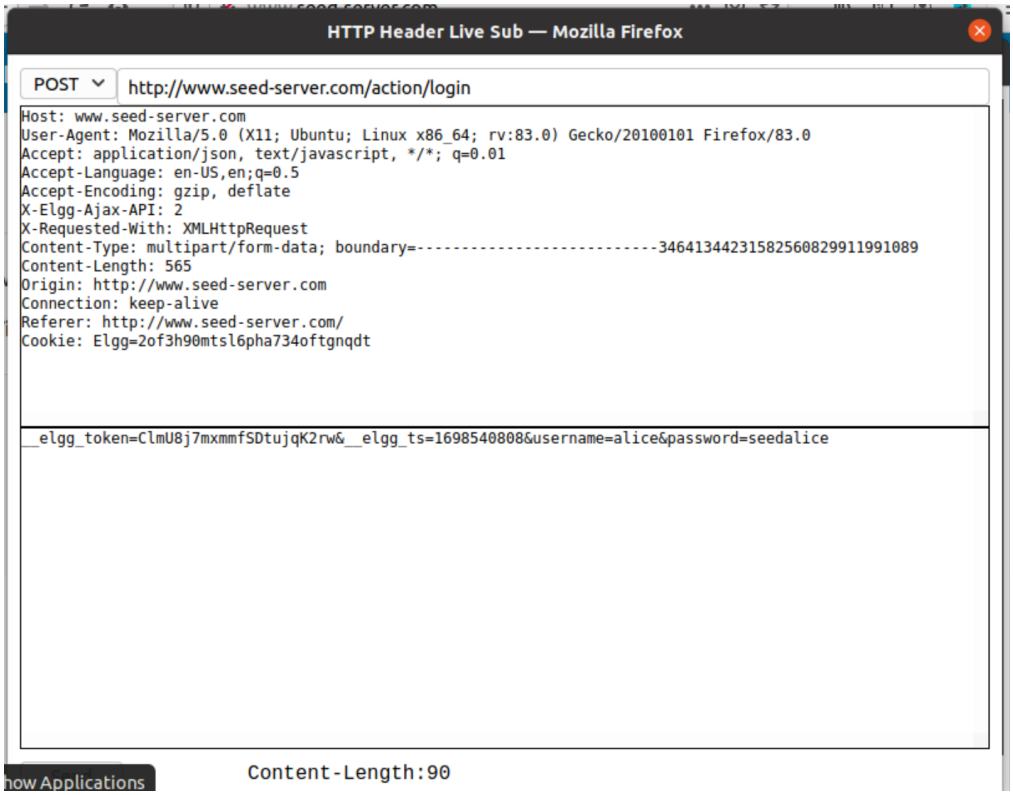
// there were many things going on but I was able to see that http sent some ling and Get : HTTP/1.1 200 OK we're response. At the end I was able to login as ALICE and GET HTTP/1.1 200 OK was the last interaction in HTTP header live main - Mozila Firefox.



// Logged in as Alice and use HTTP Header Live main to catch HTTP requests.



// this shows that e captured when alice logged in with her password seedalice. We got “POST: HTTP/1.1 200 OK”. It was a post request.



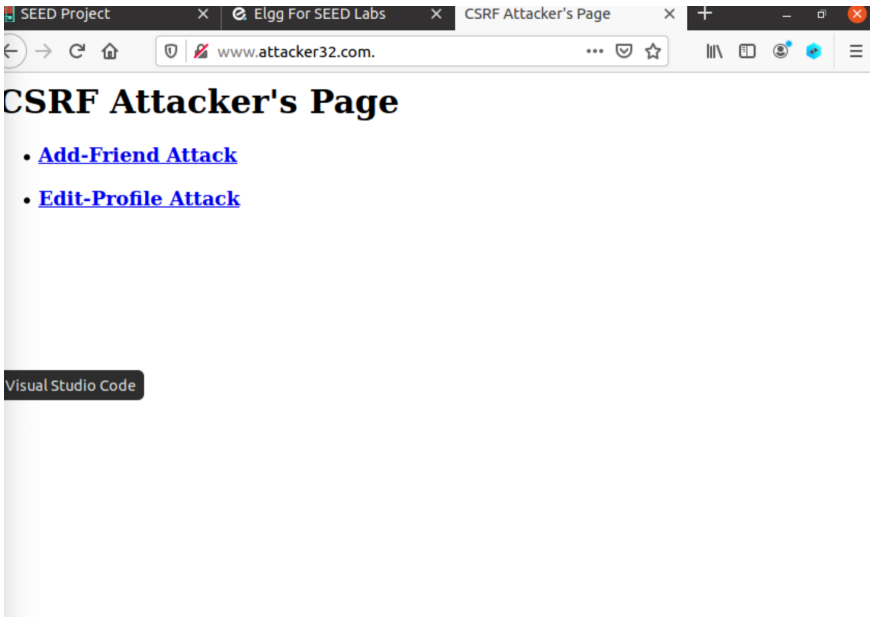
// In here we can see that http://www.seed-server.com/action/login (so login action) was done by POSt request and username=alice&password=seedalice was passed. elgg_token=ClmU8j7mxmmfSDtujqK2rw&__elgg_ts=1698540808&username=alice&password=seedalice. (it captured username and password) Above those two values are countermeasures for CSRF attack.

-> For login, HTTP used Post request and passed username with password as parameter.

3.2 Task 2: CSRF Attack using GET Request

Your job is to make the attack successful as soon as Alice visits the web page, without even making any click on the page (h**int: you can use the img tag,** which automatically triggers an HTTP GET request).

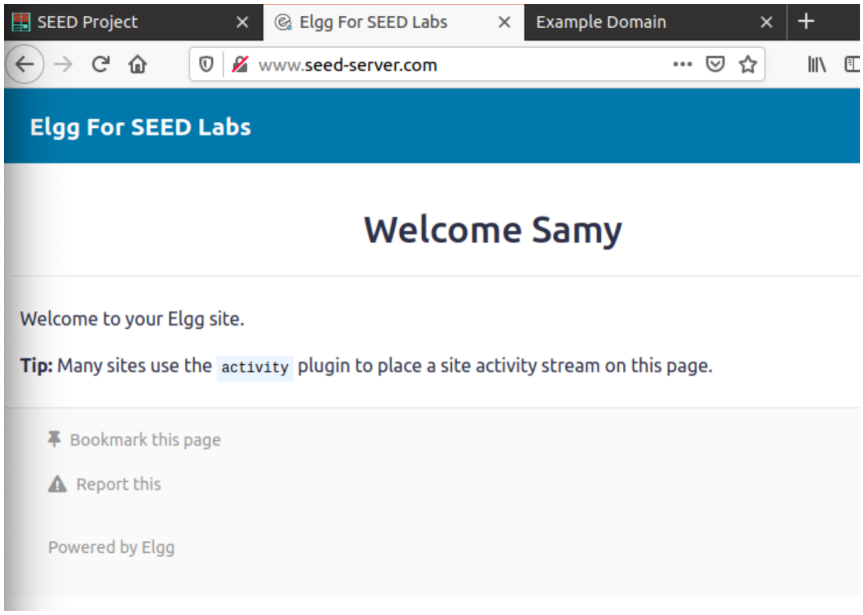
Elgg has implemented a countermeasure to defend against CSRF attacks. In Add-Friend HTTP requests, you may notice that each re**quest includes two weird-looking parameters, elgg ts and elgg token. These parameters are used by the countermea**sure, so if they do not contain correct values, the request will not be accepted by Elgg. We have disabled the countermeasure for this lab, so there is no need to include these two parameters in the forged requests.



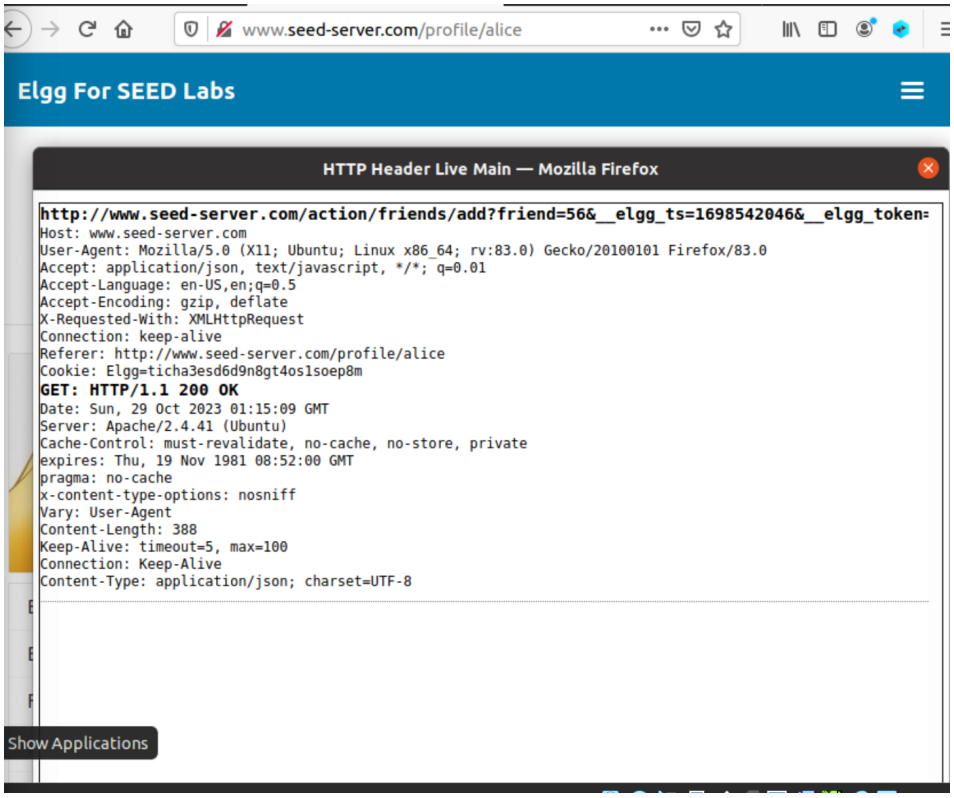
// I checked www.attacker32.com . I have to modify the “Add-Friend-Attack” link so that once Alice clicks that link, Samy will be registered as Alice’s friend.

1. Log out in “Elgg for seed Labs” website as Alice

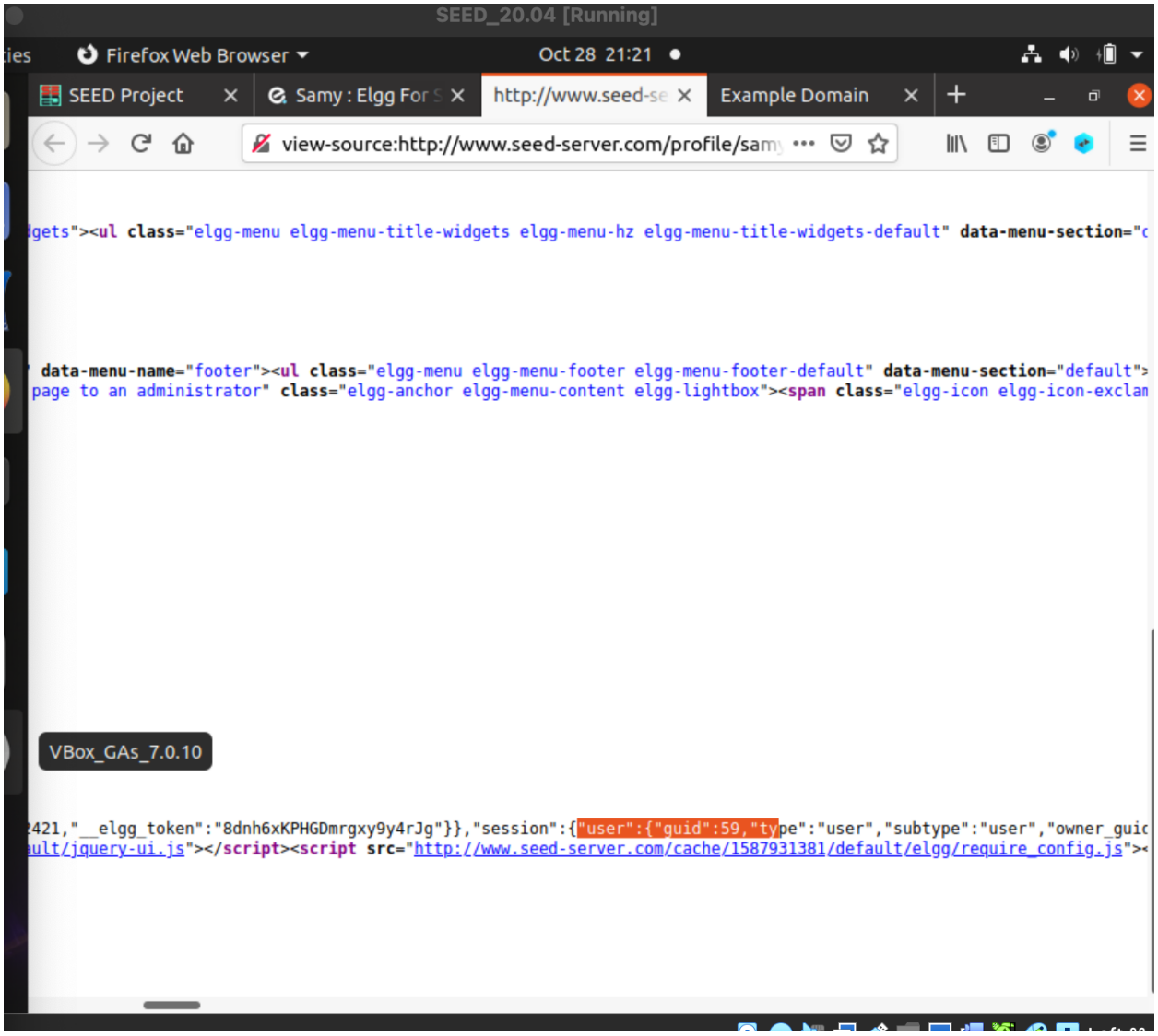
2. Login with user name: “samy” and password : “seedsamy”



