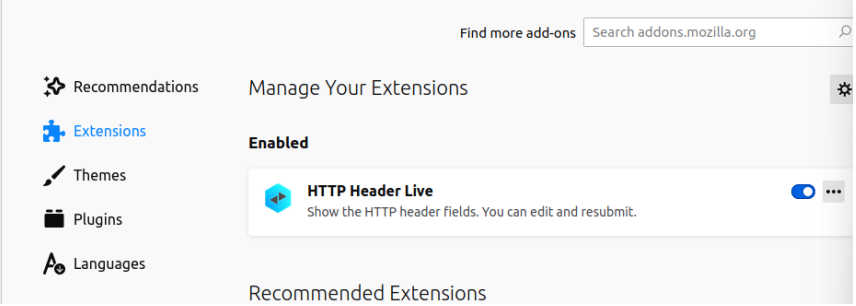
ASSIGNMENT 5

CSRF ATTACK

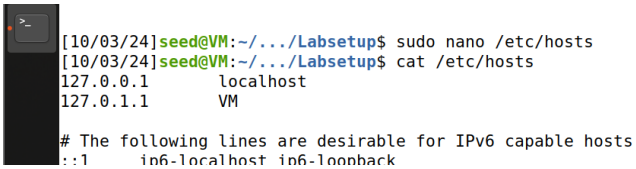
APARNAA MAHALAXMI ARULLJOTHI ( A20560995 )

## TASK 1: OBSERVING THE HTTPS REQUESTS:

- The http header live extension has been added to firefox

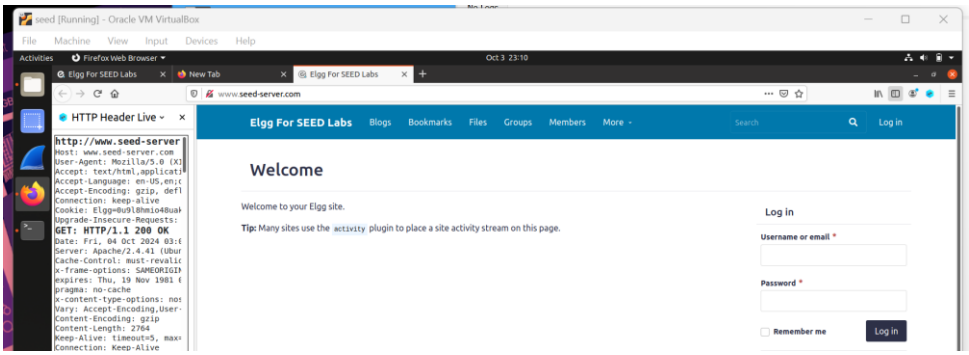



- The necessary containers for csrf attacks has been added to /etc/hosts file.
- the hosts names are mapped to their corresponding IPs.
- To get the root access to  do the DNS configuration, I have used the command
    - sudo nano /etc/hosts

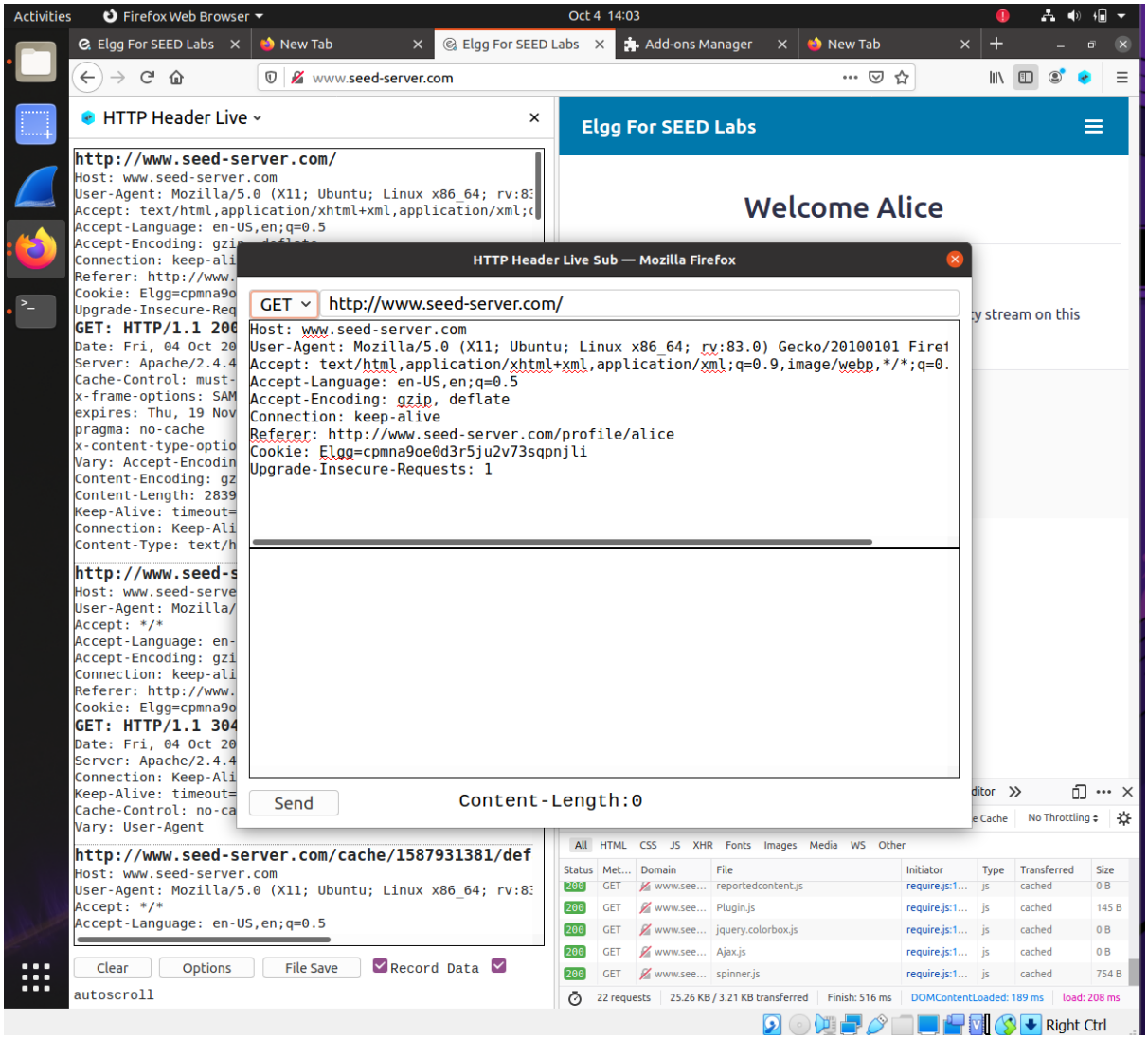


```
[10/03/24]seed@VM:~/.../Labsetup$ sudo nano /etc/hosts
[10/03/24]seed@VM:~/.../Labsetup$ cat /etc/hosts
127.0.0.1       localhost
127.0.1.1       VM

# The following lines are desirable for IPv6 capable hosts
::1     ip6-localhost ip6-loopback
```

- Using the http header live tool, the get and post requests has been captured.



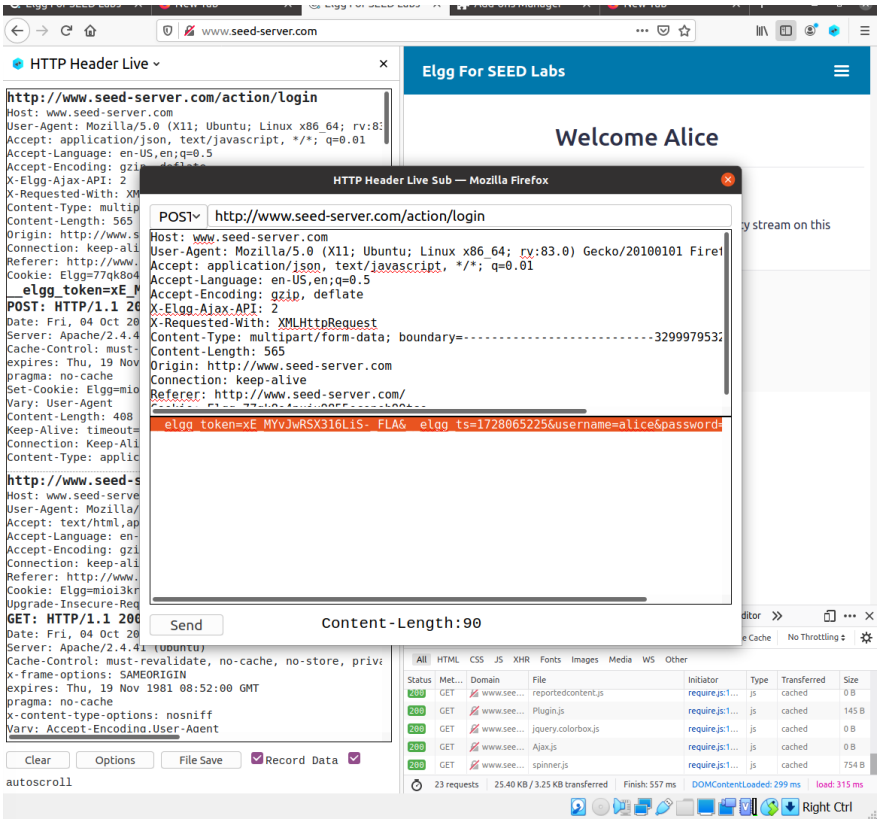
## CAPTURING THE GET REQUEST:




- **THE PARAMETERS INCLUDED HERE ARE**:
  - **URL: REFERER**
    - This states the url from which the request is originated
  - **COOKIE:**
    - This has the session data
      - **Elgg=cpmna9oe0d3r5ju2v73sqpnjli**, where Elgg is the cookie name and **cpmna9oe0d3r5ju2v73sqpnjli** is the value.

## CAPTURING THE POST REQUEST:

- The post request is captured by submitting the form.