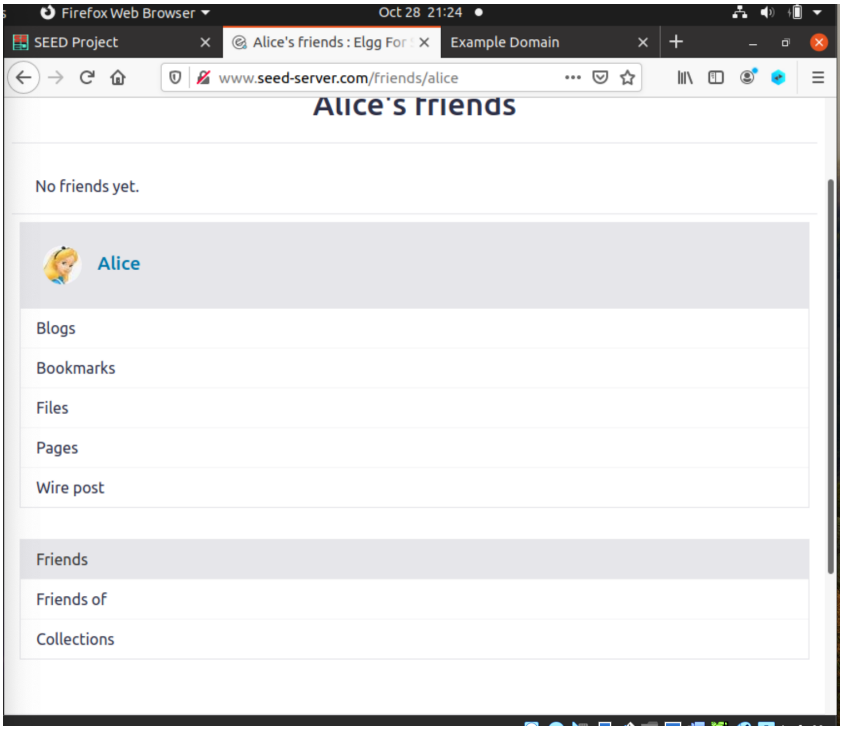
3.
4. Go to member and click ALice and see HTTP header line to capture GET request, so open HTTP Header LIve main and click “add friend” for Alice and see how HTTP interacts.
5.



// I capture the GET request, when Samy requests “add friend” to Alice. It seems Alice’s guid is 56 (it is number on the top friend=56, so 56 is guid, userId) For Task1 attack we have to disable countermeasure parameters such as elgg ts and elgg token Tokens.



// next I will find Samy’s guid. Go to samy’s profile and click page source code.
// we can see samy’s guid was 59.
// I created “textfile.txt” to store useful data for this attack.
// log out as Samy and log in again as Alicee, because we want to launch an attack on Alice, so Alice should be active/logged in.

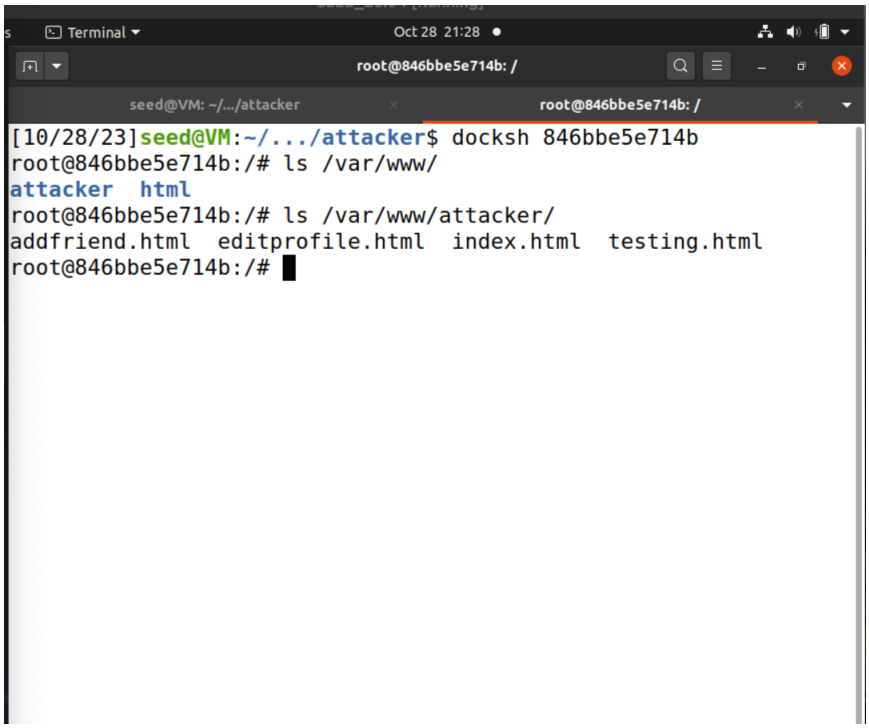


$

// before the launch attack, I checked Alice's friend. So far she doesn't have any friends.

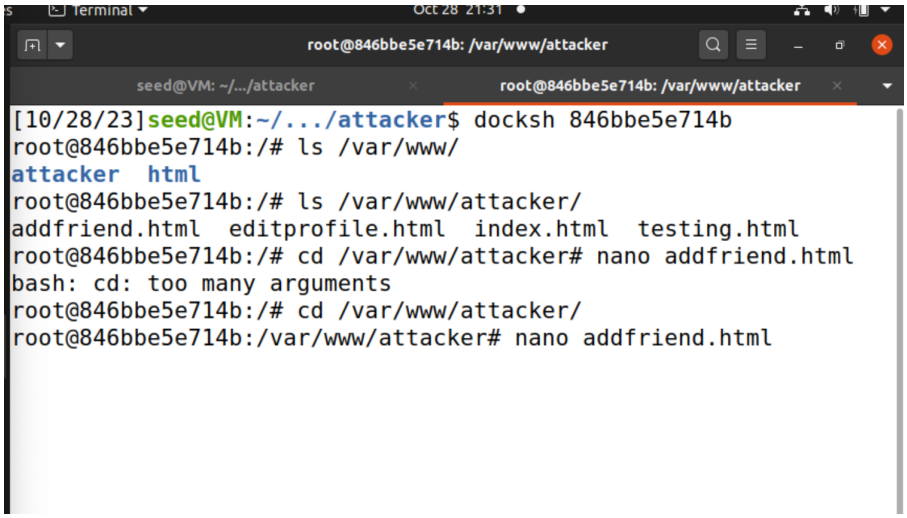
// open terminal create folder named attacker using **cd /attacker**

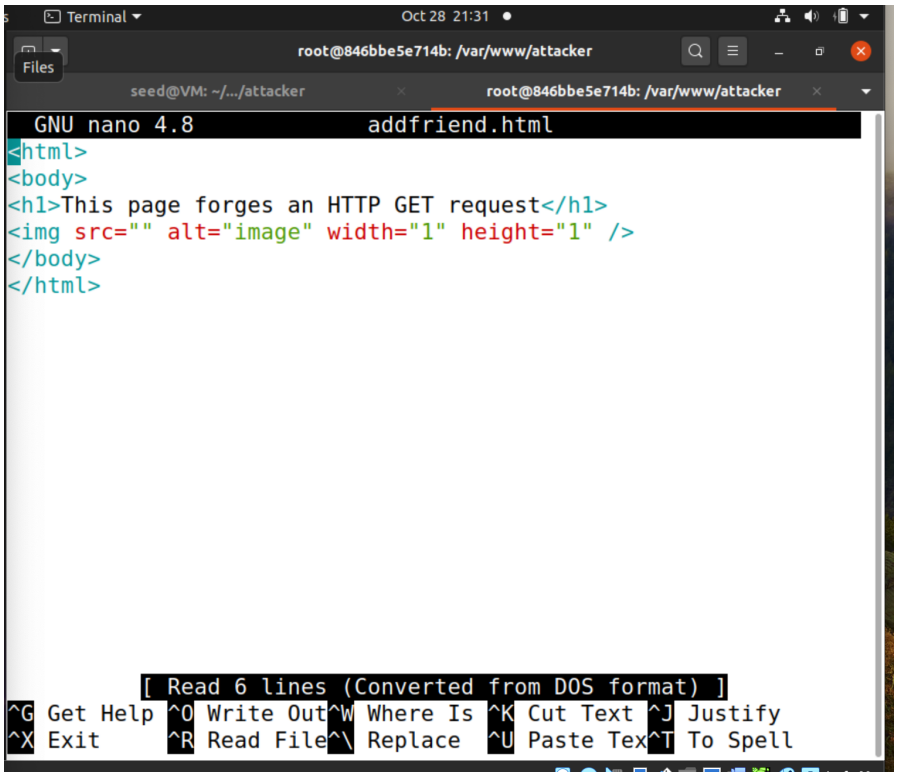
In attacker's folder, type **"docksh 846bbe5e714b**" (<- the numbers are www.attacker32.com website's shell) we want to go inside t**hat website's shell**



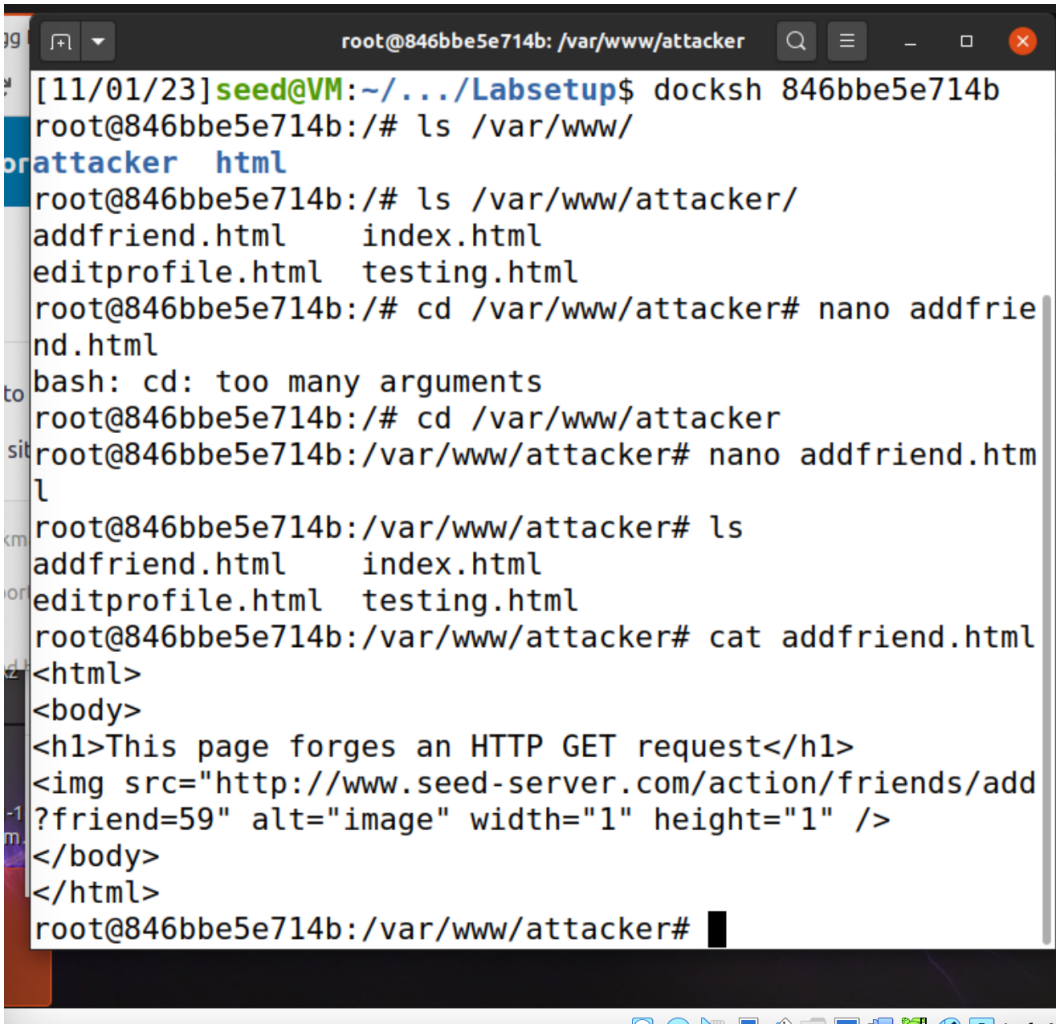
// In here I went into the www.atttacker.com's shell code . I typed **ls /var/www/ a**nd I went inside the **attacker a**nd I can see there are **4 different " .html" .** since task1 is about adding friends.