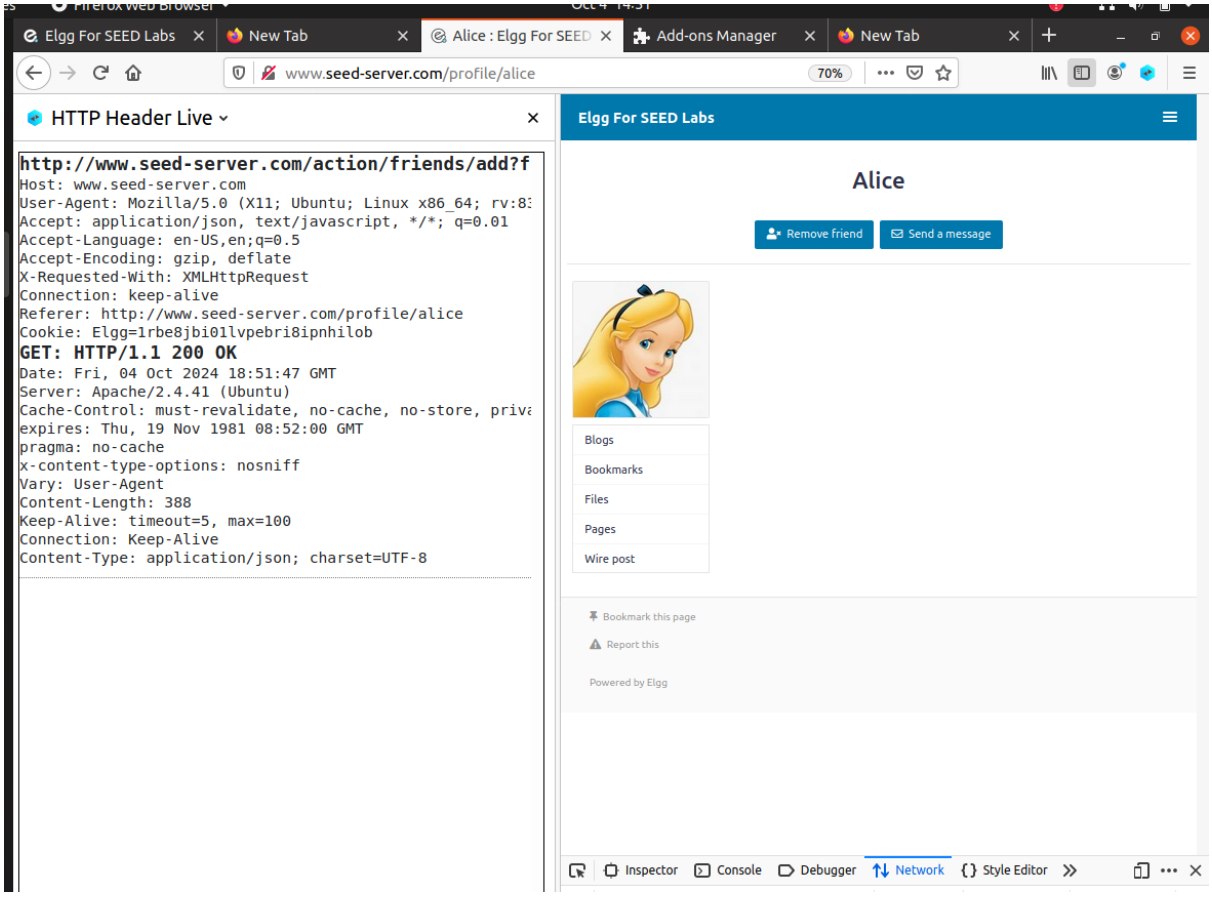

## PARAMETERS INCLUDED HERE ARE:

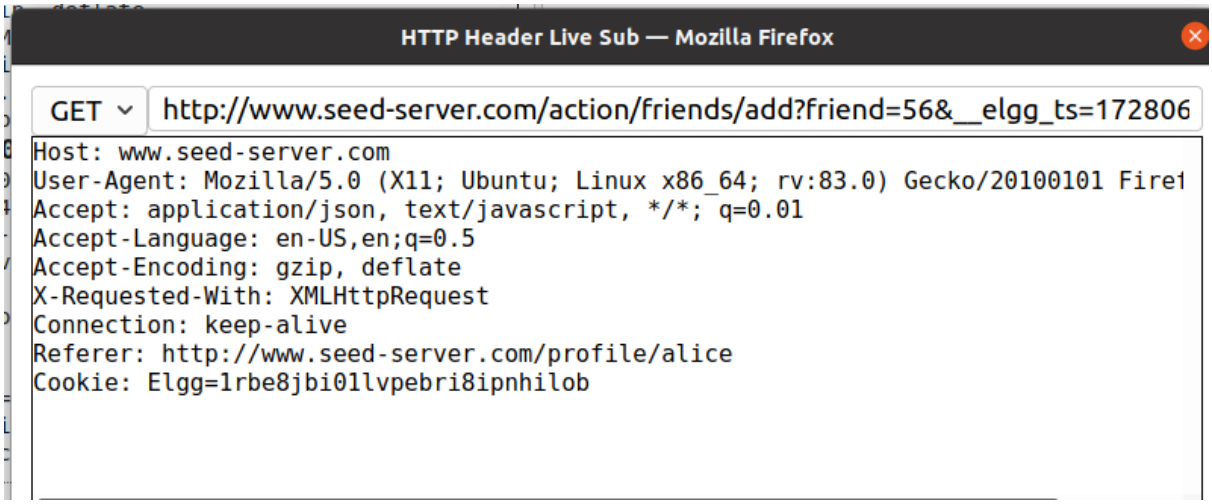
1. **__elgg_token**: it ensures that the request is coming from a legitimate source
   - **name**: __elgg_token
   - **Value**: xE_MYvJwRSX316LiS-_FLA
2. **__elgg_ts**: timestamp parameter is used for various purposes like session validation and to prevent replay attacks.
   - **name**: __elgg_ts
   - **Value**: 1728065225
3. **username**: it contains the username of the user trying to login
   - **Name**: username
   - **Value**: alice
4. **password**: it contains the user's passwords
   - **Name**: password
   - **Value**: seedalice

## TASK 2 : CSRF ATTACK USING GET REQUEST:

- Here, I logged into Samy's account and added Alice as Samy's friend.
- Then, I captured the request after adding Alice as Samy's friend
- On looking at the get request we are able to find that Alice's id is 56.
- Then by inspecting on Samy's profile we are able to get Samy's id is 59.

.



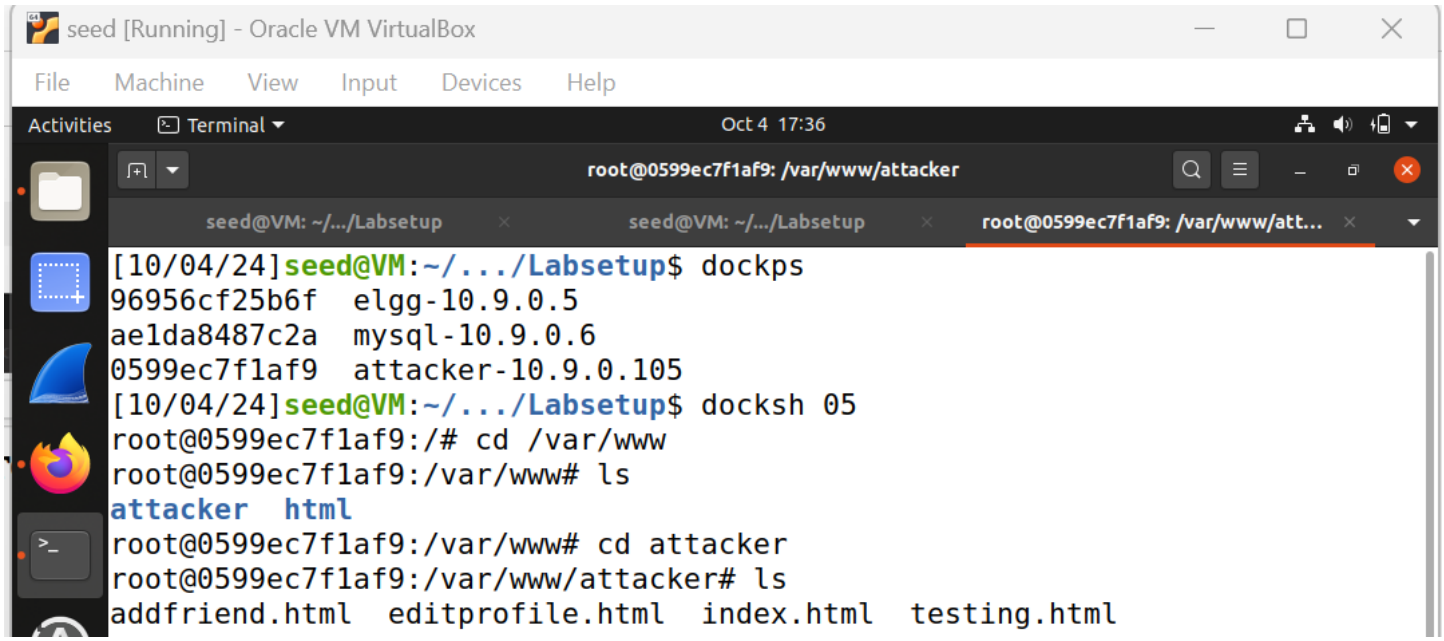



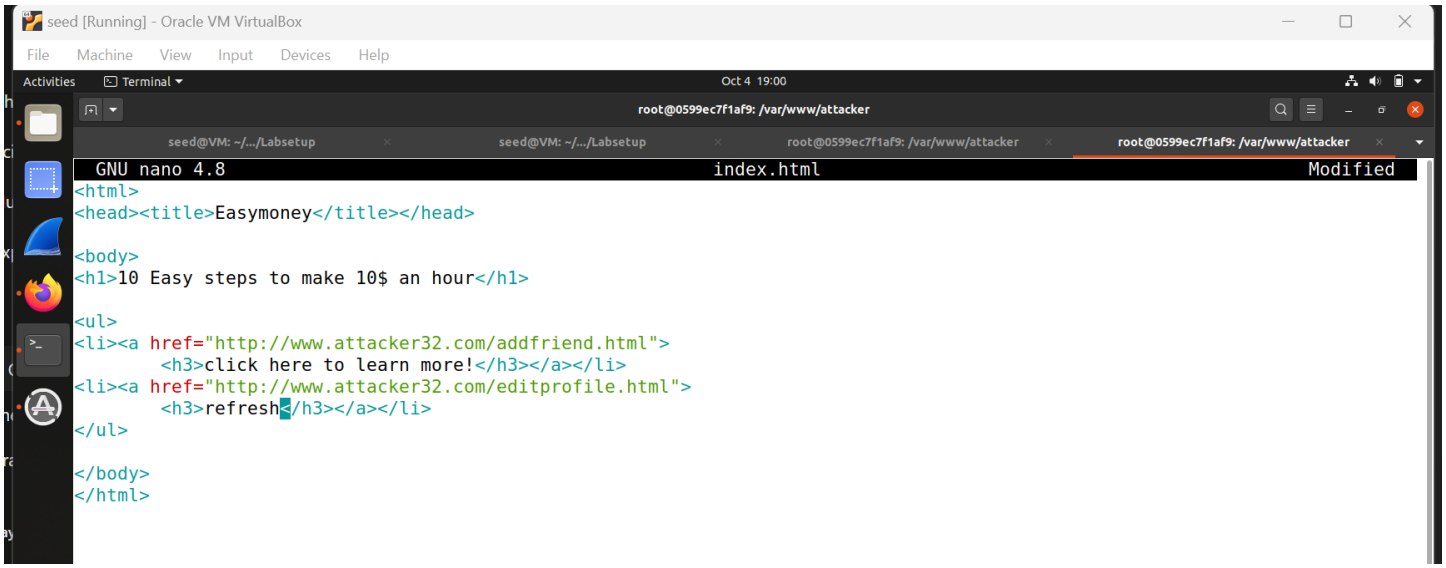

)64,"__elgg_token":"UaZtS3egQisRN6jaT1bugg"}},"session":{"user":{"guid":59,"type":"user","subtype":"user",
ılt/jquery-ui.js"></script><script src="http://www.seed-server.com/cache/1587931381/default/elgg/require

## CONSTRUCTING THE CONTAINER:

## EDITING index.html:

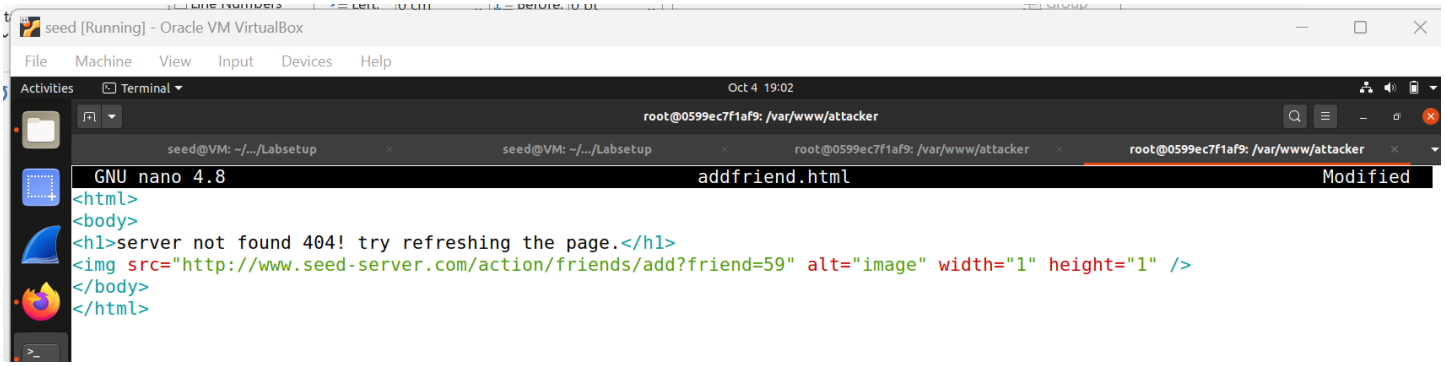
- I edited the index.html file to create a clickable site for Alice to click and do the attack successfully.
- the attacker file is under /var/www
- This code consist of a title which says 'Easymoney'. The body of the code has headers and links. The header is given as "10 easy steps to make 10$ an hour.". There are 2 links in the code. The 1st link access the addfriend.html and the second link accesses the editprofile.html.
- I edited the links of the webpage to http://www.attacker32.com/addfriend.html and http://www.attacker32.com/editprofile.html
- In the 1st link I pasted the captured add friend url and changed the id so that samy can be added as alice's friend.



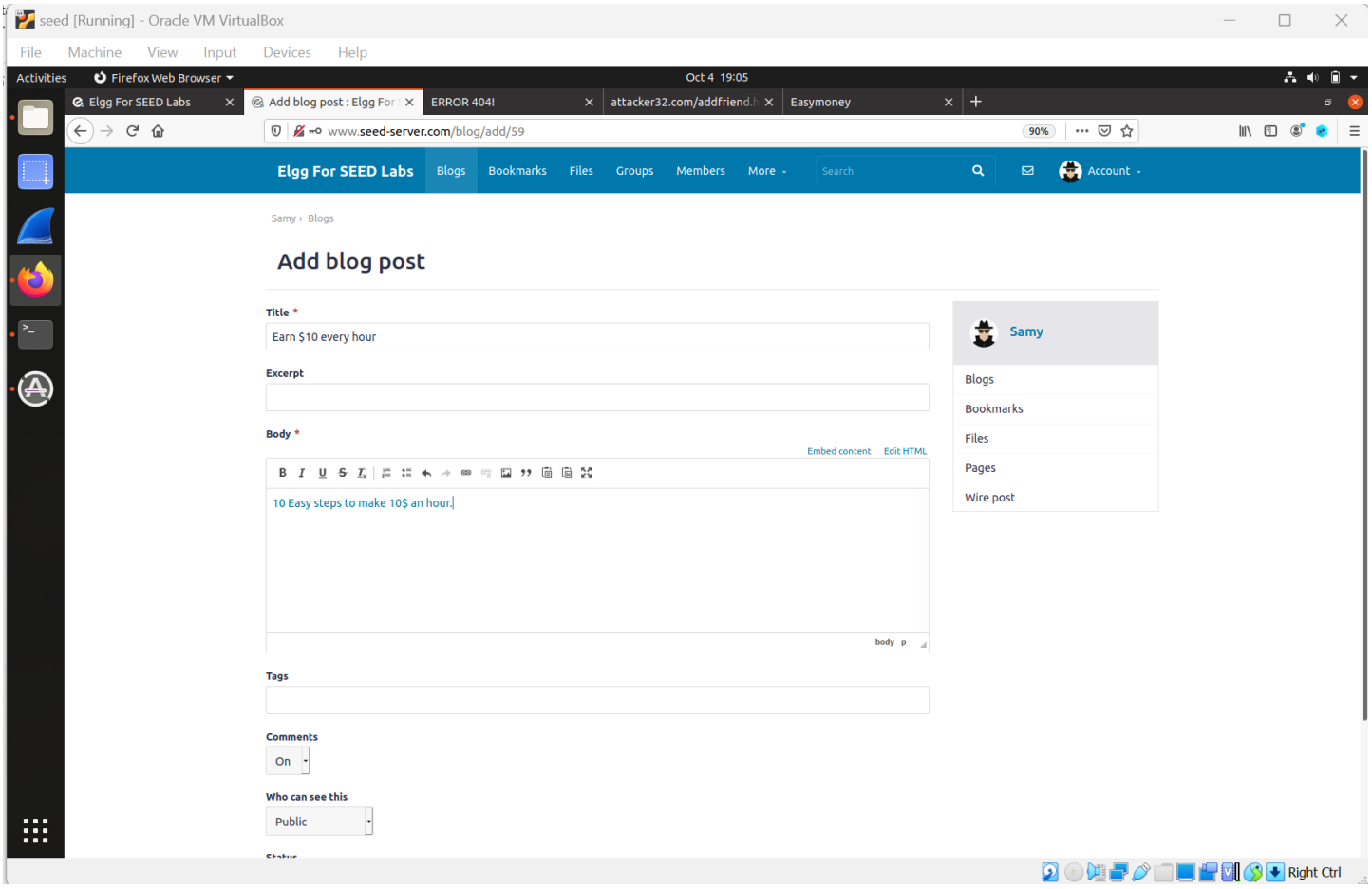



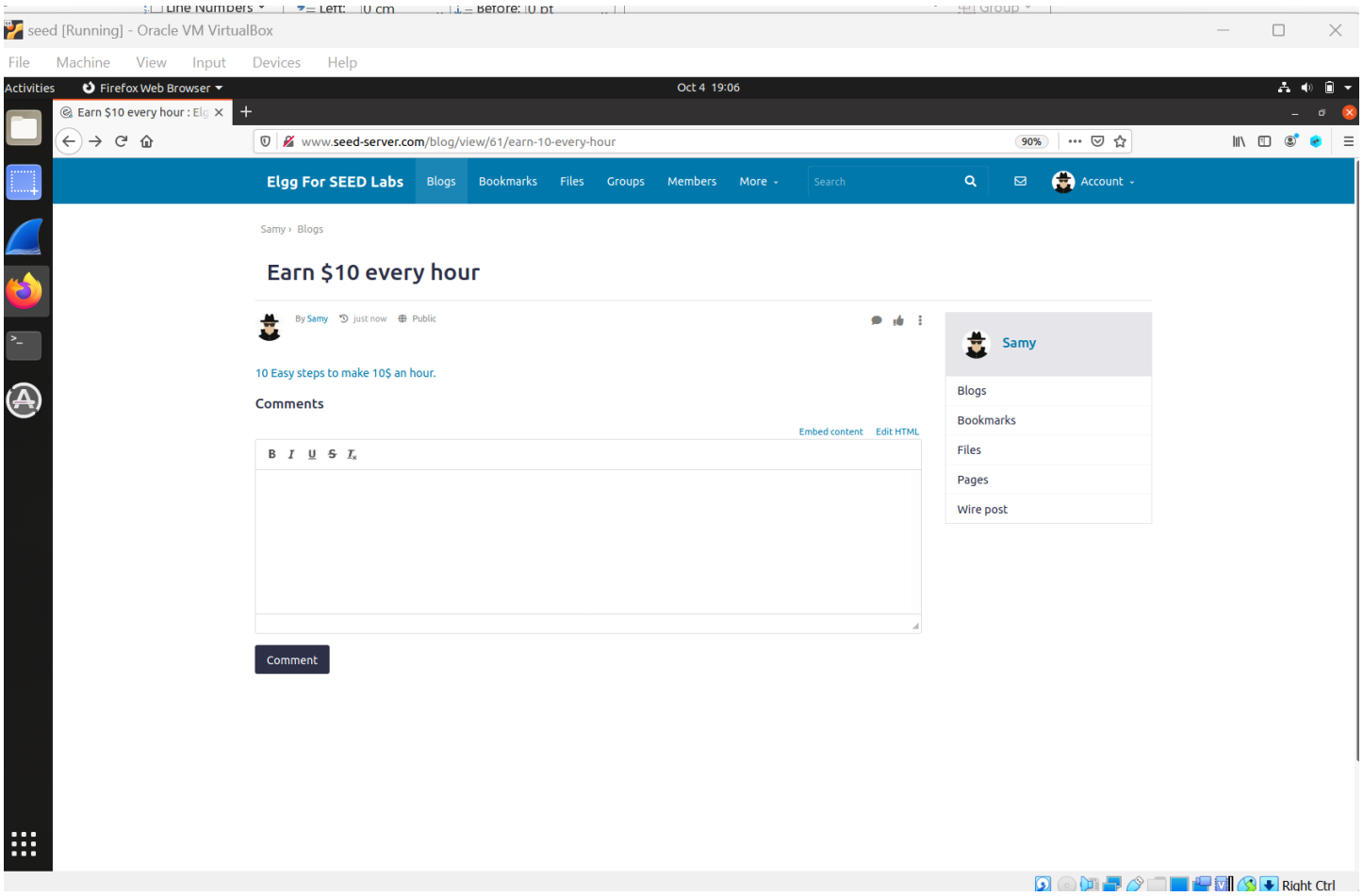
## Editing addfrind.html:

- In the addfriend.html the heading is given as “server not found 404! Try refreshing the page” so that Alice does not raise any suspicion and the URL is pasted as an image so that the attack is triggered on clicking the link. The link consists of the GUID.