We need to focus on the **“addfriend.html”** part. I went to attacker folder **using “ cd/var/www/attack# “ and used “nano addfreind.html” to ge**tinto addfreind.html’s code.





// this is how addfriend.html looks like at the beginning.
// add html link and samy’s guid in img src section.

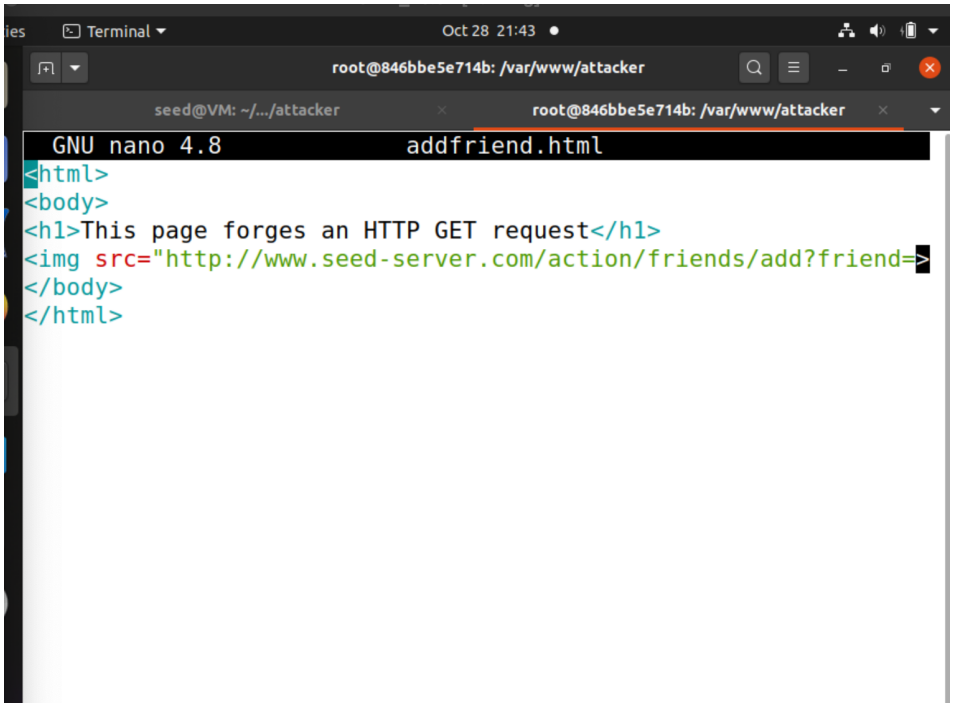


// **Use samy's guid 59 for attack, s**o whenever alice clicks it samy will be automatically added to her friend.

// I just dragged addfriend.html from the attacker folder(this is provided code in the Lab Setup zip file) into the text editor. **And in img src= section, I pasted alice's add friend's guid url inside addfriend.html (guid=5 is Alice's guid) (Lab7 instruction gives hint that I could use img tag to trigger HTTP Get() request)**

```
(gedit:3835): Tepl-WARNING **: 21:19:38.903: GVfs metadata is no
t supported. Fallback to TeplMetadataManager. Either GVfs is not
VBox_GAs_7.0.10 installed or GVfs metadata are not supported on this
platform. In the latter case, you should configure Tepl with --d
isable-gvfs-metadata.
[1]+  Done                    sudo gedit /etc/hosts &> /dev/null
[10/28/23]seed@VM:~/.../Labsetup$ cd attacker/
[10/28/23]seed@VM:~/.../attacker$ docker cp addfriend.html 846bb
e5e714b:/var/www/attacker/
[10/28/23]seed@VM:~/.../attacker$
```

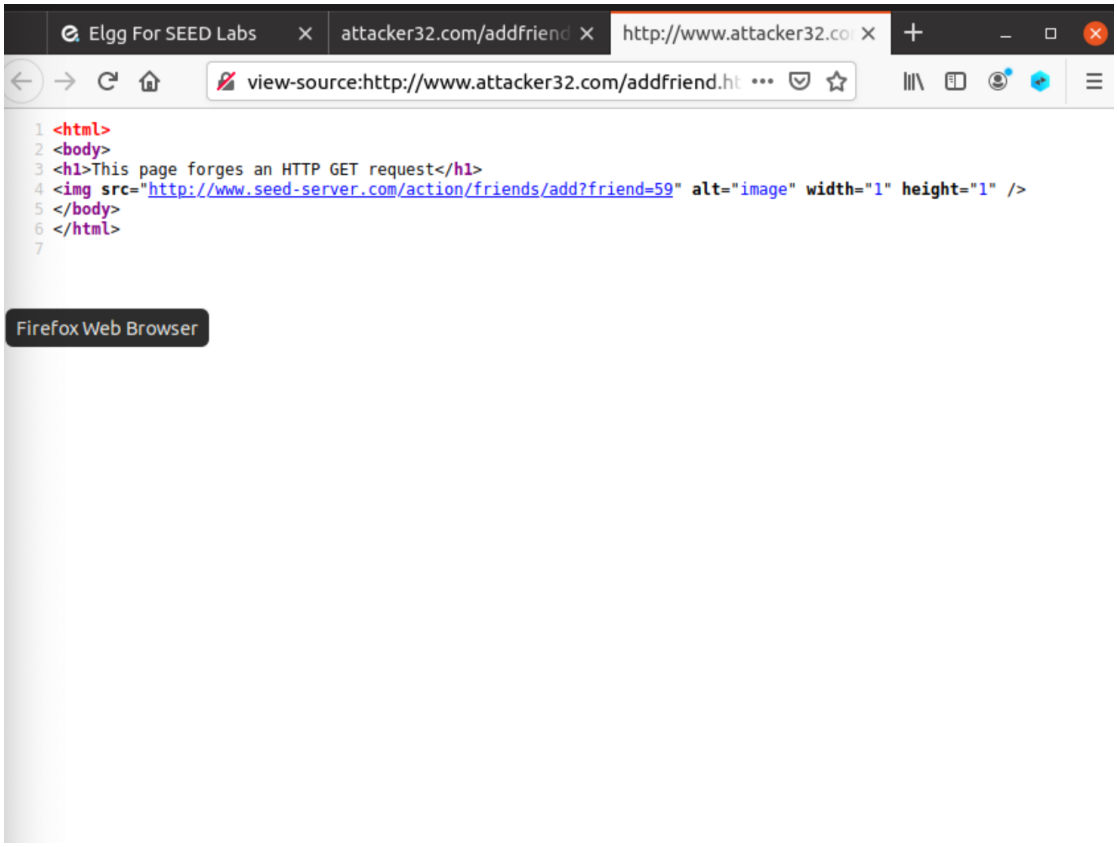
// I copied 846bbe5e714b:/var/www/attacker from attacker's shellcode and then used coker **cp to copy the containe**r



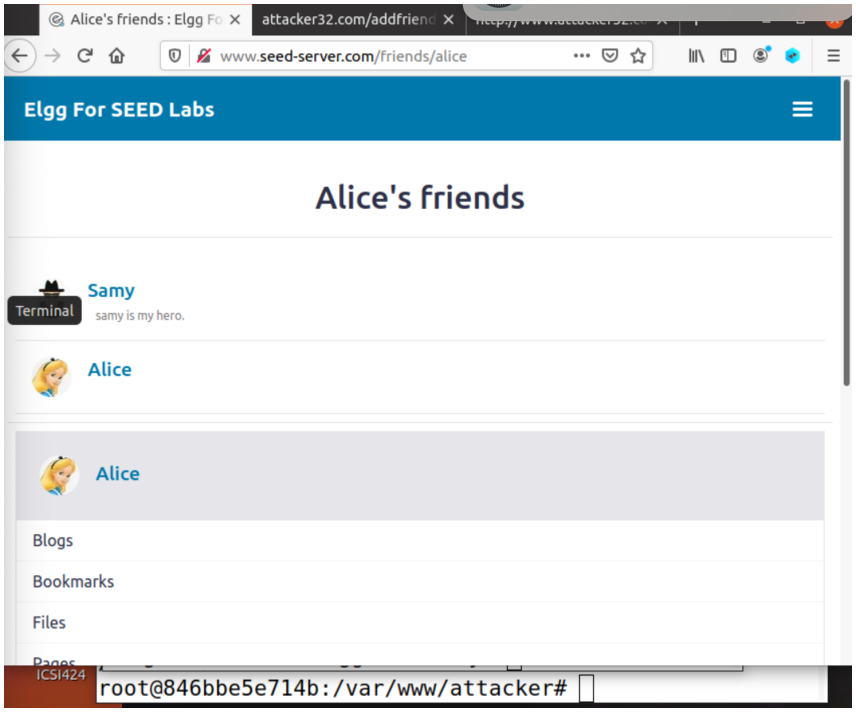
// It should be <img src= add-friend-url and samy's guid(user id)>

I used **docksh 846bbe5e714b** to get into www.attacker.com's shell code and I went into attacker's website
Using "**cd /var/www/attacker/** "
I am in an attacker's **shell and I opened addfriend.html to** see whether img scr= part was updated. (I used nano addfriend.html) It was modified.

// inorder to launch a cross site request forgery attack, Alice should be active ,so in Elgg website we logged in as alice and went to the ww.attacker32.com (malicious website) and clicked the “Add-Friend-Attack” link.
And then we will go back to the Elgg website, refresh and see whether Ailce’s friends were updated.
/ make sure Alice is active, click attacker32.com and refresh. First time I did this, even if Alice clicked the “Add-Friend-Attack " link, Samy was not added to her friend. So I just let Alice visit the “Add-Friend-Attack " link again and refresh the website.



// looks like the “add friend website” img src part is updated as what I wanted to launch. (56 is Alice’s guid and 59 is Samy's guid) .



// This shows that Samy is added as Alice's friend. So Alice is attacked by get message with Cross-site request forgery attack. // I have to make sure that Csrf.php countermeasure is off and Alice is activated(login currently) and Alice should visit/click a malicious website. Then Samy will automatically be added to Alice's friends.

3.3 Task 3: CSRF Attack using POST Request

Questions. In addition to describing your attack in full details, you also need to answer the following questions in your report:

• 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.

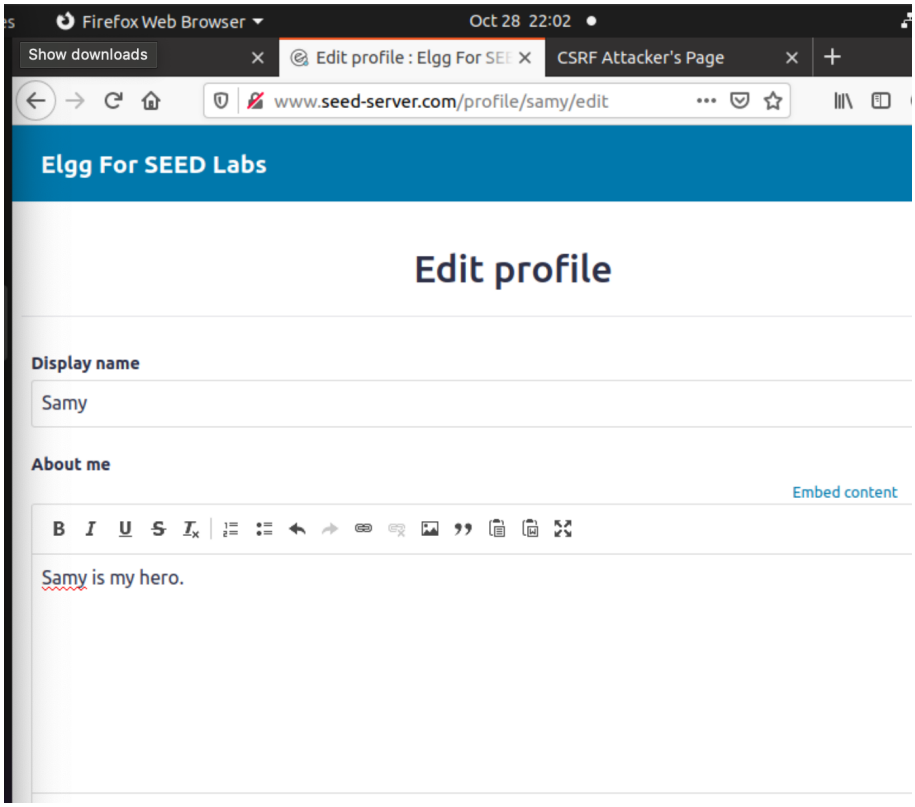
Summary : Let Boby as attacker and Alice is target user. Using HTTP request (get/put) , we can get Alice's user id. I don't think Boby needs to know ALice's Elgg password. In this task, Samy also didn't know Alice's Elgg password, but he was able to add it to Alice's friend and He was able to display "Samy is my hero" in Alice's profile description without knowing Alice's password.