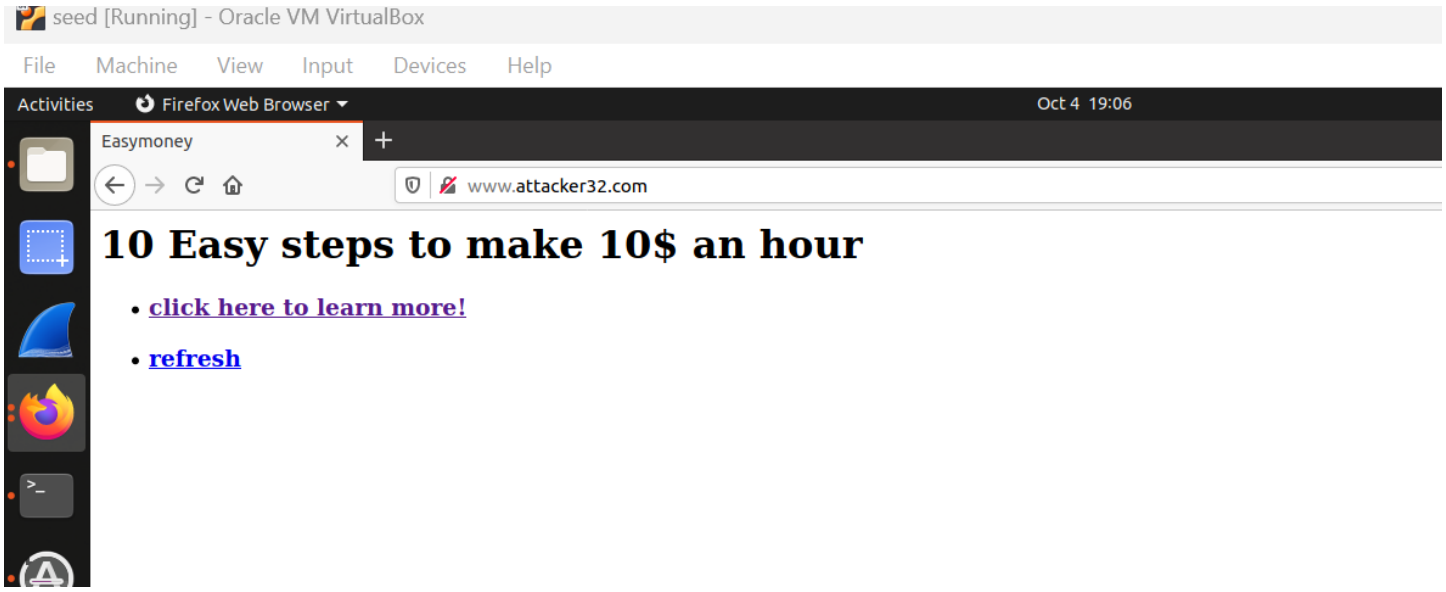



- The link is attached to the blog and posted on the elgg website. When Alice clicks the link it takes her to attacker32’s site.

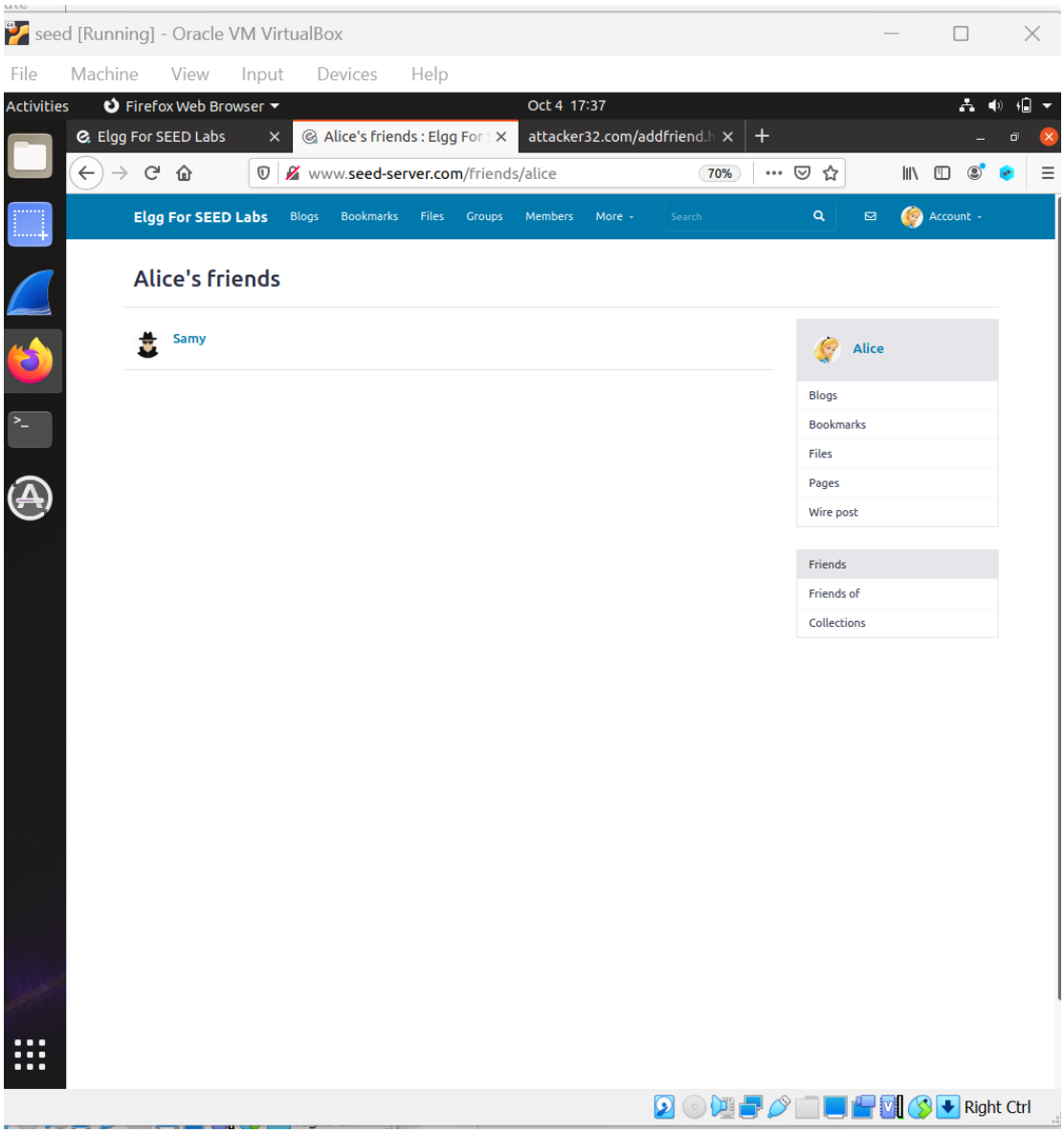





- This is the page after the link is clicked.

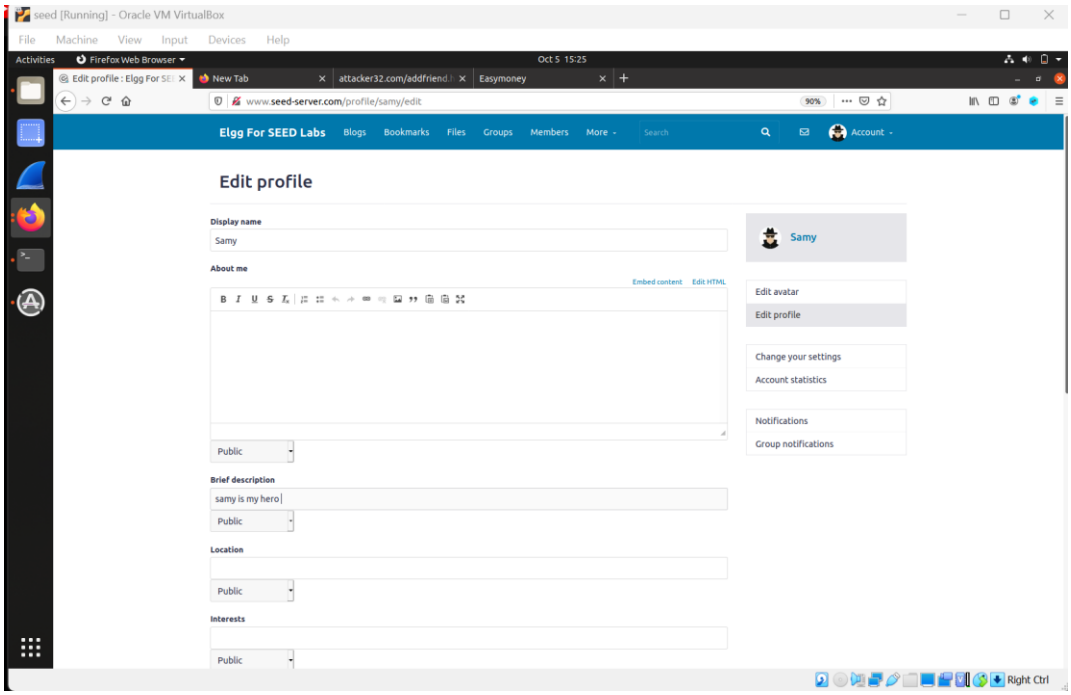


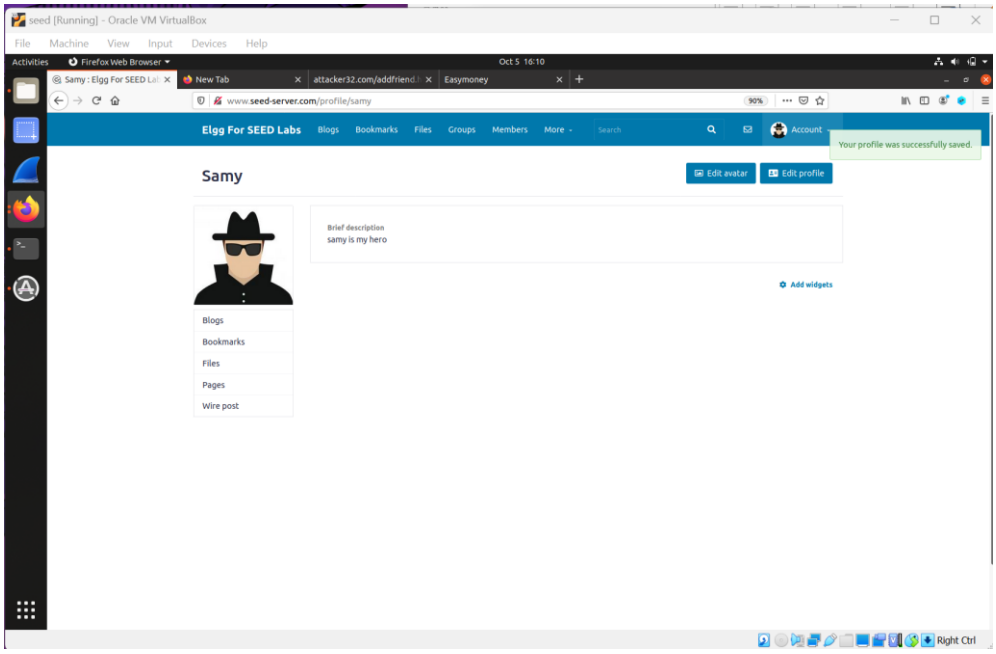

- Now we can see that the attack has been successful and we made alice accepted samy's request.



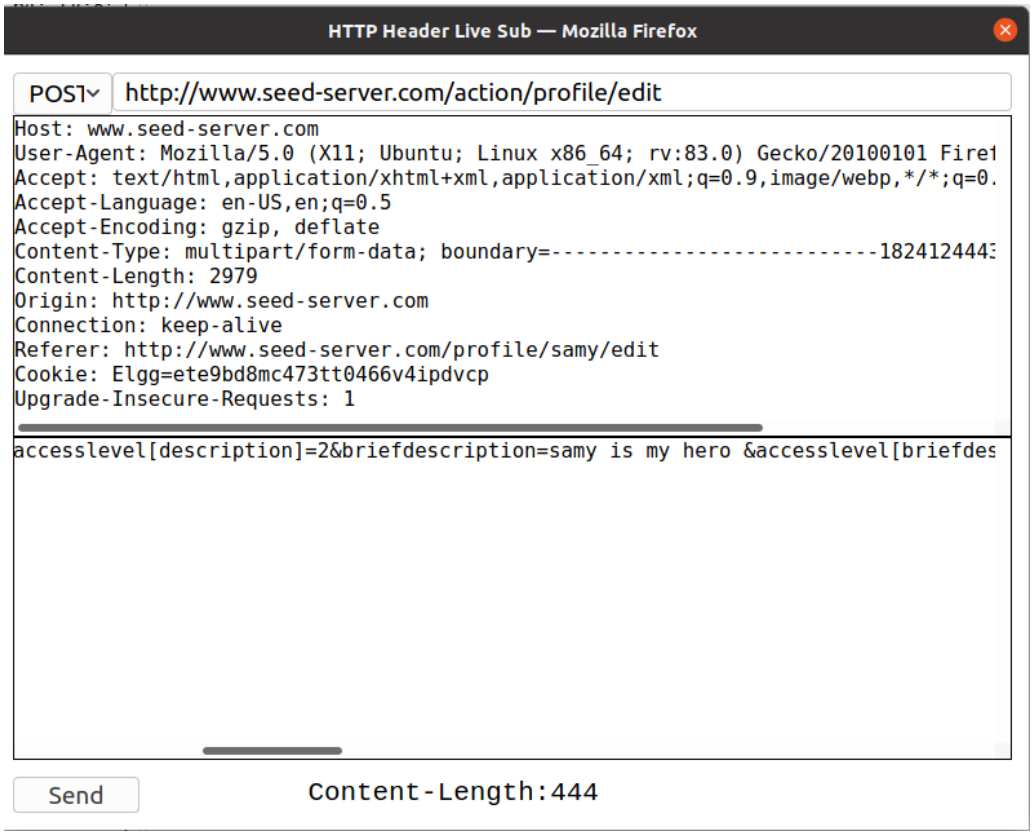
## TASK 3: CSRF ATTACK USING POST REQUEST:

- In this task I am editing Alice's profile.
- The approach here is to make Alice click the link(www.attacker32.com) in the blog that has already been posted.
- I first edited the same's profile by adding a description saying samy is my hero to capture the post request.

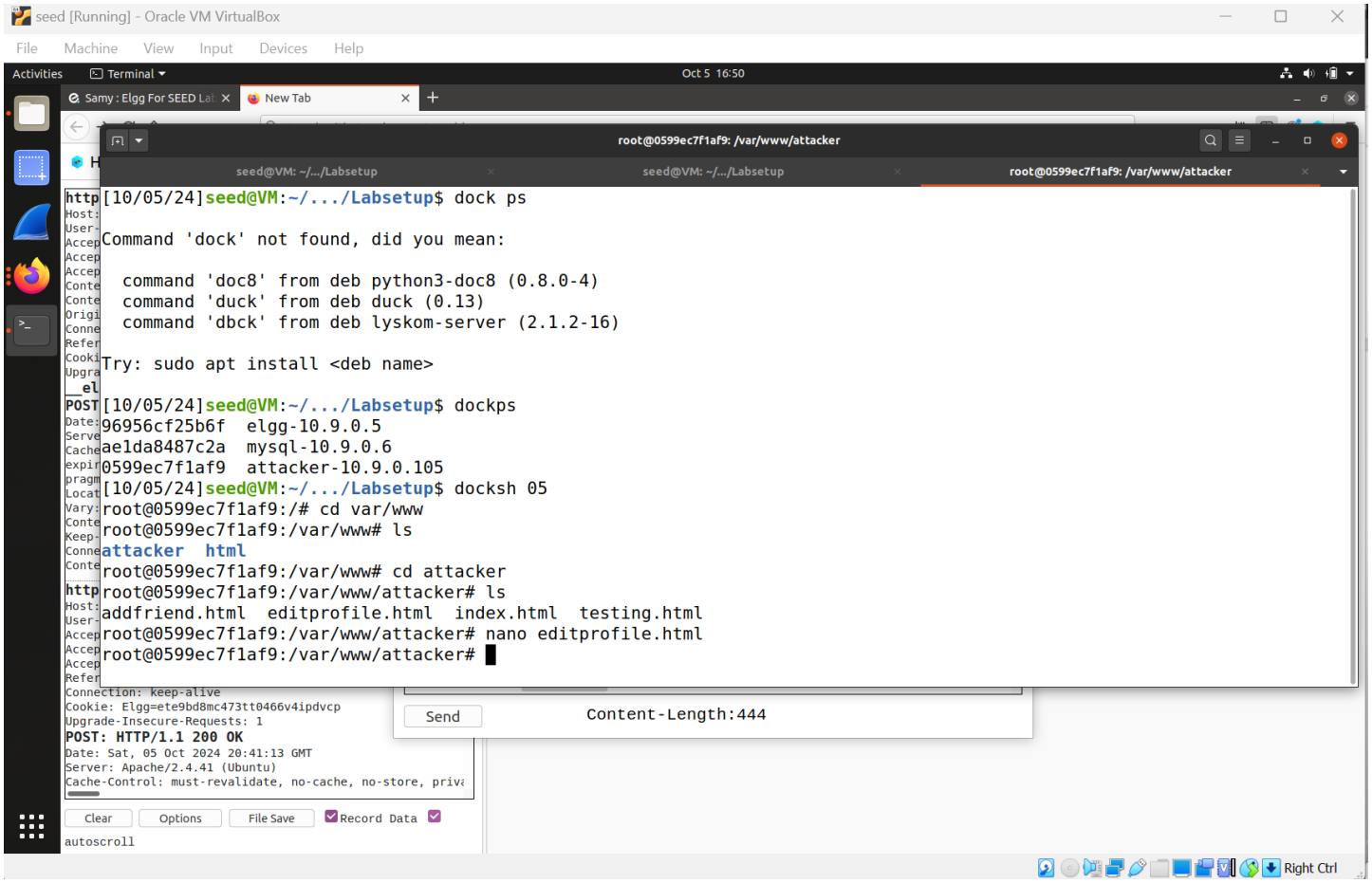


- From the post request I was able to get the profile information to modify and paste it in the Alice's page.

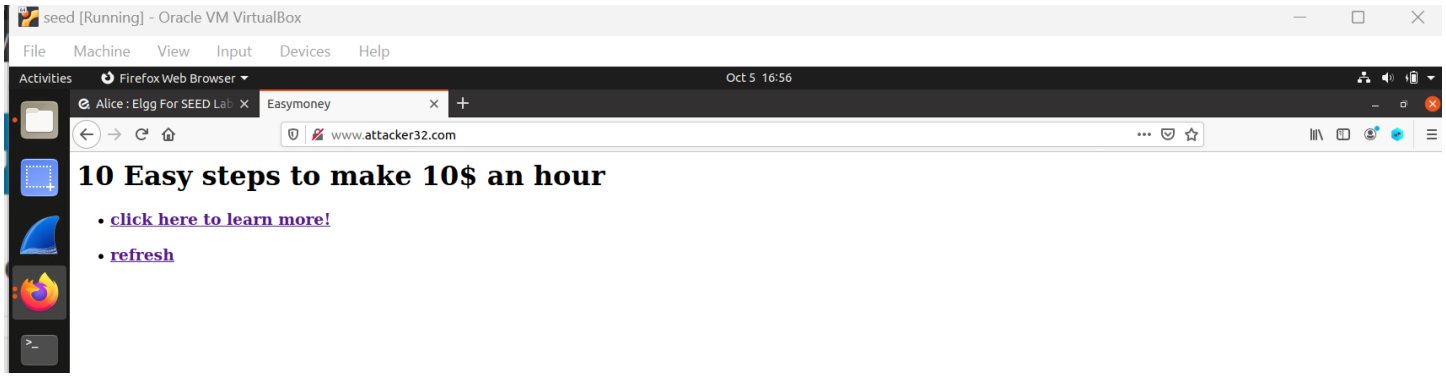



- Now, inside the attacker container, open editprofile.html with the path /var/www/attacker