Therefore, Boby doesn't necessarily need to know Alice's password to launch an attack. But he has to get Alice's GUID (userid) to attack Alice. And Boby can get Alice's GUID by inspecting

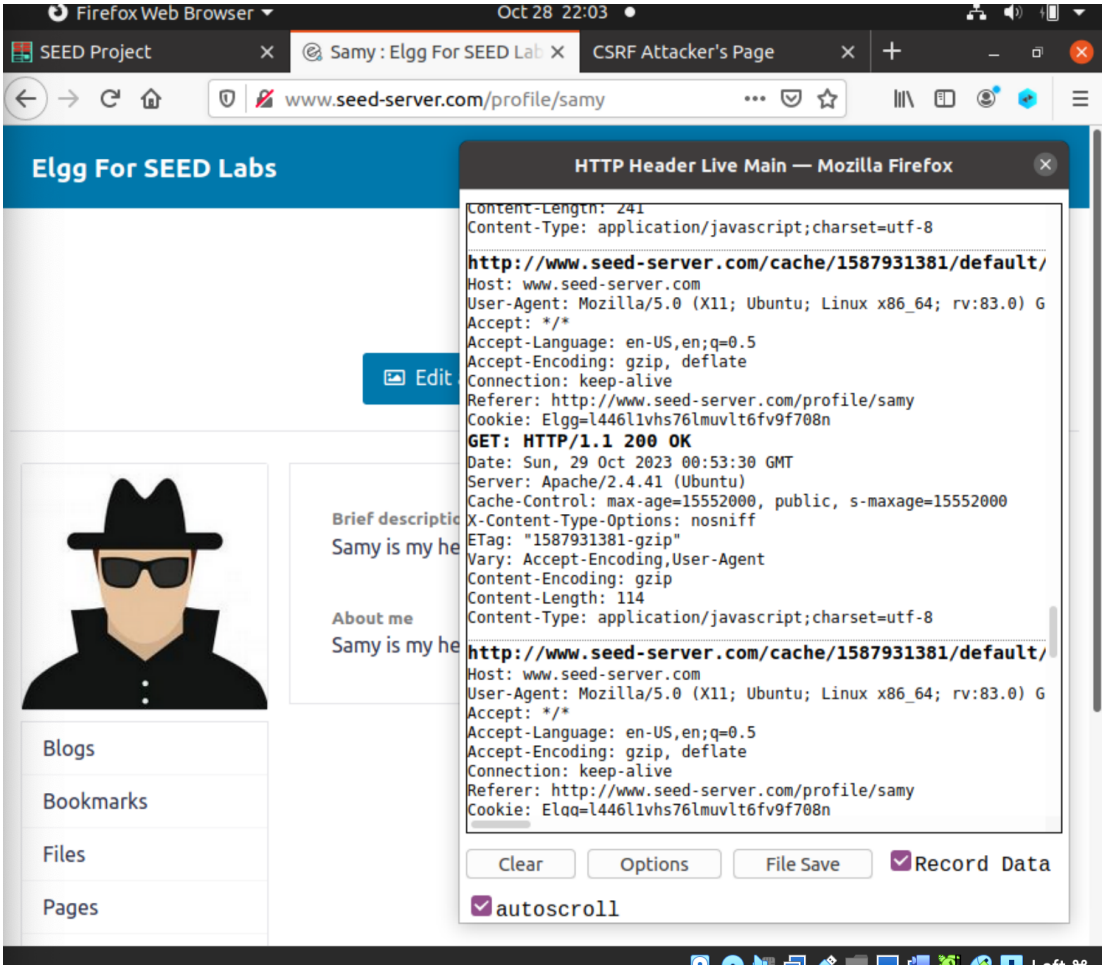
the website/element. But if we can't find Alice's GUID, then I don't think we can attack Alice using HTTP GET/Request/Response those types of methods.

• 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.

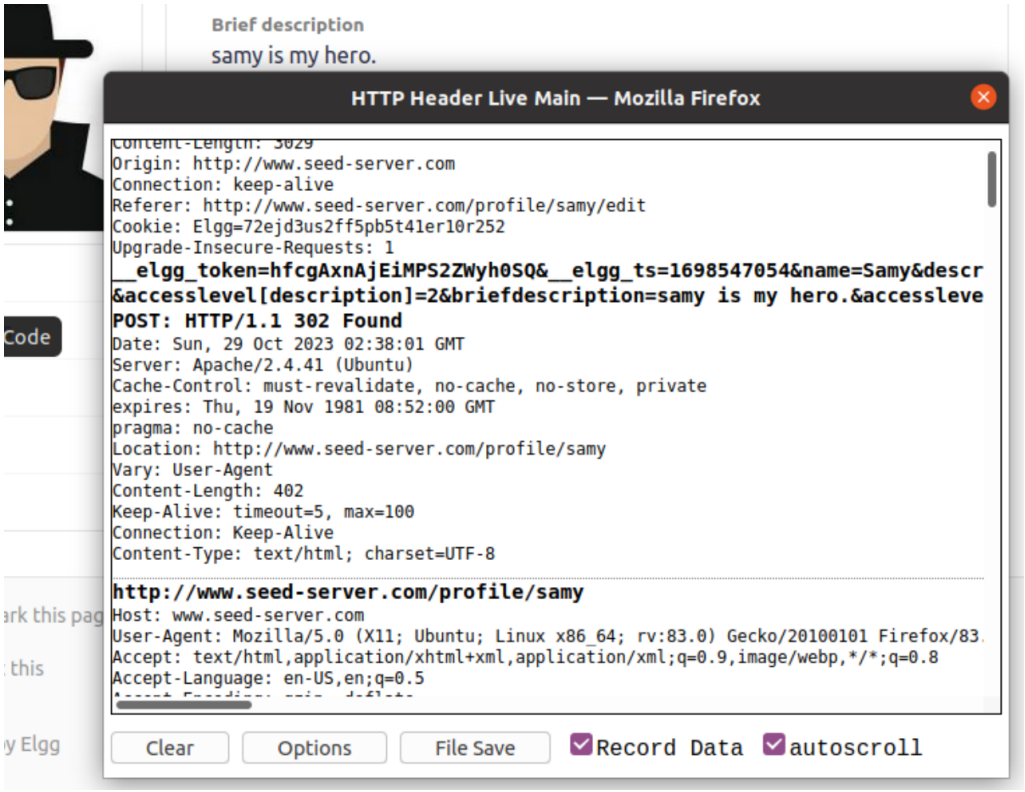
Summary: Boby is an attacker and wants to launch an attack with his malicious website. He doesn't know who is going to visit which means he doesn't have a specific target to attack. Can he launch an attack using CSRF in Elgg profile? -> Based on current lab task, we don't that samy want to attack Alice, so I was able to modify editprofile.html with target's name and GUID. (I needed the target's name (Alice) and the target's GUID(56) to launch the attack). Therefore, if Boby doesn't know who he wants to attack, then he can't launch an attack. Because he doesn't know which GUID he should provide in malicious code.



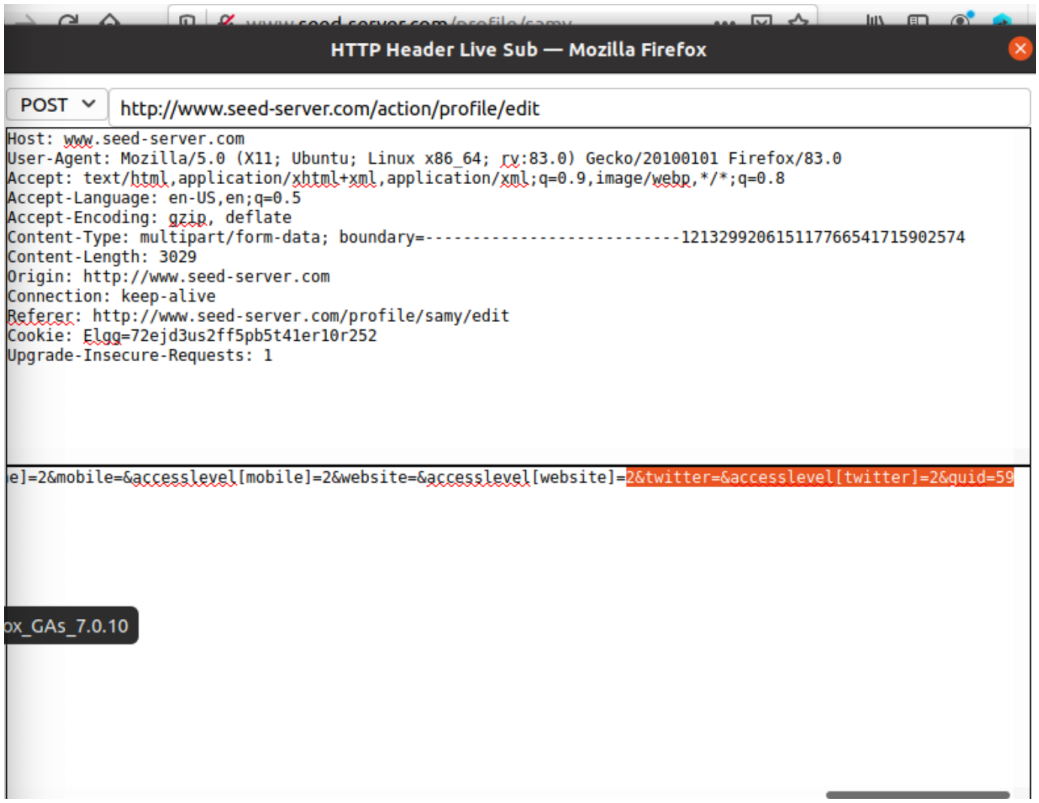
// open editprofile.html from attacker folder (it was from LabSetup.zip)
// 1. Log in as Asammy into "Elgg for seed lbs" and go to "add profile" In text field type " Samy is my hero. "
// open the HTML header line and then click save for updated samy's profile.



// here we can check that Samy's brief description part was updated with "samy is my hero" and I captured how Elgg for SED lab website updated samy's profile using HTTP Header LIve Main. I have to capture a post message.



// capture POST request we can see that briefdescription=samy is my hero. Is update date