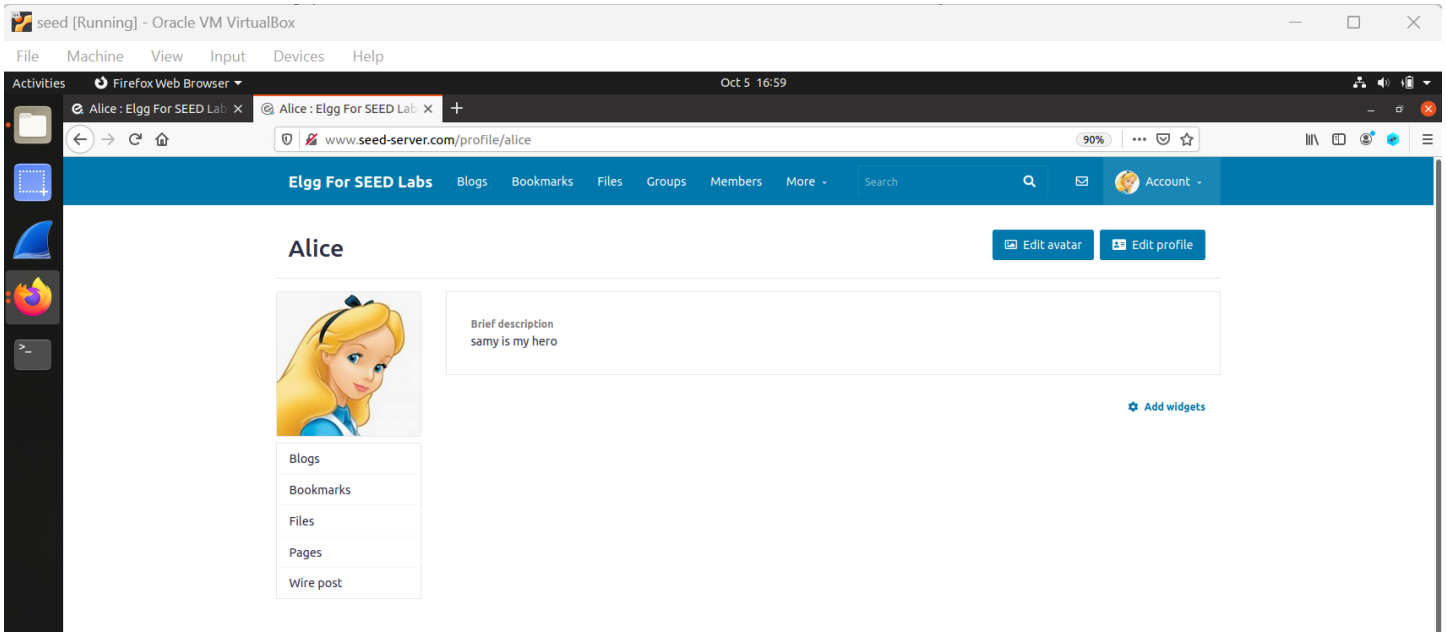


- Here, edit the name to Alice ; enter the description as samy is my hero. ; change the GUID value to 56 ; and paste the url http://www.seed-server.com/action/profile/edit in the p.action parameter.


- Now navigating back to the www.attacker32 webpage, with the 2nd link is clicked by Alice, the profile is automatically edited and saved.




- The profile of Alice is automatically updated saying Samy is my hero. So the attack has been successful.



**QUESTION 1:**

This problem can be solved using the way we found Alice's Guid. We can do a search on Alice's profile and inspect it to get the webpage owner's GUID. This does not require Alice's credentials. If the website does not contain any username then we have to try guessing the password to Alice's profile with the username and look at the http request. If that has GUID we could use that.

**QUESTION 2:**

In this case he would not be able to launch the attack because the malicious page is different than the target website. We do not have access to the target website's source code and hence cannot derive the GUID. And also the GUID is sent to the targeted website's server and not any other website. So we won't be able to get it from the https request too.

**TASK 4: IMPLEMETING A COUNTER MEASURE FOR ELGG.**

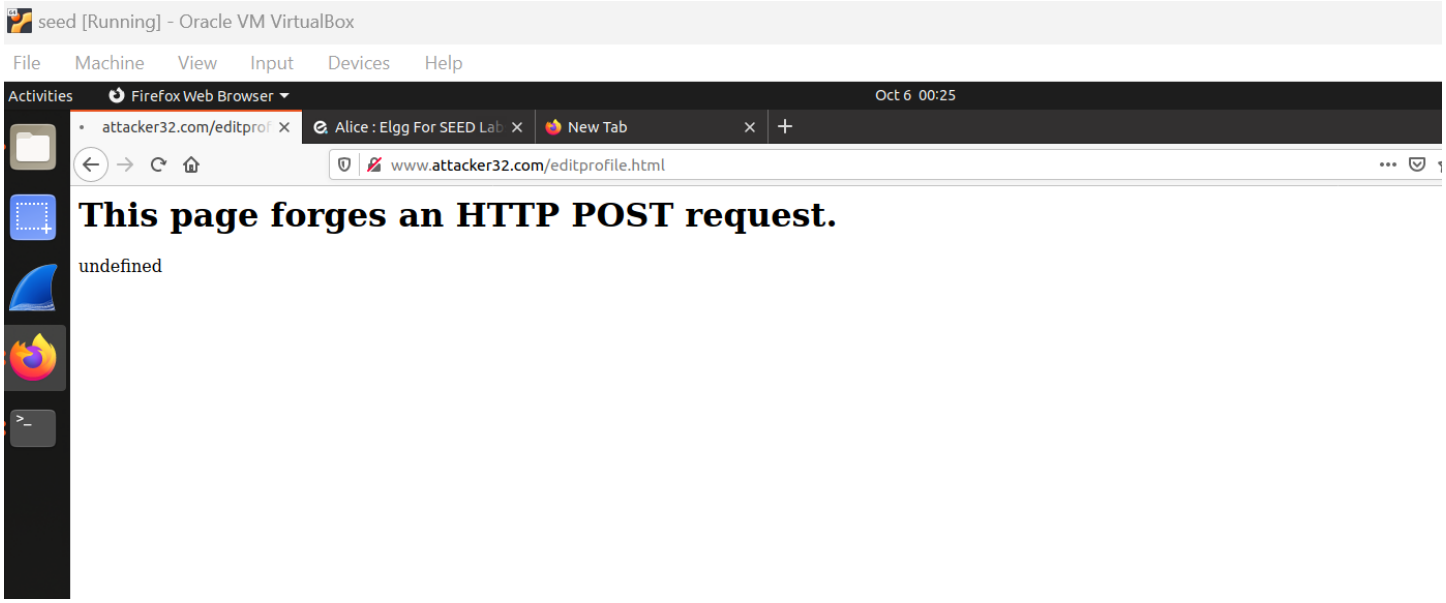
Here, we enable the CSRF attack countermeasure by commenting out the return statement. Commenting out this statement. With this requests made by the user will succeed if the __elgg_ts and __elgg_token values are include. When the CSRF attacks are performed again, they do no succeed. The code is available in the elgg continer under Csrf.php.

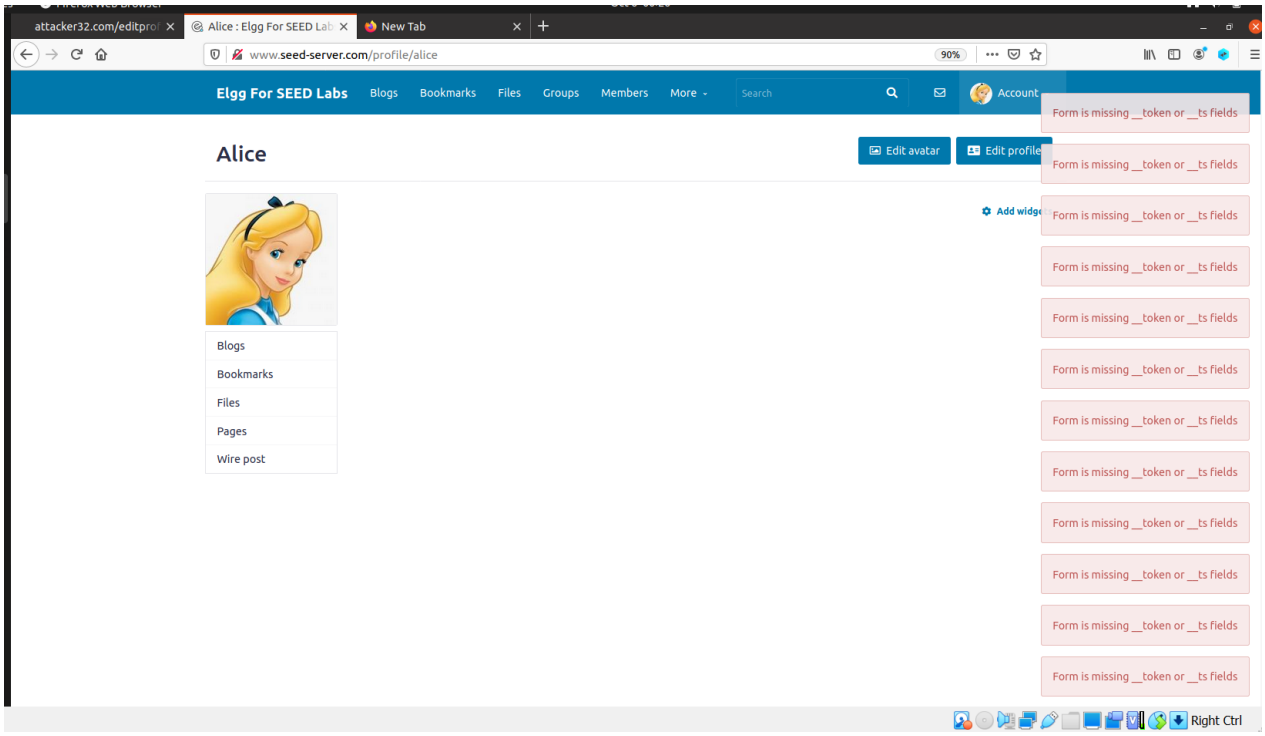
```
root@96956cf25b6f: /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
[10/05/24]seed@VM:~/.../Labsetup$ ls
attacker            image_attacker  image_www
docker-compose.yml  image_mysql     mysql_data
[10/05/24]seed@VM:~/.../Labsetup$ cd attacker
[10/05/24]seed@VM:~/.../attacker$ ls
addfriend.html  editprofile.html  index.html  testing.html
[10/05/24]seed@VM:~/.../attacker$ cd ..
[10/05/24]seed@VM:~/.../Labsetup$ dockps
96956cf25b6f  elgg-10.9.0.5
ae1da8487c2a  mysql-10.9.0.6
0599ec7f1af9  attacker-10.9.0.105
[10/05/24]seed@VM:~/.../Labsetup$ docksh 96
root@96956cf25b6f:/# /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
bash: /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security: Is a directory
root@96956cf25b6f:/# cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
root@96956cf25b6f:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# ls
Base64Url.php  Hmac.php         PasswordGeneratorService.php
Csrf.php       HmacFactory.php  UrlSigner.php
root@96956cf25b6f:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# cat Csrf.php
<?php

namespace Elgg\Security;
```

Commenting out the return :