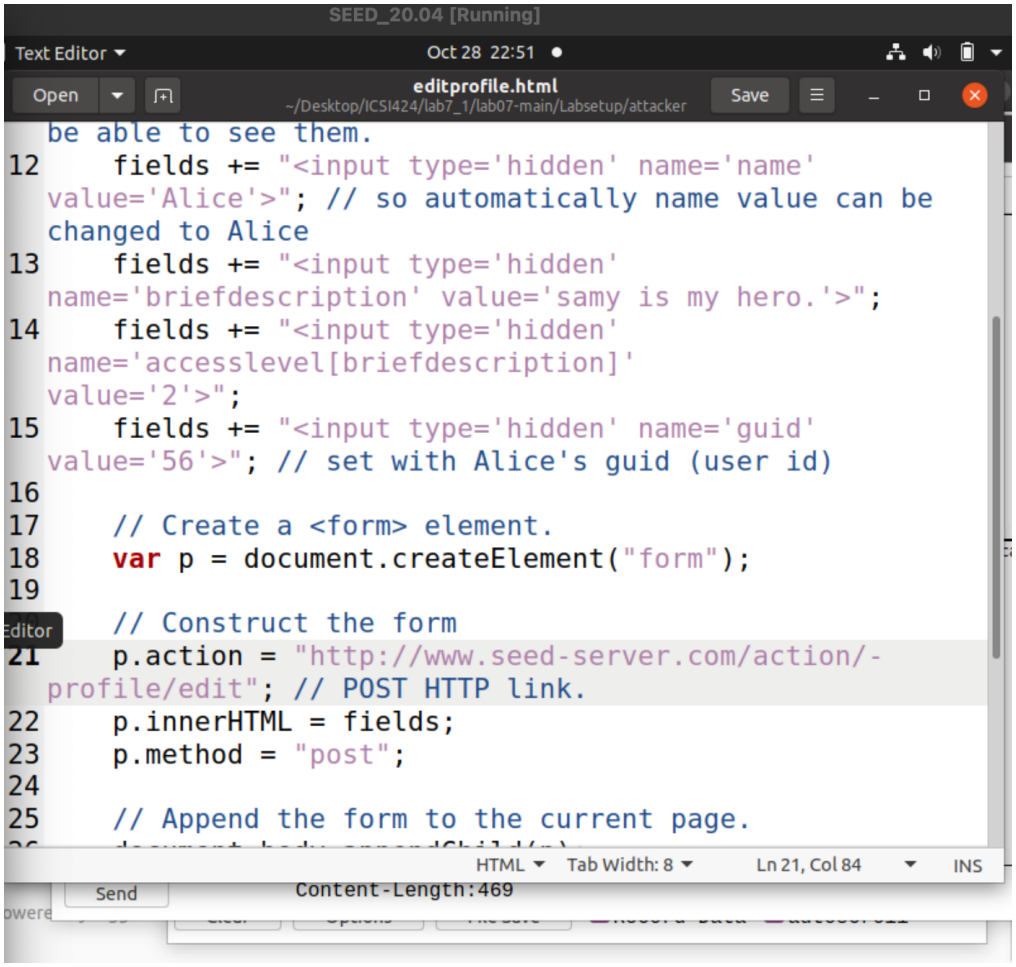


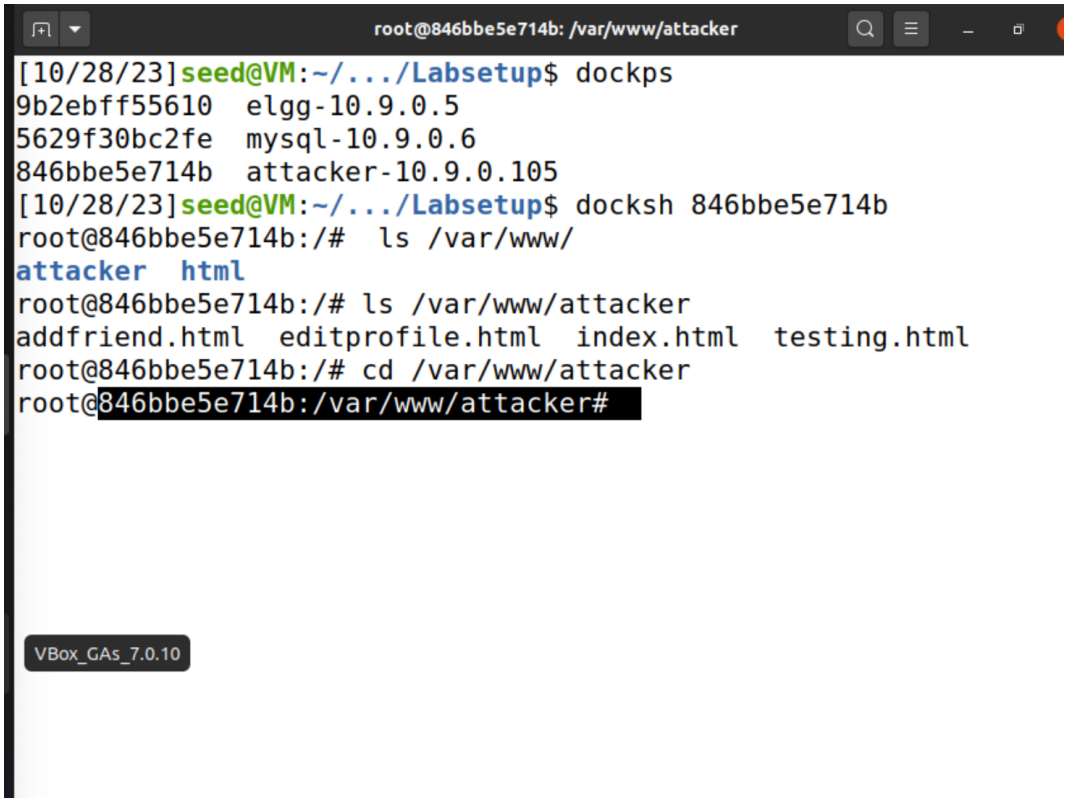
d.

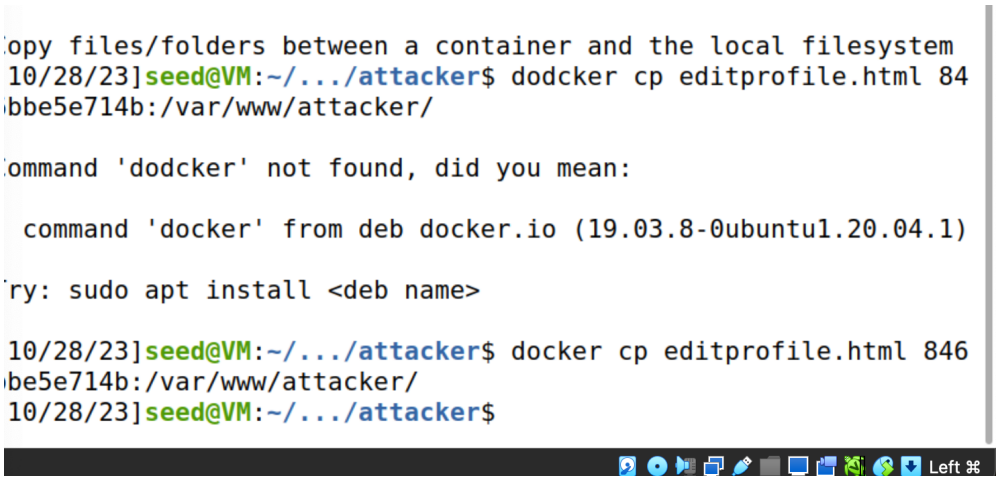
// capture post request and I can see that brief description is updated to samy is my hero and quid is 59 so it is samy. We want to post this “samy is my hero” to appear on alice’s brief description, we have to change 59 guid to alice’s guid.

// add name value to Alice and add briefdescription value as samy is my hero (this is what I updated on samy’s profile) and set guid value as 56 so we can put Alice’s guid.

// Just in case, **Alice guid is 56 and Samy’s guid is 59**



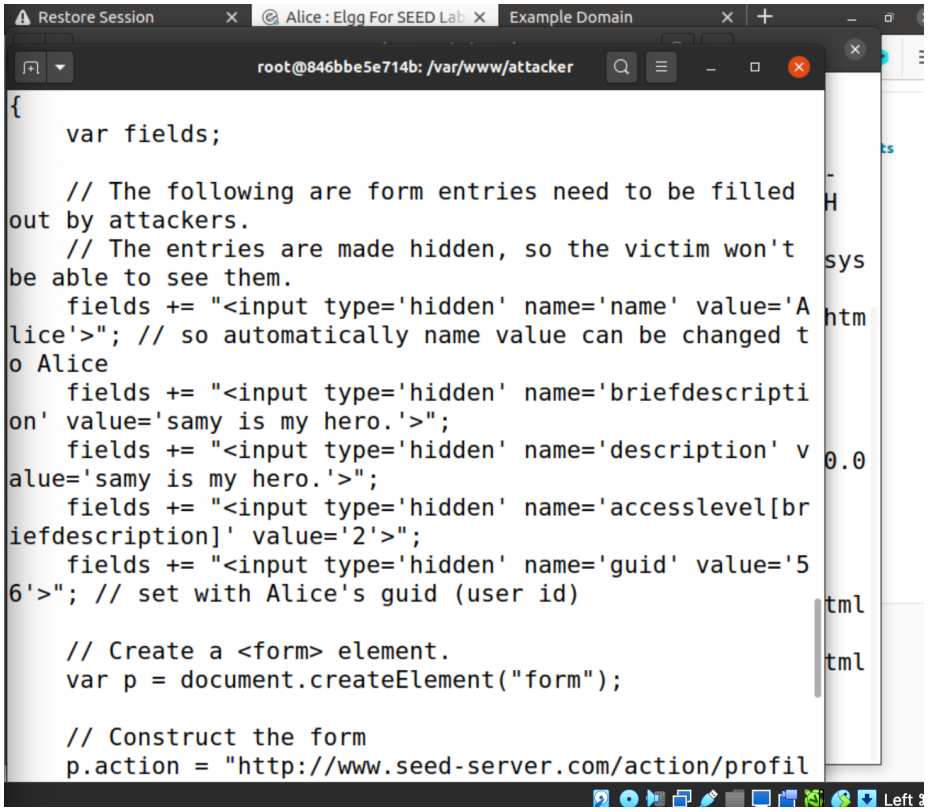




// type coker cp editprofile.html so we can modify editprofile.html and use the shellcode path that we copied.

```
[10/28/23]seed@VM:~/.../Labsetup$ dockps
9b2ebff55610  elgg-10.9.0.5
5629f30bc2fe  mysql-10.9.0.6
846bbe5e714b  attacker-10.9.0.105
[10/28/23]seed@VM:~/.../Labsetup$ docksh 846bbe5e714b
root@846bbe5e714b:/#  ls /var/www/
attacker  html
root@846bbe5e714b:/# ls /var/www/attacker
addfriend.html  editprofile.html  index.html  testing.html
root@846bbe5e714b:/# cd /var/www/attacker
root@846bbe5e714b:/var/www/attacker# cat editprofile.html
```

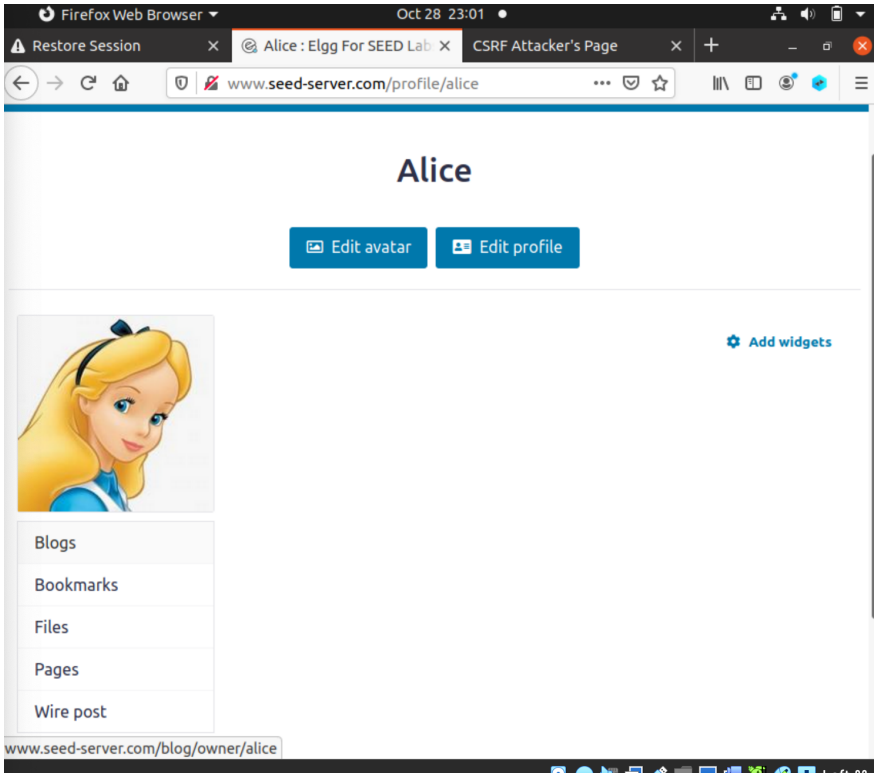
// typed cat **editprofile.html** in attacker's shell terminal to see whether editprofile.html is modified well.



// It looks like all the fields and p.action are updated to launch the attack.
// now we have to launch attack

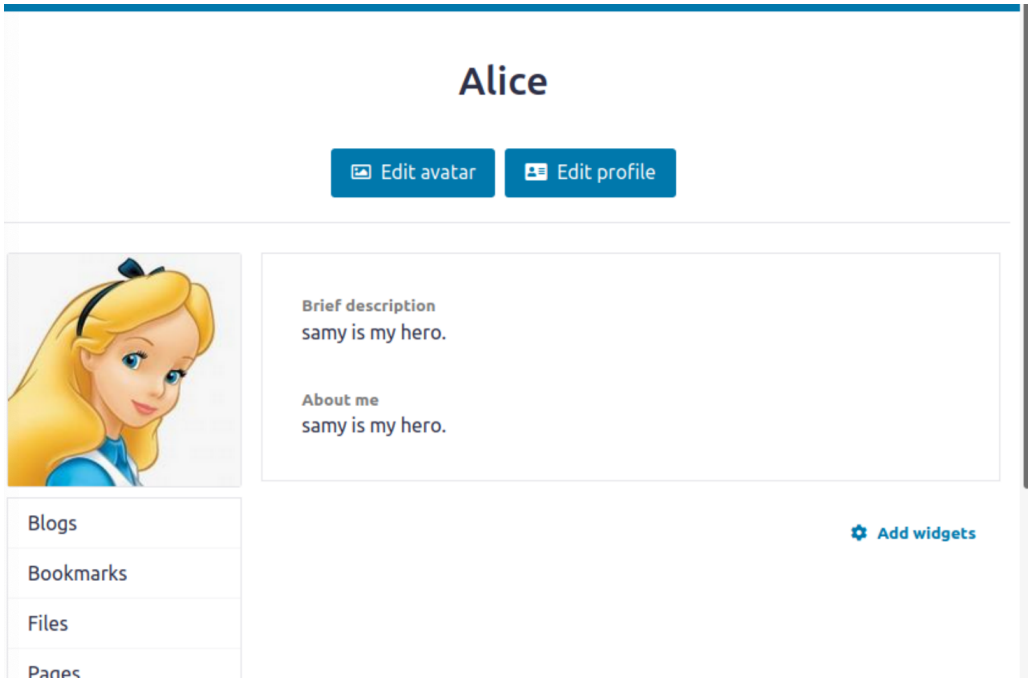
// so log in as ALice in Elgg for the SEED Lab website because Alice should be active. Go to her profile and check whether her profile is empty (it should be empty, because I didn't launch attack yet)

(I added description value too, because I changed samy' description value as samy is my hero.)



// Before launching the attack, Her profile is empty, so good!

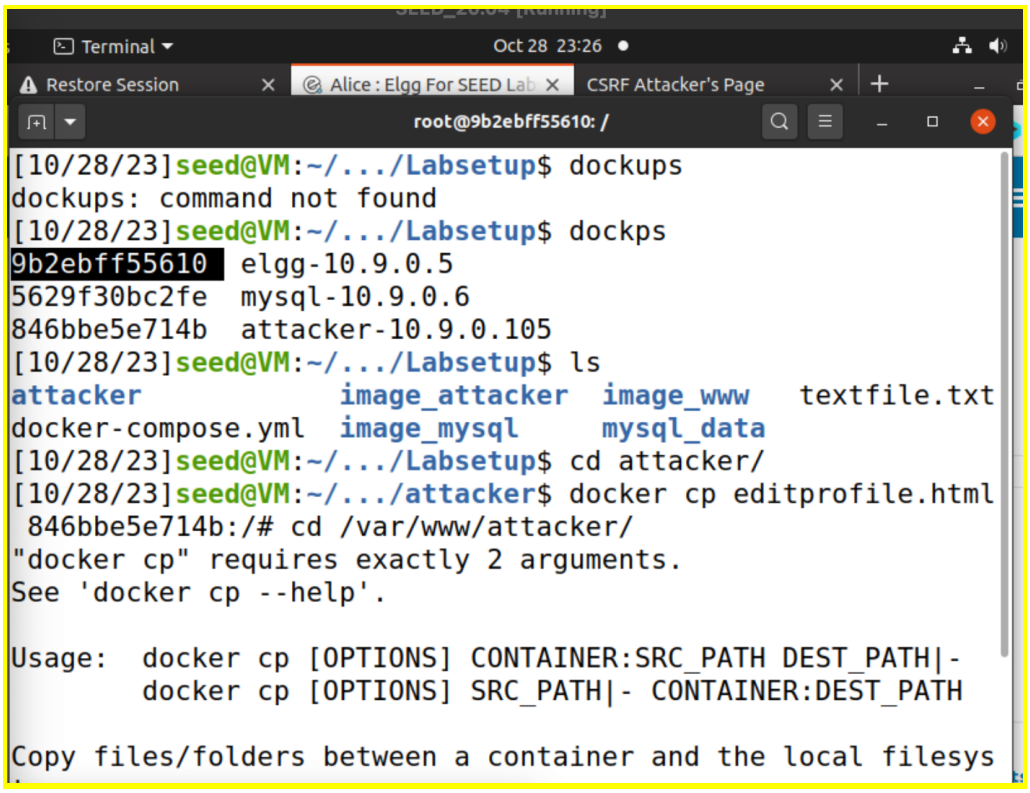
// While alice is login in Elgg for SEED lab website click " Edit-profile Attack " malicious link



// attack successed. Samy is my hero in Alice's brief description.

4.Lab Tasks: Defense

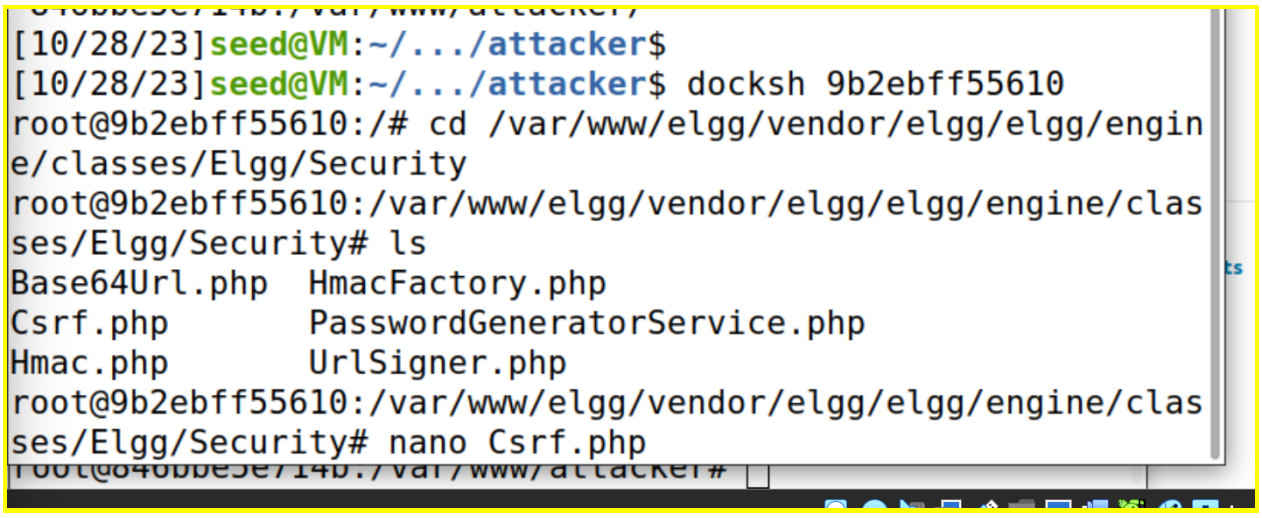
4.1 Task 4: Enabling Elgg's Countermeasure



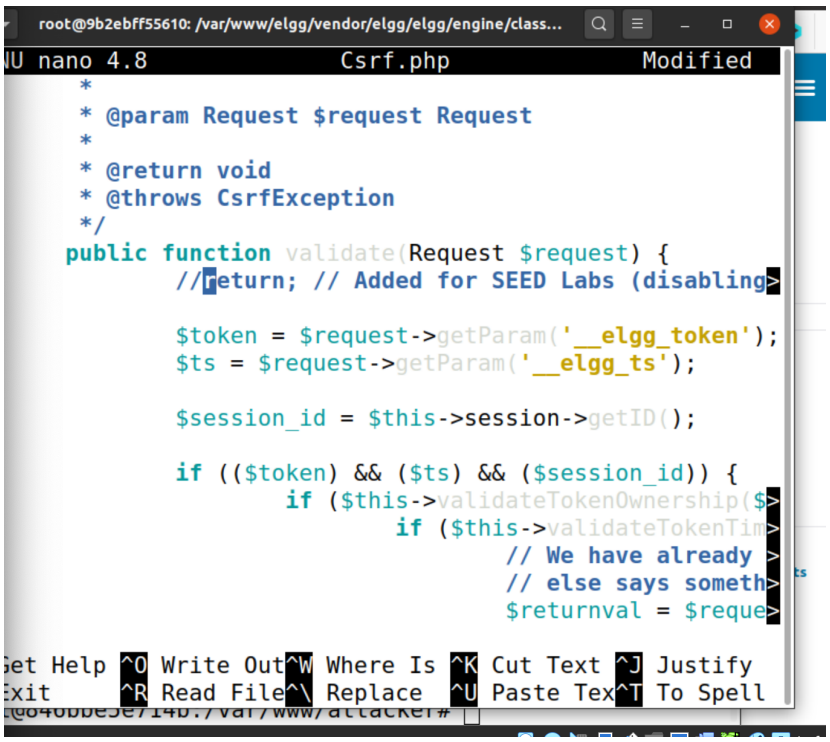
// go inside the elgg shell using docksh 9b2ebff55610

//I typed code given command line from lab instruction and goto Elgg/ Security

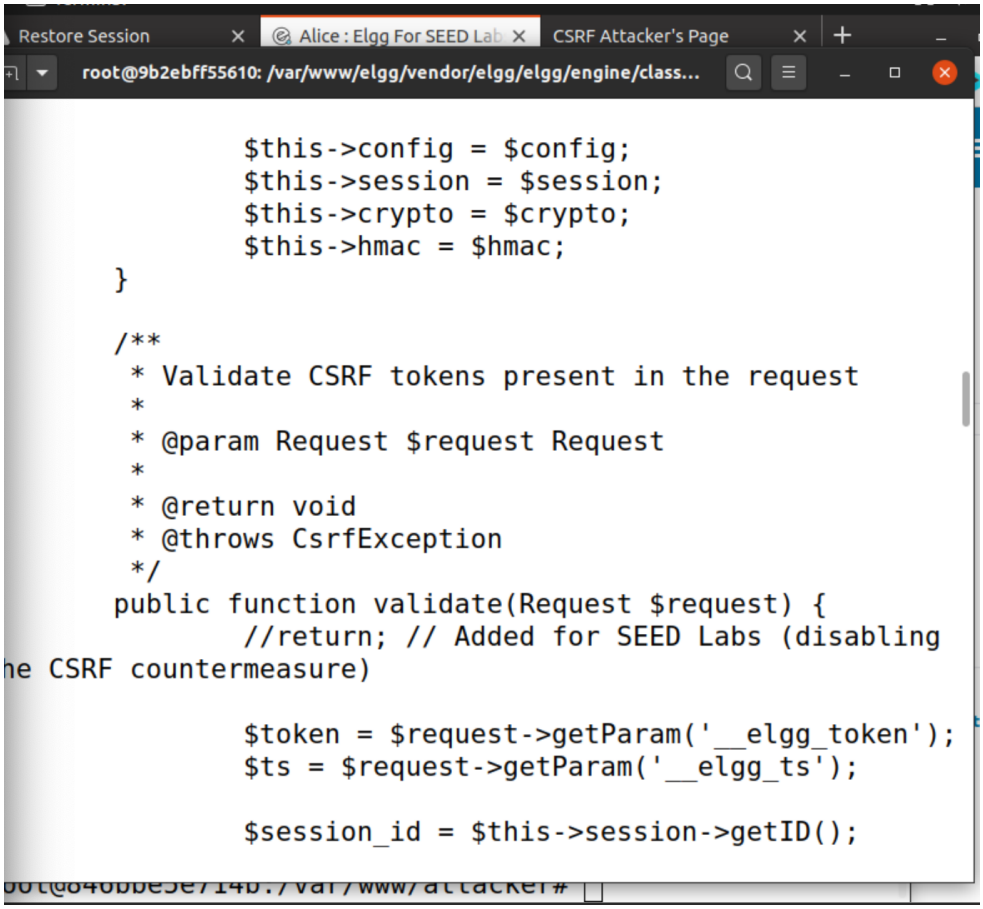
From Instruction 7 : “The secret token and timestamp are added to Elgg’s web pages by the vendor/elgg/elgg/views/ default/input/securitytoken.php module. The code snippet below shows how they are dynam- ically added to web pages.”



// type ls to see what is inside Elgg/ Security.
Type nano Csrf.php to modify Csrf.php.

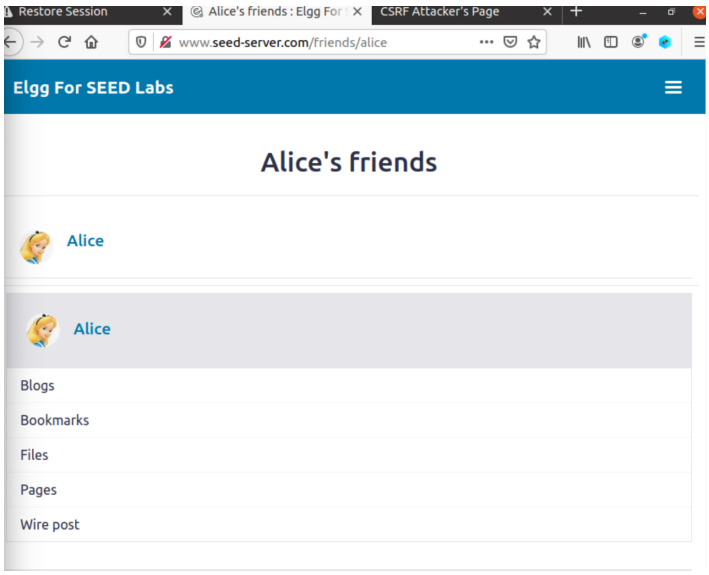


// go to validate(Request $requst) method and comment return. And save it using cat Csrf.php and check whether I commented return well.

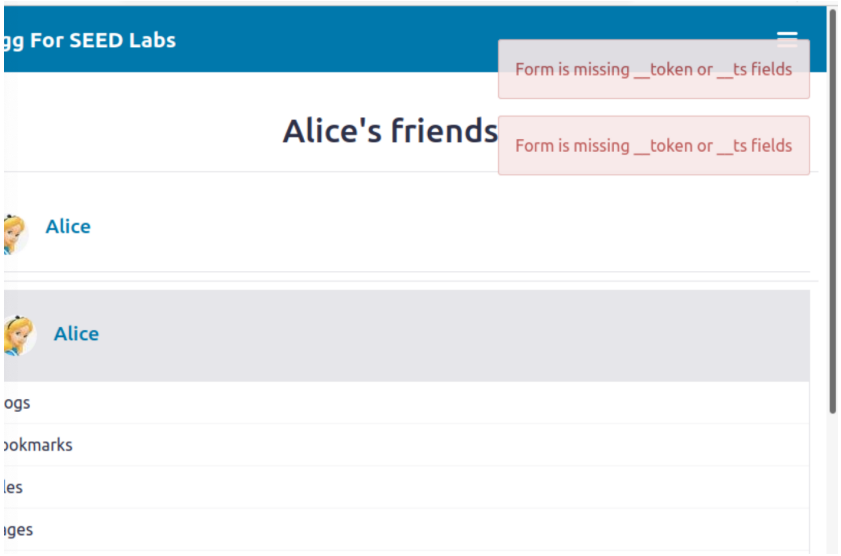


// It was commented out well.