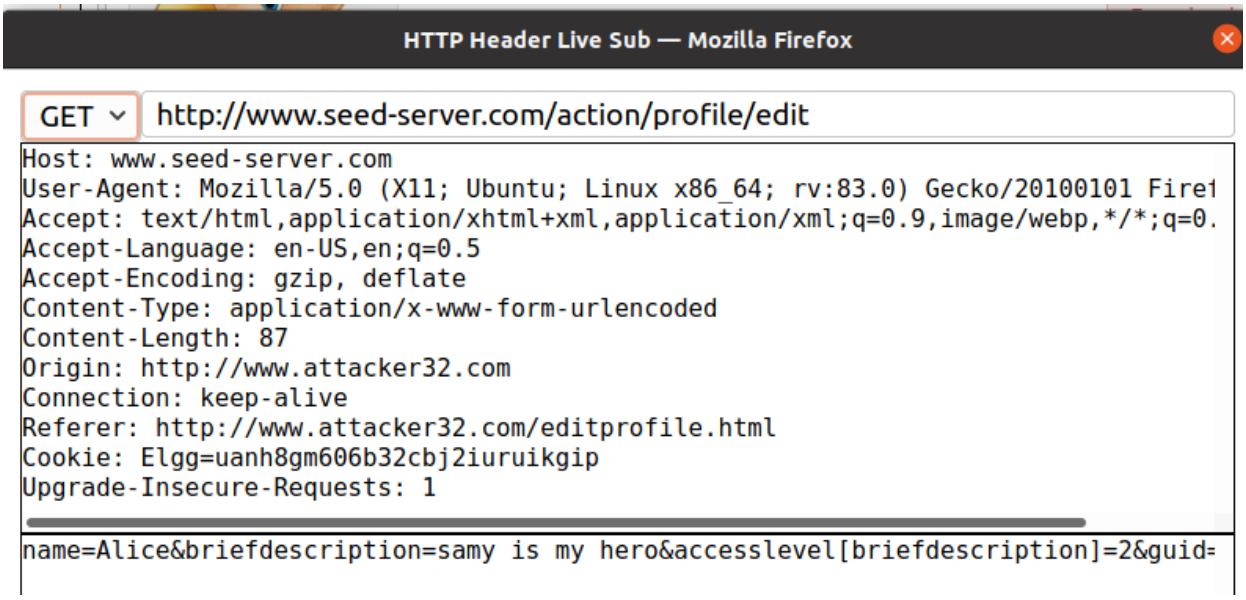
```
GNU nano 4.8                                   Csrf.php                                   Modified

        /**
         * Validate CSRF tokens present in the request
         *
         * @param Request $request Request
         *
         * @return void
         * @throws CsrfException
         */
        public function validate(Request $request) {
                #return; // Added for SEED Labs (disabling the CSRF countermeasure)

                $token = $request->getParam('__elgg_token');
                $ts = $request->getParam('__elgg_ts');

^G Get Help    ^O Write Out   ^W Where Is    ^K Cut Text    ^J Justify     ^C Cur Pos     M-U Undo       M-A Mark Text
^R Read File   ^\ Replace     ^U Paste Text  ^T To Spell    ^_ Go To Line  M-E Redo       M-6 Copy Text
```

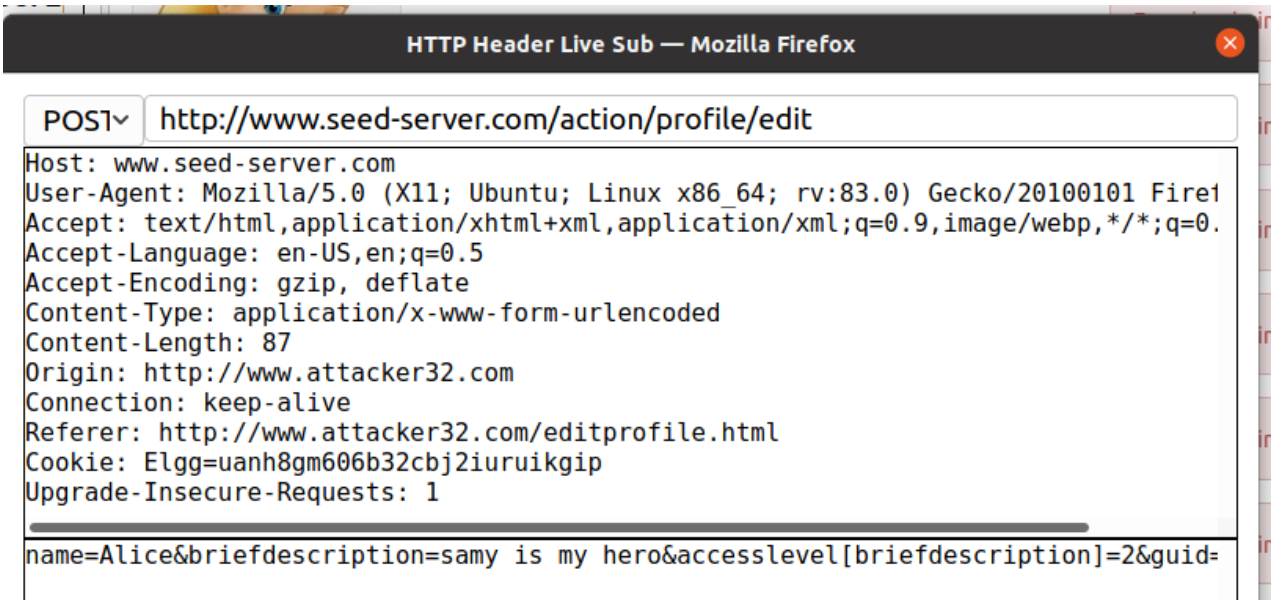
- The edit profile link in attacker32 site shows as undefined.



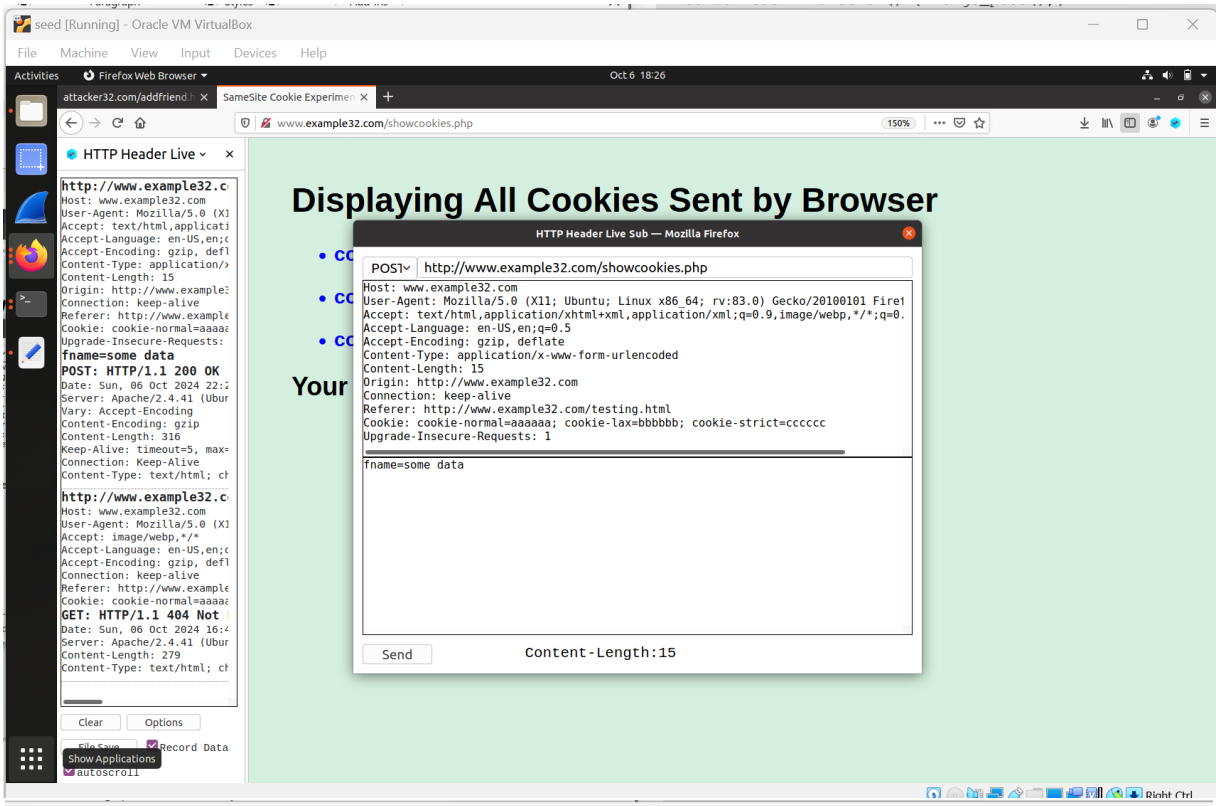
- We get error while performing the attack. We get the error that token and timestamp are missing. On looking at the http requests we can see no token and ts fields. This is because these tokens are set in the elgg's website. Only a request going from the same website will have these parameters. We can see the secret tokens **elgg ts** (elgg timestamp helps in preventing the replay attacks) and **elgg token** (this is a unique token for user's sessions.) when we log into Alice's account. Also no one can guess these values. Finding timestamp is easy but the secret token is a hash value. The secret value is stored in a database is generated randomly. So the attacker cannot find out the secret tokens.