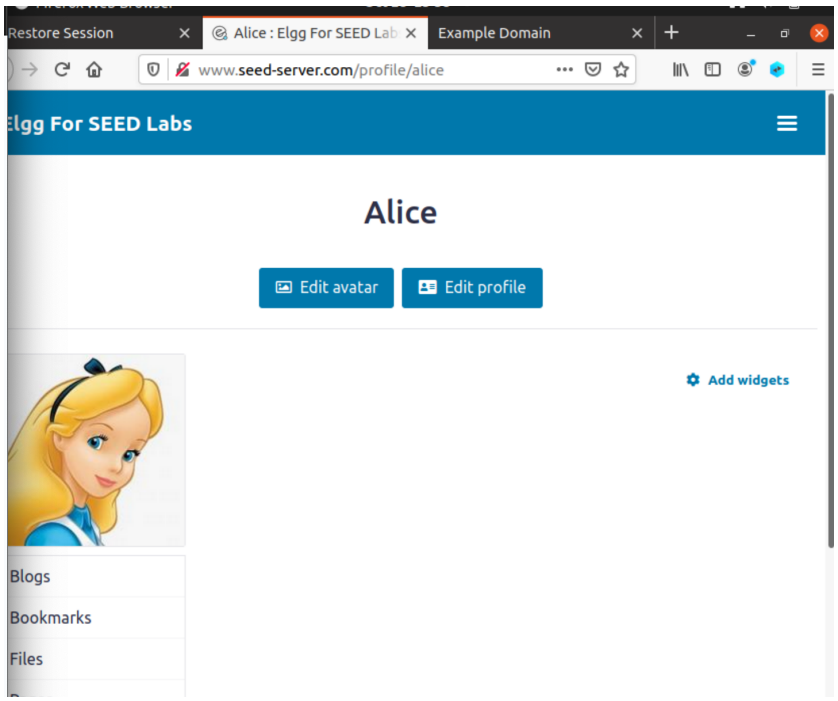
// caret empty profile for Alice (i just erased samy is hero part) and go to alice's friend and remove friend for Samy.



// Now Alice has no friend and is not showing “samy is my hero. “ sentence.



//Right side top red alert box shows that “from is missing _tokent or _ts fields, because we turned on the CRSF countermeasure, therefore malicious code/attacker can not attack Alice anymore

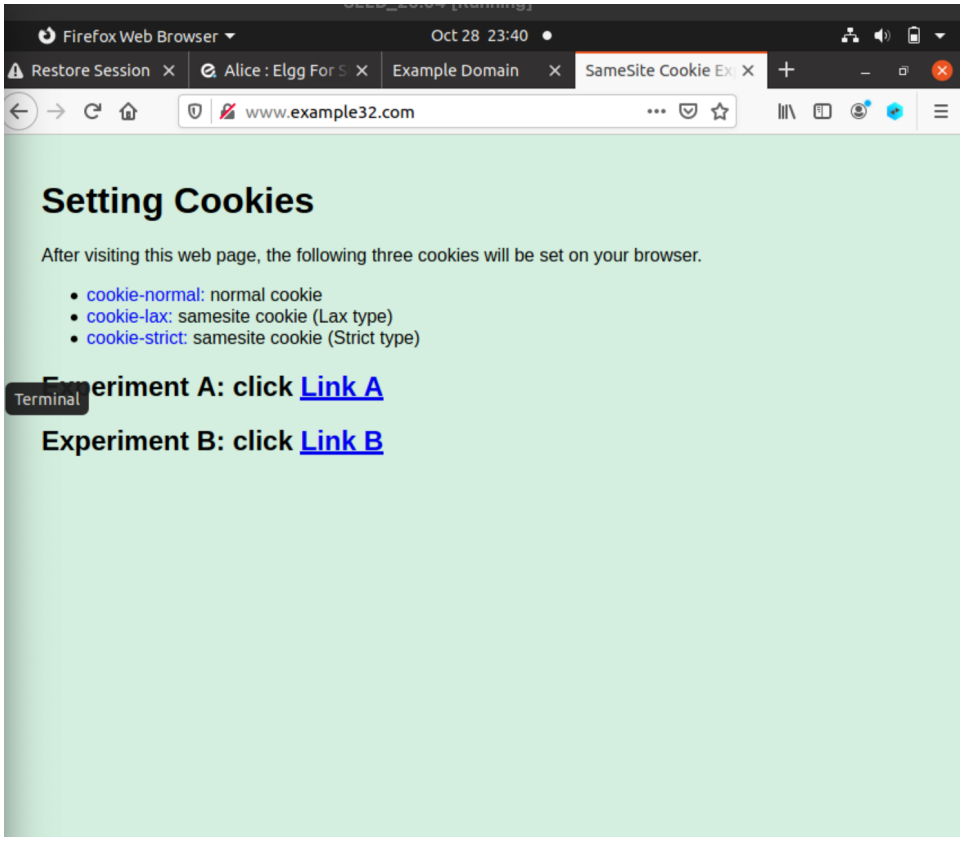


// profile was still empty even though I logged in as LIce and went to malicious code and clicked that malicious code and refreshed Alice’s profile. It didn't print out. Samy is my hero sentence.

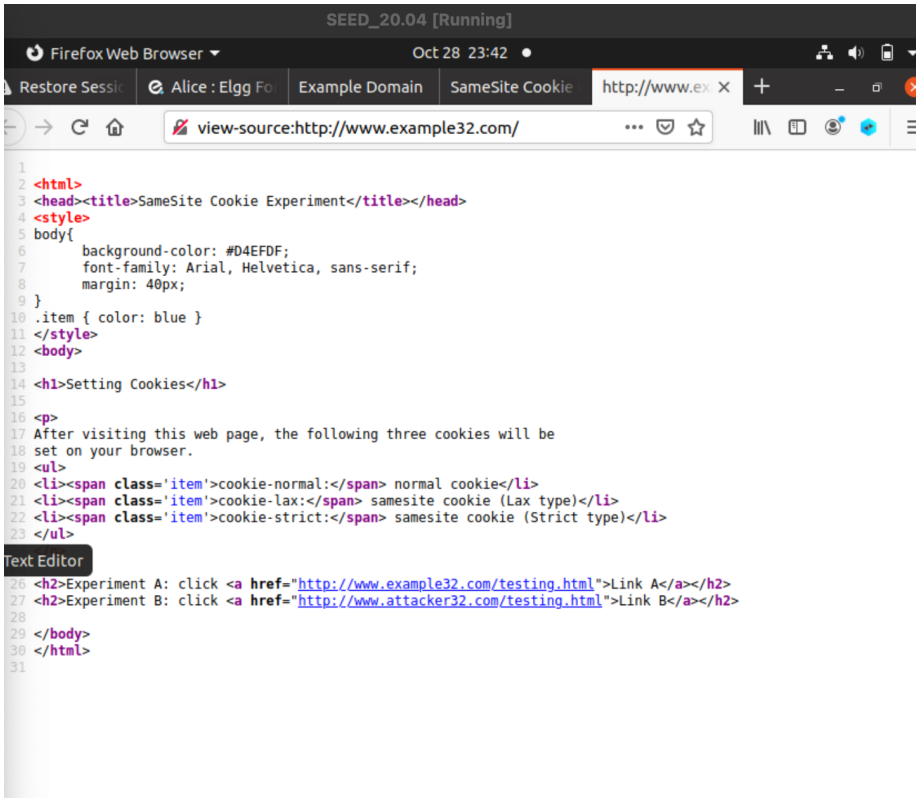
// I visited two malicious websites from attacker32.com and went back to Elgg for the SEED lab. It doesn’t attack alice’s website and keep printing out From is missing _toket or _ts fields. SO I think countermeasures are working since we can not launch a cross-site request forgery attack.

4.2 Task 5: Experimenting with the SameSite Cookie Method

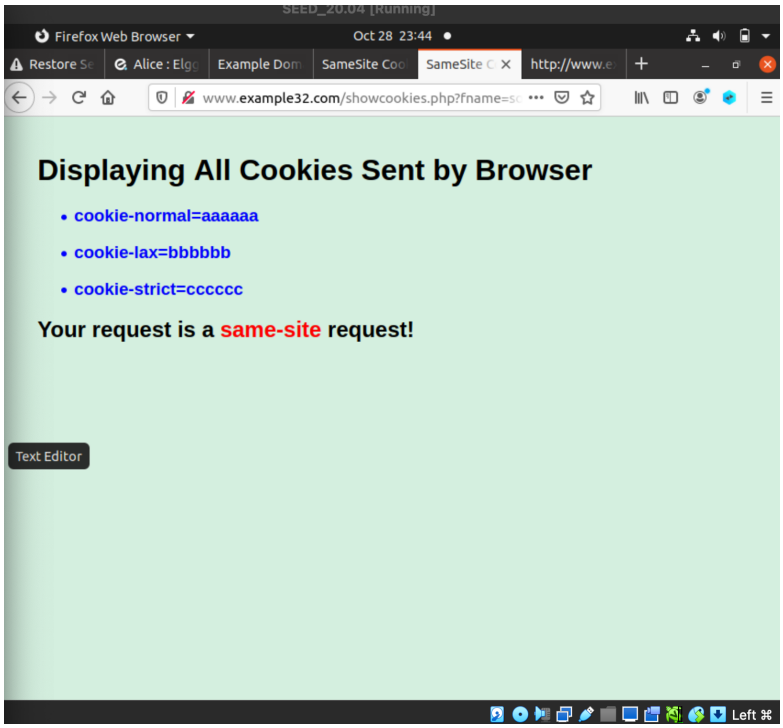
// visite www.example32.com website



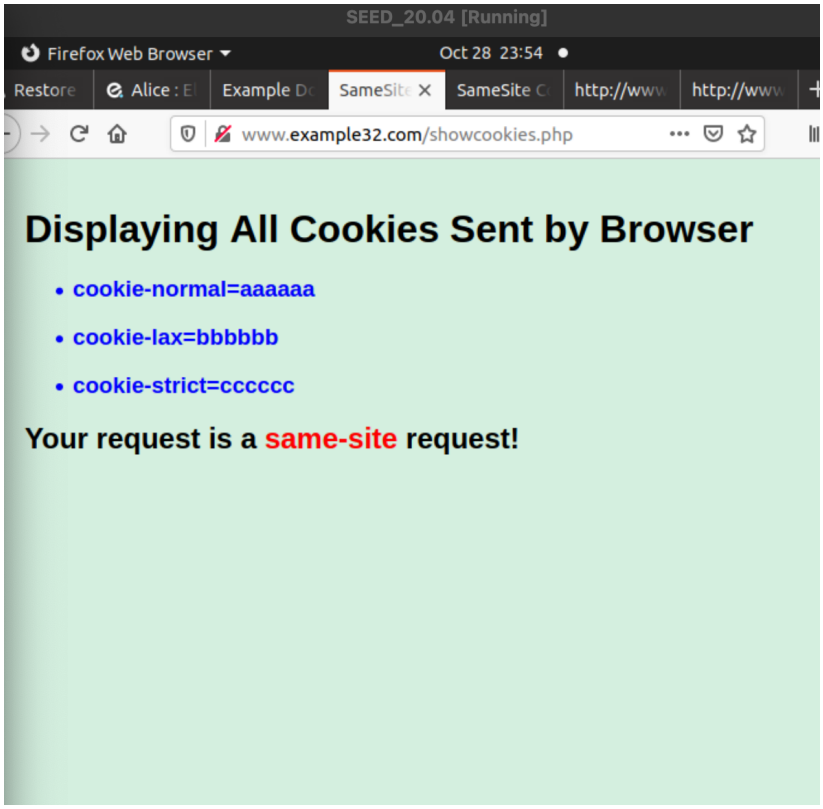
// right click and view page source code,



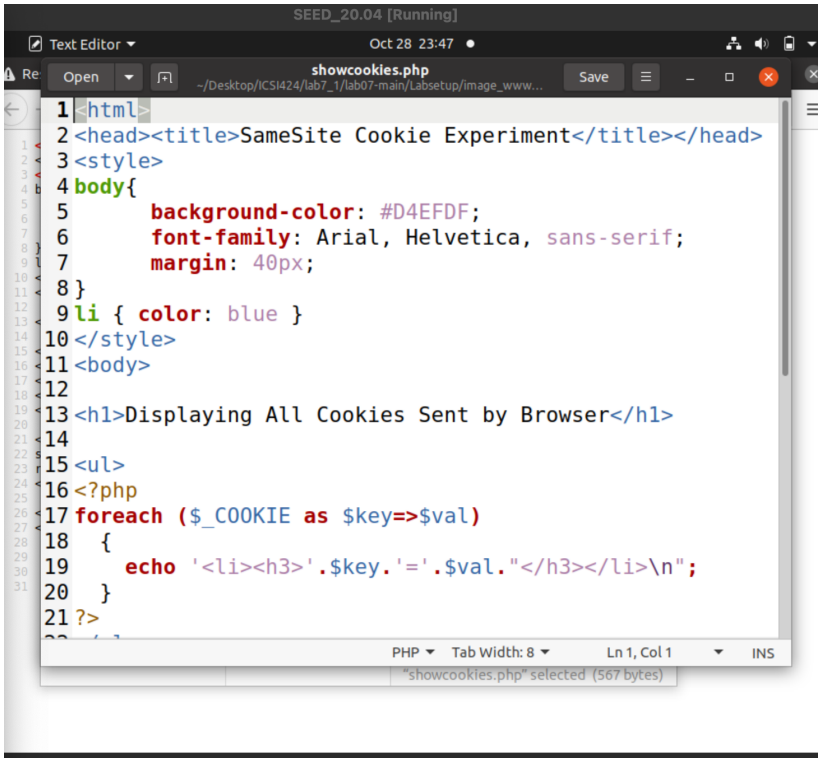
// then we can see javascript code for **a same-site cookie websi**te.
We can see tha**t Link A is same w**ebsite www.example32.com but l**ink B is from different website** which was www.attacker32.com (cross site request case)



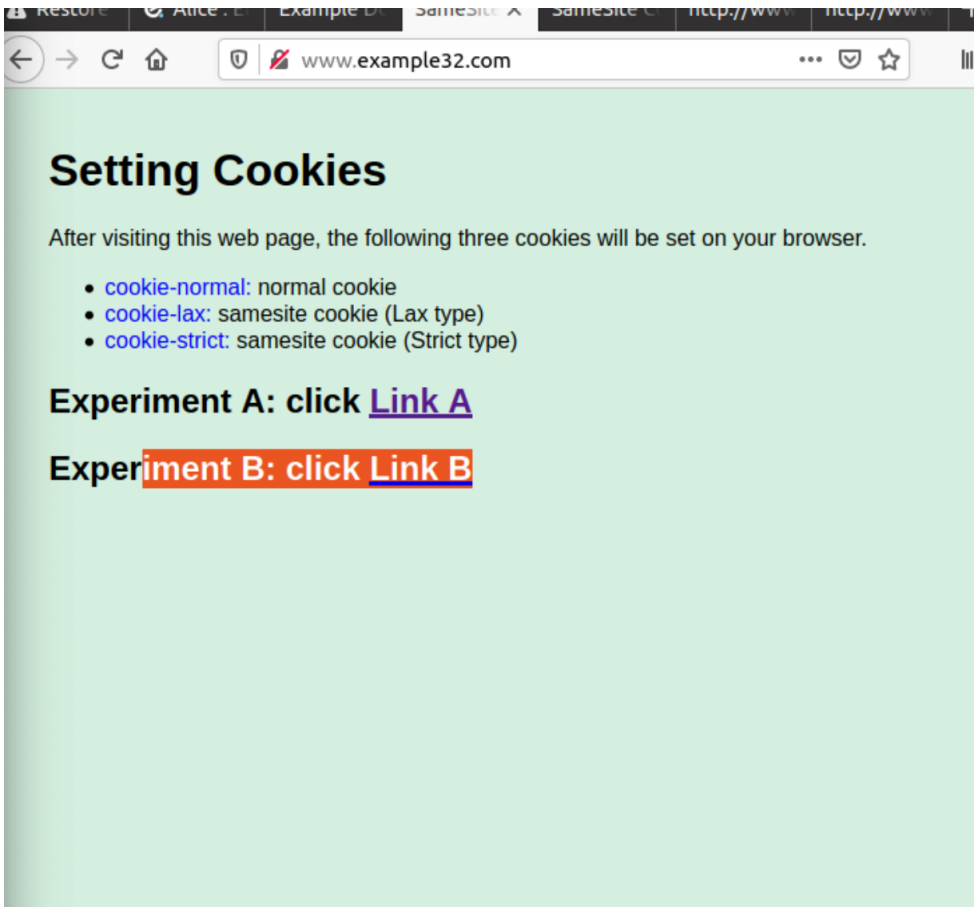
// I went to **LinkA and client submitted(GET) c**ookies and I got this website. (sent get request) it says your request is the same site request. Because I opened this website from www.example32.com and this get request cookie website is also on www.example32.com
I clicked right and saw the detail of javascript code of above website, but there won't enough information so I web to Lab Setup > image_www > sgowcookies.php > open with text editor.



// This is the result when I tried the same site request **for link A and clicked Post request**. I got cookie-normal, cookie-lax and cookie-strict.



// same site or cross site sentence were printed from this statement. Below.

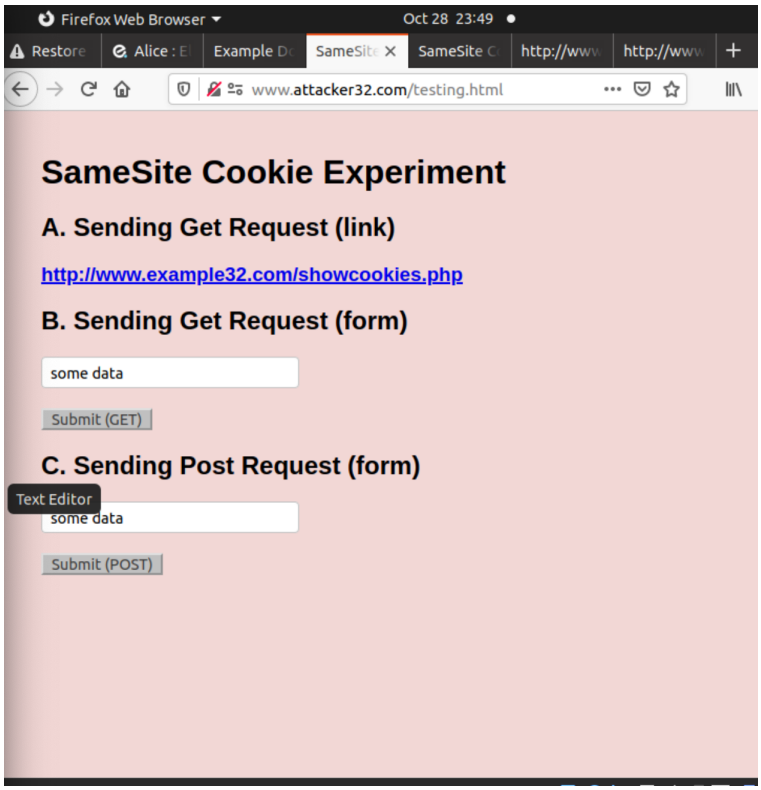


```
15 <ul>
16 <?php
17 foreach ($_COOKIE as $key=>$val)
18   {
19     echo '<li><h3>'.$key.'='.$val."</h3></li>\n";
20   }
21 ?>
22 </ul>
23
24 <h2>Your request is a <font color='red'>
25 <?php
26 if(isset($_COOKIE['cookie-strict'])) {
27    echo 'same-site ';
28 }
29 else {
30    echo 'cross-site ';
31 }
32 ?>
33 </font>
34 request!
```

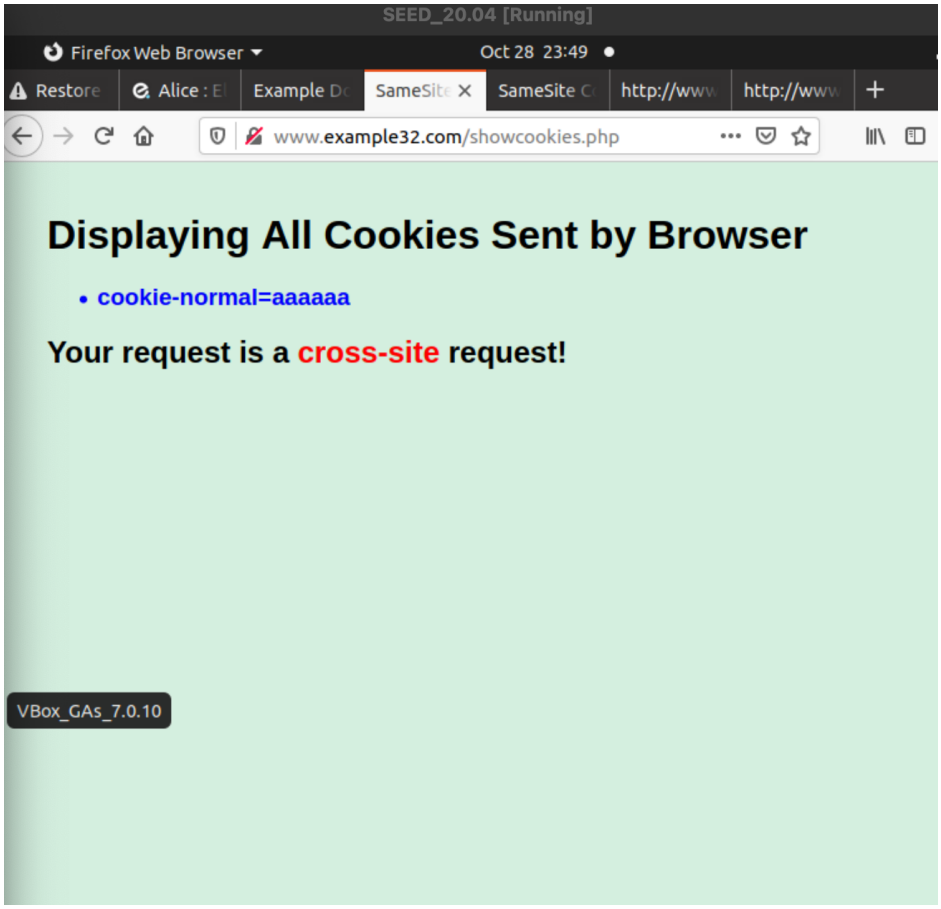
PHP Tab Width: 8 Ln 1, Col 1 INS

"showcookies.php" selected (567 bytes)

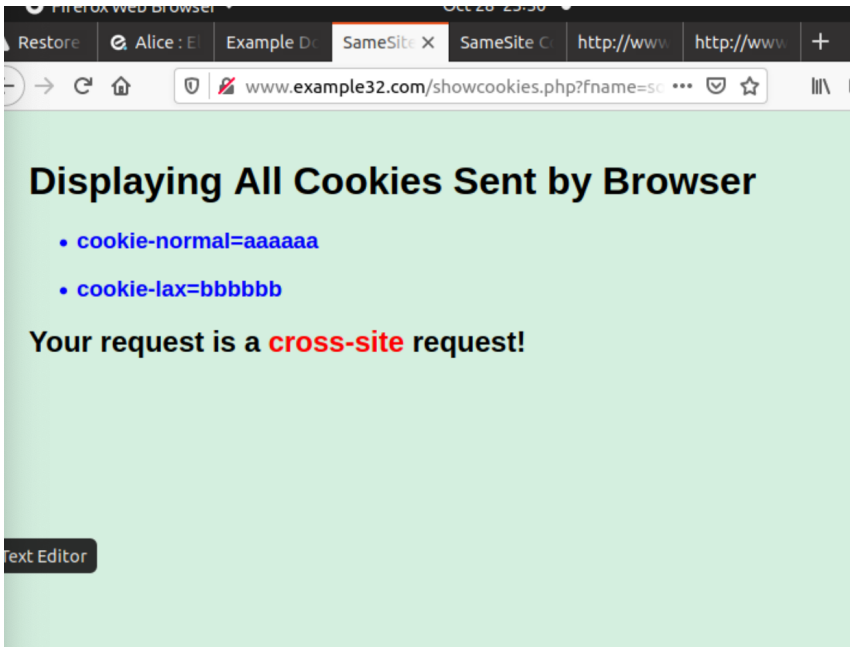
// now I will click link B



// this is how I land. I w**ill submit(POST)requ**ests.



// I got into that website and it shows that my request was a cross-site request.



// I also **tried the get request butt**on and it shows that my request was a cross site request and shows cookie-normal and cookie-lax.

What is SameSite protection using Cookies?

SameSite protects browsers from sending that cookie along with cross site requests. The main goal is to mitigate the risk of cross origin information leakage. It also provides some protection against cross-site request forgery attacks. Possible values for the flag are none, lax, or strict.

- **Please describe what you see and explain why some cookies are not sent in certain scenarios.**

  Same site Cookie is one of countermeasures for CSRF attacks. For Same-Site-Cookies, the web server is responsible for doing this and they could use this to secure the website. Because Same site Cookie attributes are setted by servers and these attributes will tell browsers whether cookies should be attached to a cross site request or not (This is from lecture ppt).

  Based on analysis, Samsite link(A) with GET request had cookie-normal, cookie-lax, and cookie-strict. Samesite link(A) with Post request had cookie-normal, cookie-lax, and cookie-strict.

  For crossSite(B) with GET requests had cookie-normal and cookie-lax. (no cookie-strict for cross site). Cross Site link(B) with post request had cookie-normal. (no Cookie-lax and Cookie-strict).

  Cookie-strict will not be sent with a cross site link.

- Based on your understanding, please describe how the **SameSite** cookies can **help a server detect whether a request is a cross-site or same-site req**uest.

Answer for this question is similar to above. SameSite cookie has specific 3 attributes and especially cookie-strike will not be sent with, if browser is a cross-site request. Which means the server can detect whether the request was cross-site or same-site by checking whether the same site cookie has “cookie-strict” value. In this case, the server can distinguish cross site requests with the same site request, therefore the server can prevent the attack.

- Please describe how you would use the **SameSite cookie mechanism to help Elgg defend against CSRF attack**s. You only need to describe **general ideas, a**nd there is no need to implement them.

In a previous lab task, Samy used malware to attack Alice. So Alice basically had to click/visit a malicious website. The malware then attacked Alice's website in Elgg and the execution attack was successful. This means that the malware is a cross-site link. If Elgg servers store same-site cookies, this website always checks whether an incoming request is a same-site or cross-site request (e.g. Samy's malicious website). Elgg servers can then distinguish between Samy's malicious websites. This is because it is a cross-site website and there are no cookie restrictions. In this way, Elgg servers can defend against CSRF attacks.