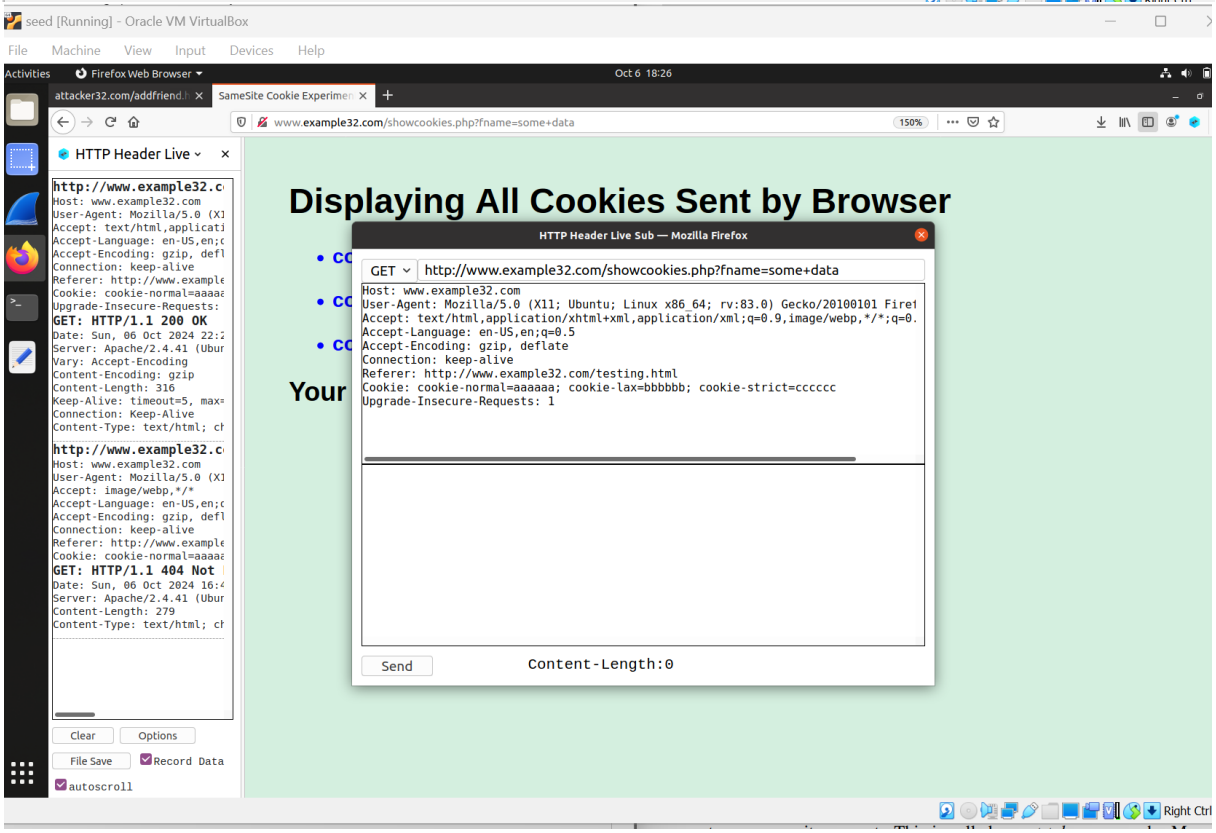


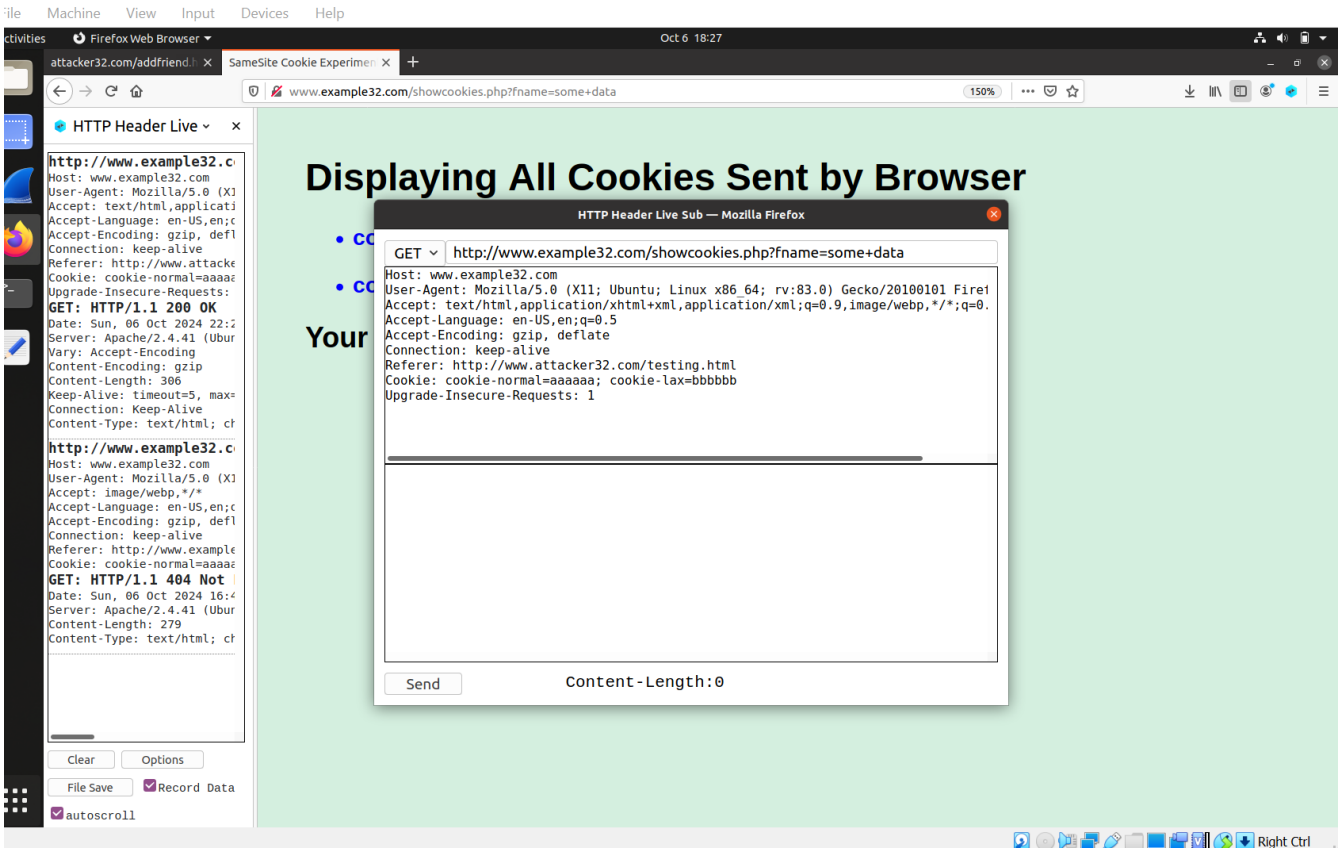
## TASK 5: Experimenting with the SameSite Cookie Method

- Link A provides all three cookies. This is because the request is coming from the same site and allows the browser to include all cookies.

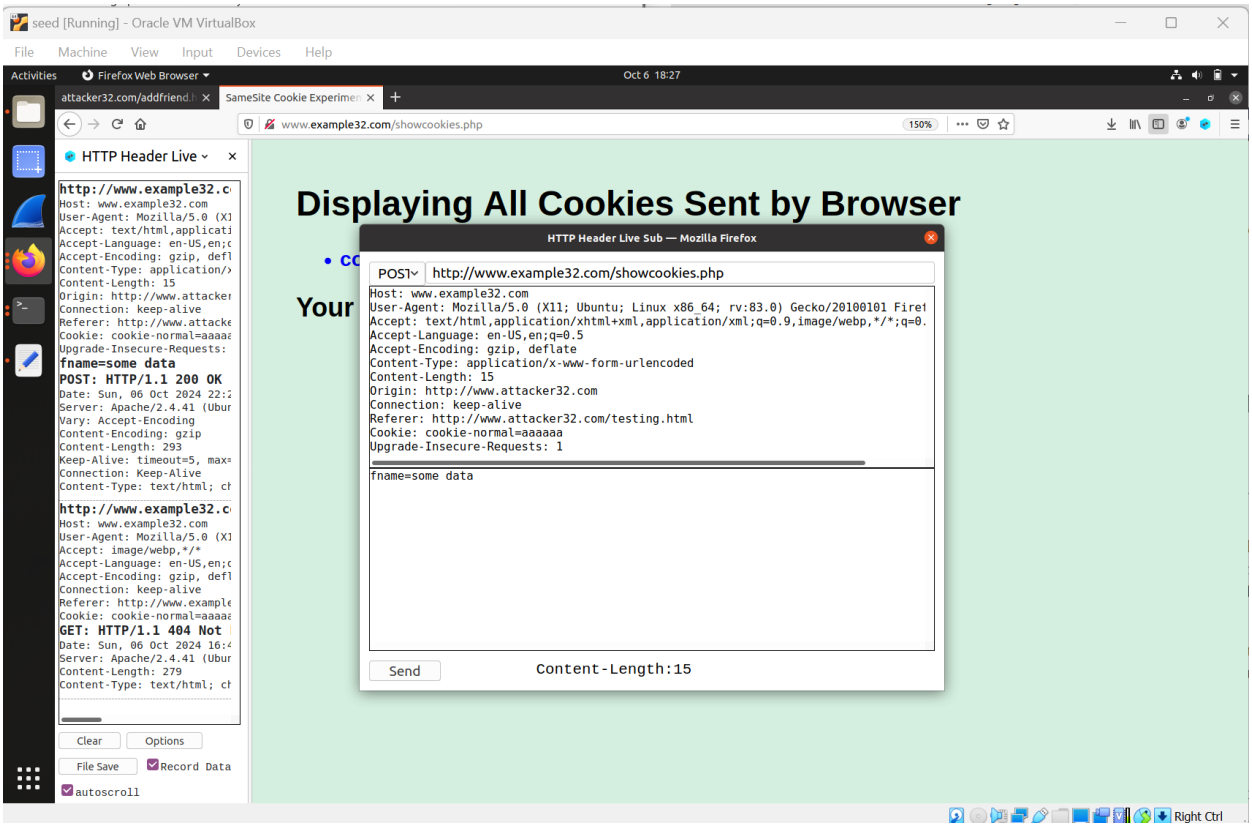


- In Link b (cross site request) all three cookies are not sent. Only the normal and lax are alone sent. This is because the normal does not have any restriction and lax is a top level navigation cookie. But the strict cookie is not sent because it is a cross site request.




- In the post request only the cookie normal is sent because both lax and strict cookies are designed to prevent cross site requests.



**Qs 1: Please describe what you see and explain why some cookies are not sent in certain scenarios**.

- I noticed that after visiting www.example32.com, my browser saved three cookies: cookie-normal, cookie-lax, and cookie-strict.
- On clicking **Link A**, which went to another page on the same site (example32.com), all three cookies were included in the request. This is because the request was coming from the same site, so there were no restrictions.
- However, when I clicked on **Link B**, which led to an external site (attacker32.com). Only the cookie-normal was sent. The cookie-lax was still included because it is counted as a top-level navigation, but the cookie-strict was left out entirely. As this was a request to a different site, the browser didn't send that cookie.

**Qs 2: 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 request.**

- SameSite cookies are a useful security feature that helps servers figure out where a request is coming from.
- The lax cookies are sent along when the user navigates to a new page. If someone tries to make a request from an external site, that cookie won't be included. So, if the server gets a request without the Lax cookie, then the request is cross-site.
- Strict cookies are even more protective. They're only sent when the request is truly coming from the same site, and they're completely left out of any cross-site requests. So, it alerts the server by telling that it is a cross site request.

**Qs 3: Please describe how you would use the SameSite cookie mechanism to help Elgg defend against CSRF attacks. You only need to describe general ideas, and there is no need to implement them.**

- To help Elgg defend against CSRF attacks using the SameSite cookie mechanism we can follow the steps like setting all session cookies to Strict. This means these cookies will only be sent with requests that come from the Elgg site itself. So if someone tries to make a request from a different site, they won't have access to the user's session information.
- For cookies that might be needed during navigation within the site, I would use Lax. This allows those cookies to be sent when users click links to other pages on Elgg, while also preventing them from being sent suring cross site requests.
- I would suggest using anti-CSRF tokens in forms. Even if a cross-site request is made, the request would be blocked without the correct tokens.